http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

SISTEMI OPERATIVI

Concetti ed esempi

S

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Indice

Parte prima Generalità

Capitolo 1 Introduzione

1.1 Cos'è un sistema operativo 3
1.2 Sistemi mainframe 7
1.3 Sistemi da scrivania 11
1.4 Sistemi con più unità d'elaborazione 12
1.5 Sistemi distribuiti 15
1.6 Batterie di sistemi 17
1.7 Sistemi d'elaborazione in tempo reale 18
1.8 Sistemi palmari 20
1.9 Migrazione delle funzioni 21
1.10 Ambienti d'elaborazione 22
1.11 Sommario 24
1.12 Esercizi 25
1.13 Note bibliografiche 26

Capitolo 2 Strutture dei sistemi di calcolo

2.1 Funzionamento di un sistema di calcolo 27
2.2 Struttura di I/O 30
2.3 Struttura della memoria 35
2.4 Gerarchia delle memorie 40
2.5 Architetture di protezione 43
2.6 Struttura delle reti di calcolatori 50
2.7 Sommario 53
2.8 Esercizi 54
2.9 Note bibliografiche 56

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 3 Strutture dei sistemi operativi

3.1 Componenti del sistema 57
3.2 Servizi di un sistema operativo 63
3.3 Chiamate del sistema 65
3.4 Programmi di sistema 74
3.5 Struttura del sistema 77
3.6 Macchine virtuali 82
3.7 Progettazione e realizzazione di un sistema 88
3.8 Generazione di sistemi 91
3.9 Sommario 93
3.10 Esercizi 94
3.11 Note bibliografiche 95

Parte seconda Gestione dei processi

Capitolo 4 Processi

4.1 Concetto di processo 99
4.2 Scheduling dei processi 103
4.3 Operazioni sui processi 108
4.4 Processi cooperanti 112
4.5 Comunicazione tra processi 114
4.6 Comunicazione nei sistemi client-server 121
4.7 Sommario 130
4.8 Esercizi 131
4.9 Note bibliografiche 132

Capitolo 5 Thread

5.1 Introduzione 133
5.2 Modelli di programmazione multithread 136
5.3 Questioni di programmazione multithread 139
5.4 Pthreads 143
5.5 Thread del sistema Solaris 2 145
5.6 Thread nel sistema Windows 2000 147
5.7 Thread nel sistema LINUX 148
5.8 Thread nel linguaggio Java 149
5.9 Sommario 151
5.10 Esercizi 152
5.11 Note bibliografiche 153

Capitolo 6 Scheduling della CPU

6.1 Concetti fondamentali 155
6.2 Criteri di scheduling 159
6.3 Algoritmi di scheduling 160
6.4 Scheduling per sistemi con più unità d'elaborazione 173
6.5 Scheduling per sistemi d'elaborazione in tempo reale 174
6.6 Valutazione degli algoritmi 176
6.7 Modelli di scheduling dei processi 182
6.8 Sommario 189
6.9 Esercizi 190
6.10 Note bibliografiche 192

Capitolo 7 Sincronizzazione dei processi

7.1 Introduzione 195
7.2 Problema della sezione critica 197
7.3 Architetture di sincronizzazione 203
7.4 Semafori 207
7.5 Problemi tipici di sincronizzazione 212
7.6 Regioni critiche 217
7.7 Monitor 220
7.8 Sincronizzazione nei sistemi operativi 228
7.9 Transazioni atomiche 231
7.10 Sommario 241
7.11 Esercizi 242
7.12 Note bibliografiche 246

Capitolo 8 Stallo dei processi

8.1 Modello del sistema 249
8.2 Caratterizzazione delle situazioni di stallo 251
8.3 Metodi per la gestione delle situazioni di stallo 255
8.4 Prevenire le situazioni di stallo 256
8.5 Evitare le situazioni di stallo 259
8.6 Rilevamento delle situazioni di stallo 266
8.7 Ripristino da situazioni di stallo 270
8.8 Sommario 272
8.9 Esercizi 272
8.10 Note bibliografiche 276

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Parte terza Gestione della memoria

Capitolo 9 Gestione della memoria

9.1 Introduzione 279
9.2 Avvicendamento dei processi 286
9.3 Assegnazione contigua della memoria 290
9.4 Paginazione 294
9.5 Segmentazione 311
9.6 Segmentazione con paginazione 317
9.7 Sommario 320
9.8 Esercizi 321
9.9 Note bibliografiche 324

Capitolo 10 Memoria virtuale

10.1 Introduzione 325
10.2 Paginazione su richiesta 328
10.3 Creazione dei processi 336
10.4 Sostituzione delle pagine 339
10.5 Assegnazione dei blocchi di memoria 354
10.6 Attività di paginazione degenere 358
10.7 Esempi tra i sistemi operativi 364
10.8 Altre considerazioni 366
10.9 Sommario 374
10.10 Esercizi 375
10.11 Note bibliografiche 380

Capitolo 11 Interfaccia del file system

11.1 Concetto di file 383
11.2 Metodi d'accesso 391
11.3 Struttura di directory 394
11.4 Montaggio di un file system 406
11.5 Condivisione di file 409
11.6 Protezione 416
11.7 Sommario 420
11.8 Esercizi 421
11.9 Note bibliografiche 423

Capitolo 12 Realizzazione del file system

12.1 Struttura del file system 425
12.2 Realizzazione del file system 427
12.3 Realizzazione delle directory 434
12.4 Metodi di assegnazione 435
12.5 Gestione dello spazio libero 446
12.6 Efficienza e prestazioni 448
12.7 Ripristino 454
12.8 File system con annotazione delle modifiche 456
12.9 NFS 458
12.10 Sommario 464
12.11 Esercizi 466
12.12 Note bibliografiche 468

Parte quarta Sistemi di I/O

Capitolo 13 Sistemi di I/O

13.1 Introduzione 471
13.2 Architetture e dispositivi di I/O 472
13.3 Interfaccia di I/O per le applicazioni 483
13.4 Sottosistema per l'I/O del nucleo 489
13.5 Trasformazione delle richieste di I/O in operazioni dei dispositivi 496
13.6 STREAMS 499
13.7 Prestazioni 501
13.8 Sommario 505
13.9 Esercizi 505
13.10 Note bibliografiche 507

Capitolo 14 Memoria secondaria e terziaria

14.1 Struttura dei dischi 509
14.2 Scheduling del disco 510
14.3 Gestione dell'unità a disco 517
14.4 Gestione dell'area d'avvicendamento 520
14.5 Strutture RAID 524
14.6 Connessione dei dischi 532
14.7 Realizzazione della memoria stabile 534
14.8 Strutture per la memorizzazione terziaria 536
14.9 Sommario 547
14.10 Esercizi 549
14.11 Note bibliografiche 555

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Parte quinta Sistemi distribuiti

Capitolo 15 Strutture dei sistemi distribuiti

15.1 Introduzione 559
15.2 Topologie 566
15.3 Tipi di reti 568
15.4 Comunicazione 572
15.5 Protocolli di comunicazione 578
15.6 Robustezza 583
15.7 Problemi di progettazione 585
15.8 Un esempio di comunicazione in rete 587
15.9 Sommario 589
15.10 Esercizi 590
15.11 Note bibliografiche 592

Capitolo 16 File system distribuiti

16.1 Introduzione 593
16.2 Nominazione e trasparenza 595
16.3 Accesso ai file remoti 599
16.4 Servizio con informazioni di stato e servizio senza informazioni di stato 604
16.5 Replicazione dei file 606
16.6 Un esempio: AFS 607
16.7 Sommario 613
16.8 Esercizi 614
16.9 Note bibliografiche 614

Capitolo 17 Coordinazione distribuita

17.1 Ordinamento degli eventi 615
17.2 Mutua esclusione 618
17.3 Atomicità 621
17.4 Controllo della concorrenza 625
17.5 Gestione delle situazioni di stallo 630
17.6 Algoritmi di elezione 638
17.7 Raggiungimento di un accordo 641
17.8 Sommario 643
17.9 Esercizi 644
17.10 Note bibliografiche 645

Parte sesta Protezione e sicurezza

Capitolo 18 Protezione

18.1 Scopi della protezione 649
18.2 Domìni di protezione 650
18.3 Matrice d'accesso 656
18.4 Realizzazione della matrice d'accesso 660
18.5 Revoca dei diritti d'accesso 663
18.6 Sistemi basati su abilitazioni 665
18.7 Protezione basata sul linguaggio 668
18.8 Sommario 674
18.9 Esercizi 675
18.10 Note bibliografiche 676

Capitolo 19 Sicurezza

19.1 Problema della sicurezza 679
19.2 Autenticazione degli utenti 681
19.3 Minacce ai programmi 686
19.4 Minacce ai sistemi 688
19.5 Migliorare la sicurezza dei sistemi 694
19.6 Rilevamento delle intrusioni 696
19.7 Crittografia 704
19.8 Classificazione della sicurezza dei sistemi di calcolo 710
19.9 Un esempio: Windows NT 711
19.10 Sommario 714
19.11 Esercizi 714
19.12 Note bibliografiche 715

Parte settima Casi di studio

Capitolo 20 Sistema operativo LINUX

20.1 Storia 719
20.2 Princìpi di progettazione 724
20.3 Moduli del nucleo 727
20.4 Gestione dei processi 731
20.5 Scheduling 735
20.6 Gestione della memoria 740
20.7 File system 748

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.8 I/O 754
20.9 Comunicazione fra processi 758
20.10 Strutture di rete 759
20.11 Sicurezza 762
20.12 Sommario 765
20.13 Esercizi 766
20.14 Note bibliografiche 767

Capitolo 21 Windows 2000

21.1 Storia 769
21.2 Princìpi di progettazione 770
21.3 Componenti del sistema 772
21.4 Sottosistemi d'ambiente 790
21.5 File system 793
21.6 Servizi di rete 801
21.7 Interfaccia per il programmatore 808
21.8 Sommario 815
21.9 Esercizi 816
21.10 Note bibliografiche 817

Capitolo 22 Prospettiva storica

22.1 Primi sistemi 819
22.2 Atlas 826
22.3 XDS-940 828
22.4 THE 828
22.5 RC 4000 829
22.6 CTSS 830
22.7 MULTICS 831
22.8 OS/360 832
22.9 Mach 833
22.10 Altri sistemi 835

Bibliografia 837

Riconoscimenti 865

Indice analitico 867

Prefazione all'edizione italiana

A oltre vent'anni dalla sua prima edizione, si può senz'altro affermare che questo testo è un vero classico per l'introduzione a livello universitario dei sistemi operativi. Il fatto che sia giunto oggi alla sesta edizione costituisce di per sé un indice dell'apprezzamento che l'opera ha ricevuto e dell'esperienza acquisita dagli autori nel presentare questi argomenti. Se inoltre si considera che l'autore che ha dato corpo a quest'opera fin dalla prima edizione è Avi Silberschatz – vice-presidente dell'*Information Sciences Research Center* dei famosi Bell Laboratories, destinatario di prestigiosi riconoscimenti per i suoi contributi scientifici nel campo dei sistemi operativi, dei sistemi distribuiti e delle basi di dati, e per le sue doti di divulgatore – non è necessario spendere molte parole per affermare che si tratta di un'opera decisamente valida.

L'esperienza didattica maturata in Italia seguendo quest'ultima edizione ha evidenziato alcune caratteristiche degne di nota: i concetti e le problematiche alla base della progettazione di qualsiasi sistema operativo sono illustrati in modo chiaro e non eccessivamente tecnico. Le scelte concettuali e implementative sono esemplificate citando la soluzione adottata nei sistemi operativi più diffusi (LINUX, Solaris, Windows e altri), e fornendo alcuni capitoli dedicati a singoli sistemi operativi per il lettore che desiderasse un ulteriore approfondimento. Questo approccio è particolarmente apprezzato sia dagli studenti sia da chi opera professionalmente sui sistemi ma non ne conosce i meccanismi interni.

La seconda caratteristica degna di nota è l'enfasi data in questa edizione alla gestione della concorrenza, alle reti e ai sistemi distribuiti che oggi influenzano in modo decisivo la progettazione e le funzionalità dei sistemi operativi: un intero capitolo è dedicato ai thread, e i modelli e le tecniche di comunicazione tra processi su diversi calcolatori sono trattati fin dai primi capitoli; sono illustrati i princìpi della Java Virtual Machine, e il trattamento dei file system include la gestione di file in sistemi di memoria secondaria distribuita; inoltre, tre capitoli sono dedicati in modo specifico ai sistemi distribuiti. Ne risulta un approccio moderno ed efficace allo studio dei sistemi operativi.

La quantità di materiale contenuto nel testo, integrato da un utilissimo sito web, è decisamente superiore a quanto richiesto da un usuale programma di studio di un corso universitario di primo livello sui sistemi operativi. Il testo fornisce dunque ampi approfondimenti per gli studenti e spunti per i docenti che volessero enfatizzare meno argomenti classici lasciando maggior spazio a temi più attuali come la sicurezza, i file system distribuiti o i problemi di coordinamento nei sistemi distribuiti.

Claudio Bettini
Professore Associato
Docente di Sistemi Operativi e di Sistemi Distribuiti
Università degli Studi di Milano – Dip. di Scienze dell'Informazione

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Prefazione

Così come i sistemi operativi sono una parte essenziale dei sistemi di calcolo, un corso sui sistemi operativi è una parte essenziale di un percorso di studio dell'informatica. Con i calcolatori ormai presenti praticamente in ogni campo, dai giochi ai più complessi e raffinati strumenti di pianificazione impiegati dagli enti governativi e dalle grandi multinazionali, questo campo si evolve a un ritmo vertiginoso. D'altra parte, i concetti fondamentali restano abbastanza chiari; su questi si fonda la trattazione svolta in questo testo.

Il presente libro è stato concepito e scritto per un corso introduttivo sui sistemi operativi, dei quali fornisce una chiara descrizione dei *concetti* di base. Prerequisiti essenziali per la comprensione del testo da parte del lettore sono la familiarità con l'organizzazione di un generico calcolatore, la conoscenza di un linguaggio di programmazione ad alto livello, ad esempio il Linguaggio C, e delle principali strutture di dati. Nel Capitolo 2 s'introducono le nozioni riguardanti l'architettura dei calcolatori necessarie alla comprensione dei sistemi operativi. Benché siano scritti prevalentemente in Linguaggio C e talvolta in Java, gli algoritmi discussi nel testo sono facilmente comprensibili anche senza una conoscenza approfondita di questi linguaggi di programmazione.

I concetti fondamentali e gli algoritmi trattati in questo testo spesso si basano su quelli impiegati nei sistemi operativi disponibili in commercio. Si è in ogni modo cercato di presentare questi concetti e algoritmi in una forma generale, non legata a un particolare sistema operativo. Nel libro sono presenti molti esempi che riguardano i più diffusi sistemi operativi, tra i quali Solaris 2, della Sun Microsystems, LINUX, Microsoft MS-DOS, Windows NT e Windows 2000, IBM OS/2, Apple Macintosh Operating System.

La presentazione si svolge attraverso descrizioni intuitive. Sono trattati importanti risultati teorici ma le dimostrazioni formali sono state omesse. Le note bibliografiche contengono i riferimenti agli articoli di ricerca in cui tali risultati sono stati presentati e dimostrati per la prima volta, e sono inoltre presenti suggerimenti per ulteriori letture. Al posto delle dimostrazioni, si è fatto ricorso a illustrazioni ed esempi per suggerire perché ci si può attendere la correttezza dei risultati presentati.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Contenuto del libro

Il testo è suddiviso nelle seguenti parti:

- **Generalità.** Nei Capitoli dall'1 al 3 si spiega cosa *sono* i sistemi operativi, cosa *fanno* e come sono *progettati* e *realizzati*. Questa spiegazione è fornita attraverso una discussione sull'evoluzione dei concetti dei sistemi operativi, sulle comuni caratteristiche e su quei servizi che un sistema operativo deve fornire agli utenti e su quelli che deve fornire agli amministratori del sistema. Essendo privi di riferimenti al funzionamento interno, questi capitoli sono consigliabili a chiunque voglia sapere cos'è un sistema operativo, senza soffermarsi sui dettagli degli algoritmi che ne controllano il funzionamento. Inoltre, nel Capitolo 2 s'introducono le nozioni sulle architetture dei calcolatori necessarie alla comprensione dei sistemi operativi. I lettori che già possiedono una certa conoscenza dell'architettura dei calcolatori, in particolare dei dispositivi di I/O, delle tecniche DMA e del funzionamento delle unità a disco, si possono limitare a dare una rapida occhiata agli argomenti trattati in questo capitolo o addirittura passare a quello successivo.
- **Gestione dei processi.** Nei Capitoli dal 4 all'8 si descrivono i concetti di processo e concorrenza che costituiscono il fondamento dei moderni sistemi operativi. Per *processo* s'intende l'unità di lavoro di un sistema, il quale sarà caratterizzato da un insieme di processi eseguiti in modo *concorrente*, alcuni dei quali sono parte del sistema operativo (quelli incaricati di eseguire il codice di sistema), mentre i rimanenti sono i processi utenti (il cui scopo è, appunto, l'esecuzione del codice d'utente). In questi capitoli si affrontano i differenti metodi impiegati per lo scheduling e la sincronizzazione dei processi, la comunicazione tra processi e la gestione delle situazioni di stallo. Tra questi argomenti è compresa una discussione sui thread.
- **Gestione della memoria.** Nei Capitoli dal 9 al 12 è trattata la gestione dei mezzi di memoria nelle loro varie forme. Durante la propria esecuzione, un processo deve risiedere, almeno parzialmente, nella memoria centrale. Al fine di migliorare sia l'utilizzo della CPU sia la velocità di risposta ai propri utenti, un calcolatore deve essere in grado di mantenere contemporaneamente più processi nella memoria. Esistono molti schemi per la gestione della memoria centrale; questi schemi riflettono diverse strategie di gestione della memoria, e l'efficacia dei diversi algoritmi dipende dal particolare contesto nel quale si applicano. Poiché la memoria centrale è di solito troppo piccola per poter contenere tutti i dati e i programmi, e non può memorizzare informazioni in modo permanente, un sistema di calcolo deve essere corredato da una memoria secondaria a sostegno della memoria centrale. La maggior parte dei moderni sistemi impiega unità a disco come principale mezzo per la memorizzazione locale delle informazioni, siano esse programmi o dati. Il file system realizza le funzioni mediante le quali è possibile memorizzare e accedere direttamente alle informazioni residenti nei dischi. Questi capitoli descrivono gli algoritmi fondamentali per la gestione della memoria, così che il lettore possa comprenderne proprietà, vantaggi e svantaggi.

- **Sistemi di I/O.** I Capitoli 13 e 14 descrivono i dispositivi che si connettono a un calcolatore. Tali dispositivi differiscono per molteplici aspetti, così come per vari aspetti sono i più lenti tra i principali componenti dei calcolatori. Poiché i dispositivi possono essere assai diversi, il sistema operativo deve fornire alle applicazioni un'ampia gamma di funzioni che consentano di controllare tutti gli aspetti dei dispositivi stessi. Questa parte tratta i sistemi di I/O in modo approfondito: progettazione dei sistemi di I/O, interfacce, strutture e funzioni interne di sistema. Siccome i dispositivi di I/O sono una strozzatura per le prestazioni, si esaminano gli aspetti riguardanti le prestazioni. Sono inoltre trattati gli argomenti relativi alla memoria secondaria e terziaria.
- **Sistemi distribuiti.** I Capitoli dal 15 al 17 trattano i *sistemi distribuiti*. Con tale locuzione di solito ci si riferisce a un insieme di unità d'elaborazione che operano senza condivisione di memoria o di clock. Un sistema di questo tipo è in grado di mettere a disposizione dei propri utenti le varie risorse da esso controllate. La possibilità d'accesso a risorse condivise consente un incremento delle prestazioni del sistema, oltre che una maggiore affidabilità e disponibilità di dati e programmi. Attraverso l'uso di file system distribuiti, utenti, server e unità di memorizzazione si possono dislocare tra i vari siti del sistema distribuito. Tale tipo di sistema deve quindi provvedere alla realizzazione di diversi meccanismi per la sincronizzazione e la comunicazione capaci di gestire particolari questioni legate alle situazioni di stallo e alle situazioni critiche che non si incontrano nei sistemi centralizzati.
- **Protezione e sicurezza.** I Capitoli 18 e 19 spiegano gli aspetti principali che riguardano la protezione e la sicurezza dei sistemi d'elaborazione. Tutti i processi di un sistema operativo devono essere reciprocamente protetti; a tale scopo esistono meccanismi capaci di garantire che solo i processi autorizzati dal sistema operativo possano impiegare le risorse del sistema, come file, segmenti di memoria, CPU, e altre risorse. La protezione è il meccanismo attraverso il quale il sistema controlla l'accesso alle risorse da parte di programmi, processi o utenti. Tale meccanismo deve fornire un metodo per definire i controlli e i vincoli ai quali gli utenti devono essere sottoposti, e i mezzi per realizzarli. La sicurezza, invece, consiste nel proteggere sia le informazioni memorizzate all'interno del sistema (dati e codice) sia le risorse fisiche del sistema di calcolo da accessi non autorizzati, tentativi di alterazione o distruzione e dal verificarsi di incongruenze nel funzionamento.
- **Casi di studio.** Nei Capitoli dal 20 al 22 del libro, e nelle Appendici dalla A alla C, presenti nel sito Web dedicato a questo volume, si integrano i concetti esposti descrivendo alcuni sistemi operativi reali, tra i quali i sistemi LINUX, Windows 2000, Free BSD, Mach, e Nachos. I sistemi LINUX e FreeBSD sono stati scelti poiché UNIX è un sistema operativo sufficientemente ridotto da poter essere studiato e compreso nei dettagli, pur non essendo un sistema operativo giocattolo. La maggior parte dei suoi algoritmi interni è stata scelta sulla base della *semplicità* e non per le prestazioni o la raffinatezza. Sia il LINUX sia il FreeBSD sono tra l'altro presenti in tutti i diparti-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

menti d'informatica delle principali università, perciò un gran numero di studenti può accedervi liberamente. Il sistema Windows 2000 è stato scelto perché offre l'opportunità di studiare un sistema operativo moderno progettato e realizzato in modo radicalmente diverso dallo UNIX. Il sistema Nachos è trattato poiché un ottimo metodo per comprendere a fondo i concetti alla base di un sistema operativo moderno consiste nell'analizzare il codice sorgente di un sistema operativo, studiarne il funzionamento a basso livello, ricostruirne parti significative e osservare gli effetti del proprio lavoro. Nel Capitolo 22 sono descritti brevemente alcuni tra i sistemi operativi che hanno maggiormente influenzato l'evoluzione di questo settore.

Sesta edizione

La sesta edizione di questo testo è stata scritta considerando i numerosi suggerimenti ricevuti riguardo alle precedenti edizioni del testo, insieme con le osservazioni proprie degli Autori, sul mutevole campo dei sistemi operativi e delle reti di calcolatori. È stato riscritto molto nella maggior parte dei capitoli, aggiornando il vecchio materiale ed eliminando quello non più interessante. È stato riscritto in Linguaggio C tutto il codice in Pascal presente nelle precedenti edizioni; è presente anche qualche esempio di codice scritto in Java. È stata fatta una revisione e sono state apportate sostanziali modifiche nella maggior parte dei capitoli. Le modifiche più importanti consistono nell'aggiunta di due nuovi capitoli e nella riorganizzazione della trattazione dei sistemi distribuiti. Poiché le reti di calcolatori e i sistemi distribuiti sono sempre più importanti nell'ambito dei sistemi operativi, una parte di tale materiale, in particolare la trattazione dei sistemi client-server, è stata inserita in capitoli precedenti.

Sono state apportate sostanziali revisioni e modifiche della struttura organizzativa nei seguenti capitoli:

- **Capitolo 3, Strutture dei sistemi operativi.** Contiene un paragrafo che tratta la macchina virtuale del linguaggio Java.
- **Capitolo 4, Processi.** Contiene nuovi paragrafi che descrivono le socket e le RPC.
- **Capitolo 5, Thread.** È un nuovo capitolo che tratta i sistemi di calcolo multithread. Molti sistemi operativi moderni dispongono di funzioni che consentono ai processi di contenere più thread di controllo.
- **Capitoli dal 6 al 10, Processi.** Sono i Capitoli dal 5 al 9, rispettivamente, della precedente edizione.
- **Capitolo 11, Interfaccia del file system.** È il Capitolo 10 della precedente edizione. È stato modificato in modo sostanziale e ora comprende la trattazione dell'NFS che nella precedente edizione stava nel Capitolo 16.

- **Capitoli 12 e 13.** Sono, rispettivamente, i Capitoli 11 e 12 della precedente edizione. Il Capitolo 13 contiene un nuovo paragrafo sul meccanismo STREAMS.
- **Capitolo 14, Memoria secondaria e terziaria.** Comprende i Capitoli 13 e 14 della precedente edizione.
- **Capitolo 15, Strutture dei sistemi distribuiti.** Comprende i Capitoli 15 e 16 della precedente edizione.
- **Capitolo 19, Sicurezza.** È il Capitolo 20 della precedente edizione.
- **Capitolo 20, Sistema operativo LINUX.** È il Capitolo 23 della precedente edizione aggiornato ai recenti sviluppi del sistema LINUX.
- **Capitolo 21, Sistema operativo Windows 2000.** Sostituisce il Capitolo 24 sul sistema Windows NT della precedente edizione.
- **Capitolo 22, Prospettiva storica.** È il Capitolo 25 della precedente edizione.
- **Appendice A.** È il Capitolo 21 sul sistema UNIX della precedente edizione, aggiornato al sistema FreeBSD.
- **Appendice B.** È il Capitolo 22 sul sistema operativo Mach della precedente edizione.
- **Appendice C.** È l'Appendice A sul sistema operativo Nachos della precedente edizione.

Le tre appendici sono disponibili nelle pagine Web dedicate al testo.

Supplementi didattici e pagine Web

Le pagine Web dedicate a questo testo contengono le tre appendici (in lingua inglese), la serie di diapositive che accompagna il testo, nei formati PDF e PowerPoint, l'errata corrige più aggiornata, e i collegamenti alle pagine Web degli autori. L'editore John Wiley & Sons tiene le pagine Web sul testo all'indirizzo

http://www.wiley.com/college/silberschatz

Per ottenere i supplementi riservati, si può contattare il proprio rappresentante commerciale alla pagina Web "Find a Rep?", all'indirizzo

http://www.jsw-edcv.wiley.com/college/findarep

Indirizzario e supplementi

È disponibile un ambiente in cui gli utenti possono comunicare tra loro e con noi. Esiste un indirizzario dei lettori col seguente indirizzo: os-book@research.bell-labs.com. Per essere inseriti in tale indirizzario è sufficiente inviare un messaggio ad avi@bell-labs.com indicando il proprio nome, appartenenza, e indirizzo di posta elettronica.

http://www.pdfultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Suggerimenti

In questa nuova edizione si è cercato di eliminare tutti gli errori ma, come accade con i sistemi operativi, qualche oscuro baco probabilmente rimane. È benvenuta qualsiasi segnalazione riguardante errori o omissioni riscontrate nel testo; e sono altrettanto benvenuti il suggerimento di miglioramenti e i contributi per nuovi esercizi. La corrispondenza deve essere inviata ad Avi Silberschatz, vicepresidente dell'Information Sciences Research Center, MH 2T-310, Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974 (avi@bell-labs.com).

Ringraziamenti

Questo testo deriva dalle precedenti edizioni, le prime tre delle quali sono state scritte insieme con James Peterson. Tra le altre persone che sono state d'aiuto per quanto riguarda le precedenti edizioni ci sono Hamid Arabnia, Randy Bentson, David Black, Joseph Boykin, Jeff Brumfield, Gael Buckley, P. C. Capon, John Carpenter, Thomas Casavant, Ajoy Kumar Datta, Joe Deck, Sudarshan K. Dhall, Thomas Doeppner, Caleb Drake, M. Rasit Eskicioglu, Hans Flack, Robert Fowler, G. Scott Graham, Rebecca Hartman, Wayne Hathaway, Christopher Haynes, Mark Holliday, Richard Kieburz, Carol Kroll, Thomas LeBlanc, John Leggett, Jerrold Leichter, Ted Leung, Gary Lippman, Carolyn Miller, Michael Molloy, Yoichi Muraoka, Jim M. Ng, Banu Ozden, Ed Posnak, Boris Putanec, Charles Qualline, John Quarterman, Jesse St. Laurent, John Stankovich, Adam Stauffer, Steven Stepanek, Hal Stern, Louis Stevens, Pete Thomas, David Umbaugh, Steve Vinoski, Tommy Wagner, John Werth e J. S. Weston.

Gli autori ringraziano le seguenti persone che hanno contribuito a questa edizione: Bruce Hillyer è stato d'aiuto nella riscrittura dei Capitoli 2, 12, 13 e 14; Mike Reiter ha rivisto ed è stato d'aiuto nella riscrittura del Capitolo 18; il Capitolo 14 deriva parzialmente da un articolo di Hillyer e Silberschatz [1996]; il Capitolo 17 deriva parzialmente da un articolo di Levy e Silberschatz [1990]; il Capitolo 20 da un manoscritto inedito di Stephen Tweedie; il Capitolo 21 da un manoscritto inedito di Cliff Martin; lo stesso Cliff Martin è stato d'aiuto nell'aggiornamento al sistema FreeBSD dell'appendice sul sistema UNIX; Mike Shapiro ha rivisto le informazioni sul sistema Solaris e Jim Mauro ha risposto ad alcune questioni sullo stesso sistema.

Si ringraziano le seguenti persone per la revisione di questa edizione: Rida Bazzi, dell'Arizona State University; Roy Campbell, della University of Illinois-Chicago; Gil Carrick, della University of Texas at Arlington; Richard Guy, della UCLA; Max Hailperin, del Gustavus Adolphus College; Ahmed Kamel, della North Dakota State University; Morty Kwestel, del New Jersey Institute of Technology; Gustavo Rodriguez-Rivera, della Purdue University; Carolyn J. C. Schauble, della Colorado State University; Thomas P. Skinner, della Boston University; Yannis Smaragdakis, del Georgia Tech; Larry L. Wear, della California State University, Chico; James M. Westall, della Clemson University; Yang Xiang, della University of Massachusetts.

I redattori Bill Zobrist e Paul Crockett sono stati delle esperte guide nella preparazione di questa edizione ed erano entrambi assistiti da Susannah Barr, che ha gestito in modo abile i dettagli di questo progetto. Katherine Hepburn è si è occupata del marketing; Ken Santor della produzione; Susan Cyr ha illustrato la copertina del libro sul progetto di Madelyn Lesure; Barbara Heaney si è occupata della supervisione del copy-editing, che è stato eseguito da Katie Habib; Katrina Avery ha svolto il ruolo di lettore di riferimento; Rosemary Simpson ha curato l'indice analitico; Anna Melhorn ha coordinato il lavoro sulle illustrazioni, e Marilyn Turnamian è stata d'aiuto nel generare le figure e aggiornare il testo, il Manuale dell'istruttore e le diapositive.

Infine alcune note personali. Avi Silberschatz desidera estendere la propria gratitudine a Krystyna Kwiecien per le devote cure alla propria madre che gli hanno dato la calma necessaria a concentrarsi nella scrittura di questo libro; Peter Baer Galvin desidera ringraziare Harry Kasparian e il suo gruppo per avergli dato la libertà di lavorare a questo progetto sostituendolo nel 'vero lavoro'; Greg Gagne desidera segnalare due importanti traguardi raggiunti dai propri figli durante il periodo di lavoro su questo testo: Tom, a cinque anni, ha imparato a leggere; Jay, a due anni, ha imparato a parlare.

Abraham Silberschatz, Murray Hill, NJ, 2001.
Peter Baer Galvin, Norton, MA, 2001.
Greg Gagne, Salt Lake City, UT, 2001.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Parte prima

Generalità

Introduzione
Strutture dei sistemi di calcolo
Strutture dei sistemi operativi

Un *sistema operativo* è un programma che agisce come intermediario tra l'utente e gli elementi fisici di un calcolatore. Lo scopo di un sistema operativo è fornire un ambiente nel quale un utente possa eseguire programmi in modo *conveniente* ed *efficiente*.

In questo libro si ripercorre lo sviluppo dei sistemi operativi, da quelli più rudimentali, passando per i sistemi multiprogrammati e a partizione del tempo d'elaborazione della CPU, ai sistemi d'elaborazione in tempo reale e agli attuali sistemi palmari. La comprensione dell'evoluzione dei sistemi operativi permette di apprezzare le operazioni gestibili da un sistema operativo e il modo in cui esse sono gestite.

Un sistema operativo deve assicurare il corretto funzionamento di un calcolatore; per evitare che i programmi utenti entrino in conflitto con le operazioni di un sistema operativo, l'hardware deve offrire meccanismi appropriati; sicché si descrivono le caratteristiche che l'architettura di un calcolatore deve possedere per rendere possibile la stesura di un sistema operativo corretto.

Un sistema operativo fornisce servizi ai programmi e ai relativi utenti per agevolare i loro compiti. I servizi offerti differiscono da un sistema operativo all'altro; in questo testo, tuttavia, si identificano e s'indagano alcune classi comuni di tali servizi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

DEFINIZIONE DI S.O.

- Allocatore di risorse = controlla, distribuisce
- Programma di controllo
  controlla l'esecuzione dei prog. utente e le operazioni dei dispositivi di I/O
- Kernel → unico prog. permanentemente in esecuzione

Capitolo 1

Introduzione

Un sistema operativo è un insieme di programmi (*software*) che gestisce gli elementi fisici di un calcolatore (*hardware*); fornisce una piattaforma ai programmi d'applicazione e agisce da intermediario fra l'utente e la struttura fisica del calcolatore. Un aspetto sorprendente dei sistemi operativi è quanto siano diversi i modi in cui eseguono questi compiti: i sistemi operativi per i *mainframe* si progettano innanzi tutto per ottimizzare l'utilizzo delle risorse; i sistemi operativi per PC consentono l'esecuzione di un'ampia varietà di programmi, dai giochi ai programmi gestionali; quelli per i sistemi palmari si progettano per fornire un ambiente in cui l'utente possa interagire facilmente col calcolatore per l'esecuzione dei programmi. Alcuni sistemi operativi si progettano per essere d'*uso agevole*, altri per essere *efficienti* e altri ancora per possedere una combinazione di tali qualità.

Per capire cosa siano i sistemi operativi occorre prima capire qual è stato il loro sviluppo, che in questo capitolo si descrive dai primi sistemi, passando per i sistemi con multiprogrammazione e partizione del tempo d'elaborazione, per arrivare fino ai PC e ai sistemi palmari. Si trattano inoltre i sistemi paralleli, i sistemi d'elaborazione in tempo reale, e i sistemi di controllo integrati negli apparecchi più vari. Passando attraverso i vari stadi, si vede come i componenti dei sistemi operativi si siano evoluti come naturale soluzione dei problemi nati con i primi calcolatori.

- Scopi principali di un S.O.
  - Eseguire i prog. utente e rendere + semplice la soluzione di varie problemi.
  - Rendere il sistema di calcolo semplice sia usato.

1.1 Cos'è un sistema operativo

Un sistema operativo è un elemento importante di quasi tutti i sistemi di calcolo. Un sistema di calcolo si può suddividere in quattro componenti: i *dispositivi fisici*, il *sistema operativo*, i *programmi d'applicazione* e gli *utenti* (Figura 1.1).

I dispositivi composti dall'unità centrale d'elaborazione, CPU (*central processing unit*), dalla memoria e dai dispositivi di immissione ed emissione dei dati, I/O (*input/output*) forniscono le risorse di calcolo fondamentali. I programmi d'applicazione (elaboratori di testi, fogli di calcolo, compilatori e programmi di consultazione del Web) definiscono il

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

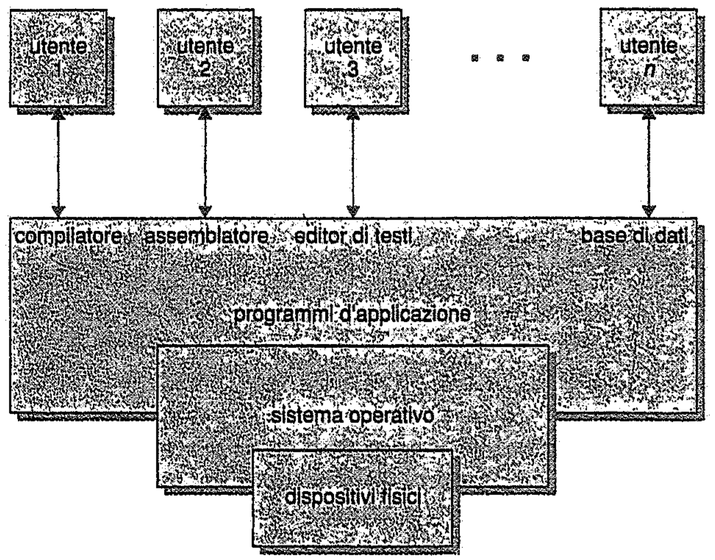


Figura 1.1 Componenti di un sistema di calcolo.

modo in cui si usano queste risorse per risolvere i problemi computazionali degli utenti. Il sistema operativo controlla e coordina l'uso dei dispositivi da parte dei programmi d'applicazione per gli utenti.

I componenti di un sistema di calcolo sono i suoi elementi fisici, i programmi e i dati. Il sistema operativo offre gli strumenti per impiegare in modo corretto queste risorse; non compie operazioni di per sé utili, ma definisce semplicemente un *ambiente* nel quale altri programmi possono lavorare in modo utile. Un sistema operativo si può considerare da due punti di vista: quello degli utenti e quello del sistema.

1.1.1 Punto di vista dell'utente

La percezione di un calcolatore da parte di un utente dipende principalmente dall'interfaccia impiegata. La maggior parte degli utenti usa PC, composti da uno schermo, una tastiera, un mouse e un'unità di sistema. I PC sono progettati per un singolo utente, che impiega le risorse in modo esclusivo, con lo scopo di massimizzare la quantità di lavoro che l'utente può svolgere. In questo caso il sistema operativo si progetta considerando principalmente la facilità d'uso, con qualche attenzione alle prestazioni, ma nessuna all'utilizzo delle risorse. Le prestazioni sono importanti per l'utente, ma non è importante se la maggior parte dei componenti del sistema rimane inutilizzato nelle attese dovute alla lentezza dell'utente nelle operazioni di I/O.

Alcuni utenti usano terminali connessi a **mainframe** o **minicalcolatori** condividendone le risorse con altri utenti anche loro connessi tramite terminali. In questo caso il sistema operativo si progetta per massimizzare l'utilizzo delle risorse; per garantire che tutto il tempo disponibile della CPU, la memoria e le periferiche di I/O siano impiegati in modo equo ed efficiente.

Altri utenti usano stazioni di lavoro, connesse a reti di altre stazioni di lavoro e a server; dispongono di risorse loro riservate; altre devono condividerle, come la rete e i server, per l'accesso ai file e per i servizi di stampa e di elaborazione. Tutto ciò richiede che il sistema operativo sia progettato ponderando l'adeguatezza all'uso individuale e l'utilizzo ottimale delle risorse.

Negli ultimi tempi si sono diffusi calcolatori palmari dei tipi più svariati; si tratta prevalentemente di unità a sé stanti, usate singolarmente da ciascun utente. Alcuni sono collegati a reti di calcolatori, direttamente tramite cavi o più spesso tramite dispositivi d'accesso senza fili. A causa della loro scarsa autonomia e interfaccia limitata, svolgono un numero piuttosto ridotto di operazioni remote. I loro sistemi operativi si progettano principalmente per facilitare l'uso individuale, ma anche per ridurre il consumo delle batterie.

Alcuni calcolatori sono integrati nei prodotti più vari (*embedded system*), e hanno poca o nessuna visibilità per gli utenti: i calcolatori integrati negli elettrodomestici e nelle automobili, ad esempio, possono avere una tastiera numerica, e possono accendere o spegnere alcuni indicatori luminosi per segnalare il proprio stato, ma questi apparati e i relativi sistemi operativi, nella maggior parte dei casi, si progettano per funzionare senza l'intervento degli utenti.

1.1.2 Punto di vista del sistema

Dal punto di vista del calcolatore, il sistema operativo è il programma più strettamente correlato ai suoi elementi fisici. È possibile considerare un sistema operativo come un **assegnatore di risorse**. Un sistema di calcolo dispone di risorse (fisiche e programmi) utili per la risoluzione di un problema: tempo di CPU, spazio di memoria, spazio per la registrazione di file, dispositivi di I/O e così via. Il sistema operativo agisce come gestore di tali risorse. Di fronte a numerose ed eventualmente conflittuali richieste di risorse, il sistema operativo deve decidere come assegnarle agli specifici programmi e utenti affinché il sistema di calcolo operi in modo equo ed efficiente.

Una visione leggermente diversa di un sistema operativo enfatizza la necessità di controllare i dispositivi di I/O e i programmi utenti: un sistema operativo è in effetti un programma di controllo. Un **programma di controllo** gestisce l'esecuzione dei programmi utenti in modo da impedire che si verifichino errori o che il calcolatore sia usato in modo scorretto, soprattutto per quel che riguarda il funzionamento e il controllo dei dispositivi di I/O.

Tuttavia, in generale, non si dispone di una definizione completa ed esauriente di sistema operativo. I sistemi operativi esistono poiché rappresentano una soluzione ragionevole al problema di realizzare un sistema di calcolo che si possa impiegare in modo utile, per eseguire i programmi utenti e facilitare la soluzione dei problemi degli utenti, ed

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

è proprio per questo scopo che sono stati costruiti i calcolatori; ma, poiché un calcolatore di per sé non è molto facile da utilizzare, sono stati sviluppati i programmi d'applicazione. Questi programmi sono diversi tra loro, ma richiedono alcune funzioni comuni, ad esempio il controllo dei dispositivi di I/O. Tali funzioni comuni, di controllo e assegnazione delle risorse, sono state racchiuse in un unico insieme coerente di programmi: il sistema operativo.

Non si dispone nemmeno di una definizione universalmente accettata di cosa faccia parte di un sistema operativo. Un semplice punto di vista è considerare che esso comprenda tutto quello che il venditore fornisce quando gli si richiede 'il sistema operativo'. Comunque, i requisiti della memoria e della capacità di registrazione in dischi e nastri, e le funzioni richieste variano molto da sistema a sistema. La capacità di registrazione di un sistema si misura in gigabyte (un kilobyte o KB è pari a 1024 byte, un megabyte o MB è 1024^2 byte e un gigabyte o GB è 1024^3 byte; spesso i costruttori di calcolatori approssimano questi valori esprimendosi in termini di un milione di byte per un megabyte e di un miliardo di byte per un gigabyte). Alcuni sistemi usano meno di un megabyte di memoria e non possiedono neppure un *editor* a pieno schermo, mentre altri richiedono centinaia di megabyte di memoria e sono interamente basati su interfacce grafiche a finestre. Una definizione più comune è quella secondo cui il sistema operativo è il solo programma che funziona sempre nel calcolatore, generalmente chiamato **nucleo** (*kernel*), mentre tutti gli altri sono programmi d'applicazione. Quest'ultima definizione è quella generalmente seguita in questo testo. La questione riguardante i componenti di un sistema operativo ha assunto una certa rilevanza anche in seguito all'azione legale promossa dal Dipartimento della giustizia degli Stati Uniti contro la Microsoft, accusata di includere troppe funzioni nel sistema operativo e quindi di concorrenza sleale nei confronti dei produttori di applicazioni.

1.1.3 Scopi del sistema

È più facile definire un sistema operativo rispetto a ciò che *fa* piuttosto che a ciò che esso *è*; ma anche questo potrebbe essere solo un espediente. Lo scopo principale di alcuni sistemi operativi è essere *agevoli per gli utenti*. I sistemi operativi hanno lo scopo di facilitare le computazioni. Questa definizione appare più chiara se ci si riferisce ai sistemi operativi per i PC.

Lo scopo principale di altri sistemi operativi è il funzionamento *efficiente* del sistema di calcolo; è il caso dei sistemi di grandi dimensioni condivisi da più utenti. Questi sistemi sono molto costosi, sicché è auspicabile massimizzarne l'efficienza. Questi due scopi, comodità d'uso ed efficienza, sono a volte in contrasto. L'efficienza è stata a lungo considerata più importante della comodità d'uso (Paragrafo 1.2.1), quindi gran parte della teoria dei sistemi operativi è imperniata sull'uso ottimale delle risorse di calcolo. Col passare del tempo i sistemi operativi si sono evoluti notevolmente. Lo UNIX, ad esempio, inizialmente possedeva come interfaccia solo una tastiera e una stampante, e il

suo uso era assai poco agevole. In seguito, con l'evoluzione dell'architettura dei calcolatori, è stato dotato d'interfacce dall'uso assai più semplice. Si potevano sovrapporre al sistema diverse **interfacce d'utente grafiche** (*grafic user interface* — GUI), consentendo una maggiore facilità d'uso, ma lasciando che il sistema fosse ancora orientato all'efficienza.

La progettazione di un sistema operativo è un compito complesso e chi se ne occupa deve affrontare molti compromessi, sia nella fase di progettazione sia in quella di realizzazione. Tutto ciò impegna molte persone, anche nelle successive fasi di costante revisione e aggiornamento. Quanto un sistema operativo soddisfi gli obiettivi stabiliti nella fase di progettazione è una questione soggettiva riguardante i diversi utenti.

Per capire che cosa siano e che cosa facciano i sistemi operativi, è opportuno considerare come si sono sviluppati negli ultimi quarantacinque anni. Delineando quest'evoluzione si possono identificare gli elementi comuni ai vari sistemi operativi e capire come e perché tali sistemi si siano sviluppati in un certo modo.

I sistemi operativi e l'architettura dei calcolatori si sono influenzati reciprocamente in modo notevole. I ricercatori hanno sviluppato i sistemi operativi per facilitare l'uso dei calcolatori; gli utenti, dal canto loro, hanno proposto modifiche dell'architettura dei calcolatori che consentissero di semplificare i sistemi operativi. In questa breve prospettiva storica, si vuole evidenziare come l'identificazione di problemi connessi ai sistemi operativi conduca all'introduzione di nuove caratteristiche nell'architettura dei calcolatori.

1.2 Sistemi mainframe

I sistemi di calcolo di tipo **mainframe** sono stati i primi calcolatori impiegati per affrontare molte applicazioni scientifiche e commerciali. In questo paragrafo si descrive l'evoluzione dei sistemi mainframe; dai semplici **sistemi a lotti** (*batch*), nei quali il calcolatore eseguiva una, e una sola, applicazione alla volta; ai **sistemi a partizione del tempo d'elaborazione** — o, più brevemente, **sistemi a partizione del tempo** (*time sharing*) —, che permettono un uso interattivo dei sistemi di calcolo.

1.2.1 Sistemi a lotti

I primi calcolatori erano enormi macchine il cui funzionamento si controllava da una console. Gli usuali dispositivi di immissione dei dati erano i lettori di schede e le unità a nastro; gli usuali dispositivi di emissione dei dati erano le stampanti, le unità a nastro e i perforatori di schede. L'utente non interagiva direttamente col sistema di calcolo ma preparava un lavoro d'elaborazione (*job*) — composto del programma, i dati e le informazioni sul tipo di lavoro (schede di controllo) — e lo affidava, di solito sotto forma di una pila di schede perforate, all'operatore del sistema. Dopo un certo tempo (minuti, ore, giorni) si ottenevano i risultati del programma e, in presenza d'errori nel programma, l'immagine del contenuto finale della memoria e dei registri (*dump*); utile per l'individuazione e la correzione degli errori (*debugging*).

utente ≠ operatore

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

Figura 1.2 Configurazione della memoria per un sistema a lotti.

Il sistema operativo di tali primi calcolatori era abbastanza semplice; il suo compito principale consisteva nel trasferire automaticamente il controllo da un lavoro al successivo ed era sempre residente nella memoria (Figura 1.2).

Per accelerare l'elaborazione, si raggruppavano insieme i lavori con requisiti simili e si facevano eseguire dal calcolatore in un unico blocco. Così, i programmatori affidavano i loro programmi all'operatore, che ordinava i programmi in lotti con requisiti analoghi e, non appena il calcolatore era disponibile, li faceva eseguire. Quindi, si inviava l'esito di ogni lavoro al relativo programmatore.

In tale ambiente d'esecuzione la CPU era spesso inattiva poiché i dispositivi meccanici di I/O sono intrinsecamente più lenti dei dispositivi elettronici. I tempi caratteristici di una CPU, anche se lenta, sono misurabili in microsecondi. Quindi, mentre questa eseguiva migliaia di istruzioni al secondo, un lettore di schede veloce poteva leggere 1200 schede al minuto, vale a dire 20 al secondo. Di conseguenza, la differenza di velocità tra la CPU e i dispositivi di I/O era di tre o più ordini di grandezza. Col tempo i progressi tecnologici e l'introduzione dei dischi magnetici hanno portato allo sviluppo di dispositivi di I/O più veloci, ma la velocità delle CPU è aumentata ancora più rapidamente, perciò il problema non solo non si è risolto, ma si è aggravato.

Spooling
Le operazioni di I/O di 1 job vengono sovrapposte con l'elaborazione di 1 altro job

L'introduzione dei dischi magnetici ha permesso al sistema operativo di mantenere tutti i lavori in un disco, anziché in un lettore di schede seriale. Disponendo di un mezzo ad accesso diretto, il sistema operativo poteva organizzare la sequenza d'esecuzione dei lavori (*job scheduling*) in modo da usare le risorse e portare a termine i propri compiti in modo efficiente. Alcuni aspetti importanti di quest'ultimo argomento sono discussi in questa sede; una trattazione più dettagliata si trova nel Capitolo 6.

job pool
Struttura dati che permette al S.O. di selezionare il job che verrà successivamente portato in memoria al fine di aumentare l'utilizzo della CPU => Scheduling dei job

1.2.2 Sistemi multiprogrammati

L'aspetto più importante della possibilità di organizzare la sequenza d'esecuzione dei lavori è quello di poter effettuare la multiprogrammazione. In generale, un singolo utente non può tenere costantemente occupati la CPU o i dispositivi di I/O. La multiprogrammazione consente di aumentare l'utilizzo della CPU organizzando i lavori in modo tale da mantenerla in continua attività.

L'idea su cui si fonda questo concetto è la seguente: il sistema operativo tiene contemporaneamente nella memoria centrale diversi lavori (Figura 1.3); tale insieme di lavori (*job*) è generalmente un sottoinsieme dei lavori presenti nel lotto, ciò a causa delle limitate dimensioni della memoria centrale, che di solito consentono di contenere meno lavori di quanti ne siano presenti nel lotto. Il sistema operativo ne sceglie uno tra quelli contenuti nella memoria e inizia a eseguirlo. Esso, a un certo punto, potrebbe trovarsi nell'attesa di qualche evento, come il completamento di un'operazione di I/O. In questi casi, in un sistema non multiprogrammato, la CPU rimane inattiva. In un sistema con multiprogrammazione, invece, il sistema operativo passa semplicemente a un altro lavoro e lo esegue. Quando il primo lavoro ha terminato l'attesa, la CPU ne riprende l'esecuzione. Finché c'è almeno un lavoro da eseguire, la CPU non è mai inattiva.

Ritroviamo quest'idea anche in altre circostanze della vita comune. Un avvocato, ad esempio, non lavora per un solo cliente alla volta: mentre un caso attende di essere dibattuto o si attende la stesura dei relativi documenti, l'avvocato può lavorare a un altro caso; se ha abbastanza clienti, l'avvocato non sarà mai inattivo per mancanza di lavoro. Gli avvocati inattivi tendono a trasformarsi in politici, quindi tenere occupati gli avvocati ha un certo valore sociale.

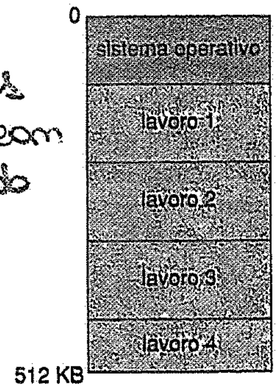


Figura 1.3 Configurazione della memoria per un sistema con multiprogrammazione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La multiprogrammazione è la prima situazione in cui il sistema operativo deve prendere decisioni per gli utenti; i sistemi operativi multiprogrammati sono perciò abbastanza raffinati. Tutti i lavori che entrano nel sistema sono mantenuti in un gruppo, che consiste di tutti i processi d'elaborazione presenti nel disco che attendono il caricamento nella memoria centrale. Se più lavori sono pronti per essere caricati ma non c'è abbastanza spazio per tutti, il sistema deve scegliere tra essi: in tale decisione consiste il cosiddetto *job scheduling*, trattato nel Capitolo 6. Quando il sistema operativo sceglie un lavoro dal gruppo lo carica nella memoria per l'esecuzione. La presenza contemporanea di più programmi nella memoria implica una gestione della memoria stessa (argomento trattato nei Capitoli 9 e 10). Inoltre, se in un dato momento più lavori sono pronti per essere eseguiti, il sistema deve scegliere quello da eseguire. Questa decisione è il cosiddetto *CPU scheduling*, trattato nel Capitolo 6. Infine, la possibilità d'influenza reciproca di più lavori eseguiti in modo concorrente deve essere limitata in tutte le fasi del sistema operativo, compresi lo scheduling dei processi, la scrittura nei dischi e la gestione della memoria.

1.2.3 Sistemi a partizione del tempo d'elaborazione

Un sistema a lotti con multiprogrammazione forniva un ambiente in cui le risorse del sistema (per esempio CPU, memoria, dispositivi periferici) erano impiegate in modo efficiente, ma non forniva un sistema d'interazione con l'utente. La partizione del tempo d'elaborazione (*time sharing* o *multitasking*) è un'estensione logica della multiprogrammazione; la CPU esegue più lavori commutando le loro esecuzioni con una frequenza tale da permettere a ciascun utente d'interagire col proprio programma durante la sua esecuzione.

Un sistema di calcolo interattivo permette la comunicazione diretta tra utente e sistema. L'utente impartisce le istruzioni direttamente al sistema operativo oppure a un programma, attraverso una tastiera o un mouse, e attende una risposta immediata. Il tempo di risposta dovrebbe perciò essere breve.

Un sistema operativo a partizione del tempo d'elaborazione permette a più utenti di condividere contemporaneamente il calcolatore. Poiché le azioni e i comandi eseguiti in un sistema a partizione del tempo sono tendenzialmente brevi, a ciascun utente basta poter usare una piccola parte del tempo di calcolo della CPU. Il sistema passa rapidamente da un utente all'altro, quindi ogni utente ha l'impressione di disporre dell'intero calcolatore; che in realtà è condiviso da molti utenti.

Un sistema operativo a partizione del tempo d'elaborazione si avvale dello scheduling della CPU, e della multiprogrammazione, per assicurare a ciascun utente una piccola frazione del tempo di calcolo. Ciascun utente dispone di almeno un proprio programma nella memoria. Un programma caricato nella memoria e predisposto per la fase d'esecuzione è noto come processo d'elaborazione — o, più brevemente, processo. Normalmente un processo, durante la sua esecuzione, impegna la CPU per un breve periodo di tempo prima di richiedere operazioni di I/O o di terminare; tali operazioni possono essere interattive, i risultati sono inviati a uno schermo a disposizione dell'utente, il quale immette dati tramite una tastiera, un mouse, o altro. I tempi di tali operazioni risentono

• SWAP IN → i job vengono spostati dal disco alla memoria
• SWAP OUT → " " " " " " " dalla memoria al disco

della lentezza del lavoro umano: l'immissione di dati e comandi tramite una tastiera, ad esempio, è limitata dalla velocità di battitura dell'utente, che, per elevata che sia, è sempre molto bassa rispetto ai tempi d'elaborazione di un calcolatore: sette caratteri al secondo sono tanti per una persona, ma pochissimi per un calcolatore. Anziché lasciare inattiva la CPU, durante l'immissione interattiva il sistema operativo commuta rapidamente la CPU al programma di un altro utente.

I sistemi operativi a partizione del tempo sono ancora più complessi dei sistemi operativi multiprogrammati. In entrambi i casi è prevista la coesistenza di più lavori nella memoria; sicché il sistema deve disporre di una forma di gestione e protezione della memoria (Capitolo 9). Per ottenere un tempo di risposta ragionevole, i lavori si possono scaricare dalla memoria centrale in un disco e viceversa; in questi casi il disco serve come memoria ausiliaria della memoria centrale. Un modo comune per ottenere tale risultato è l'uso della cosiddetta **memoria virtuale**, una tecnica che consente l'esecuzione di lavori d'elaborazione anche non interamente caricati nella memoria (Capitolo 10). Il più evidente vantaggio della memoria virtuale è che i programmi possono avere dimensioni maggiori della **memoria fisica**; inoltre, essa astrae la memoria centrale in un grande e uniforme vettore, separando la **memoria logica**, vista dall'utente, dalla memoria fisica; ciò solleva i programmatori dai problemi legati ai limiti della memoria. I sistemi a partizione del tempo devono inoltre fornire un *file system* (Capitoli 11 e 12) residente in un insieme di dischi, i quali a loro volta necessitano di una gestione (Capitolo 14). I sistemi a partizione del tempo dispongono di meccanismi per l'esecuzione concorrente dei processi, che richiede raffinati schemi di scheduling della CPU (Capitolo 6), e per assicurare che tale esecuzione sia disciplinata devono fornire meccanismi per la comunicazione e sincronizzazione dei processi (Capitolo 7); tali sistemi devono inoltre assicurare che i processi non s'inceppino in una situazione di stallo: in un'indefinita attesa reciproca (Capitolo 8).

Il concetto di partizione del tempo fu dimostrato fin dal 1960, ma a causa delle difficoltà e degli elevati costi di costruzione, i sistemi a partizione del tempo si sono diffusi soltanto dai primi anni Settanta. Benché si eseguano ancora elaborazioni a lotti, oggigiorno la maggior parte dei sistemi è a partizione del tempo d'elaborazione. La multiprogrammazione e la partizione del tempo sono argomenti centrali dei moderni sistemi operativi e quindi sono anche argomenti centrali di questo libro.

gestiti completamente dall'utente.

1.3 Sistemi da scrivania

I PC sono apparsi negli anni Settanta, e durante i primi dieci anni dalla loro comparsa le loro CPU non possedevano le caratteristiche necessarie per proteggere un sistema operativo dai programmi utenti. Tali sistemi operativi non erano quindi né multiutente né a partizione del tempo. Tuttavia gli scopi di questi sistemi operativi sono cambiati col passare del tempo; invece di tendere al massimo utilizzo della CPU e dei dispositivi periferici, si è mirato alla comodità e prontezza d'uso per l'utente. Questi sistemi comprendono sia i PC col sistema operativo Microsoft Windows sia gli Apple Macintosh. Il sistema operativo

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

MS-DOS della Microsoft è stato soppiantato dalle varie versioni del Microsoft Windows, l'IBM ha sostituito il proprio PC-DOS col sistema a partizione del tempo OS/2; il sistema operativo Apple Macintosh è stato trasferito su un'architettura più moderna e ora include nuove caratteristiche come la memoria virtuale e la partizione del tempo d'elaborazione. Il nucleo e le funzioni fondamentali del sistema MacOS X sono basate sulle relative funzioni dei sistemi Mach e UNIX FreeBSD, per le loro funzioni e doti di scalabilità e prestazioni, ma mantengono una semplice e completa interfaccia d'utente grafica. Il LINUX, un sistema operativo disponibile per i PC, ispirato allo UNIX, si sta diffondendo sempre di più.

I sistemi operativi di questi calcolatori hanno beneficiato per diversi aspetti dello sviluppo dei sistemi operativi per i mainframe. I microcalcolatori hanno potuto adottare subito alcune tecnologie sviluppate per i sistemi operativi più grandi. I costi relativi alla struttura fisica dei microcalcolatori sono sufficientemente bassi da permetterne l'uso a una sola persona, e l'utilizzo della CPU non è più l'obiettivo principale. Di conseguenza alcune fra le innovazioni adottate per i sistemi operativi dei mainframe non sono adatte ai sistemi più piccoli. Inoltre si applicano altri princìpi: se in un sistema personale la protezione dei file non era necessaria, lo diviene in seguito alla tendenza a collegare in rete anche questi sistemi. Infatti, in questo caso, altri calcolatori e altri utenti possono accedere ai file in essi contenuti. L'assenza di tale protezione ha reso semplice ad alcuni programmi 'malvagi' la distruzione di dati in sistemi operativi come l'MS-DOS e il Macintosh OS. Questi programmi possono essere autoreplicanti ed espandersi rapidamente, tramite meccanismi detti worm o virus, fino a distruggere intere reti aziendali e persino reti geografiche. Benché raffinate, le funzioni di protezione dei file e della memoria, di cui si possono dotare i sistemi a partizione del tempo, non sono sufficienti a proteggere un sistema dagli attacchi alla sua sicurezza. Questi argomenti sono trattati nei Capitoli 18 e 19.

1.4 Sistemi con più unità d'elaborazione (Sistemi Paralleli)

Oggi la maggior parte dei sistemi è costituita da sistemi con un'unica unità d'elaborazione, cioè con una sola CPU. Tuttavia cresce l'importanza dei sistemi dotati di più unità d'elaborazione, noti come **sistemi paralleli** (o **sistemi strettamente connessi** — *tightly coupled system*). Questo tipo di sistemi dispone di più unità d'elaborazione in stretta comunicazione, che condividono i canali di comunicazione all'interno del calcolatore (*bus*), i temporizzatori dei cicli di macchina (*clock*) e talvolta i dispositivi di memorizzazione e periferici.

Tali sistemi hanno tre vantaggi principali.

1. **Maggiore produttività** (*throughput*). Aumentando il numero di unità d'elaborazione è possibile svolgere un lavoro maggiore in meno tempo. Con n unità d'elaborazione la velocità non aumenta comunque di n volte, ma in misura minore. Infatti, se più unità d'elaborazione collaborano nell'esecuzione di un compito, il sistema operativo deve gestire le operazioni che garantiscono che tutti i componenti funzionino correttamente. Questo sovraccarico, unito alla contesa delle risorse condivise, riduce il guadagno atteso dalla disponibilità di più unità d'elaborazione; così come un gruppo di n programmatori che lavorano insieme non produce n volte quanto produrrebbe un solo programmatore.

Risparmio Economico: condivisione di risorse fisiche

2. **Economia di scala.** I sistemi con più unità d'elaborazione possono inoltre consentire risparmi rispetto a più sistemi dotati di una sola unità d'elaborazione, poiché nei primi si possono condividere dispositivi periferici, mezzi di registrazione dei dati e alimentatori elettrici. Se più programmi devono operare sullo stesso insieme di dati, è economicamente più conveniente registrarli in dischi condivisi da tutte le unità d'elaborazione, piuttosto che avere più calcolatori con i rispettivi dischi locali e più copie degli stessi dati.

3. **Incremento dell'affidabilità.** Se le funzioni si possono distribuire adeguatamente tra più unità d'elaborazione, un guasto ad alcune di esse non blocca il sistema, semplicemente lo rallenta: ciascuna delle unità d'elaborazione rimanenti assume su di sé una parte del lavoro che svolgevano le unità d'elaborazione guaste; l'intero sistema non si ferma ma funziona a una velocità ridotta. La capacità di continuare il servizio in misura proporzionale al livello di dispositivi correttamente funzionanti si chiama **degradazione controllata** (*graceful degradation*). I sistemi strutturati in modo da garantire una degradazione controllata delle prestazioni si dicono **tolleranti i guasti** (*fault-tolerant*).

Affinché il funzionamento del sistema continui anche alla presenza di guasti, occorre un meccanismo che permetta l'identificazione, la diagnosi e, se è possibile, la correzione dei guasti stessi. Per garantire la propria continuità di funzionamento nonostante il verificarsi di guasti, il sistema Tandem impiega un duplicato dei dispositivi e dei programmi. Il sistema è costituito di due unità d'elaborazione identiche collegate tra loro da un bus, una di esse è l'unità d'elaborazione primaria, mentre l'altra è di riserva (*backup*); entrambe sono provviste di una propria memoria locale. Ogni processo viene duplicato: una copia per l'unità d'elaborazione primaria, una per quella di riserva. Durante l'esecuzione del sistema, in corrispondenza di determinati punti di verifica (*checkpoint*) si copia l'informazione di stato di ogni processo, compresa una copia dell'immagine della memoria locale, dall'unità d'elaborazione primaria a quella di riserva. Se si individua un guasto, si attiva la copia di riserva riavviandola a partire dal più prossimo punto di verifica. Naturalmente si tratta di una soluzione costosa, giacché richiede una notevole duplicazione dei dispositivi.

I sistemi paralleli più comuni oggi impiegano il modello della **multielaborazione simmetrica** (*symmetric multiprocessing* — SMP), nel quale ciascuna unità d'elaborazione esegue un'identica copia del sistema operativo; queste copie comunicano tra loro quando è necessario. Alcuni sistemi impiegano la **multielaborazione asimmetrica** (*asymmetric multiprocessing* — AMP), nella quale a ogni unità d'elaborazione si assegna un compito specifico. Un'unità d'elaborazione principale controlla il sistema, le altre attendono istruzioni dall'unità principale oppure hanno compiti predefiniti. Questo schema definisce una relazione gerarchica; l'unità d'elaborazione principale organizza e assegna il lavoro alle unità d'elaborazione secondarie.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il modello SMP mette sullo stesso piano tutte le unità d'elaborazione; tra esse non esiste una relazione gerarchica e ciascuna esegue una copia del sistema operativo. La Figura 1.4 illustra una tipica architettura SMP. Un esempio del modello di multielaborazione simmetrica è dato dalla versione Encore dello UNIX per il calcolatore Multimax. Questo calcolatore si può configurare in modo da usare decine di unità d'elaborazione, ciascuna delle quali esegue una copia del sistema operativo. Il vantaggio offerto da questo modello è dato dal fatto che si possono eseguire molti processi contemporaneamente (*n* processi se si hanno *n* CPU) senza causare un rilevante calo delle prestazioni. Per essere certi che i dati raggiungano le unità d'elaborazione giuste occorre però controllare con molta attenzione le operazioni di I/O. Inoltre, poiché le unità d'elaborazione sono separate, una potrebbe essere inattiva mentre un'altra è sovraccarica, e ciò determinerebbe un'inefficienza che si può evitare se le unità d'elaborazione condividono alcune strutture di dati. Un sistema con più unità d'elaborazione di questo tipo permette infatti di condividere dinamicamente processi e risorse — ad esempio la memoria — tra le varie unità d'elaborazione, riducendo la varianza tra di esse. Come si vede nel Capitolo 7, un sistema di questo tipo deve essere scritto con molta cura. Tutti i sistemi operativi moderni (tra i quali il Windows 2000, Solaris, Digital UNIX, OS/2 e il LINUX) gestiscono la multielaborazione simmetrica.

La differenza tra la multielaborazione simmetrica e asimmetrica può derivare sia da caratteristiche dell'architettura del sistema sia da caratteristiche del sistema operativo: una speciale architettura può differenziare le unità d'elaborazione, oppure si può avere un sistema operativo che definisce una sola unità d'elaborazione primaria e più unità d'elaborazione secondarie. Ad esempio, la versione 4 del sistema operativo SunOS della Sun consentiva la multielaborazione asimmetrica, mentre la versione 5 (Solaris 2) consente, sulla stessa architettura, la multielaborazione simmetrica.

Con la riduzione dei costi e l'aumento della potenza delle unità d'elaborazione, si delegano sempre più funzioni del sistema operativo alle unità d'elaborazione secondarie. È abbastanza semplice, ad esempio, impiegare un'unità d'elaborazione e la sua memoria per gestire un'unità a disco. L'unità d'elaborazione può ricevere una sequenza di richieste dall'unità d'elaborazione principale, la CPU, e gestire la propria coda d'attesa per l'accesso al disco eseguendo direttamente il relativo algoritmo. Questo sistema evita all'unità d'elaborazione principale l'onere della gestione dell'accesso al disco. La tastiera dei PC, ad esempio, contiene una piccola unità d'elaborazione che converte le sequenze di pressione dei tasti in codici da inviare alla CPU. Questo tipo di impiego delle unità d'elaborazione s'è diffuso a tal punto che non si considera più una multielaborazione.

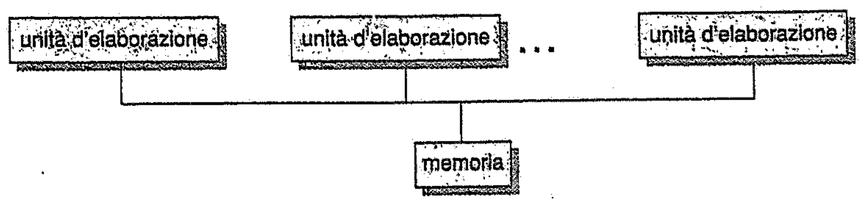


Figura 1.4 Architettura per la multielaborazione simmetrica.

distribuzione
l'elaborazione
tra + processori fisici

VANTAGGI:
- condivisione delle risorse
- Aumento delle prestazioni
- Affidabilità

1.5 Sistemi distribuiti: generalizzazioni

Una rete si può considerare, nei più semplici termini, come un canale di comunicazione tra due o più sistemi. I sistemi distribuiti si basano sulle reti per realizzare le proprie funzioni, sfruttano la capacità di comunicazione per cooperare nella soluzione dei problemi di calcolo e per fornire agli utenti un ricco insieme di funzioni.

Le reti differiscono per i protocolli usati, per le distanze tra i nodi e per il mezzo attraverso il quale avviene la comunicazione. Sebbene siano largamente usati sia il protocollo ATM sia altri protocolli, il più diffuso protocollo di comunicazione è il TCP/IP. È piuttosto varia anche la disponibilità dei diversi protocolli di rete nei diversi sistemi operativi. La maggior parte dei sistemi operativi, inclusi i sistemi Windows e UNIX, impiega il TCP/IP. Alcuni sistemi dispongono di propri protocolli che soddisfano esigenze specifiche. Affinché un sistema operativo possa gestire un protocollo di rete è necessaria la presenza di un dispositivo d'interfaccia — un adattatore di rete, ad esempio — con un driver per gestirlo, oltre ai programmi che impaccano i dati secondo il protocollo di comunicazione, per poterli inviare, e che li depaccano a destinazione.

Le reti si classificano secondo le distanze tra i loro nodi: una rete locale (LAN) comprende nodi all'interno della stessa stanza, piano o edificio; una rete geografica (WAN) si estende a gruppi di edifici, città, o al territorio di una regione o di uno stato. Una società multinazionale, ad esempio, potrebbe disporre di una rete WAN per connettere i propri uffici nel mondo. Queste reti possono funzionare con uno o più protocolli e il continuo sviluppo di nuove tecnologie fa sì che si definiscano nuovi tipi di reti. Le reti metropolitane (MAN) ad esempio collegano gli edifici di un'intera città; i dispositivi BlueTooth comunicano a breve distanza, dell'ordine delle decine di metri, creando essenzialmente una microrete (*small-area network*).

I mezzi di trasmissione che s'impiegano nelle reti sono altrettanto vari: fili di rame, fibre ottiche, trasmissioni via satellite, sistemi a microonde e sistemi radio; anche il collegamento dei dispositivi di calcolo ai telefoni cellulari crea una rete, così come per creare una rete si può usare anche la capacità di comunicazione a brevissima distanza dei dispositivi a raggi infrarossi. In breve, ogni volta che comunicano, i calcolatori usano o creano reti; che ovviamente si differenziano per prestazioni e affidabilità.

1.5.1 Sistemi client-server

Con la disponibilità di PC più veloci, più potenti e più economici, i progettisti hanno abbandonato le architetture centralizzate. I terminali connessi a sistemi centralizzati sono stati sostituiti dai PC, mentre le funzioni relative alle interfacce d'utente, che erano fornite direttamente dai sistemi centralizzati, sono sempre più gestite dai PC. I sistemi centralizzati fungono oggi da sistemi server pensati per soddisfare le richieste generate da sistemi client; la Figura 1.5 illustra la struttura generale di un sistema client-server.

I sistemi server si possono dividere nelle due categorie principali di server di calcolo e server di file.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

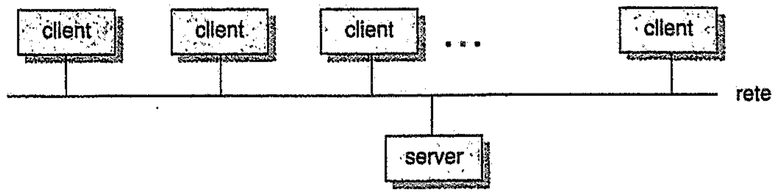


Figura 1.5 Struttura generale di un sistema client-server.

Un sistema server di calcolo offre un'interfaccia cui ciascun client può inviare una richiesta d'esecuzione di un'azione alla quale risponde eseguendo l'azione e riportando il risultato al client.

Un sistema server di file, offre un'interfaccia al file system, attraverso la quale ogni client può creare, aggiornare, leggere e cancellare file.

1.5.2 Sistemi paritetici (peer to peer)

La crescita delle reti di calcolatori — specialmente della rete mondiale Internet e del World Wide Web (WWW) — ha avuto una profonda influenza sui recenti sviluppi dei sistemi operativi. Quando furono introdotti, negli anni Settanta, i PC furono progettati per l'uso personale e considerati come stazioni di lavoro isolate. Negli anni Ottanta con la diffusione della rete Internet, e il diffondersi dell'uso della posta elettronica, e dei servizi *ftp* e *gopher*, molti PC si connettevano alla Rete (Internet). L'introduzione del Web a metà degli anni Novanta trasformò la capacità di connettersi alla Rete in una funzione essenziale di un sistema di calcolo.

Tutti i PC e le moderne stazioni di lavoro dispongono di programmi di consultazione dei documenti ipertestuali del Web. I sistemi operativi (Windows, OS/2, MacOS, UNIX, ecc.) comprendono i programmi di sistema (TCP/IP, PPP ecc.) che permettono a un calcolatore di accedere alla rete Internet attraverso una rete locale o una connessione telefonica. Alcuni includono anche i programmi di consultazione del Web, i client e i server di gestione della posta elettronica, per l'uso di calcolatori remoti e per il trasferimento di file.

Rispetto ai sistemi strettamente connessi discussi nel Paragrafo 1.4, le reti di calcolatori usate in queste applicazioni sono costituite di un insieme di unità d'elaborazione (processori) che non condividono la memoria, né un clock; ciascuna ha la propria memoria locale e comunicano tra loro per mezzo di linee di comunicazione di vario genere, ad esempio bus ad alta velocità o linee telefoniche. Questi sistemi di solito si chiamano sistemi debolmente connessi (*loosely coupled system*) o sistemi distribuiti.

Alcuni sistemi operativi interpretano il concetto di rete e di sistema distribuito in modo più radicale rispetto alla capacità di collegarsi tramite una rete. Un sistema operativo di rete è un sistema operativo che offre funzioni come la condivisione di file tra i calcolatori della rete e che include uno schema di comunicazione che permette lo scambio

di messaggi tra diversi processi di diversi calcolatori nella rete. Un calcolatore che esegue un sistema operativo di rete si comporta autonomamente rispetto agli altri calcolatori della rete, sebbene riconosca la rete e possa comunicare con tutti i calcolatori connessi. Un sistema operativo distribuito è un ambiente meno autonomo, in cui più sistemi operativi comunicano tra loro in modo abbastanza armonico da dare l'illusione che vi sia un unico sistema operativo che controlla la rete. Le reti di calcolatori e i sistemi distribuiti sono trattati nei Capitoli dal 15 al 17.

1.6 Batterie di sistemi

In modo analogo ai sistemi paralleli, le batterie di sistemi (*cluster system*) sono basate sull'uso congiunto di più unità d'elaborazione riunite per svolgere attività d'elaborazione comuni. Differiscono dai sistemi paralleli per il fatto che sono composte di due o più calcolatori completi collegati tra loro. In realtà non esiste una definizione precisa; c'è un dibattito aperto tra i fornitori delle varie offerte commerciali su tale definizione e sul perché una soluzione sia migliore di un'altra. La definizione generalmente accettata è che si tratta di calcolatori che condividono la memoria di massa, e sono connessi per mezzo di una rete locale.

Questo tipo di soluzione di solito si adotta per offrire un'elevata disponibilità. Ciascun calcolatore esegue una serie di programmi che forma uno strato di gestione della batteria di sistemi; ogni nodo può tenere sotto controllo (attraverso la LAN) uno o più degli altri nodi. Se si presenta un malfunzionamento, il calcolatore che svolge il controllo può appropriarsi dei mezzi di memorizzazione del calcolatore malfunzionante e riavviare le applicazioni che erano in esecuzione. Gli utenti e i client delle applicazioni notano solo una breve interruzione del servizio, anche se il calcolatore guasto smette di funzionare.

Nelle batterie asimmetriche un calcolatore rimane nello stato di attesa attiva (*hot standby*) mentre l'altro esegue le applicazioni. Il primo non fa altro che tenere sotto controllo il server attivo (il secondo calcolatore), se questo presenta un problema, il calcolatore di controllo diventa il server attivo. Nelle batterie simmetriche, due o più calcolatori eseguono le applicazioni e allo stesso tempo si controllano reciprocamente; in questo modo si ottiene una maggiore efficienza, poiché si utilizzano meglio le risorse, ma richiede che siano disponibili più applicazioni da eseguire.

Altre forme sono le batterie di sistemi paralleli e le batterie di sistemi connessi attraverso reti geografiche (WAN). Le prime permettono a più calcolatori di accedere agli stessi dati nella memoria di massa condivisa. Poiché la maggior parte dei sistemi operativi non consente quest'accesso simultaneo ai dati da parte di più calcolatori, si ricorre a programmi specifici e particolari versioni delle applicazioni. Ad esempio, l'Oracle Parallel Server è una versione del sistema di gestione delle basi di dati Oracle progettata per funzionare in questo tipo di sistemi. Ogni calcolatore esegue l'applicazione Oracle e uno strato di programmi controlla l'accesso ai dischi condivisi, in questo modo ogni calcolatore del sistema ha accesso alla base di dati.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Nonostante i progressi fatti nel campo dell'elaborazione distribuita, la maggior parte dei sistemi non offre le funzioni di un file system distribuito d'uso generale. È per questo motivo che le batterie di sistemi di solito non permettono l'accesso condiviso ai dati presenti nei dischi. Per realizzare queste funzioni, i file system distribuiti devono disporre di meccanismi di controllo degli accessi e di bloccaggio dei file che assicurino l'assenza di operazioni conflittuali. Questo tipo di servizio è noto come **gestione distribuita degli accessi** (*distributed lock manager* — DLM). Si stanno sviluppando file system distribuiti d'uso generale e aziende come la Sun Microsystems hanno annunciato i loro progetti d'integrazione della gestione distribuita degli accessi nel sistema operativo.

La tecnologia delle batterie di sistemi d'elaborazione si sta evolvendo molto rapidamente in diverse direzioni, una delle quali è quella dei sistemi globali, in cui i calcolatori potrebbero essere ovunque nel mondo (in qualsiasi locazione raggiungibile da una WAN). Questi progetti sono attualmente nelle fasi di ricerca e sviluppo.

L'uso e le funzioni delle batterie di sistemi d'elaborazione dovrebbero espandersi notevolmente col diffondersi delle **reti di memorizzazione d'area** (*storage-area network* — SAN), descritte nel Paragrafo 14.6.3, che permettono una semplice associazione di più unità di memorizzazione a più calcolatori. Le attuali batterie d'elaborazione sono normalmente limitate a due o quattro calcolatori, proprio a causa della complessa connessione dei calcolatori alle unità di memorizzazione.

1.7 Sistemi d'elaborazione in tempo reale

Un'altra forma di sistema operativo è il sistema d'elaborazione in tempo reale. Un sistema d'elaborazione in tempo reale (*real-time*) si usa quando è necessario fissare rigidi vincoli di tempo per le operazioni della CPU o il flusso di dati. S'impiega spesso nella gestione dei dispositivi di controllo per applicazioni specifiche: i sensori forniscono i dati al calcolatore, che li analizza ed eventualmente regola i dispositivi di controllo in modo da modificare i segnali provenienti dagli stessi sensori. I sistemi che controllano esperimenti scientifici, come quelli per la rappresentazione d'immagini in medicina, i sistemi di controllo industriale, sono tutti sistemi d'elaborazione in tempo reale. Si possono considerare tali anche alcuni sistemi d'alimentazione a iniezione per autovetture, i controllori di apparecchiature domestiche e alcuni sistemi d'arma.

Un sistema d'elaborazione in tempo reale presenta vincoli di tempo fissati e ben definiti entro i quali *si deve* effettuare l'elaborazione. Ad esempio, non è accettabile che il braccio di un robot riceva l'istruzione di fermarsi *dopo* aver distrutto la macchina che avrebbe dovuto costruire. Un sistema di questo tipo funziona correttamente solo se riporta i risultati attesi entro i limiti di tempo stabiliti. Questi requisiti non sono richiesti nei sistemi a partizione del tempo, dove è desiderabile, ma non indispensabile, avere una risposta immediata, oppure nei sistemi a lotti, che sono spesso privi di vincoli temporali.

Esistono due tipi di sistemi d'elaborazione in tempo reale: i sistemi d'elaborazione in tempo reale stretto e i sistemi d'elaborazione in tempo reale debole. Un sistema d'elaborazione in tempo reale stretto (*hard real-time*) assicura che i compiti critici siano completati in un dato intervallo di tempo; tale requisito si può garantire se tutti i tempi d'attesa nel sistema sono ben delimitati, dal prelievo dei dati registrati, al tempo necessario al sistema operativo per completare le operazioni richieste su tali dati. I vincoli temporali impongono l'uso di funzioni che sono generalmente presenti nei sistemi d'elaborazione in tempo reale stretto, nei quali è invece limitata, o addirittura mancante, una memoria secondaria di qualsiasi tipo; poiché i dati si memorizzano in una memoria a breve termine o in una di sola lettura (ROM). Quest'ultima, diversamente dalla maggior parte degli altri tipi di memorie, è posta in dispositivi di memorizzazione non volatili che conservano il proprio contenuto anche nel caso d'interruzione dell'energia elettrica. I sistemi d'elaborazione in tempo reale stretto mancano di una gran parte delle caratteristiche comuni agli altri sistemi operativi più progrediti; questi ultimi tendono infatti a separare l'utente dalla struttura fisica del sistema rendendo incerto il tempo d'esecuzione di un'operazione. La memoria virtuale, trattata nel Capitolo 10, è una di queste caratteristiche. I requisiti dei sistemi d'elaborazione in tempo reale stretto sono quindi in conflitto con il modo di funzionamento dei sistemi a partizione del tempo. Poiché nessuno degli attuali sistemi operativi d'impiego generale dispone di funzioni di tempo reale stretto, in questo testo non ci si occuperà ulteriormente di essi.

I sistemi d'elaborazione in tempo reale debole (*soft real-time*) hanno caratteristiche meno restrittive: i processi d'elaborazione in tempo reale critici hanno priorità sugli altri processi e la mantengono sino al completamento dell'esecuzione. Come nei sistemi d'elaborazione in tempo reale stretto, le attese del nucleo del sistema operativo devono essere limitate: un processo in tempo reale non può attendere indefinitamente che il nucleo lo faccia eseguire. Le caratteristiche di capacità d'elaborazione in tempo reale debole si possono inserire in sistemi di altro tipo, tali sistemi hanno comunque un'utilità più limitata dei sistemi d'elaborazione in tempo reale stretto, infatti la mancanza di funzioni di gestione di elaborazioni con vincoli di tempo ne rende pericoloso l'impiego nel controllo industriale e nella robotica. Sono comunque molto utili in diverse aree tra cui la multimedialità, la realtà virtuale, e progetti di ricerca scientifica, come l'esplorazione sottomarina e planetaria. Essi necessitano di caratteristiche dei sistemi operativi che i sistemi d'elaborazione in tempo reale stretto non possono avere. L'uso sempre più diffuso delle funzioni d'elaborazione in tempo reale debole ne ha indotto l'introduzione nei più attuali sistemi operativi, comprese le principali versioni dello UNIX.

Nel Capitolo 6 sono esposte le caratteristiche di scheduling necessarie alla realizzazione delle funzioni d'elaborazione in tempo reale debole in un sistema operativo. Nel Capitolo 10 si descrive la gestione della memoria nelle computazioni in tempo reale. Infine, nel Capitolo 21 si trova una descrizione dei componenti per l'elaborazione in tempo reale del sistema operativo Windows 2000.

http://www.adultpdf.com
Created by TIFF Tools. Trial version, remove this mark, please register this software.

CARATTERISTICHE: • Memoria limitata
• Processori lenti → x non consumare troppa energia
• Video Display di piccole dimensioni

1.8 Sistemi palmari

I sistemi palmari comprendono gli assistenti digitali, noti come PDA (*personal digital assistant*), come i *Palm-pilot* o i telefoni cellulari che possono connettersi alle reti. I progettisti di sistemi e applicazioni per i sistemi palmari devono affrontare molti problemi, alcuni dei quali correlati alle dimensioni di questi dispositivi. I PDA più diffusi sono alti circa dodici centimetri, larghi circa otto, e pesano tra cento e duecentocinquanta grammi. A causa delle piccole dimensioni, la maggior parte dei dispositivi palmari dispone di una memoria limitata, unità d'elaborazione lente e schermi piccoli.

La quantità di memoria di molti dispositivi palmari varia da un megabyte a qualche decina di megabyte (assai meno di quella di un comune PC o stazione di lavoro, che può essere di diverse centinaia di megabyte). Per questo motivo, il sistema operativo e le applicazioni devono gestire la memoria in modo efficiente. Ciò implica, tra l'altro, restituire al gestore della memoria l'intera memoria assegnata, una volta che questa non è più usata. Il Capitolo 10 tratta le tecniche di memoria virtuale, che permettono ai programmatori di scrivere programmi che si comportano come se il sistema avesse più memoria di quella fisicamente disponibile. Attualmente, molti dispositivi palmari non impiegano tecniche di memoria virtuale, quindi impongono ai programmatori di lavorare entro i limiti della memoria fisica.

Un secondo aspetto che riguarda i programmatori è la velocità delle CPU di tali dispositivi, che di solito è solo una frazione di quella di una CPU per PC. D'altra parte, le CPU più veloci consumano più energia, quindi richiederebbero l'impiego di batterie di dimensioni maggiori e la necessità di cambiarle (o ricaricarle) più spesso. Per ridurre al minimo le dimensioni dei dispositivi palmari di solito si usano CPU più piccole, più lente e che consumano meno energia. Quindi, il sistema operativo e le applicazioni si devono progettare in modo da non gravare eccessivamente la CPU.

L'ultimo aspetto che i programmatori devono considerare riguarda le dimensioni dello schermo, tipicamente molto ridotte. Mentre nei PC sono comuni schermi di 21 pollici (circa 36 × 28 centimetri), quelli dei dispositivi palmari di solito non superano i 6 × 9 centimetri. Attività comuni come la lettura della posta elettronica o la consultazione del Web, si devono condensare in schermi molto piccoli. Un metodo per mostrare il contenuto di pagine del Web è il servizio di 'ritagli di pagine' (*Web clipping*), che prevede l'invio di sottoinsiemi delle pagine da mostrare nel dispositivo palmare.

Alcuni dispositivi palmari possono servirsi di tecnologie di comunicazione senza fili, come il sistema BlueTooth (Paragrafo 1.5), che consentono l'accesso al servizio di posta elettronica e al Web. I telefoni cellulari che permettono la connessione alla rete Internet appartengono a questa categoria. Tuttavia, molti PDA attualmente non prevedono l'accesso senza fili; i dati da trasferire a questi dispositivi di solito si trasferiscono prima a un PC o a una stazione di lavoro e successivamente al PDA. Alcuni permettono la copiatura diretta dei dati fra dispositivi per mezzo di una connessione a raggi infrarossi. Generalmente, l'utilità e la trasportabilità dei PDA compensano i limiti delle loro funzioni. Si stanno diffondendo insieme con la possibilità di connessione alle reti e la disponibilità di altri accessori — macchine fotografiche, lettori MP3, ecc. — che li rendono sempre più utili.

1.9 Migrazione delle funzioni

Un esame complessivo dei sistemi operativi per mainframe e microcalcolatori mostra che caratteristiche una volta disponibili solo nei mainframe sono state adottate anche per i microcalcolatori. Gli stessi concetti si adattano alle diverse classi di calcolatori (mainframe, minicalcolatori, microcalcolatori e sistemi palmari). In questo libro sono trattate molte tra le caratteristiche rappresentate nella Figura 1.6. Per cominciare a comprendere i concetti su cui si fondano i sistemi operativi moderni, è necessario essere consapevoli della migrazione delle tante caratteristiche e funzioni dei sistemi operativi, e conoscerne la lunga storia.

Un valido esempio di questo passaggio si è avuto con il sistema operativo MULTICS (*MULTiplexed Information and Computing Services*), sviluppato, tra il 1965 e il 1970 al Massachusetts Institute of Technology (MIT), come servizio d'elaborazione. Questo sistema operativo era eseguito da un mainframe molto grande e complesso, il GE 645. Molti concetti sviluppati originariamente per il MULTICS furono usati in seguito ai Bell Laboratories, che avevano originariamente collaborato allo sviluppo del MULTICS, nella progettazione del sistema UNIX. Il sistema operativo UNIX fu progettato intorno al 1970 per un minicalcolatore PDP-11. Intorno al 1980 le caratteristiche dello UNIX divennero la base per i sistemi operativi per microcalcolatori da esso derivati; attualmente sono state introdotte nei più recenti sistemi operativi come il Microsoft Windows NT, l'IBM OS/2

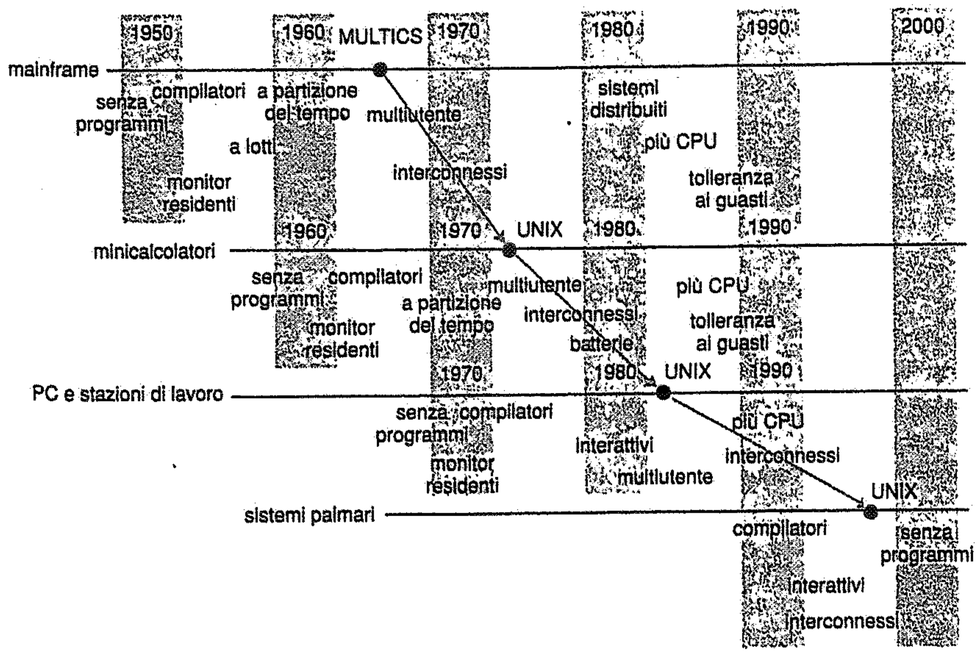


Figura 1.6 Migrazione dei concetti e delle caratteristiche dei sistemi operativi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

e il Macintosh Operating System. Le caratteristiche sviluppate per un grande sistema mainframe sono pertanto passate col tempo anche ai microcalcolatori.

Mentre le caratteristiche dei grandi sistemi operativi si adattavano ai PC, si sviluppavano architetture più potenti, più veloci e più raffinate. Le stazioni di lavoro sono potenti PC, come i calcolatori Sun SPARCstation, HP/Apollo o IBM RS/6000 e i sistemi basati sulle versioni più potenti delle CPU Intel Pentium, con i sistemi operativi Windows 2000 o di derivazione UNIX. Tante università e uffici commerciali hanno molte stazioni di lavoro collegate fra loro tramite reti locali. A mano a mano che i PC progrediscono sia nell'architettura fisica sia nel sistema operativo, la linea che separa le due categorie (mainframe e microcalcolatori) diventa sempre più sfumata.

1.10 Ambienti d'elaborazione

Dopo aver ripercorso lo sviluppo dei sistemi operativi, dai primi rudimentali sistemi a quelli dei PC e dei sistemi palmari, attraverso i sistemi con multiprogrammazione e a partizione del tempo, in questo paragrafo si presenta una breve rassegna dell'uso di tali sistemi nei diversi ambienti d'elaborazione.

1.10.1 Elaborazione tradizionale

Con l'evoluzione delle tecniche d'elaborazione, i confini tra molti ambienti d'elaborazione tradizionale diventano sempre più sfumati. Si consideri ad esempio un tipico ambiente d'ufficio, solo pochi anni fa consisteva di PC connessi in rete, con server che fornivano servizi di accesso ai file e di stampa. L'accesso remoto era difficoltoso e la trasportabilità si otteneva con i calcolatori portatili che permettevano di trasportare una parte dello spazio di lavoro dell'utente. I terminali connessi ai mainframe erano ancora i più diffusi nelle grandi aziende, con ancora meno funzioni d'accesso remoto e di trasportabilità.

Attualmente si tende ad avere più modi d'accesso a questi ambienti; le tecnologie del Web stanno estendendo i confini del calcolo tradizionale, le aziende realizzano i cosiddetti 'portali' che permettono l'accesso tramite il Web ai propri server interni.

I calcolatori di rete sono essenzialmente terminali adatti all'elaborazione basata sul Web. I sistemi palmari possono sincronizzarsi con i PC per permettere un uso estremamente portatile delle informazioni aziendali, possono anche connettersi a reti senza fili per accedere al portale Web dell'azienda (e alle tantissime altre risorse del Web).

Nelle case, la maggior parte degli utenti aveva un solo calcolatore con una lenta connessione via modem all'ufficio, alla rete Internet, o a entrambi. Le connessioni di rete veloci, una volta possibili a costi molto alti, sono ora disponibili a prezzi abbastanza bassi e permettono l'accesso a maggiori quantità di dati sia nell'azienda sia nel Web. Queste connessioni veloci permettono ai calcolatori di casa di trasformarsi in Web server e di formare reti con stampanti, PC client e server. Alcuni ambienti d'elaborazione domestici sono dotati anche di barriere antintrusione (*firewall*) che proteggono dagli attacchi informatici esterni; si tratta di sistemi che solo qualche anno fa costavano migliaia di euro e dieci anni fa nemmeno esistevano.

1.10.2 Elaborazione basata sul Web

Il Web si è diffuso ovunque, consentendo l'accesso ai dati a una varietà di dispositivi inimmaginabile solo alcuni anni fa. I PC sono ancora i dispositivi predominanti per l'accesso al Web, insieme con le stazioni di lavoro (PC di fascia alta), i PDA e persino i telefoni cellulari, anch'essi usati per l'accesso ai dati.

L'elaborazione basata sul Web ha accresciuto l'importanza delle reti; i dispositivi che in passato non avevano accesso ad alcuna rete, ora sono connessi tramite cavi o tramite sistemi senza fili. Grazie al miglioramento delle tecnologie delle reti e all'ottimizzazione dei programmi che realizzano i sistemi di rete, le connessioni alle reti sono assai più veloci.

L'elaborazione basata sul Web ha condotto all'introduzione di nuovi servizi, come il bilanciamento del carico (*load balancing*), che distribuisce le connessioni di rete tra un insieme di server. I sistemi operativi come il Windows 95, che funzionavano come Web client, si sono evoluti in Windows ME e Windows 2000, che possono agire sia come Web client sia come Web server. In generale, essendo diventata una funzione indispensabile, la capacità di connessione al Web ha fatto aumentare la complessità dei dispositivi.

1.10.3 Dispositivi d'elaborazione integrati

I dispositivi d'elaborazione integrati (*embedded*) sono il tipo di calcolatori notevolmente più diffuso; incorporano sistemi operativi per l'elaborazione in tempo reale e si trovano ovunque: nei motori d'automobili, nelle macchine industriali, videoregistratori, forni a microonde, ecc.; di solito svolgono compiti molto specifici. Si tratta tipicamente di sistemi piuttosto semplici, che non dispongono di caratteristiche raffinate come la memoria virtuale e i dischi, e che offrono una scarna o addirittura nessuna interfaccia d'utente, ma che dànno la priorità al controllo e alla gestione di dispositivi fisici, come i motori d'automobili e le macchine industriali.

Come esempio, si possono considerare i già citati sistemi di barriera antintrusione e i sistemi di bilanciamento del carico. Alcuni di questi sono calcolatori d'uso generale con sistemi operativi ordinari — come lo UNIX — e applicazioni specifiche che svolgono tali funzioni; altri sono dispositivi dotati di un opportuno sistema operativo integrato che svolge solo la funzione desiderata.

La diffusione e l'importanza dei dispositivi d'elaborazione integrati, già in espansione, sia come unità separate sia come componenti di reti e del Web, aumenterà decisamente nel futuro. Nelle case, un calcolatore centrale, ordinario o un sistema d'elaborazione integrato, potrebbe controllare il riscaldamento e l'illuminazione, il sistema d'allarme e anche la macchina del caffè. L'accesso al Web permetterebbe di istruire il proprio calcolatore affinché accenda il riscaldamento per scaldare i locali prima dell'arrivo a casa, e chiami il negozio di generi alimentari appena il frigorifero rileva che il latte è finito.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

1.11 Sommario

I sistemi operativi sono stati sviluppati negli ultimi quarantacinque anni soprattutto per due motivi: innanzi tutto, organizzano le attività d'elaborazione in modo da assicurare un buon rendimento del sistema di calcolo; inoltre, offrono un ambiente idoneo allo sviluppo e all'esecuzione dei programmi. Inizialmente i calcolatori si controllavano da una console; programmi, come assemblatori, caricatori, collegatori e compilatori, rendevano più agevole la programmazione del sistema, ma richiedevano un elevato tempo di preparazione. Per ridurre tale tempo, si impiegarono operatori che riunivano in lotti (*batch*) i lavori d'elaborazione (*job*) simili.

I sistemi a lotti permisero di eseguire in sequenza i lavori in modo automatico grazie a un sistema operativo residente, migliorando notevolmente il rendimento complessivo del sistema di calcolo, il quale non doveva più attendere l'esecuzione delle operazioni da parte dell'operatore. Tuttavia, l'utilizzo della CPU continuava a essere basso, a causa della lentezza dei dispositivi di I/O rispetto alla velocità della CPU. L'elaborazione fuori linea da parte dei dispositivi lenti consentì, per ciascuna CPU, l'impiego di più sistemi per il trasferimento dei dati dai lettori di schede ai nastri magnetici e di più sistemi per il trasferimento dei dati dai nastri magnetici alle stampanti.

Per migliorare il rendimento generale del sistema di calcolo si introdusse il concetto di multiprogrammazione. Con la multiprogrammazione più lavori si possono tenere contemporaneamente nella memoria; la CPU passa continuamente tra l'esecuzione di un lavoro e l'altro, è così sfruttata al massimo e si riduce il tempo totale necessario all'esecuzione di tutti i lavori.

La multiprogrammazione consente anche l'impiego della tecnica di partizione del tempo d'elaborazione (*time sharing*), che permette a più utenti (da uno a parecchie centinaia) di usare contemporaneamente e interattivamente uno stesso sistema.

I PC sono microcalcolatori assai più piccoli e meno costosi dei *mainframe*. I sistemi operativi di questi calcolatori hanno tratto vantaggio dallo sviluppo dei sistemi operativi per mainframe. Tuttavia, poiché un utente ha l'uso esclusivo di un calcolatore, l'utilizzo della CPU non ha più un ruolo predominante. Di conseguenza, i criteri progettuali seguiti per i sistemi operativi per mainframe possono non essere adatti a tali sistemi più piccoli. Poiché attualmente i PC sono spesso connessi ad altri calcolatori e utenti tramite reti e tramite il Web, altri criteri progettuali, come quelli relativi alla sicurezza sono appropriati sia per i piccoli sia per i grandi sistemi.

I sistemi paralleli hanno più unità d'elaborazione in stretta comunicazione; le unità d'elaborazione condividono i canali di comunicazione all'interno del calcolatore (*bus*) e talvolta anche la memoria e i dispositivi periferici. Tali sistemi consentono una maggiore produttività e affidabilità. I sistemi distribuiti consentono la condivisione di risorse tra calcolatori distribuiti in diverse aree geografiche. Le batterie di sistemi (*cluster system*) permettono a un gruppo di calcolatori di eseguire elaborazioni di dati contenuti in mezzi di memorizzazione condivisi, e consentono che l'elaborazione prosegua anche nel caso di guasti in un sottoinsieme dei membri del gruppo.

I sistemi d'elaborazione in tempo reale stretto s'impiegano spesso come dispositivi di controllo in applicazioni specifiche, e si caratterizzano per il loro funzionamento nel rispetto di ben definiti vincoli di tempo. L'elaborazione *deve* avvenire nell'ambito di tali limiti. I sistemi d'elaborazione in tempo reale debole hanno invece caratteristiche meno restrittive.

Recentemente, l'influenza del World Wide Web ha incoraggiato lo sviluppo di sistemi operativi che includono come funzioni proprie programmi di consultazione del Web, d'interconnessione in rete e di comunicazione.

In questo capitolo è illustrata la progressione logica dello sviluppo dei sistemi operativi, condotta dall'inclusione nell'architettura delle CPU di caratteristiche ormai indispensabili per migliorare le funzioni dei sistemi operativi. Questa stessa tendenza si può riscontrare oggi nell'evoluzione dei PC, dove dispositivi economici si perfezionano sempre più per consentire il miglioramento delle prestazioni.

1.12 Esercizi

1.1 Dite quali sono i tre scopi principali di un sistema operativo.

1.2 Elencate le quattro operazioni da compiere per far eseguire un programma da un calcolatore riservato alla sua sola esecuzione.

1.3 Dite qual è il vantaggio principale offerto dalla multiprogrammazione.

1.4 Dite quali sono le differenze principali tra i sistemi operativi per mainframe e per PC.

1.5 In un ambiente con multiprogrammazione e a partizione del tempo più utenti condividono il sistema. Questa situazione può causare diversi problemi di sicurezza.

a) Dite quali sono questi problemi.
b) Dite se in un sistema a partizione del tempo si può assicurare lo stesso grado di sicurezza di un sistema riservato a un solo utente. Giustificate la risposta.

1.6 Definite le caratteristiche fondamentali dei seguenti tipi di sistema operativo:

a) a lotti;
b) interattivi;
c) a partizione del tempo;
d) elaborazione in tempo reale;
e) di rete;
f) paralleli;
g) distribuiti;
h) batterie d'elaborazione;
i) palmari.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

1.7 Sono state sottolineate le caratteristiche che un sistema operativo deve possedere affinché si faccia un uso efficiente dei dispositivi fisici. Dite in quali casi è appropriato lasciar cadere tale principio e 'sprecare' risorse, e spiegate perché in tali sistemi non si tratta propriamente di spreco.

1.8 Dite in quali circostanze l'uso di un sistema a partizione del tempo può risultare più conveniente rispetto a un PC o a una stazione di lavoro per un singolo utente.

1.9 Descrivete le differenze tra multielaborazione simmetrica e asimmetrica, e indicate tre vantaggi e uno svantaggio dei sistemi paralleli.

1.10 Dite qual è la maggiore difficoltà che un programmatore incontra durante la scrittura di un sistema operativo per un ambiente in tempo reale.

1.11 Considerate le diverse definizioni di *sistema operativo*. Riflettete sull'opportunità che un sistema operativo includa programmi di consultazione del Web e di gestione della posta elettronica. Discutete le posizioni pro e contro e argomentate le risposte.

1.12 Dite quali sono i compromessi intrinseci nei sistemi palmari.

1.13 Considerate una batteria di sistemi, costituita di due nodi, per l'elaborazione di una base di dati. Descrivete due modi in cui i programmi di tale sistema possono gestire l'accesso ai dati. Discutete i vantaggi e gli svantaggi di ciascuno.

1.13 Note bibliografiche

I sistemi a partizione del tempo d'elaborazione furono proposti per la prima volta da [Strachey 1959]. Tali primissimi sistemi furono il Compatible Time-Sharing System (CTSS), sviluppato al Massachusetts Institute of Technology (MIT) [Corbato et al. 1962], e il sistema SDC Q-32, costruito dalla System Development Corporation [Schwartz et al. 1964], [Schwartz e Weissman 1967]. Altri, tra i primi, ma più progrediti sistemi, comprendevano il sistema (MULTICS) MULTIplexed Information and Computing Services, sviluppato al MIT [Corbato e Vyssotsky 1965], il sistema XDS-940, sviluppato alla University of California at Berkeley [Lichtenberger e Pirtle 1965] e l'IBM TSS/360 [Lett e Konigsford 1968].

L'MS-DOS e i PC sono descritti da [Norton 1986] e [Norton e Wilton 1988]. Una trattazione di carattere generale sull'architettura e il sistema operativo degli Apple Macintosh è fornita da [Apple 1987]. Una descrizione del sistema operativo OS/2 è fornita da [Microsoft 1989], ulteriori informazioni si trovano in [Letwin 1988] e [Deitel e Kogan 1992]. [Solomon e Russinovich 2000] tratta la struttura del sistema operativo Microsoft Windows 2000.

[Buyya 1999] offre una buona trattazione dell'elaborazione con le batterie di sistemi; [Ahmed 2000] ne presenta i progressi più recenti.

[Murray 1998] e [Rhodes e McKeehan 1999] trattano i sistemi palmari.

Esistono molti libri di testo sui sistemi operativi, tra i quali [Milenkovic 1987], [Finkel 1988], [Deitel 1990], [Stallings 2000b], [Nutt 1999] e [Tanenbaum 2001].

Capitolo 2

Strutture dei sistemi di calcolo

Per potersi addentrare nello studio dei dettagli di un sistema operativo è necessaria una conoscenza generale della struttura di un calcolatore. Lo scopo del presente capitolo è proprio consolidare le basi di questa conoscenza proponendo una trattazione introduttiva dei diversi elementi di questa struttura. La trattazione riguarda principalmente le architetture dei calcolatori, quindi chi già conosce questo argomento può saltare il capitolo o limitarsi a un'occhiata. I primi argomenti trattati comprendono l'avviamento del sistema, i sistemi di I/O e la memorizzazione.

Un sistema operativo deve assicurare il corretto funzionamento di un calcolatore. Al fine di evitare che i programmi utenti interferiscano con le operazioni proprie del sistema, l'architettura del calcolatore deve fornire meccanismi appropriati per assicurarne il corretto comportamento. Nella sua parte conclusiva, il capitolo contiene una descrizione delle caratteristiche fondamentali che un'architettura dovrebbe possedere per permettere lo sviluppo di un efficace sistema operativo, e termina con un'introduzione all'architettura delle reti di calcolatori

2.1 Funzionamento di un sistema di calcolo

Un moderno calcolatore d'uso generale è composto da una CPU e da un certo numero di controllori di dispositivi connessi attraverso un canale di comunicazione comune (*bus*) che permette l'accesso alla memoria condivisa dal sistema (Figura 2.1). Ciascuno di questi controllori si occupa di un particolare tipo di dispositivo fisico (ad esempio, unità a disco, dispositivi audio e unità video). La CPU e questi controllori possono operare in modo concorrente; alcuni possono anche contendere alla CPU i cicli d'accesso alla memoria. La sincronizzazione degli accessi alla memoria è garantita dalla presenza di un controllore di memoria.

L'avviamento del sistema, conseguente all'accensione fisica di un calcolatore, così come il riavvio di un calcolatore già acceso, richiede la presenza di uno specifico programma iniziale, di solito non troppo complesso, detto programma d'avviamento (*bootstrap program*), in genere contenuto in una memoria a sola lettura (*read only memory* — ROM),

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

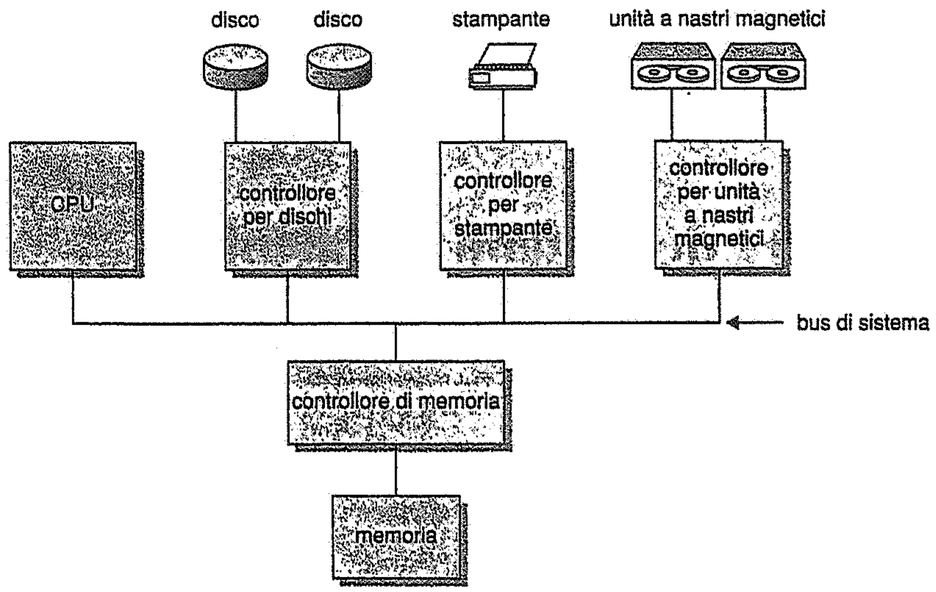


Figura 2.1 Moderno sistema di calcolo.

ad esempio di tipo microprogrammato (*firmware*) o EEPROM, facente parte della struttura fisica del calcolatore. La sua funzione consiste nell'inizializzare i diversi componenti del sistema, dai registri della CPU ai controllori dei diversi dispositivi, fino al contenuto della memoria centrale. Il programma d'avviamento deve caricare nella memoria il sistema operativo e avviarne l'esecuzione, perciò individua e carica nella memoria il nucleo del sistema operativo, quindi il sistema operativo avvia l'esecuzione del primo processo d'elaborazione, ad esempio `init`, e attende che si verifichi qualche evento. Un evento è di solito segnalato da una 'interruzione' dell'attuale sequenza d'esecuzione della CPU, che può essere causata da un dispositivo fisico o da un programma. Nel primo caso si parla di segnale d'interruzione o, più brevemente, interruzione (*interrupt*); si tratta di segnali che i controllori dei dispositivi e altri elementi dell'architettura possono inviare alla CPU, di solito attraverso il bus di sistema. Nel secondo caso si parla di segnale di eccezione o, più brevemente, eccezione (*exception* o *trap*), che può essere causata da un programma in esecuzione a seguito di un evento eccezionale, riconosciuto tramite l'architettura della CPU (ad esempio un errore: una divisione per zero o un accesso alla memoria non valido); oppure a seguito di una richiesta specifica effettuata da un programma utente per ottenere l'esecuzione di un servizio del sistema operativo, attraverso una speciale istruzione detta **chiamata del sistema** (*system call*) o **chiamata del supervisore** (*supervisor call* — SVC).

I sistemi operativi moderni sono caratterizzati dal fatto di essere **guidati dalle interruzioni**: se non ci sono processi da eseguire, dispositivi di I/O da servire o utenti con cui interagire, un sistema operativo resta inattivo, nell'attesa che accada qualcosa. L'occorrenza di un evento è di solito segnalata da un'interruzione. La natura guidata dalle interruzioni di un sistema operativo definisce tale struttura generale del sistema. Per ciascun tipo d'interruzione, diversi segmenti di codice del sistema operativo determinano l'azione da intraprendere; è presente una procedura di servizio del tipo d'interruzione individuata.

Ogniqualvolta riceve un segnale d'interruzione, la CPU interrompe l'elaborazione corrente e trasferisce immediatamente l'esecuzione a una fissata locazione della memoria. Di solito, questa locazione contiene l'indirizzo iniziale della procedura di servizio per quel dato segnale d'interruzione. Una volta completata l'esecuzione della procedura richiesta, la CPU riprende l'elaborazione precedentemente interrotta. La Figura 2.2 mostra il diagramma temporale della gestione di un simile evento.

I segnali d'interruzione sono un elemento importante nell'architettura di un calcolatore, e ciascun tipo di calcolatore ha il proprio meccanismo delle interruzioni; ciò nonostante molte funzioni sono comuni. Un segnale d'interruzione deve causare il trasferimento del controllo all'appropriata procedura di servizio dell'evento a esso associato. Il modo più semplice per gestire quest'operazione è quello di impiegare una procedura generale che esamina le informazioni presenti nel segnale d'interruzione, e invoca la procedura di gestione dello specifico segnale d'interruzione. D'altra parte, la gestione di un'interruzione deve essere molto rapida perciò, considerando che il numero dei possibili segnali d'interruzione è predefinito, si può usare una tabella di puntatori alle specifiche procedure. In questo modo, l'attivazione delle procedure di servizio delle interruzioni avviene in modo indiretto attraverso questa tabella, senza procedure intermediarie. In genere la tabella di puntatori contenente gli indirizzi delle procedure di servizio delle interruzioni è mantenuta nella memoria bassa (ad esempio, le prime 100 locazioni).

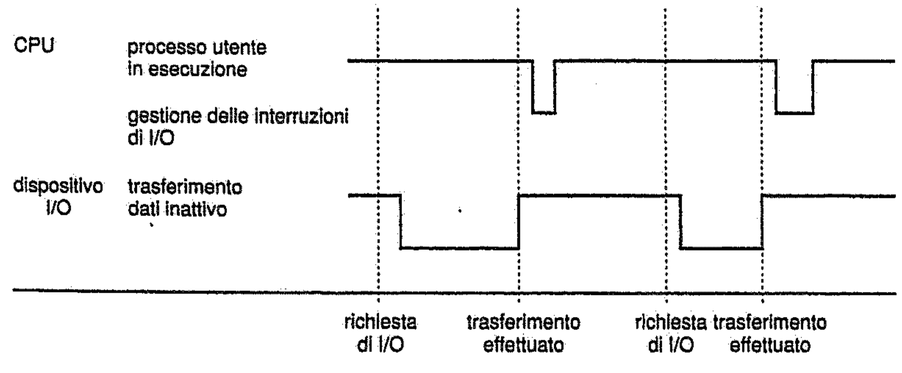


Figura 2.2 Diagramma temporale delle interruzioni per un singolo processo che emette dati.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'accesso a questa sequenza d'indirizzi, detta **vettore delle interruzioni**, avviene per mezzo di un indice, codificato nello stesso segnale d'interruzione, allo scopo di fornire l'indirizzo della procedura di servizio relativa all'evento segnalato dall'interruzione. Sistemi operativi radicalmente differenti, come l'MS-DOS e lo UNIX, usano lo stesso meccanismo di gestione delle interruzioni.

L'architettura di gestione delle interruzioni deve anche salvare l'indirizzo dell'istruzione interrotta. Molti sistemi di vecchia concezione si limitavano a memorizzare l'indirizzo dell'istruzione interrotta in una locazione fissata o indicizzata dal numero di dispositivo, architetture più recenti memorizzano l'indirizzo di ritorno nella pila (*stack*) di sistema. Se la procedura di gestione dell'interruzione richiede la modifica dello stato della CPU, ad esempio, modificando il contenuto di qualche registro, deve salvare esplicitamente lo stato corrente per poterlo ripristinare prima di restituire il controllo. Terminato il servizio dell'interruzione, si carica nel **contatore di programma** (*program counter*) — detto anche, in modo più appropriato, **puntatore d'istruzione** (*instruction pointer*), che contiene l'indirizzo della prossima istruzione da eseguire — l'indirizzo di ritorno precedentemente salvato, consentendo la ripresa della computazione interrotta come se nulla fosse accaduto.

Una chiamata del sistema si può effettuare in modi diversi, secondo le funzioni messe a disposizione dalla CPU. In qualsiasi forma sia specificata, si tratta del modo in cui un processo richiede al sistema operativo di eseguire una qualsiasi funzione. Di solito le chiamate del sistema prendono la forma di un segnale di eccezione che fa riferimento a una specifica locazione del vettore delle interruzioni. Tale eccezione si può ottenere tramite una generica istruzione `trap`, anche se alcuni sistemi (come la famiglia Mips R2000) hanno una specifica istruzione `syscall`.

2.2 Struttura di I/O

Come si è accennato nel Paragrafo 2.1, un calcolatore d'uso generale è composto di una CPU e di un insieme di controllori di dispositivi connessi mediante un bus comune. Ciascun controllore deve occuparsi di un particolare tipo di dispositivo e, secondo la sua natura, può gestire uno o più dispositivi a esso connessi. Un controllore SCSI (acronimo di *small computer systems interface*) ad esempio è capace di controllare sette o più dispositivi. Un controllore di dispositivo dispone di una propria memoria interna, detta memoria di transito (*buffer*), e di un insieme di registri. Il controllore è responsabile del trasferimento dei dati tra i dispositivi periferici a esso connessi e la propria memoria di transito. La dimensione di quest'ultima varia da controllore a controllore, secondo le caratteristiche del dispositivo da gestire. La dimensione della memoria di transito di un controllore per unità a disco ad esempio corrisponde alla più piccola porzione di disco indirizzabile, detta **settore**, (di solito 512 byte) o a un suo multiplo.

2.2.1 Interruzioni di I/O

Per iniziare un'operazione di I/O, la CPU carica i registri appropriati del controllore del dispositivo con il quale intende comunicare. Il controllore, a sua volta, esamina il contenuto di questi registri allo scopo di determinare l'azione da compiere. Ad esempio, se la CPU richiede l'esecuzione di un'operazione di lettura, il controllore provvede al trasferimento dei dati dal dispositivo alla propria memoria di transito. Completato il trasferimento, il controllore informa la CPU del completamento dell'operazione emettendo un segnale d'interruzione.

Di solito questa situazione si verifica in seguito a una richiesta di I/O da parte di un processo utente. Una volta iniziata l'operazione di I/O, il corso dell'esecuzione può seguire due percorsi distinti. Nel caso più semplice, si restituisce il controllo al processo utente solamente dopo il completamento dell'operazione di I/O; questo modo di operare si chiama **I/O sincrono**. L'altra possibilità, detta **I/O asincrono**, prevede la restituzione immediata del controllo al processo utente, senza attendere il completamento dell'operazione di I/O; in questo modo, l'I/O può proseguire mentre il sistema esegue altre operazioni (Figura 2.3).

METODO SINCRONO

L'attesa del completamento di un'operazione di I/O si può realizzare in due modi diversi. Alcuni calcolatori hanno una particolare istruzione d'attesa, `wait`, che sospende la CPU fino al successivo segnale d'interruzione. Nei calcolatori che non prevedono un'istruzione del genere si può impiegare un ciclo d'attesa:

```
Attesa: JUMP Attesa
```

Questo ciclo stretto persiste fino all'arrivo di un'interruzione che trasferisce il controllo a un'altra sezione del sistema operativo. Il precedente ciclo deve inoltre verificare le eventuali richieste di comunicazione di tutti i controllori dei dispositivi di I/O che non emettono segnali d'interruzione, ma che impostano un indicatore (*flag*) in uno dei propri registri e attendono che il sistema lo noti.

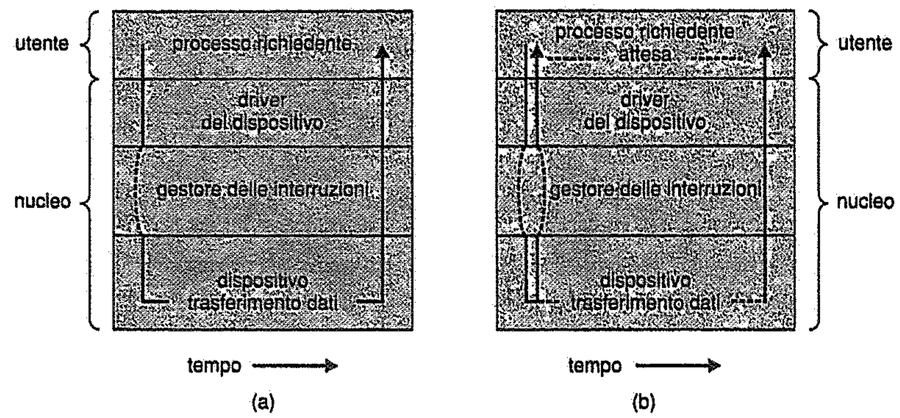


Figura 2.3 Due metodi di I/O: a) sincrono; b) asincrono.

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

Se la CPU attende sempre il completamento di un'operazione di I/O si ha al più una richiesta pendente alla volta. Quindi, ogniqualvolta si presenta un'interruzione di I/O, il sistema operativo riconosce immediatamente il dispositivo che ha emesso il segnale d'interruzione. D'altra parte, questo metodo esclude la possibilità di operazioni di I/O concorrenti su più dispositivi, così come esclude la possibilità di sovrapporre utili elaborazioni con l'I/O.

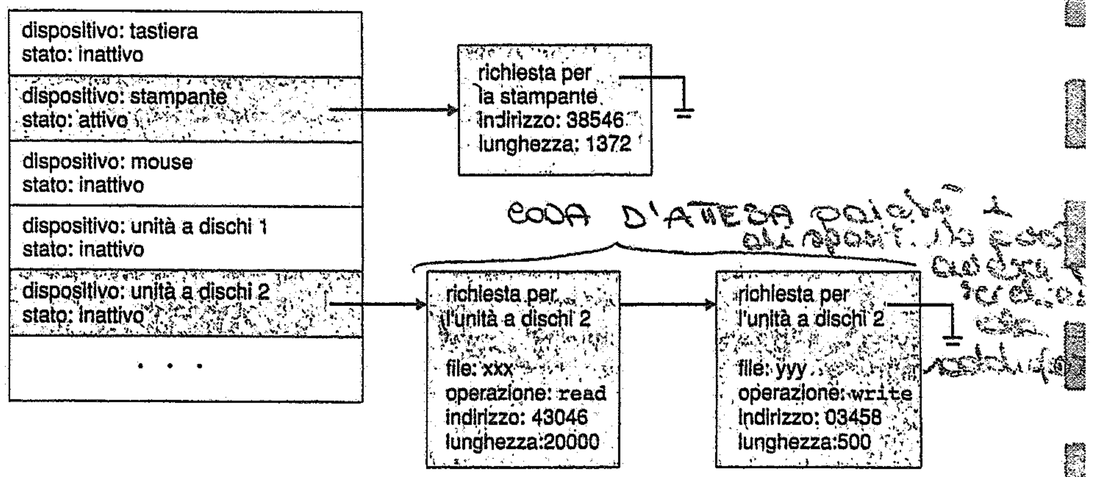
METODO ASINCRONO NO

Un'alternativa migliore consiste nell'avviare l'operazione di I/O quindi proseguire con l'elaborazione di altro codice del sistema operativo o di un programma utente. In questo caso è necessaria la presenza di una chiamata del sistema che consenta al programma utente di attendere, se è richiesto, il completamento dell'operazione. Se nessun programma utente è pronto per l'esecuzione, e il sistema operativo non ha altro lavoro da fare, si ha ancora bisogno dell'istruzione wait o del ciclo d'attesa di cui si è parlato in precedenza; il sistema deve inoltre poter tenere traccia delle varie richieste di I/O eventualmente attive allo stesso istante. A questo scopo, il sistema operativo mantiene una **tabella di stato dei dispositivi** (Figura 2.4) composta da tanti elementi quanti sono i dispositivi di I/O connessi al calcolatore. Ciascun elemento della tabella specifica il tipo di dispositivo cui fa riferimento, il suo indirizzo e il suo stato (disattivato, inattivo, occupato). Se un dispositivo è impegnato nel soddisfare una precedente richiesta, si memorizza il tipo di richiesta insieme con altri parametri nell'elemento della tabella corrispondente a quel dispositivo. Poiché altri processi possono sottoporre una richiesta allo stesso dispositivo, il sistema operativo mantiene una coda d'attesa per ciascun dispositivo di I/O.

Quando un controllore di un dispositivo di I/O invia un segnale d'interruzione al sistema per richiedere un servizio, il sistema operativo innanzi tutto individua il controllore del dispositivo che ha emesso il segnale d'interruzione, quindi accede alla tabella dei dispositivi, risale allo stato in cui il dispositivo si trova e modifica l'elemento della tabella indicando l'occorrenza dell'interruzione. La maggior parte dei dispositivi impiega le interruzioni per segnalare il completamento di una richiesta di I/O. Se ci sono ulteriori richieste nella coda d'attesa di quel dispositivo, il sistema operativo avvia l'elaborazione della prima richiesta.

Infine, si restituisce il controllo acquisito tramite l'interruzione di I/O. Se un processo attendeva il completamento di tale richiesta (come risulta dalla tabella di stato del dispositivo) è ora possibile restituirgli il controllo. Altrimenti, il sistema riprende ciò che stava facendo prima dell'interruzione di I/O: l'esecuzione del programma utente (il programma ha attivato l'operazione di I/O appena conclusa senza però attendere il suo completamento) o l'esecuzione del ciclo d'attesa (in questo caso il programma ha sottoposto al sistema due o più operazioni di I/O e attende che una di esse, ma non quella attualmente terminata, sia completata). In un sistema a partizione del tempo d'elaborazione il sistema operativo potrebbe cedere il controllo a un processo pronto per l'esecuzione diverso da quello che ha richiesto l'operazione di I/O.

Gli schemi adottati dagli specifici dispositivi di immissione di dati possono seguire uno schema differente. Molti sistemi interattivi consentono all'utente di immettere dati con la propria tastiera ancora prima che il programma cui sono diretti ne faccia richiesta. In questo caso, il controllore del dispositivo può inviare al sistema un'interruzione, con cui segnala la presenza d'informazioni in arrivo dal terminale, anche se le informazioni



CODA D'ATTESA

Figura 2.4 Tabella di stato dei dispositivi. → 1 riga x ciascun dispositivo di I/O, contenente tipo, indirizzo e stato.

registrate nella tabella del suo stato indicano che nessun programma ha richiesto dati da quel dispositivo. Per consentire quest'eventualità è necessario fornire un'area della memoria (talvolta, anche in questo caso si parla di *buffer*) nella quale memorizzare i dati inseriti dall'utente nell'attesa che un programma ne faccia richiesta. In generale si fa corrispondere un'area della memoria a ciascun dispositivo d'immissione di dati connesso al sistema.

Il principale vantaggio di tale schema di I/O asincrono è un incremento dell'efficienza del sistema. Durante il servizio di una richiesta di I/O, la CPU può proseguire nelle proprie elaborazioni, o avviare operazioni di I/O su altri dispositivi. Poiché le operazioni di I/O possono essere lente rispetto alla velocità della CPU, il sistema può sfruttare molto meglio gli strumenti di cui dispone. Nel Paragrafo 2.2.2 s'introduce un altro meccanismo mediante il quale è possibile migliorare ulteriormente le prestazioni di un calcolatore.

2.2.2 Accesso diretto alla memoria

In un semplice driver di un dispositivo di immissione di dati, quando si deve leggere una riga dal dispositivo, s'invia al calcolatore il primo carattere inserito. Una volta ricevuto il carattere, il canale di comunicazione asincrona, o porta seriale, cui è connesso il dispositivo, invia un segnale d'interruzione che la CPU riceve prima di iniziare l'esecuzione di una nuova istruzione. (Se la CPU riceve il segnale d'interruzione durante l'esecuzione di un'istruzione, si ritarda il suo riconoscimento fino al completamento dell'esecuzione in corso.) Salvato l'indirizzo dell'istruzione interrotta, si trasferisce il controllo alla procedura di servizio relativa al dispositivo richiedente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La procedura di servizio dell'interruzione provvede a sua volta alla memorizzazione del contenuto dei registri della CPU che intende usare, verifica che la più recente operazione d'immissione di dati non abbia generato alcuna condizione d'errore, quindi preleva il carattere dal controllore del dispositivo per memorizzarlo in una locazione dell'area della memoria riservata a tale dispositivo. Un altro compito della procedura è l'aggiornamento dei puntatori e dei contatori, al fine di assicurare che il successivo carattere ricevuto dal dispositivo sia effettivamente memorizzato nella locazione successiva dell'area suddetta. Dopodiché imposta il valore di un indicatore contenuto nella memoria per segnalare agli altri elementi del sistema operativo, a loro volta responsabili dell'elaborazione dei dati ricevuti e del loro trasferimento al programma che ne ha richiesto l'invio (Paragrafo 2.5), che il carattere immesso è stato effettivamente ricevuto. Quindi, la procedura termina la propria funzione ripristinando i registri con i valori precedentemente memorizzati e restituendo il controllo dell'esecuzione all'istruzione interrotta.

Supponendo di scrivere caratteri a un terminale capace di una velocità di trasmissione di 9600 bps, la frequenza con la quale il dispositivo può ricevere e inviare i caratteri è approssimativamente di un carattere ogni millisecondo, o 1000 microsecondi. Una procedura ben codificata può trasferire nella memoria un carattere ogni due microsecondi, lasciando alla CPU 998 microsecondi su 1000 per altre elaborazioni o per la gestione di altre interruzioni. Questi parametri evidenziano i motivi per i quali alle operazioni di I/O gestite in modo asincrono di solito si attribuiscono segnali d'interruzione a bassa priorità, consentendo a quelli più urgenti di essere gestiti prima o, addirittura, di interrompere la gestione di interruzioni a bassa priorità. D'altra parte, un dispositivo ad alta velocità, come un'unità a nastro, un'unità a disco o una rete di trasmissione di dati, può trasmettere a una velocità paragonabile a quella della memoria, perciò la CPU richiederebbe due microsecondi per gestire interruzioni che si verificano, ad esempio, ogni quattro microsecondi, lasciando poco tempo all'esecuzione dei processi.

Il problema si può risolvere con l'ausilio della tecnica di **accesso diretto alla memoria** (*direct memory access* — DMA). Una volta impostata l'area della memoria, i puntatori e i contatori per il dispositivo di I/O, il controllore trasferisce un intero blocco di dati dalla propria memoria di transito direttamente nella memoria centrale, o viceversa, senza alcun intervento da parte della CPU. In questo modo il trasferimento richiede una sola interruzione per ogni blocco di dati trasferito, piuttosto che per ogni byte (o parola) come avviene nella gestione dei dispositivi più lenti.

Per quel che riguarda le operazioni di base della CPU il principio è lo stesso. Un programma utente, o lo stesso sistema operativo, richiede il trasferimento di un insieme di informazioni. Il sistema individua l'area della memoria interessata (vuota se si tratta di un'operazione di immissione di dati, piena se si tratta di un'operazione di emissione di dati); le dimensioni di queste aree di memoria variano di solito da 128 a 4096 byte, secondo il tipo di dispositivo. Quindi un elemento del sistema operativo, detto **driver di dispositivo**, imposta gli appositi registri del controllore DMA affinché questo impieghi gli appropriati indirizzi di sorgente e destinazione e la lunghezza del blocco da trasferire. Il controllore DMA quindi riceve l'istruzione d'avvio dell'operazione di I/O; mentre il controllore DMA esegue il trasferimento dei dati, la CPU è libera di eseguire altri compiti. Siccome la memoria generalmente può trasferire una sola parola per volta, il controllore DMA sottrae alla CPU cicli d'accesso alla memoria; ciò può rallentare l'esecuzione da parte della CPU mentre è in atto il trasferimento DMA. Il controllore DMA notifica il completamento dell'operazione inviando un'interruzione alla CPU.

2.3 Struttura della memoria

Per essere eseguito, un programma per calcolatore deve risiedere nella memoria centrale, detta anche **memoria ad accesso diretto** (*random access memory* — RAM). Infatti, la memoria centrale è la sola area di memoria di grandi dimensioni (dai milioni ai miliardi di byte) direttamente accessibile dalla CPU. È realizzata con una tecnologia basata sui semiconduttori detta **memoria dinamica ad accesso diretto** (*dynamic random access memory* — DRAM), ed è strutturata come un vettore di parole di memoria. Ciascuna parola possiede un proprio indirizzo. L'interazione avviene per mezzo di una sequenza di istruzioni `load` e `store` opportunamente indirizzate. L'istruzione `load` trasferisce il contenuto di una parola della memoria centrale in uno dei registri interni della CPU, mentre la `store` copia il contenuto di uno di questi registri nella locazione di memoria specificata. Oltre agli accessi dovuti alle operazioni `load` e `store`, che si richiedono in modo esplicito, la CPU preleva automaticamente dalla memoria centrale le istruzioni da eseguire.

La tipica sequenza d'esecuzione di un'istruzione, in un sistema con architettura di von Neumann, comincia con il prelievo (*fetch*) di un'istruzione dalla memoria centrale e il suo trasferimento nel registro d'istruzione. Quindi si decodifica l'istruzione che eventualmente può richiedere il trasferimento di alcuni operandi dalla memoria in alcuni registri interni. Una volta terminata l'esecuzione dell'istruzione sugli operandi, il risultato si può scrivere nella memoria. Si noti che l'unità di memoria 'vede' soltanto una sequenza d'indirizzi di memoria; non importa né il modo in cui questi sono stati generati (dal contatore di programma, per indicizzazione, riferimento indiretto, indirizzamento immediato, e così via) né tanto meno a cosa fanno riferimento (istruzioni o dati). Di conseguenza, anche la presente trattazione non si cura di *come* questi indirizzi siano generati all'interno dei programmi e si occupa semplicemente delle sequenze d'indirizzi della memoria generate dai programmi in esecuzione.

In teoria, si vorrebbe che sia i programmi sia i dati da essi trattati potessero risiedere in modo permanente nella memoria centrale. Questo non è possibile per i seguenti due motivi:

1. la capacità della memoria centrale non è di solito sufficiente a contenere in modo permanente tutti i programmi e i dati richiesti;
2. la memoria centrale è un dispositivo di memorizzazione *volatile*, sicché perde il proprio contenuto quando si spegne il sistema o si ha un'interruzione dell'alimentazione elettrica.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per queste ragioni la maggior parte dei sistemi di calcolo comprende una **memoria secondaria** come estensione della memoria centrale. La caratteristica fondamentale di questi dispositivi è la capacità di conservare in modo permanente grandi quantità di informazioni.

Il dispositivo più comunemente impiegato a questo scopo è l'unità a disco magnetico, adoperato per la memorizzazione sia di programmi sia di dati. La maggior parte dei programmi (elaboratori di testi, fogli di calcolo, programmi di consultazione del Web, compilatori, e così via) è mantenuta in un disco sino al momento del caricamento nella memoria, e fa uso di un disco come sorgente e destinazione delle informazioni elaborate. Come si spiega nel Capitolo 14, una corretta gestione delle unità a disco è di fondamentale importanza per un sistema di calcolo.

Occorre comunque dire che la struttura proposta (composta da registri, memoria centrale e unità a disco) rappresenta semplicemente una delle possibili configurazioni del sistema di memorizzazione di un calcolatore. Esistono altri tipi di memorie, ad esempio le memorie cache, i CD-ROM e i nastri magnetici. Qualsiasi architettura fornisce le funzioni fondamentali che consentono la memorizzazione di un dato e il suo mantenimento fino all'uso successivo. Le caratteristiche che differenziano i diversi sistemi di memorizzazione sono la velocità, il costo, le dimensioni e la volatilità. Nei Paragrafi 2.3.1, 2.3.2 e 2.3.3 si descrivono la memoria centrale, i dischi e i nastri magnetici, che rappresentano le caratteristiche generali dei più importanti dispositivi di memorizzazione disponibili in commercio. Nel Capitolo 14 si descrivono le caratteristiche di alcuni dispositivi specifici, come le unità a dischi magnetici, le unità a dischetti, le unità a dischi ottici (CD, DVD).

2.3.1 Memoria centrale

Gli unici dispositivi di memoria direttamente accessibili dalla CPU sono la memoria centrale e i registri interni alla stessa CPU. Esistono istruzioni di macchina che accettano indirizzi di memoria come argomenti, ma nessuna istruzione accetta indirizzi di un disco. Per questo motivo tutte le istruzioni in esecuzione, unitamente ai dati da esse elaborati, devono risiedere in questi dispositivi direttamente accessibili. Se i dati non risiedono nella memoria devono esservi trasferiti prima che la CPU possa operare su di essi.

Nel caso di operazioni di I/O, come s'è menzionato nel Paragrafo 2.1, ciascun controllore di I/O è dotato di registri nei quali mantenere i comandi e i dati da trasferire. Di solito speciali istruzioni di I/O permettono il trasferimento dei dati tra questi registri e la memoria centrale. Per rendere più agevole l'accesso ai dispositivi di I/O molte architetture di calcolatori forniscono una tecnica di I/O detta **I/O associato alla memoria** (*memory-mapped I/O*). Questa tecnica consiste nel far corrispondere i registri dei dispositivi a intervalli dello spazio d'indirizzi della CPU. In questo modo, ogni operazione di lettura o scrittura a tali indirizzi — che la CPU tratta come se fossero indirizzi della memoria, nonostante, è bene sottolinearlo, non si tratti d'indirizzi di locazioni della memoria centrale — comporta un trasferimento diretto di dati con i registri del dispositivo. Questo metodo è particolarmente adatto alla gestione di dispositivi dotati di brevi tempi di risposta,

I/O associato alla memoria consente di far corrispondere i registri dei dispositivi ad indirizzi della CPU

come i controllori video. Nei PC, ad esempio, i controllori video dispongono di un'ampia memoria, detta memoria video, le cui locazioni corrispondono alle posizioni sullo schermo. Le locazioni di tale memoria sono messe in corrispondenza, dall'architettura del sistema, con un sottoinsieme dello spazio d'indirizzi della CPU, che vi accede come se si trattasse d'indirizzi dell'ordinaria memoria centrale. Quindi per mostrare un testo su uno schermo si deve semplicemente scrivere il testo agli appropriati indirizzi di memoria che corrispondono alla memoria video.

L'uso di questa tecnica è conveniente anche per altri dispositivi come le porte seriali e parallele, che si usano ad esempio per collegare ai calcolatori modem e stampanti. La CPU trasferisce i dati attraverso questi tipi di dispositivi, detti porte di I/O, leggendo e scrivendo in alcuni registri in essi contenuti. Per inviare una sequenza di byte tramite una porta seriale i cui registri si fanno corrispondere a una serie d'indirizzi di memoria, la CPU scrive un byte nel registro dei dati, quindi imposta un bit nel registro di controllo per indicare che il byte è disponibile. Il dispositivo riceve tale byte e cancella il bit del registro di controllo per segnalare che è pronto a ricevere un nuovo byte; la CPU può così trasferire il byte successivo. Il metodo in cui la CPU impiega l'interrogazione ciclica (*polling*) del bit di controllo, per verificare se il dispositivo è pronto, si chiama **I/O programmato** (PIO). Il metodo che non prevede l'interrogazione ciclica ma l'attesa di un'interruzione che segnala che il dispositivo è pronto per il byte successivo si chiama **I/O guidato dalle interruzioni**.

La maggior parte delle CPU può decodificare una o più istruzioni ed eseguire una o più semplici operazioni sul contenuto dei registri in un solo ciclo di clock. Per quel che riguarda la memoria, poiché vi si accede tramite una transazione nel bus di memoria, per completare un accesso, potrebbero invece essere richiesti molti cicli del clock della CPU; in tal caso la CPU deve **congelare** la propria esecuzione poiché non dispone dei dati richiesti per portare a termine l'istruzione in corso d'esecuzione. Data la frequenza degli accessi alla memoria, questa situazione è chiaramente intollerabile; un rimedio consiste nell'interporre una memoria ad alta velocità d'accesso tra la CPU e la memoria centrale. Tali dispositivi di memoria, usati per conciliare la differenza di velocità tra la CPU e i dispositivi esterni a essa, si chiamano **cache** e sono descritti nel Paragrafo 2.4.1.

2.3.2 Dischi magnetici

I dischi magnetici sono il mezzo fondamentale di memoria secondaria dei moderni sistemi di calcolo. Concettualmente, i dischi (Figura 2.5) sono relativamente semplici: i piatti dei dischi hanno una forma piana e rotonda come quella dei CD, con un diametro che comunemente varia tra 1,8 e 5,25 pollici, e le due superfici ricoperte di materiale magnetico simile a quello dei nastri magnetici; le informazioni si memorizzano registrandole magneticamente sui piatti.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

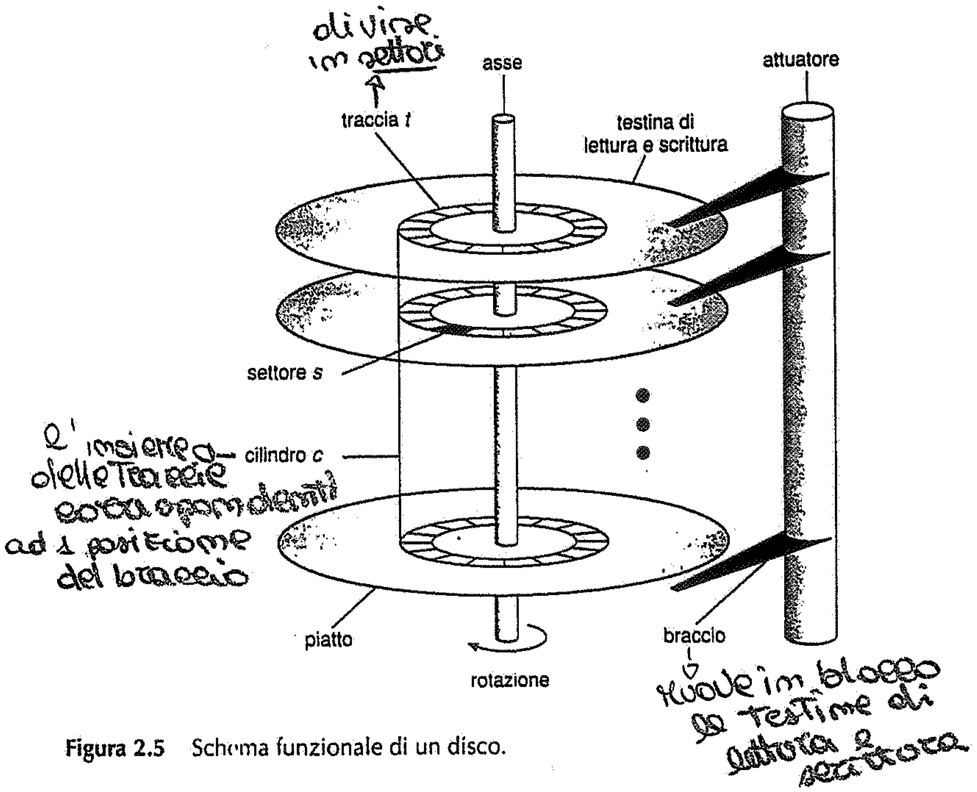


Figura 2.5 Schema funzionale di un disco.

Le testine di lettura e scrittura sono sospese su ciascuna superficie d'ogni piatto e sono attaccate al **braccio del disco** che le muove in blocco. La superficie di un piatto è divisa logicamente in **tracce circolari** a loro volta suddivise in **settori**; l'insieme delle tracce corrispondenti a una posizione del braccio (equidistanti dal centro dei piatti) costituisce un **cilindro**. In un'unità a disco possono esservi migliaia di cilindri concentrici e ogni traccia può contenere centinaia di settori. La capacità di memorizzazione di una comune unità a disco è dell'ordine delle decine di gigabyte.

Quando un disco è in funzione, un motore lo fa ruotare ad alta velocità; la maggior parte dei dischi ruota a velocità comprese fra 3600 e 12.000 giri al minuto. L'efficienza di un disco è caratterizzata da due valori: la **velocità di trasferimento**, cioè la velocità con cui i dati fluiscono dall'unità a disco al calcolatore, e il **tempo di posizionamento**, talvolta detto **tempo d'accesso diretto**, che consiste nel tempo necessario a spostare il braccio del disco in corrispondenza del cilindro desiderato, detto **tempo di ricerca**, e nel tempo necessario affinché il settore desiderato si porti, tramite la rotazione del disco, sotto la testina, detto **latenza di rotazione**. Tipicamente, i dischi possono trasferire parecchi megabyte di dati al secondo e hanno un tempo di ricerca e una latenza di rotazione di diversi millisecondi.

Poiché le testine di un disco sono sospese su un cuscino d'aria sottilissimo (dell'ordine dei micron), esiste il pericolo che la testina urti la superficie del disco, in tal caso, nonostante i piatti del disco siano ricoperti da un sottile strato protettivo, la testina può danneggiare la superficie magnetica. Tale incidente, detto **crollo della testina**, di solito non può essere riparato e comporta la sostituzione dell'intera unità a disco.

Un disco può essere **rimovibile**, ciò permette che diversi dischi siano montati secondo le necessità. I dischi magnetici rimovibili consistono generalmente in un piatto contenuto in un involucro di materiale plastico, che serve a evitare danni che possono verificarsi quando il disco non è inserito nella propria unità. I dischetti (*floppy disk*) sono dischi magnetici economici e rimovibili costituiti da un involucro di plastica che contiene un piatto flessibile. Poiché generalmente la testina delle unità a dischetti poggia direttamente sulla superficie del disco, per ridurne l'usura, si fa ruotare il dischetto molto più lentamente dei piatti di un'unità a disco (*hard disk*). La capacità di memorizzazione di un dischetto è dell'ordine del megabyte. Sono disponibili dischi rimovibili che funzionano in modo assai simile alle normali unità a disco e che hanno capacità dell'ordine dei gigabyte.

Un'unità a disco è connessa a un calcolatore attraverso un insieme di fili detto **bus di I/O**; esistono diversi tipi di tale bus, tra i quali i bus EIDE (*enhanced integrated drive electronics*), ATA (*advanced technology attachment*) e SCSI. Il trasferimento dei dati in un bus è eseguito da speciali unità di elaborazione dette **controllori**: gli **adattatori** (*adapter*) o **controllori di macchina** (*host controller*) sono i controllori posti all'estremità relativa al calcolatore del bus; i **controllori dei dischi** (*disk controller*) sono incorporati in ciascuna unità a disco. Per eseguire un'operazione di I/O il calcolatore inserisce un comando nell'adattatore — tipicamente mediante porte di I/O associate allo spazio degli indirizzi di memoria, com'è descritto nel Paragrafo 2.3.1 —, l'adattatore invia il comando al controllore del disco, che agisce sugli elementi elettromeccanici dell'unità a disco per portare a termine il comando. I controllori dei dischi di solito hanno una cache incorporata: il trasferimento dei dati nell'unità a disco avviene tra la cache e la superficie del disco; il trasferimento dei dati tra la cache e l'adattatore avviene alla velocità propria dei dispositivi elettronici.

2.3.3 Nastri magnetici

I **nastri magnetici** sono stati i primi mezzi di memorizzazione secondaria. Pur avendo la capacità di memorizzare in modo permanente un'enorme quantità di dati, queste unità sono caratterizzate da un tempo d'accesso molto elevato rispetto a quello della memoria centrale. Inoltre il tempo d'accesso diretto dei nastri magnetici (essendo fisicamente ad accesso sequenziale) è migliaia di volte maggiore di quello dei dischi magnetici, ciò li rende inadatti come mezzo di memoria secondaria. Gli usi principali dei nastri sono la creazione di copie di riserva dei dati (*backup*), la registrazione di dati poco usati e il trasferimento di informazioni tra diversi sistemi di calcolo.

Il nastro è avvolto in bobine e scorre su una testina di lettura e scrittura. Il posizionamento sul settore richiesto può richiedere alcuni minuti, anche se, una volta raggiunta la posizione desiderata, l'unità a nastro può leggere o scrivere informazioni a una velocità paragonabile a quella di un'unità a disco. La capacità varia secondo il particolare tipo di unità a nastro. Alcuni tipi di nastro contengono 2 o 3 volte più dati di quanto può contenere un disco molto capace. I nastri e le relative unità sono classificati secondo la larghezza; esistono nastri di 4, 8, 19 millimetri e di 1/4 e 1/2 pollice.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

2.4 Gerarchia delle memorie

I numerosi componenti di memoria di un sistema di calcolo si possono organizzare, secondo la velocità e il costo, in una struttura gerarchica (Figura 2.6). I dispositivi che occupano le posizioni più alte sono caratterizzati da un costo elevato e da un elevata velocità di servizio. Scendendo nella gerarchia il costo per bit generalmente diminuisce, mentre di solito aumenta il tempo d'accesso. Questo è un compromesso ragionevole, poiché se, mantenendo invariate le altre proprietà, esistesse un dispositivo di memoria rapido e al tempo stesso economico non ci sarebbe la necessità di impiegare unità di memoria più lente o più costose. Ora che i nastri magnetici e le **memorie a semiconduttori** hanno raggiunto elevate prestazioni e prezzi contenuti, i vecchi sistemi a nastro perforato o a memorie a nuclei sono relegati nei musei.

I tre livelli di memoria più alti della Figura 2.6 si possono costruire impiegando memorie a semiconduttori. Oltre che differire per il costo e la velocità, i diversi tipi di memoria possono essere volatili o non volatili. Le **memorie volatili** perdono il proprio contenuto quando manca l'alimentazione elettrica del dispositivo. In assenza di costosi sistemi dotati di batterie o generatori ausiliari, si deve salvaguardare l'integrità delle informazioni tramite la registrazione in **memorie non volatili**. Nella struttura gerarchica proposta nella Figura 2.6 gli elementi sopra i dischi RAM sono memorie volatili, mentre quelli sotto sono memorie non volatili. Un disco RAM si può progettare in modo da

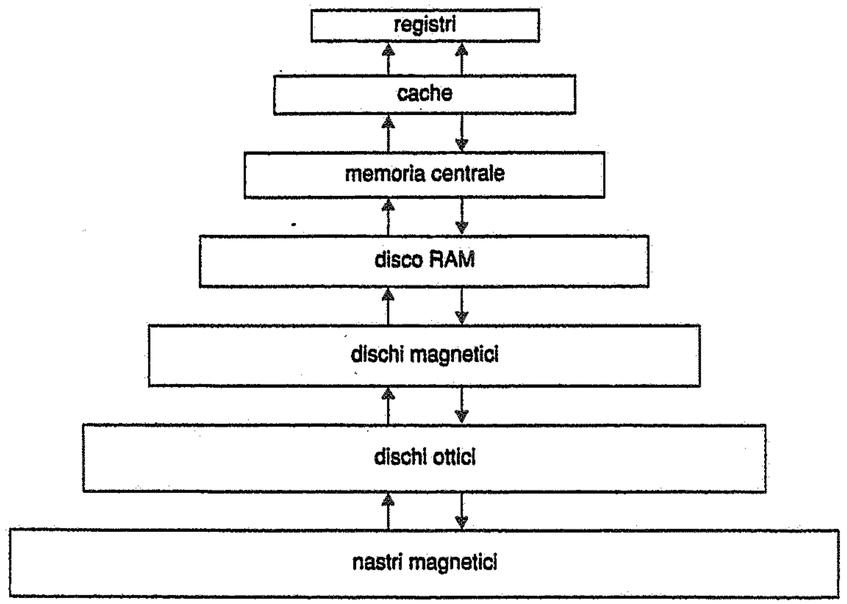


Figura 2.6 Gerarchia dei dispositivi di memoria.

essere volatile o non volatile. La progettazione di un completo sistema di memorizzazione deve bilanciare il rapporto fra tutti questi fattori, cercando di usare le memorie costose in misura strettamente necessaria, e di fornire al tempo stesso la maggiore quantità possibile di memoria non volatile e a basso costo. Per migliorare le prestazioni nei casi in cui c'è una grande differenza tra i tempi d'accesso o di trasferimento di due componenti si può ricorrere alle memorie cache.

2.4.1 Cache

Il concetto di **cache** è un principio importante in un sistema di calcolo. Di norma le informazioni sono mantenute in unità di memoria come la memoria centrale; al momento del loro uso si copiano temporaneamente in un'unità più veloce: la cache. Quando si deve accedere a una particolare informazione, innanzi tutto si controlla se è già presente all'interno della cache; in tal caso si adopera direttamente la copia contenuta nella cache, altrimenti la si preleva dalla memoria centrale e la si copia nella cache poiché si suppone che questa informazione presto servirà ancora.

Inoltre, i registri programmabili presenti all'interno della CPU, come i registri d'indice, rappresentano per la memoria centrale una cache ad alta velocità. Il programmatore (o il compilatore) codifica gli algoritmi di assegnazione e aggiornamento dei registri in modo da stabilire quali informazioni mantenere nei registri e quali nella memoria centrale. Esistono anche cache che sono interamente gestite dall'architettura del sistema, ad esempio la maggior parte dei sistemi è dotata di una cache per la memorizzazione delle istruzioni che presumibilmente saranno eseguite dopo l'istruzione corrente. Senza di essa, la CPU dovrebbe attendere parecchi cicli prima che un'istruzione sia prelevata dalla memoria. Per simili motivi, la maggior parte dei sistemi è dotata di una o più cache di dati nella gerarchia delle memorie. In questo testo non ci si occupa di questi dispositivi, poiché sono componenti non controllabili dal sistema operativo.

La gestione della cache, data la capacità limitata di questi dispositivi, è un importante problema di progettazione. Un'attenta selezione delle dimensioni e dei criteri di aggiornamento della cache può comportare che una percentuale dall'80 al 99 per cento di tutti gli accessi si limiti alla cache, con un notevole incremento delle prestazioni del sistema. Nel Capitolo 10 si discutono alcuni algoritmi per la sostituzione degli elementi contenuti nelle cache controllabili dal sistema operativo.

La memoria centrale si può considerare una cache per la memoria secondaria, giacché i dati in essa contenuti devono essere riportati nella memoria centrale per poter essere usati e qui devono risiedere prima di essere trasferiti nella memoria secondaria. I dati del file system, che risiedono in modo permanente nella memoria secondaria, possono apparire a diversi livelli della gerarchia di memorie. Al livello più alto, il sistema operativo può mantenere una cache dei dati del file system nella memoria centrale. Anche i dischi RAM si possono impiegare come veloci sistemi di memoria volatile cui si accede tramite l'interfaccia del file system. La memorizzazione secondaria si effettua generalmente in dischi magnetici; copie di riserva del loro contenuto spesso si registrano in nastri

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

magnetici o dischi rimovibili. Alcuni sistemi, per ridurre i costi di memorizzazione, archiviano automaticamente i vecchi file di dati trasferendoli dalla memoria secondaria alla memoria terziaria, costituita ad esempio da *juke-box* di nastri (Capitolo 14). Il movimento delle informazioni tra i vari livelli della gerarchia può essere quindi sia implicito sia esplicito, secondo l'architettura e il sistema operativo che la controlla. Ad esempio, il trasferimento dei dati tra la cache e i registri della CPU è di solito svolto dall'architettura del sistema senza alcun intervento del sistema operativo. Diversamente, il trasferimento dei dati dai dischi alla memoria è di solito gestito dal sistema operativo.

2.4.2 Coerenza

In una struttura gerarchica come quella appena introdotta, può accadere che gli stessi dati siano mantenuti contemporaneamente in diversi livelli del sistema di memorizzazione. Ad esempio, si supponga di incrementare di 1 il valore di un numero intero *n* contenuto in un file *f*, registrato in un disco magnetico. L'operazione d'incremento prevede innanzi tutto l'esecuzione di un'operazione di I/O per copiare nella memoria centrale il blocco di disco contenente il valore di *n*. Questa operazione è di solito seguita dalla copiatura di *n* all'interno della cache, e di qui in uno dei registri interni della CPU. Quindi, esistono diverse copie di *n* memorizzate nei vari dispositivi coinvolti nel trasferimento: nel disco magnetico, nella memoria centrale, nella cache e in un registro interno della CPU (Figura 2.7). Dopo l'incremento del valore della copia contenuta nel registro interno, il valore di *n* sarà diverso da quello assunto dalle altre copie della stessa variabile. Le diverse copie di *n* avranno lo stesso valore solamente dopo che il nuovo valore sarà stato riportato dal registro interno della CPU alla copia di *n* residente nel disco.

In un ambiente di calcolo che ammette l'esecuzione di un solo processo alla volta, questo modo di operare non pone particolari difficoltà, poiché ogni accesso a *n* coinvolge la copia posta al livello più alto della gerarchia. Viceversa, nei sistemi a partizione del tempo d'elaborazione, nei quali il controllo della CPU passa da un processo all'altro, è necessario prestare una particolare attenzione al fine di assicurare che qualsiasi processo che desideri accedere a *n* ottenga dal sistema il valore della variabile aggiornato più di recente.

La situazione si complica ulteriormente negli ambienti con più unità d'elaborazione, dove ciascuna di esse, oltre i registri interni, contiene anche una cache locale. Nei sistemi del genere possono esistere più copie simultanee di *n* mantenute in cache differenti. Poiché le diverse unità d'elaborazione possono operare in modo concorrente, è necessario che l'aggiornamento del valore di una qualsiasi delle copie di *n* contenute nelle cache locali si rifletta immediatamente in tutte le cache in cui *n* risiede. Questa situazione, nota come **coerenza della cache**, di solito si risolve al livello dell'architettura del sistema, quindi a un livello più basso di quello del sistema operativo.

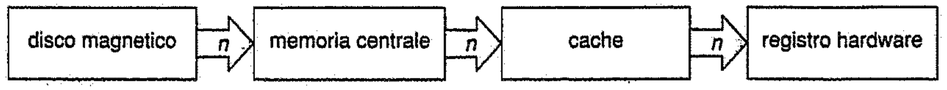


Figura 2.7 Migrazione di un intero *n* da un disco a un registro.

In un sistema distribuito la situazione diventa ancora più complessa, poiché può accadere che più copie (chiamate in questo caso repliche) dello stesso file siano mantenute in differenti calcolatori, dislocati in luoghi fisici diversi. Essendoci la possibilità di accedere e modificare in modo concorrente queste repliche, è necessario fare in modo che ogni modifica a una qualunque replica si rifletta quanto prima sulle altre. Nel Capitolo 16 si discutono diversi metodi che consentono di soddisfare questo requisito.

2.5 Architetture di protezione

I primi sistemi di calcolo ammettevano la presenza di un solo utente alla volta ed erano gestiti direttamente dai programmatori che, operando dalla console del calcolatore, avevano il completo controllo dell'intero sistema. Con l'evoluzione dei programmi di sistema, questo controllo passò gradualmente al sistema operativo. I primi sistemi operativi si chiamavano *monitor residenti*, e con essi il sistema operativo cominciò a eseguire molte funzioni, in particolare le operazioni di I/O, che precedentemente erano di competenza diretta del programmatore.

Inoltre, al fine di migliorare l'utilizzo del sistema, i sistemi operativi resero possibile la *condivisione* simultanea delle risorse del sistema tra più programmi. La gestione asincrona delle operazioni di I/O (*spooling*) consentiva l'esecuzione di un programma, mentre altri processi erano impegnati in operazioni di I/O; i dischi consentivano di contenere contemporaneamente i dati di più processi; con la multiprogrammazione si potevano far risiedere contemporaneamente più programmi nella memoria.

Se da una parte queste condivisioni incrementarono l'utilizzo del sistema, dall'altra aumentarono i problemi da risolvere. Nei sistemi che non ammettevano forme di condivisione delle risorse, il verificarsi di un errore all'interno di un programma causava problemi solamente al programma in esecuzione. Con l'avvento della condivisione delle risorse ci si trovò nella situazione in cui un errore in un programma poteva alterare il comportamento di molti processi.

A titolo d'esempio, si consideri un sistema operativo a lotti (Paragrafo 1.2.1), la cui unica funzione era la sequenzializzazione automatica dei lavori d'elaborazione. Se un programma s'inceppasse in un ciclo infinito durante la lettura delle proprie schede, esaurirebbe la lettura dei propri dati e, se niente lo fermasse, proseguirebbe con la lettura delle schede riguardanti i lavori d'elaborazione successivi, compromettendone il corretto funzionamento.

Errori ancor più subdoli si potevano verificare nei sistemi multiprogrammati, dove il verificarsi di un errore in un programma poteva alterare il codice o i dati di un altro programma, o addirittura dello stesso monitor residente. L'MS-DOS e le vecchie versioni del Macintosh OS sono due esempi di sistemi operativi che permettono il verificarsi di questo genere d'errori.

Senza un adeguata protezione ci si trova nella situazione di dover scegliere se consentire l'esecuzione di un solo processo alla volta, oppure considerare sospetto qualsiasi risultato ottenuto. Un sistema operativo ben progettato deve garantire che un programma non possa inavvertitamente (o volontariamente) compromettere il funzionamento degli altri programmi presenti nel sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Molti errori di programmazione sono riconosciuti direttamente dall'architettura del sistema e sono di solito gestiti dal sistema operativo. Se un programma utente compie un'operazione illecita, ad esempio tentando di eseguire un'istruzione non consentita o di accedere a una locazione di memoria fuori del proprio spazio d'indirizzi, l'architettura di protezione invia un segnale di eccezione al sistema operativo: il segnale di eccezione determina il trasferimento del controllo, come accade con i segnali d'interruzione, al sistema operativo attraverso il vettore delle interruzioni. Ogniqualvolta si verifica un errore in un programma, il sistema operativo deve porre fine alla sua esecuzione. Questa situazione è gestita dallo stesso codice che il sistema operativo impiega quando un utente richiede la terminazione anormale del proprio programma. In questi casi di solito si scrive in un file l'immagine della memoria (*dump*), ciò permette all'utente o al programmatore di esaminarla allo scopo di correggere e riavviare il programma.

2.5.1 Duplice modo di funzionamento

Per garantire il corretto funzionamento del sistema è necessario proteggere sia il sistema operativo sia gli altri programmi, e i relativi dati, da qualsiasi programma malfunzionante. In realtà la protezione deve essere garantita per qualsiasi risorsa condivisibile del sistema. Il metodo seguito da molti sistemi operativi consiste nel disporre di specifiche caratteristiche dell'architettura del sistema che consentono di gestire differenti modi di funzionamento. Sono necessari almeno due diversi modi di funzionamento: modo d'utente e modo di sistema (detto anche modo del supervisore, modo monitor, o modo privilegiato). Per indicare quale modo è attivo, l'architettura della CPU deve essere dotata di un bit, chiamato appunto bit di modo: di sistema (0) o d'utente (1). Questo bit consente di stabilire se l'istruzione corrente si esegue per conto del sistema operativo o per conto di un utente. Si potrà constatare come questa evoluzione dell'architettura sia utile per molti altri aspetti del funzionamento del sistema.

All'avviamento del sistema, il bit è posto nel modo di sistema. Si carica il sistema operativo che provvede all'esecuzione dei processi utenti nel modo d'utente. Ogni volta che si verifica un'interruzione o un'eccezione si passa dal modo d'utente al modo di sistema, cioè si pone a 0 il bit di modo. Perciò quando il sistema operativo riprende il controllo del calcolatore si trova nel modo di sistema. Prima di passare il controllo al programma utente, il sistema ripristina il modo d'utente riportando a 1 il valore del bit.

Il duplice modo di funzionamento (*dual mode*) consente sia di proteggere il sistema operativo dal comportamento degli utenti sia di proteggere gli utenti dagli altri utenti. Questo livello di protezione si ottiene definendo le istruzioni di macchina che possono causare danni allo stato del sistema come istruzioni privilegiate. Poiché la CPU consente l'esecuzione di queste istruzioni soltanto nel modo di sistema, se si tenta di far eseguire nel modo d'utente un'istruzione privilegiata, la CPU non la esegue ma la tratta come un'istruzione illegale inviando un segnale di eccezione al sistema operativo.

Il concetto di istruzione privilegiata fornisce i mezzi che consentono all'utente di interagire col sistema operativo richiedendo che il sistema esegua dei compiti che solo il sistema operativo può svolgere. Ciascuna di queste richieste si effettua tramite l'esecuzione di un'istruzione privilegiata. Questa richiesta è nota come chiamata di funzione del

sistema operativo (*operating system function call*) o, più brevemente, chiamata del sistema (*system call*), com'è descritto nel Paragrafo 2.1.

Quando un programma utente esegue una chiamata del sistema, questa è gestita dalla CPU come un'interruzione, anche se non si tratta di un segnale d'interruzione proveniente da un dispositivo fisico. Il controllo passa, tramite il vettore delle interruzioni, all'apposita procedura di servizio presente all'interno del sistema operativo e si pone il bit di modo nel modo di sistema. La procedura di servizio della chiamata del sistema è parte integrante del sistema operativo. Il sistema esamina l'istruzione che ha causato l'eccezione al fine di stabilire la natura della chiamata del sistema, mentre un parametro di tale istruzione definisce il tipo di servizio richiesto dal programma utente. Ulteriori informazioni indispensabili per il completamento della richiesta si possono copiare nei registri, sulla pila o direttamente nella memoria centrale (in locazioni il cui indirizzo è memorizzato nei registri). Si verifica la correttezza e la legalità dei parametri, si soddisfa la richiesta e si restituisce il controllo dell'esecuzione all'istruzione immediatamente seguente la chiamata del sistema.

Se la CPU non dispone di questi due modi di funzionamento il sistema operativo può andare incontro a serie limitazioni. Il sistema MS-DOS ad esempio è stato sviluppato per l'architettura 8088 Intel priva del bit di modo e, quindi, dei due modi di funzionamento. Un programma utente ha la possibilità di cancellare il sistema operativo scrivendo nuove informazioni nelle locazioni in cui questo è memorizzato, e più programmi possono scrivere contemporaneamente nello stesso dispositivo, con la possibilità di ottenere risultati disastrosi. Versioni più recenti e progredite delle CPU Intel, come il Pentium, sono dotate del duplice modo di funzionamento. I più recenti sistemi operativi sviluppati per tali architetture, come il Microsoft Windows 2000 e l'IBM OS/2, traggono vantaggio da questa caratteristica e garantiscono una maggiore protezione del sistema operativo.

2.5.2 Protezione dell'I/O

Un programma utente può alterare il normale funzionamento del sistema operativo effettuando operazioni illegali di I/O, accedendo a locazioni di memoria all'interno dello stesso sistema operativo o rifiutando di rilasciare il controllo della CPU. Esistono diversi meccanismi per garantire che ciò non possa avvenire.

Allo scopo di impedire l'esecuzione di operazioni illegali di I/O da parte dell'utente, si definiscono privilegiate tutte le istruzioni di I/O, in modo che l'utente non possa usarle direttamente, ma attraverso il sistema operativo. Affinché la protezione dell'I/O sia totale, è necessario evitare che l'utente possa in qualsiasi modo ottenere il controllo del calcolatore quando questo si trova nel modo di sistema; se così non fosse, si potrebbe compromettere l'integrità delle operazioni di I/O.

Si supponga che un calcolatore si trovi momentaneamente nel modo d'utente. Il verificarsi di un'interruzione o di un'eccezione riporterebbe immediatamente il calcolatore nel modo di sistema, e trasferirebbe il controllo dell'esecuzione all'indirizzo contenuto nel vettore delle interruzioni. Se un programma utente riuscisse a modificare il contenuto di questo vettore, potrebbe scrivervi un nuovo indirizzo con un riferimento all'interno del proprio codice. Quindi, al verificarsi dell'interruzione o dell'eccezione

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

corrispondente, la CPU passerebbe al modo di sistema e si trasferirebbe il controllo, attraverso il vettore delle interruzioni alterato, al programma utente. Il processo utente riuscirebbe così a ottenere il controllo del calcolatore nel modo di sistema. In effetti, i programmi utenti possono ottenere il controllo di un calcolatore nel modo di sistema in molti altri modi. Inoltre, si scoprono quotidianamente nuovi bachi che si possono sfruttare per aggirare le protezioni dei sistemi; questi argomenti sono discussi nei Capitoli 18 e 19. Così per compiere un'operazione di I/O, un programma utente usa una chiamata del sistema per richiedere che il sistema operativo esegua l'I/O per suo conto (Figura 2.8). Il sistema operativo, in esecuzione nel modo di sistema, controlla che la richiesta sia valida e, in questo caso, porta a termine l'operazione di I/O e restituisce il controllo al programma utente.

2.5.3 Protezione della memoria

Al fine di garantire il corretto funzionamento del sistema, è necessario proteggere il vettore delle interruzioni da ogni possibile alterazione da parte dei programmi utenti. Inoltre, è necessario proteggere dalle modifiche anche le procedure di servizio dei segnali d'interruzione contenute nel codice del sistema operativo. Anche se un tentativo non autorizzato di assunzione del controllo del sistema dovesse fallire, l'alterazione di tali procedure di sistema finirebbe per compromettere il corretto funzionamento del calcolatore, in particolare dei sistemi di gestione asincrona delle operazioni di I/O (*spooling*) e di memorizzazione transitoria dei dati di I/O (*buffering*).

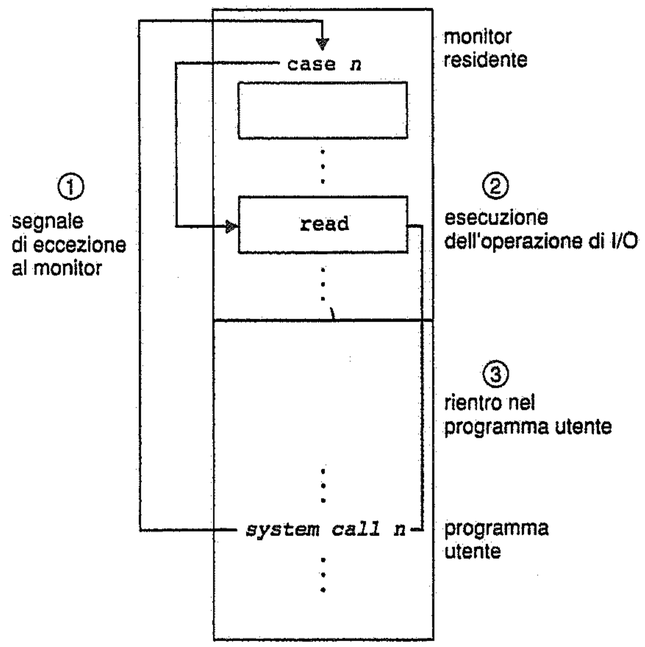


Figura 2.8 Uso di una chiamata del sistema per l'esecuzione di un'operazione di I/O.

Quel che s'è detto dimostra la necessità di proteggere dagli accessi dei programmi utenti la memoria riservata al vettore delle interruzioni, e alle relative procedure di servizio. In generale, si deve proteggere il sistema operativo dagli accessi dei programmi utenti, e i programmi utenti dagli accessi di altri programmi utenti. Questa protezione deve essere garantita dall'architettura del sistema e si può realizzare in diversi modi, come si descrive con maggiori dettagli nel Capitolo 9. In questa sede ci si limita ad accennare a una delle possibili soluzioni.

Per separare lo spazio di memoria di ogni singolo programma serve la capacità di determinare l'intervallo d'indirizzi cui il programma può accedere, e proteggere la memoria fuori di esso. Questo tipo di protezione si realizza impiegando due registri, di solito chiamati di base e di limite (Figura 2.9). Il **registro di base** contiene il più basso indirizzo della memoria fisica al quale il programma dovrebbe accedere, mentre il **registro di limite** contiene la dimensione dell'intervallo. Ad esempio, se i registri di base e di limite contengono rispettivamente i valori 300040 e 120900, al programma si consente l'accesso alle locazioni di memoria di indirizzi compresi tra 300040 e 420940, estremi inclusi.

Per ottenere questa protezione, la CPU confronta *ciascun* indirizzo generato nel modo d'utente con i valori contenuti nei due registri. Qualsiasi tentativo da parte di un programma eseguito nel modo d'utente di accedere alle aree di memoria riservate al sistema operativo o a una qualsiasi area di memoria riservata ad altri utenti comporta l'invio di un segnale di eccezione al sistema operativo, che gestisce l'evento come un errore fatale

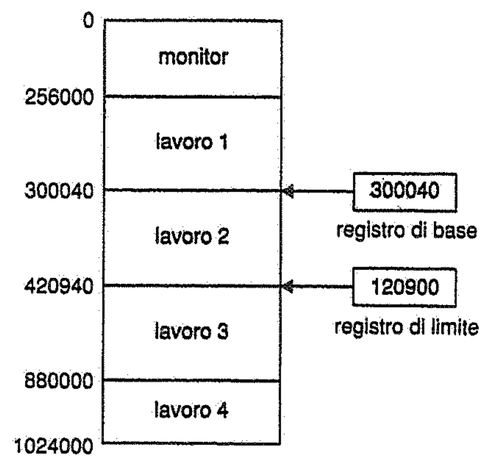


Figura 2.9 Un registro di base e un registro di limite definiscono uno spazio d'indirizzi logico.

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

Marina & Checco 18/04/03

(Figura 2.10). Questo schema impedisce a qualsiasi programma utente di alterare (accidentalmente o volontariamente) il codice o le strutture di dati, sia del sistema operativo sia degli altri utenti.

Poiché per modificare il contenuto dei registri di base e di limite si deve usare una speciale istruzione che si può eseguire solamente nel modo di sistema, e poiché solamente il sistema operativo può funzionare in questo modo, tale schema consente al sistema operativo di modificare il valore di questi registri ma impedisce la stessa operazione ai programmi utenti.

Funzionando nel modo di sistema, il sistema operativo ha la possibilità di accedere indiscriminatamente sia alla memoria a esso riservata sia a quella riservata agli utenti. Questo privilegio consente al sistema di caricare i programmi utenti nelle relative aree di memoria; se si verificano errori, di generare copie del contenuto di queste regioni di memoria (*dump*); di leggere e modificare i parametri delle chiamate del sistema; e così via.

2.5.4 Protezione della CPU

Oltre a proteggere le funzioni di I/O e la memoria, occorre assicurare che il sistema operativo mantenga il controllo dell'elaborazione; cioè impedire che un programma utente entri in un ciclo infinito o non richieda servizi del sistema senza più restituire il controllo al sistema operativo. A tale scopo si può usare un **temporizzatore** programmabile affinché invii un segnale d'interruzione alla CPU a intervalli di tempo specificati, che possono essere fissi (ad esempio, di 1/60 di secondo) o variabili (ad esempio, da un millisecondo a un secondo). Un **temporizzatore variabile** di solito si realizza mediante un generatore di impulsi a frequenza fissa e un contatore. Il sistema operativo assegna un valore al contatore, che si decrementa a ogni impulso e quando raggiunge il valore 0 si genera un segnale d'interruzione. Ad esempio, un contatore di 10 bit e un generatore di

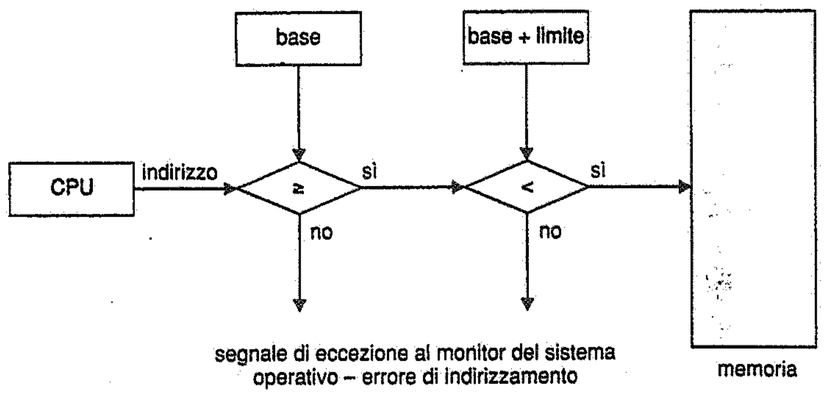


Figura 2.10 Architettura di protezione degli indirizzi con registri di base e di limite.

impulsi con periodo di 1 millisecondo consente la generazione di interruzioni a intervalli compresi tra 1 e 1024 millisecondi, con incrementi di 1 millisecondo.

Prima di restituire all'utente il controllo dell'esecuzione, il sistema operativo assegna un valore al temporizzatore. Se il temporizzatore esaurisce questo intervallo genera un'interruzione che causa il trasferimento del controllo al sistema operativo, che può decidere se gestire le interruzioni come un errore fatale o concedere altro tempo al programma. Ovviamente, anche le istruzioni usate dal sistema per modificare il funzionamento del temporizzatore si possono eseguire soltanto nel modo di sistema.

La presenza di un temporizzatore garantisce quindi che nessun programma utente possa essere eseguito troppo a lungo. Una tecnica semplice consiste nell'impostare un contatore con un valore pari al tempo concesso al programma per la propria esecuzione. Ad esempio, se il programma richiede 7 minuti si dovrebbe impostare il contatore al valore 420. Il temporizzatore genererà un'interruzione ogni secondo e il contatore sarà decrementato di 1; fintanto che il valore resta positivo, il controllo ritorna al programma utente; quando il contatore raggiunge un valore negativo, il sistema operativo termina l'esecuzione del programma per il superamento del tempo a esso assegnato.

Più comunemente i temporizzatori si usano per realizzare la partizione del tempo d'elaborazione. Nel caso più immediato, s'istruisce il temporizzatore affinché interrompa l'esecuzione ogni n millisecondi, dove n è il quanto **di tempo** concesso a ciascun utente prima che il controllo della CPU passi all'utente successivo. Allo scadere d'ogni quanto di tempo il controllo ritorna al sistema operativo, che esegue le necessarie operazioni d'amministrazione del sistema, ad esempio sommare n all'informazione che (per scopi di contabilizzazione delle risorse) specifica la quantità di tempo che il programma utente ha speso nell'esecuzione fino a quel momento. Il sistema inoltre salva i valori dei registri, delle proprie variabili interne e il contenuto delle proprie strutture di dati, inoltre aggiorna molti altri parametri allo scopo di preparare l'esecuzione del programma successivo. Questa procedura prende il nome di **cambio di contesto** (*context switch*) ed è descritta con maggiori dettagli nel Capitolo 4. Dopo ogni cambio di contesto il programma che riceve il controllo della CPU riprende l'esecuzione dal punto esatto in cui era stata interrotta allo scadere del suo precedente quanto di tempo.

Un altro impiego dei temporizzatori è il calcolo dell'ora corrente. Le interruzioni generate dal temporizzatore segnalano il trascorrere di una certa quantità di tempo, rispetto alla quale il sistema operativo può calcolare l'ora corrente con riferimento a una certa ora iniziale. Sapendo che il temporizzatore genera un'interruzione ogni secondo, e sapendo che all'istante corrente il sistema ha ricevuto 1427 interruzioni dall'istante in cui era stato informato che erano le 13.00.00, si può stabilire che attualmente sono le 13.23.47. Alcuni calcolatori usano questo metodo per determinare l'ora corrente, anche se la precisione dei calcoli deve essere molto accurata, poiché il tempo necessario al servizio delle interruzioni, e altri fattori, come la disabilitazione delle interruzioni, tendono a rallentare l'orologio. Per questa ragione la maggior parte dei calcolatori è dotata di un vero e proprio orologio indipendente dal sistema operativo.

http://www.adultpdf.com
Created by TIFF Trial version. To remove this mark, please register this software.

2.6 Struttura delle reti di calcolatori

Esistono due fondamentali tipi di rete: le **reti locali** (*local-area network* — LAN) e le **reti geografiche** (*wide-area network* — WAN). La principale differenza tra le due consiste nella loro distribuzione geografica. Le reti locali sono formate da calcolatori distribuiti su piccole aree geografiche, come un unico edificio o alcuni edifici adiacenti. Le reti geografiche, invece, sono formate da un certo numero di calcolatori autonomi distribuiti su una vasta area geografica, ad esempio l'Europa. Queste differenze implicano notevoli differenze relativamente alla velocità e all'affidabilità delle reti di comunicazione, e si riflettono nella progettazione dei sistemi operativi distribuiti.

2.6.1 Reti locali

Le reti locali sono nate nei primi anni Settanta con lo scopo di sostituire i sistemi di calcolo di tipo mainframe. Per molte aziende era più economico disporre di tanti piccoli calcolatori, ciascuno con le proprie applicazioni interne, anziché di un unico sistema di grandi dimensioni. Poiché è probabile che ogni piccolo calcolatore necessiti di una serie completa di dispositivi periferici, come dischi e stampanti, e poiché è probabile che in un'unica società esista qualche forma di condivisione di dati, è naturale collegare questi piccoli sistemi in una rete.

Generalmente le LAN sono progettate per coprire piccole aree geografiche, come un solo edificio o alcuni edifici adiacenti, e di solito si usano in ambienti di uffici. In questi sistemi tutti i siti sono vicini fra loro, i collegamenti tendono ad avere una maggiore velocità e una minor frequenza d'errori rispetto alle reti geografiche. Per ottenere tale velocità e affidabilità sono necessari (costosi) cavi di elevata qualità. Inoltre si possono impiegare i cavi esclusivamente per il traffico dei dati della rete; per lunghe distanze il costo dovuto all'impiego di cavi di elevata qualità è enorme, e l'uso esclusivo dei cavi tende a diventare proibitivo.

Nelle reti locali i collegamenti generalmente si realizzano con cavi a doppini intrecciati e a fibre ottiche. Le configurazioni più diffuse sono le reti con bus multiaccesso, ad anello e a stella. La velocità di comunicazione varia dall'ordine dei megabit al secondo per le reti come l'Appletalk, le comunicazioni a raggi infrarossi e le recenti reti locali a onde radio del tipo Bluetooth, a un gigabit al secondo per le reti gigabit Ethernet; la velocità più comune è di 10 megabit al secondo ed è la velocità dell'**Ethernet 10 BaseT**; l'**Ethernet 100 BaseT**, che sta diventando assai comune, richiede un cablaggio di alta qualità ma funziona alla velocità di 100 megabit al secondo. È in crescita anche l'uso delle interconnessioni FDDI basate su fibre ottiche; queste reti sono basate sul passaggio di contrassegno (*token*) e funzionano a 100 megabit al secondo.

Una tipica LAN è formata da diversi tipi di calcolatori (mainframe, portatili, PDA), vari dispositivi periferici condivisi (come stampanti laser o unità a nastri magnetici), e uno o più *gateway* (unità d'elaborazione specializzate) che dànno accesso ad altre reti (Figura 2.11). Per costruire le LAN comunemente si usa uno schema Ethernet che, non avendo un controllore centrale ed essendo una rete con bus multiaccesso, consente di aggiungere facilmente nuovi elementi.

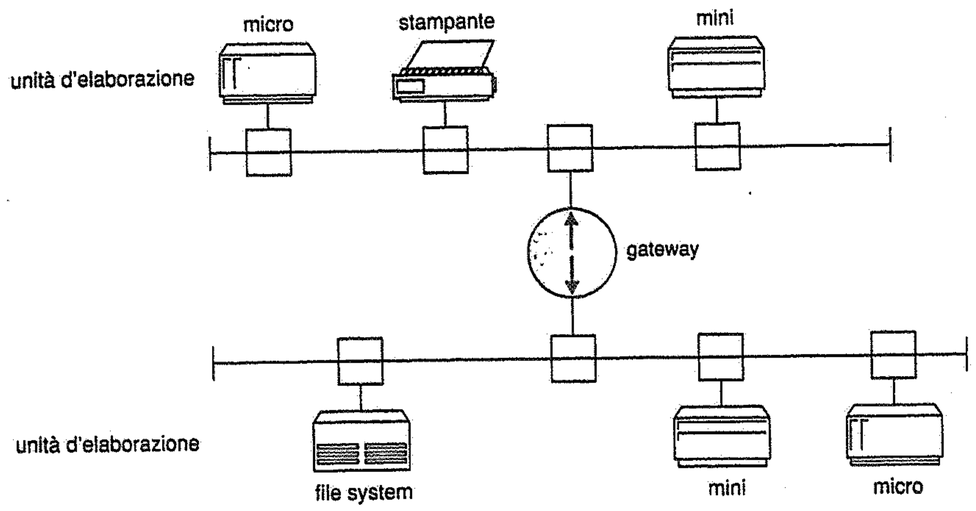


Figura 2.11 Rete locale.

2.6.2 Reti geografiche

Le reti geografiche sono nate alla fine degli anni Sessanta come progetto di ricerca accademica per fornire un sistema di comunicazione efficiente tra siti, e permettere a un'ampia comunità di utenti di condividere in modo conveniente ed economico le risorse di calcolo. La prima rete progettata e sviluppata è stata *Arpanet*. Nata nel 1968 come rete sperimentale, è cresciuta da quattro siti a una rete mondiale di reti, Internet, che comprende milioni di sistemi di calcolo.

Poiché i siti di una WAN sono fisicamente distribuiti su una vasta zona geografica, i collegamenti per le comunicazioni sono per loro natura relativamente lenti e inaffidabili. Tipicamente s'impiegano le linee telefoniche, i collegamenti a microonde e i canali via satellite. Questi collegamenti sono controllati da speciali **elaboratori di comunicazione** (Figura 2.12) responsabili della definizione dell'interfaccia attraverso la quale i siti comunicano nella rete, e del trasferimento delle informazioni tra i diversi siti.

Ad esempio, la WAN Internet offre ai calcolatori in siti geograficamente separati la possibilità di comunicare tra loro. Generalmente, i calcolatori differiscono per tipo, velocità, lunghezza delle parole di memoria, sistema operativo e così via. I calcolatori sono inseriti in LAN, che a loro volta sono collegate alla rete Internet per mezzo di reti regionali. Le reti regionali, come la NSFnet del Nord-Est degli Stati Uniti, sono interconnesse tramite instradatori (*router*) — descritti nel Paragrafo 15.4.2 — per formare la rete mondiale. I collegamenti tra le reti si servono spesso di un servizio telefonico, chiamato T1, che fornisce una velocità di trasferimento di 1,544 megabit al secondo su una linea a noleggio. Per siti che richiedono un accesso più veloce alla rete Internet, i T1 sono raccolti in gruppi che lavorano in parallelo per fornire una maggiore produttività. Ad esem-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

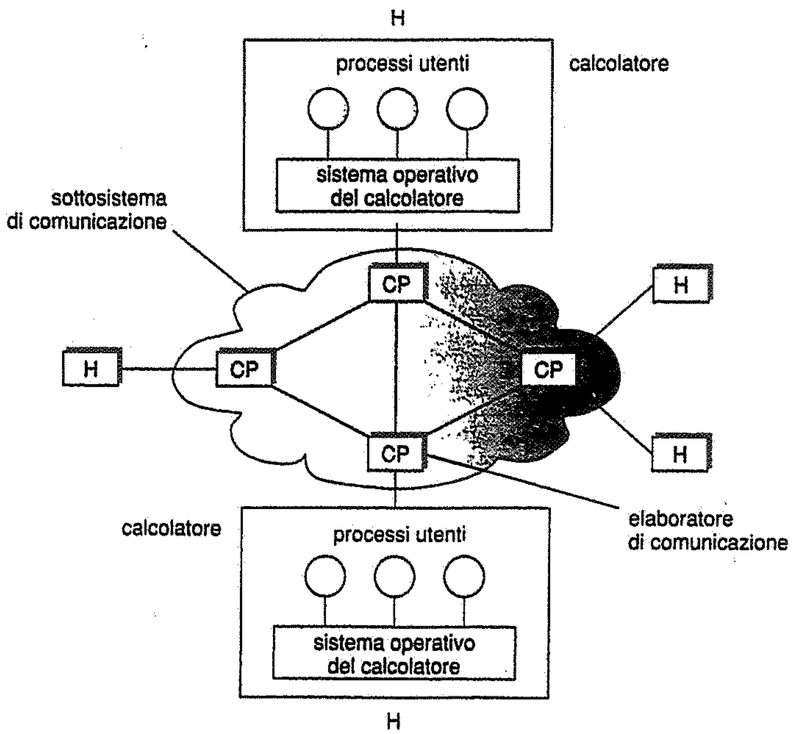


Figura 2.12 Elaboratori di comunicazione in una rete geografica.

pio, un T3 è composto da 28 connessioni T1 e ha una velocità di trasferimento di 45 megabit al secondo. Gli instradatori controllano i percorsi seguiti da ogni messaggio attraverso la rete; l'instradamento può essere dinamico, per incrementare l'efficienza delle comunicazioni, oppure statico, per ridurre i rischi per la sicurezza o per permettere di calcolare le spese di comunicazione secondo la durata delle connessioni.

Altre WAN impiegano ordinarie linee telefoniche come mezzo di comunicazione principale. I modem sono dispositivi che ricevono dati digitali da un calcolatore e li convertono nei segnali analogici del sistema telefonico; un modem nel sito di destinazione riconverte il segnale analogico in forma digitale e il destinatario riceve i dati. La rete di scambio di messaggi di posta elettronica dell'ambiente UNIX, UUCP, permette ai sistemi di comunicare tra loro via modem a ore prefissate per scambiarsi messaggi. I messaggi s'instradano ad altri sistemi vicini e in questo modo si propagano a tutti i calcolatori della rete (messaggi pubblici), oppure si trasferiscono alla loro destinazione (messaggi privati). Le WAN sono generalmente più lente delle LAN; le loro velocità di trasmissione variano da 1200 bit al secondo a oltre 1 megabit al secondo.

L'UUCP è stato soppiantato dal protocollo PPP (*point-to-point protocol*); si tratta di un protocollo che funziona con connessioni realizzate tramite modem e consente il collegamento alla rete Internet dei calcolatori domestici.

2.7 Sommario

I sistemi multiprogrammati e a partizione del tempo migliorano le prestazioni sovrapponendo le operazioni della CPU e di I/O di un singolo calcolatore. Questa sovrapposizione richiede che il trasferimento dei dati tra la CPU e i dispositivi di I/O sia gestito con l'accesso alle porte di I/O per mezzo dell'interrogazione ciclica o tramite le interruzioni, oppure col trasferimento dei dati con DMA.

Per essere eseguiti, i programmi devono risiedere nella memoria centrale del calcolatore. Questa è infatti la sola area di memoria di grandi dimensioni direttamente accessibile dalla CPU. Essa è un vettore, di parole o byte, di dimensioni variabili tra le centinaia di migliaia e i miliardi di byte. Ciascuna parola possiede il proprio indirizzo. La memoria centrale è un dispositivo volatile poiché perde il proprio contenuto quando manca l'alimentazione elettrica. La maggior parte dei sistemi di calcolo dispone di una memoria secondaria come estensione della memoria centrale. Il requisito fondamentale della memoria secondaria è la capacità di memorizzare in modo permanente grandi quantità di dati. Il più comune dispositivo di memoria secondaria è il disco magnetico; si tratta di un dispositivo di memoria non volatile che può memorizzare sia dati sia programmi e che consente l'accesso diretto ai suoi elementi. I nastri magnetici si usano prevalentemente per la registrazione di copie di riserva di dati, per la registrazione d'informazioni poco usate e come mezzo per il trasferimento di informazioni tra sistemi diversi.

I sistemi di memorizzazione di un calcolatore si possono organizzare in modo gerarchico secondo la velocità e il costo. I livelli più alti rappresentano i dispositivi più rapidi, ma più costosi. Scendendo nella gerarchia, il costo per bit generalmente decresce, mentre di solito aumentano i tempi d'accesso.

Il sistema operativo deve assicurare il corretto funzionamento del calcolatore. Per evitare che i programmi utenti interferiscano tra di essi e col sistema operativo, la CPU ha due modi di funzionamento: il modo d'utente e il modo di sistema. Diverse istruzioni (come quelle di I/O e di arresto del calcolatore) sono privilegiate e si possono eseguire solamente nel modo di sistema. Anche l'area della memoria nella quale risiede il sistema operativo deve essere protetta dai tentativi di modifiche da parte degli utenti. La presenza di un temporizzatore evita il verificarsi di cicli infiniti. Queste funzioni (duplice modo di funzionamento, istruzioni privilegiate, protezione della memoria, interruzioni del temporizzatore) costituiscono gli elementi fondamentali impiegati dal sistema operativo per ottenere un corretto funzionamento del sistema. Nel Capitolo 3 la discussione prosegue con i dettagli relativi alle funzioni offerte dai sistemi operativi.

Le LAN e le WAN sono i due principali tipi di reti. Le LAN sono di solito connesse tramite costosi cavi a doppini intrecciati o a fibre ottiche, e permettono la comunicazione tra calcolatori distribuiti in una piccola area geografica. Connesse tramite linee telefoniche, linee a noleggio, collegamenti a microonde, o canali via satellite, le WAN consentono la comunicazione tra calcolatori distribuiti in ampie aree geografiche. Le LAN trasmettono tipicamente anche più di 100 megabit al secondo, mentre le più lente WAN trasmettono dai 1200 bit al secondo a più di un megabit al secondo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

2.8 Esercizi

2.1 Il prelievo anticipato è un metodo che consente la sovrapposizione dell'I/O di un processo con le sue proprie elaborazioni. L'idea è semplice. Una volta che un'operazione di lettura è terminata e che il processo s'appresta a iniziare l'elaborazione dei dati, s'istruisce il controllore del dispositivo affinché cominci immediatamente la successiva operazione di lettura. In questo modo, sia la CPU sia il controllore del dispositivo sono contemporaneamente occupati. Se si è fortunati, quando il processo è pronto per elaborare i dati successivi, il controllore ha terminato la loro lettura. La CPU è quindi pronta a elaborare i dati letti mentre il controllore del dispositivo può iniziare a leggerne di nuovi. Un metodo simile si può applicare all'emissione di dati. In questo caso il processo crea i dati che restano nella memoria fino a quando un dispositivo (nastro, disco, stampante, ecc.) potrà accettarli. Confrontate lo schema del prelievo anticipato con quello della gestione asincrona dell'elaborazione e dell'I/O (il cosiddetto *spooling*), nel quale la CPU sovrappone l'immissione di dati per un processo all'elaborazione e all'emissione di dati per altri processi.

2.2 Spiegate come funziona la distinzione tra modo d'utente e modo di sistema come forma rudimentale di protezione dei sistemi. by hardware

2.3 Dite quali sono le differenze tra un segnale d'interruzione e uno di eccezione e spiegate il funzionamento di ciascuno. by prog execution

2.4 Dite per quali tipi di operazioni è utile il DMA e spiegate perché. ✓ (operazioni di I/O)

2.5 Dite quali tra le seguenti istruzioni dovrebbero essere privilegiate:

a) impostazione del temporizzatore;
b) lettura del clock;
c) pulizia della memoria; ✓
d) disabilitazione delle interruzioni; ✓
e) passaggio dal modo d'utente al modo di sistema. ✓

2.6 L'architettura di alcuni sistemi di calcolo non ha un modo di funzionamento riservato al sistema operativo. Dite se per questi calcolatori si possono realizzare sistemi operativi sicuri. Argomentate la risposta. NO, manca protezione della

2.7 Alcuni tra i primi calcolatori proteggevano il sistema operativo caricandolo in una partizione della memoria che né i lavori d'elaborazione degli utenti né lo stesso sistema operativo potevano modificare. Descrivete due problemi che questo schema può introdurre.

2.8 La protezione del sistema operativo è cruciale al fine di assicurarne il corretto funzionamento ed è la ragione fondamentale dei concetti di duplice modo di funzionamento, protezione della memoria e presenza del temporizzatore. D'altra parte, per garantire la massima flessibilità vorreste ridurre al minimo le limitazioni imposte agli utenti.

Quella che segue è una lista delle istruzioni che di solito sono protette; dite qual è l'insieme *minimo* di istruzioni che devono essere protette:

a) passaggio al modo d'utente; ✓
b) lettura dalla memoria del sistema operativo;
c) scrittura nella memoria del sistema operativo; ✓
d) prelievo di un'istruzione dalla memoria del sistema operativo;
e) attivazione delle interruzioni del temporizzatore; ✓
f) disattivazione delle interruzioni del temporizzatore. ✓

2.9 Fornite due motivi per i quali le cache sono utili, quali problemi risolvono e quali problemi introducono. Potendo disporre di una cache delle dimensioni del dispositivo cui è associata (ad esempio, una cache delle dimensioni di un disco), dite se è possibile usarla eliminando il dispositivo.

2.10 La realizzazione di un sistema operativo che possa funzionare senza interferenze da parte di programmi utenti malvagi o malfunzionanti richiede specifiche caratteristiche dell'architettura del sistema. Elencate, in questo senso, tre caratteristiche utili al sistema operativo e descrivete come si possono usare insieme per la sua protezione.

2.11 Alcune CPU offrono più di due modi di funzionamento. Dite due possibili usi di questi modi multipli. geografici → locali

2.12 Dite quali sono le principali differenze tra una WAN e una LAN.

2.13 Dite quale configurazione di rete è più adatta ai seguenti ambienti:

a) un piano di un pensionato universitario; LAN
b) un *campus* universitario; LAN
c) una regione; MAN
d) una nazione. WAN

http://www.adulpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

2.9 Note bibliografiche

[Hennessy e Patterson 1996] fornisce una trattazione dei sistemi di I/O, dei bus, e più in generale delle architetture di sistema. [Tanenbaum 1990] descrive le architetture dei microcalcolatori, partendo dai dettagli degli elementi fisici che li compongono.

Una discussione riguardante le tecnologie dei dischi magnetici è presentata da [Freedman 1983] e [Harker et al. 1981]. I dischi ottici sono affrontati da [Kenville 1982], [Fujitani 1984], [O'Leary e Kitts 1985], [Gait 1988] e [Olsen e Kenly 1989]. Discussioni sui dischetti si trovano in [Pechura e Schoeffler 1983] e [Sarisky 1983].

Le memorie cache, comprese le memorie associative, sono descritte e analizzate da [Smith 1982]; questo articolo contiene tra l'altro un'estesa bibliografia su tale argomento.

Una discussione generale riguardante la tecnologia delle memorie di massa è offerta da [Chi 1982] e [Hoagland 1985].

[Tanenbaum 1996] e [Halsall 1992] offrono una trattazione generale delle reti di calcolatori. [Fortier 1989] presenta una trattazione dettagliata degli aspetti ingegneristici e sistemistici delle reti di calcolatori.

Capitolo 3

Strutture dei sistemi operativi

I sistemi operativi forniscono l'ambiente in cui si eseguono i programmi. Essendo organizzati secondo criteri che possono essere assai diversi, anche la struttura interna che li caratterizza può essere molto diversa. La progettazione di un nuovo sistema operativo è un compito assai complesso, perciò è necessario definirne in modo chiaro gli scopi. Il tipo di sistema desiderato definisce i criteri di scelta dei metodi e degli algoritmi necessari.

Un sistema operativo si può considerare da diversi punti di vista: secondo i servizi che esso fornisce o l'interfaccia messa a disposizione degli utenti e dei programmatori; inoltre si può analizzare il funzionamento dei singoli elementi che lo costituiscono e studiarne le mutue relazioni; si possono cioè considerare i punti di vista degli utenti, dei programmatori e dei progettisti di sistemi operativi. In questo capitolo si analizzano i servizi che un sistema operativo offre, il modo in cui questi sono offerti e i metodi da adottare per la sua progettazione.

3.1 Componenti del sistema

Un sistema grande e complesso come un sistema operativo si può creare solo se si scompone in piccole parti, ciascuna delle quali deve essere una porzione ben delineata del sistema, con interfacce e funzioni definite con estrema cura. Naturalmente, anche se non tutti i sistemi hanno la stessa struttura, molti hanno in comune lo scopo, che consiste nell'incorporare i componenti di sistema descritti nei Paragrafi dal 3.1.1 al 3.1.8.

3.1.1 Gestione dei processi

Un programma fa qualcosa soltanto se la CPU esegue le istruzioni che lo costituiscono. Benché la sua definizione sia più generale, un processo d'elaborazione — o, più brevemente, processo —, si può in prima istanza considerare come un programma in esecuzione. Un programma d'utente, come un compilatore, eseguito in un ambiente a partizione del tempo d'elaborazione, è un processo; un programma d'elaborazione di testi eseguito da un PC per un singolo utente è un processo; così come un servizio di sistema,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ad esempio l'invio di dati a una stampante, è un processo. Per il momento ci si può limitare a considerare un processo come un lavoro d'elaborazione (*job*) o un programma eseguito in un ambiente a partizione del tempo d'elaborazione, anche se il concetto è più generale. Come si descrive nel Capitolo 4, si possono avere chiamate del sistema che permettono ai processi di creare sottoprocessi da eseguire in modo concorrente.

Per svolgere i propri compiti, un processo necessita di alcune risorse, tra cui tempo di CPU, memoria, file e dispositivi di I/O. Queste risorse si possono attribuire al processo al momento della sua creazione, oppure si possono assegnare durante l'esecuzione. Oltre alle diverse risorse fisiche e logiche assegnate a un processo durante la sua creazione, si possono considerare anche alcuni dati d'inizializzazione da passare di volta in volta al processo stesso. Ad esempio, un processo che ha lo scopo di mostrare su uno schermo lo stato di un file, riceve il nome del file ed esegue le appropriate istruzioni e chiamate del sistema che gli consentono di ottenere e mostrare sullo schermo le informazioni desiderate; quando il processo termina, il sistema operativo riprende il controllo delle risorse impiegate dal processo.

Si sottolinea che un programma di per sé non è un processo d'elaborazione; un programma è un'entità *passiva*, come il contenuto di un file memorizzato in un disco, mentre un processo è un'entità *attiva*, con un contatore di programma che indica la successiva istruzione da eseguire. L'esecuzione di un processo deve essere sequenziale: la CPU esegue le istruzioni del processo una dopo l'altra finché il processo termina, inoltre a ogni istante si esegue al più una istruzione del processo. Quindi, anche se due processi possono corrispondere allo stesso programma, si considerano in ogni caso due sequenze d'esecuzione separate. Succede spesso che un programma durante la propria esecuzione generi molti processi.

Il processo è l'unità di lavoro di un sistema. Tale sistema è costituito di un gruppo di processi, alcuni dei quali del sistema operativo (che eseguono codice di sistema), mentre altri sono processi utenti (che eseguono codice d'utente). Tutti questi processi si possono potenzialmente eseguire in modo concorrente, avvicendandosi nell'uso della CPU. Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei processi:

- creazione e cancellazione dei processi utenti e di sistema;
- sospensione e ripristino dei processi;
- fornitura di meccanismi per la sincronizzazione dei processi;
- fornitura di meccanismi per la comunicazione tra processi;
- fornitura di meccanismi per la gestione delle situazioni di stallo (*deadlock*).

Le tecniche di gestione dei processi sono trattate in modo approfondito nei Capitoli dal 4 al 7.

Ricordare: è il dispositivo di memoria centrale. Perché il suo contenuto x quanti dall'esterno

3.1.2 Gestione della memoria centrale

Come si è accennato nel Capitolo 1, la memoria centrale è fondamentale per il funzionamento di un moderno sistema di calcolo. Si tratta di un vasto vettore di dimensioni che variano tra le centinaia di migliaia e i miliardi di parole, ciascuna delle quali è dotata del proprio indirizzo. È un magazzino di dati velocemente accessibile ed è condivisa dalla CPU e da alcuni dispositivi di I/O. La CPU legge le istruzioni dalla memoria centrale durante il ciclo di prelievo delle istruzioni, oltre a leggere e scrivere i dati nella memoria centrale durante il ciclo d'accesso ai dati. Anche il funzionamento dell'I/O realizzato col DMA legge e scrive dati nella memoria centrale. Generalmente la memoria centrale è l'unico ampio dispositivo di memorizzazione cui la CPU può far riferimento e accedere in modo diretto. Ad esempio, affinché la CPU possa gestire i dati di un disco, occorre che essi siano prima trasferiti nella memoria centrale attraverso le richieste di I/O generate dalla CPU. In modo analogo, la CPU può eseguire le istruzioni solo se si trovano nella memoria.

Per eseguire un programma è necessario che questo sia associato a indirizzi assoluti e sia caricato nella memoria. Durante l'esecuzione del programma, la CPU accede alle proprie istruzioni e ai dati provenienti dalla memoria, generando i suddetti indirizzi assoluti. Quando il programma termina, si dichiara disponibile il suo spazio di memoria; a questo punto si può caricare ed eseguire il programma successivo.

Per migliorare l'utilizzo della CPU e la rapidità con la quale il calcolatore risponde ai suoi utenti occorre tenere molti programmi nella memoria. Esistono diversi schemi di gestione della memoria; l'efficacia di ogni algoritmo dipende dalla situazione specifica. La scelta di un particolare schema di gestione della memoria dipende da molti fattori, principalmente dal tipo di *architettura* del sistema, poiché ogni algoritmo richiede specifiche caratteristiche.

Il sistema operativo è responsabile delle seguenti attività connesse alla gestione della memoria centrale:

- tenere traccia di quali parti della memoria sono attualmente usate e da che cosa;
- decidere quali processi si debbano caricare nella memoria quando vi sia spazio disponibile;
- assegnare e revocare lo spazio di memoria secondo le necessità.

Le tecniche di gestione della memoria sono trattate in profondità nei Capitoli 9 e 10.

3.1.3 Gestione dei file

La gestione dei file è uno dei componenti più visibili di un sistema operativo. I calcolatori possono registrare le informazioni su molti mezzi fisici diversi; i più diffusi sono il nastro magnetico, il disco magnetico e il disco ottico. Ciascuno ha caratteristiche proprie e una propria organizzazione fisica, ed è controllato da un dispositivo, come un'unità a disco o a nastro, aventi anch'esse caratteristiche proprie: velocità, capacità, velocità di trasferimento dei dati, metodi d'accesso (diretto o sequenziale).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per rendere conveniente l'utilizzo del calcolatore, il sistema operativo fornisce una visione logica uniforme del processo di registrazione delle informazioni. Il sistema operativo astrae le caratteristiche fisiche dei dispositivi per definire un'unità di memorizzazione logica, il file. Il sistema operativo associa i file ai mezzi fisici e vi accede attraverso i dispositivi che li controllano.

Un file è una raccolta d'informazioni correlate definite dal loro creatore. Comunemente, i file rappresentano programmi, sia sorgente sia oggetto, e dati. I file di dati possono essere numerici, alfabetici o alfanumerici; la loro forma può essere libera, come nei file di testo, oppure rigidamente formattata, ad esempio in campi fissati. I file sono formati da una sequenza di bit, byte, righe o *record*, i cui significati sono definiti dai loro creatori. Il concetto di file è molto generale.

Il sistema operativo realizza il concetto astratto di file gestendo i mezzi di memoria di massa, come nastri e dischi, e i dispositivi che li controllano. I file sono generalmente organizzati in directory, che ne facilitano l'uso. Infine, se più utenti hanno accesso ai file, si potrebbe voler controllare chi ha la possibilità di accedervi e in che modo (ad esempio, lettura, scrittura, aggiunta).

Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei file:

- creazione e cancellazione di file;
- creazione e cancellazione di directory;
- fornitura delle funzioni fondamentali per la gestione di file e directory;
- associazione dei file ai dispositivi di memoria secondaria;
- creazione di copie di riserva (*backup*) dei file su dispositivi di memorizzazione non volatili.

Le tecniche di gestione dei file sono trattate nei Capitoli 11 e 12.

3.1.4 Gestione del sistema di I/O

Uno tra gli scopi di un sistema operativo è nascondere all'utente le caratteristiche degli specifici dispositivi. Nello UNIX, ad esempio, le caratteristiche dei dispositivi di I/O sono nascoste alla maggior parte dello stesso sistema operativo dal sottosistema di I/O, che è composto delle seguenti parti:

- un componente di gestione della memoria comprendente la gestione delle regioni della memoria riservate ai trasferimenti di I/O (*buffer*), la gestione delle cache e la gestione asincrona delle operazioni di I/O e dell'esecuzione di più processi (*spooling*);
- un'interfaccia generale per i driver dei dispositivi;
- i driver per gli specifici dispositivi.

Soltanto il driver del dispositivo conosce le caratteristiche dello specifico dispositivo cui è assegnato.

Nel Capitolo 2 è presentato l'uso dei gestori dei segnali d'interruzione e dei driver dei dispositivi per la costruzione di sottosistemi di I/O efficienti. Nel Capitolo 13 si discute per esteso il modo in cui il sottosistema di I/O interagisce con gli altri componenti del sistema, come gestisce i dispositivi, trasferisce i dati e individua il completamento delle operazioni di I/O.

3.1.5 Gestione della memoria secondaria

Lo scopo principale di un sistema di calcolo è quello di eseguire programmi. Durante l'esecuzione, i programmi, insieme con i dati cui accedono, devono trovarsi nella memoria centrale. Giacché la memoria centrale è troppo piccola per contenere tutti i dati e tutti i programmi, e il suo contenuto va perduto se il sistema si spegne, il calcolatore deve disporre di una memoria secondaria a sostegno della memoria centrale. La maggior parte dei moderni sistemi di calcolo impiega i dischi come principale mezzo di memorizzazione secondaria, sia per i programmi sia per i dati. I dischi contengono la maggior parte dei programmi, compresi i compilatori, gli assemblatori, le procedure di ordinamento e gli editor. Questi programmi rimangono nel disco fino al momento del caricamento nella memoria e si servono del disco sia come sorgente sia come destinazione delle loro computazioni; per tale ragione è fondamentale che la registrazione nei dischi sia gestita correttamente.

Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei dischi:

- gestione dello spazio libero;
- assegnazione dello spazio;
- scheduling del disco.

Il frequente uso della memoria secondaria impone una sua gestione efficiente. Infatti, l'efficienza complessiva di un calcolatore può dipendere dalla velocità del sottosistema di gestione dei dischi e dagli algoritmi che lo gestiscono. Le tecniche di gestione della memoria secondaria sono trattate nel Capitolo 14.

3.1.6 Reti

Un sistema distribuito è un insieme di unità d'elaborazione che non condividono la memoria, i dispositivi periferici o un clock. Ciascuna unità d'elaborazione dispone di una propria memoria locale e di un suo clock; tutte le unità d'elaborazione comunicano tra loro tramite varie linee di comunicazione, come bus ad alta velocità o reti. Le unità d'elaborazione che compongono un sistema distribuito hanno dimensioni e funzioni variabili: possono essere CPU, stazioni di lavoro, minicalcolatori e grandi sistemi di calcolo.

In un sistema distribuito le unità d'elaborazione sono collegate da una rete di comunicazione che può essere configurata in vari modi, e può essere totalmente o parzialmente connessa.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Nella progettazione di una rete di comunicazione si deve tenere conto dei metodi di connessione e d'instradamento dei messaggi, oltre che dei problemi di contesa e sicurezza.

Un sistema distribuito è un insieme, anche eterogeneo, di sistemi fisicamente separati, organizzato in modo da costituire un unico ambiente coerente che offre all'utente l'accesso alle proprie risorse. L'accesso a una risorsa condivisa permette di accelerare il calcolo, di migliorare la funzionalità, di aumentare la disponibilità dei dati e di incrementare l'affidabilità. Normalmente i sistemi operativi generalizzano l'accesso alla rete come una forma d'accesso ai file, dove i particolari riguardanti l'interconnessione sono contenuti nel driver del dispositivo d'interfaccia della rete stessa. I protocolli che definiscono un sistema distribuito possono avere notevoli conseguenze sull'utilità e la diffusione di un sistema. Le innovazioni introdotte dal World Wide Web (più in breve, Web) sono dovute alla creazione di un nuovo metodo d'accesso alle informazioni condivise: ha migliorato i già esistenti protocolli FTP (*file transfer protocol*) e NFS (*network file system*) eliminando la necessità di una procedura preliminare ed esplicita per l'accesso a una risorsa condivisa. Definisce un nuovo protocollo, l'**http** (*hypertext transfer protocol*), per la comunicazione tra i server del World Wide Web (*Web server*) e i programmi di consultazione dello stesso World Wide Web (*Web browser*). Per ricevere informazioni (testi, immagini, collegamenti ad altre informazioni), un programma di consultazione del Web deve semplicemente farne richiesta a un calcolatore remoto che svolge il ruolo di Web server, che provvederà a inviarle al richiedente. Tale semplificazione ha favorito l'enorme crescita dell'uso del protocollo http e in generale del Web. Le reti e i sistemi distribuiti sono trattati nei Capitoli dal 15 al 17.

3.1.7 Sistema di protezione

Se un sistema di calcolo ha più utenti e consente che più processi siano eseguiti in modo concorrente, i diversi processi devono essere protetti dalle attività di altri processi. Per tale ragione esistono meccanismi che assicurano che i file, i segmenti della memoria, la CPU e altre risorse possano essere controllate solo dai processi che hanno ricevuto l'autorizzazione dal sistema operativo.

Ad esempio, l'architettura d'indirizzamento della memoria assicura che un processo possa svolgersi solo all'interno del proprio spazio d'indirizzi. Il temporizzatore assicura che nessun processo possa acquisire il controllo della CPU senza poi restituirlo al sistema operativo. Infine, agli utenti non è permesso l'accesso ai registri di controllo dei dispositivi, perciò risulta protetta l'integrità dei diversi dispositivi periferici.

La protezione è definita da ogni meccanismo che controlla l'accesso da parte di programmi, processi o utenti alle risorse di un sistema di calcolo. Questo meccanismo deve fornire i mezzi che specifichino quali controlli si debbano eseguire e i mezzi per effettuarli.

La protezione può migliorare l'affidabilità individuando errori latenti nelle interfacce tra componenti di sottosistemi. Una sollecita individuazione di errori in un'interfaccia può spesso impedire la 'contaminazione' di un sottosistema integro da parte di un altro sottosistema malfunzionante. Una risorsa priva di protezione non può difendersi dall'uso, o dall'abuso, di un utente non autorizzato o incompetente. Un sistema protetto offre la possibilità di distinguere tra uso autorizzato e non autorizzato. Questo argomento è trattato nel Capitolo 18.

3.1.8 Interprete dei comandi

Uno dei programmi di sistema più importanti di un sistema operativo è l'**interprete dei comandi**; si tratta dell'interfaccia tra l'utente e il sistema operativo. Alcuni sistemi operativi contengono l'interprete dei comandi nel nucleo. Altri sistemi operativi, come l'MS-DOS e lo UNIX, trattano l'interprete dei comandi come un programma speciale che si esegue quando si avvia un lavoro d'elaborazione, oppure quando un utente inizia una sessione di lavoro in un sistema a partizione del tempo.

Molti comandi si impartiscono al sistema operativo attraverso **istruzioni di controllo**. Quando in un sistema a lotti si inizia un nuovo lavoro d'elaborazione, oppure quando un utente inizia una sessione di lavoro in un sistema a partizione del tempo, il sistema esegue automaticamente un programma che legge e interpreta le istruzioni di controllo. Questo programma ha diversi nomi: **interprete di schede di controllo**, **interprete di riga di comando**; spesso è noto come *shell*. La sua funzione è semplice: prelevare ed eseguire la successiva istruzione di comando.

Spesso i sistemi operativi si differenziano proprio dal punto di vista dell'interprete dei comandi. Esistono interpreti 'amichevoli' che rendono il sistema più adatto a certi utenti; ad esempio i sistemi basati su mouse, finestre e menu, come il Macintosh e il Microsoft Windows. Usando il mouse, l'utente indica sullo schermo immagini, o **icone**, che rappresentano programmi, file e funzioni del sistema. Secondo la posizione del puntatore, premendo un pulsante del mouse si può avviare l'esecuzione di un programma, selezionare un file o una directory (in questo contesto chiamata **cartella**), oppure aprire un menu che contiene un elenco di comandi. Interpreti più potenti, complessi e di difficile apprendimento sono apprezzati da altri utenti. In alcuni di questi interpreti i comandi si immettono con la tastiera e appaiono su uno schermo; il tasto Invio segnala la fine di un comando, indicando che il comando è pronto per essere eseguito. Gli interpreti dei comandi dell'MS-DOS e dello UNIX funzionano in questo modo.

Le istruzioni di comando riguardano la creazione e la gestione dei processi, la gestione dell'I/O, la gestione della memoria secondaria, la gestione della memoria centrale, l'accesso al file system, la protezione e la comunicazione tramite la rete.

3.2 Servizi di un sistema operativo

Un sistema operativo offre un ambiente in cui eseguire i programmi e fornire servizi ai programmi e ai loro utenti. Naturalmente, i servizi specifici variano secondo il sistema operativo, ma si possono identificare alcune classi di servizi comuni. Per rendere più agevole la programmazione, si offrono i seguenti servizi:

- Esecuzione di un programma. Il sistema deve poter caricare un programma nella memoria ed eseguirlo. Il programma deve poter terminare la propria esecuzione in modo normale o anormale (indicando l'errore).
- Operazioni di I/O. Un programma in esecuzione può richiedere un'operazione di I/O che implica l'uso di un file o di un dispositivo di I/O. Per particolari dispositivi

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

possono essere necessarie funzioni speciali, come il riavvolgimento di un'unità a nastro, oppure la cancellazione dello schermo di un tubo a raggi catodici (CRT). Per motivi di efficienza e protezione, di solito un utente non può controllare direttamente i dispositivi di I/O, quindi il sistema operativo deve offrire mezzi adeguati.

- **Gestione del file system.** Il file system riveste un interesse particolare. I programmi richiedono l'esecuzione di operazioni di lettura e scrittura su file, oltre a creare e cancellare file.
- **Comunicazioni.** In molti casi un processo ha bisogno di scambiare informazioni con un altro processo. Ciò avviene principalmente in due modi: tra processi in esecuzione nello stesso calcolatore e tra processi in esecuzione in calcolatori diversi collegati per mezzo di una rete. La comunicazione si può realizzare tramite una **memoria condivisa** o attraverso lo **scambio di messaggi**, in questo caso il sistema operativo trasferisce pacchetti d'informazioni tra i vari processi.
- **Rilevamento d'errori.** Il sistema operativo deve essere sempre capace di rilevare eventuali errori che possono verificarsi nella CPU e nei dispositivi di memoria, come un errore di memoria o un guasto all'alimentazione elettrica; nei dispositivi di I/O, come un errore di parità in un nastro, il guasto di una connessione di rete, la mancanza di carta nella stampante; e in un programma utente, come una divisione per zero, un tentativo d'accesso a una locazione di memoria illegale, un uso eccessivo del tempo di CPU. Per ciascun tipo d'errore il sistema operativo deve saper intraprendere l'azione giusta per assicurare un'elaborazione corretta e coerente.

Esiste anche un'altra serie di funzioni del sistema operativo che non riguarda direttamente gli utenti, ma assicura il funzionamento efficiente del sistema stesso. Sistemi con più utenti possono guadagnare in efficienza condividendo le risorse del calcolatore tra i diversi utenti.

- **Assegnazione delle risorse.** Se sono in corso più sessioni di lavoro di utenti o sono contemporaneamente in esecuzione più processi, il sistema operativo provvede all'assegnazione delle risorse necessarie a ciascuno di essi. Alcune di queste risorse, come i cicli di CPU, la memoria centrale e la registrazione in file, possono avere un codice di assegnazione speciale, mentre altre, come i dispositivi di I/O, possono avere un codice di richiesta e di rilascio più generale. Ad esempio, per determinare come utilizzare al meglio la CPU, i sistemi operativi impiegano le procedure di scheduling della CPU, che tengono conto della velocità della CPU, dei processi da eseguire, del numero di registri disponibili e di altri fattori. Esistono anche procedure per l'assegnazione di risorse a uso di un processo; una procedura di questo tipo localizza una risorsa disponibile e modifica una tabella interna per registrare il nuovo utente di tale risorsa. Per cancellare la tabella si usa un'altra procedura.

- **Contabilizzazione dell'uso delle risorse.** È possibile registrare quali utenti usino il calcolatore, segnalando quali e quante risorse impieghino. Questo tipo di registrazione si può usare per contabilizzare l'uso delle risorse, per addebitare il costo agli utenti, oppure per redigere statistiche; queste ultime possono essere un valido strumento per i ricercatori che desiderano riconfigurare il sistema per migliorarne i servizi di calcolo.
- **Protezione.** I proprietari di informazioni memorizzate in un sistema di calcolo multiutente possono voler controllare l'uso di tali informazioni. Più processi non correlati e in esecuzione concorrente non devono influenzarsi o interferire con il sistema operativo. La protezione assicura che l'accesso alle risorse del sistema sia controllato. È importante anche proteggere il sistema dagli estranei. La sicurezza di un sistema comincia con l'obbligo da parte di ciascun utente di farsi identificare dal sistema, di solito attraverso parole d'ordine che permettono l'accesso alle risorse; si estende alla difesa dei dispositivi di I/O (compresi i modem e gli adattatori di rete) dai tentativi d'accesso illegali e provvede al loro rilevamento. Se un sistema deve essere protetto e sicuro, al suo interno devono esistere precauzioni ovunque; ricordando che la forza di una catena è esattamente quella del suo anello più debole.

3.3 Chiamate del sistema

Le chiamate del sistema costituiscono l'interfaccia tra un processo e il sistema operativo. Sono generalmente disponibili in forma di istruzioni in linguaggio assemblativo e sono elencate nei manuali per i programmatori che usano questo linguaggio.

Certi sistemi possono permettere che le chiamate del sistema siano invocate direttamente da un programma scritto in un linguaggio di alto livello; in tal caso s'invocano funzioni o procedure predefinite. Questi sistemi possono generare direttamente le chiamate del sistema, oppure possono generare una chiamata a una speciale procedura dell'ambiente d'esecuzione che effettua la chiamata del sistema.

Parecchi linguaggi, ad esempio il Linguaggio C, il C++, e il Perl sono stati definiti allo scopo di sostituire il linguaggio assemblativo per la programmazione dei sistemi. Questi linguaggi consentono di eseguire direttamente le chiamate del sistema. Le chiamate del sistema dello UNIX ad esempio si possono invocare direttamente da un programma scritto in Linguaggio C o in C++. Le chiamate del sistema per i moderni sistemi Microsoft Windows sono disponibili tramite l'API Win32 (*application programmer interface*), per tutti i compilatori scritti per i sistemi operativi Microsoft Windows.

Come esempio si può considerare la scrittura di un semplice programma che legge i dati da un file e li trascrive in un altro file. La prima informazione di cui il programma necessita è costituita dei nomi dei due file: il file sorgente e il file di destinazione. Questi file si possono indicare in molti modi diversi, secondo la struttura del sistema operativo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un primo metodo consiste nel chiedere i nomi dei due file all'utente del programma. In un sistema interattivo questa operazione richiede una sequenza di chiamate del sistema, prima per scrivere un messaggio di richiesta sullo schermo e quindi per leggere dalla tastiera i caratteri che compongono i nomi dei due file. Nei sistemi basati su mouse e finestre viene normalmente mostrato in una finestra un menu contenente i nomi dei file. L'utente può usare il mouse per scegliere il nome del file sorgente, dopodiché è possibile aprire una finestra simile alla precedente nella quale specificare il nome del file di destinazione.

Una volta ottenuti i nomi, il programma deve aprire il file sorgente e creare il file di destinazione. Ciascuna di queste operazioni richiede un'altra chiamata del sistema e può andare incontro a condizioni d'errore. Ad esempio, quando il programma tenta di aprire il file sorgente, può scoprire che non esiste alcun file con quel nome, oppure che l'accesso al file è negato. In questi casi il programma deve scrivere un messaggio nello schermo della console (altra sequenza di chiamate del sistema) e quindi terminare in maniera anormale la propria elaborazione (altra chiamata del sistema). Se il file sorgente esiste, è necessario creare il file di destinazione. È possibile che esista già un file col nome indicato per il file di destinazione; questa situazione potrebbe causare l'interruzione del programma (una chiamata del sistema) o la cancellazione del file esistente (un'altra chiamata del sistema) e la creazione di uno nuovo. Un'altra possibilità, in un sistema interattivo, prevede di chiedere all'utente (sequenza di chiamate del sistema per emettere il messaggio di richiesta e leggere la risposta dal terminale) se si debba sostituire il file già esistente o se si debba terminare l'esecuzione del programma.

Una volta predisposti i due file si entra in un ciclo che legge dal file sorgente (una chiamata del sistema) e scrive nel file di destinazione (altra chiamata del sistema). Ciascuna lettura (`read`) e ogni scrittura (`write`) deve riportare informazioni di stato relative alle possibili condizioni d'errore. Nel file sorgente il programma può rilevare che è stata raggiunta la fine del file, oppure che nella lettura si è riscontrato un errore del dispositivo, ad esempio un errore di parità. Nella fase di scrittura si possono verificare vari errori, la cui natura dipende dal dispositivo impiegato. Esempi tipici sono l'esaurimento dello spazio nei dischi, la fine di un nastro, la mancanza della carta in una stampante, e così via.

Infine, una volta copiato tutto il file, il programma può chiudere entrambi i file (altre chiamate del sistema), inviare un messaggio alla console (più chiamate del sistema) e infine terminare normalmente (ultima chiamata del sistema). Come s'è visto, anche programmi molto semplici possono fare un intenso uso del sistema operativo.

La maggior parte degli utenti però non deve considerare tutti questi dettagli: il sistema d'ausilio alla fase d'esecuzione (*run-time*) — l'insieme di funzioni fornite con un compilatore — dei linguaggi di programmazione, di solito offre un'interfaccia assai più semplice. Un'istruzione count del C++, è probabilmente compilata come una chiamata a una procedura d'ausilio alla fase d'esecuzione che emette le chiamate del sistema necessarie, controlla la presenza di eventuali errori e infine restituisce il controllo al programma utente. In questo modo, il compilatore e il suo insieme di funzioni d'ausilio alla fase d'esecuzione nascondono al programmatore la maggior parte dei dettagli d'interfaccia del sistema operativo.

Le chiamate del sistema si presentano in modi diversi, secondo il calcolatore in uso. Spesso sono richieste maggiori informazioni oltre alla semplice identità della chiamata del sistema desiderata. Il tipo e l'entità esatta delle informazioni variano secondo lo specifico sistema operativo e la specifica chiamata del sistema. Per ottenere l'immissione di un dato, ad esempio, può essere necessario specificare il file o il dispositivo da usare come sorgente e anche l'indirizzo e l'ampiezza dell'area della memoria in cui depositare i dati letti. Naturalmente, il dispositivo o il file e l'ampiezza possono essere impliciti nella chiamata del sistema.

Per passare parametri al sistema operativo si usano tre metodi generali. Il più semplice consiste nel passare i parametri in *registri*; si possono però presentare casi in cui vi sono più parametri che registri. In questo caso, generalmente si memorizzano i parametri in un *blocco* o tabella di memoria e si passa l'indirizzo del blocco, in forma di parametro, in un registro (Figura 3.1). È il metodo seguito dal sistema operativo LINUX. Il programma può anche collocare (*push*) i parametri in una pila da cui sono prelevati (*pop*) dal sistema operativo. Alcuni sistemi operativi preferiscono i metodi del blocco o della pila, poiché non limitano il numero o la lunghezza dei parametri da passare.

Le chiamate del sistema si possono classificare approssimativamente in cinque categorie principali: **controllo dei processi, gestione dei file, gestione dei dispositivi, gestione delle informazioni e comunicazioni.** Nei Paragrafi dal 3.3.1 al 3.3.5 sono trattati brevemente i tipi di chiamate del sistema forniti da un sistema operativo.

La maggior parte di queste chiamate del sistema implicano, o presuppongono, concetti e funzioni che sono trattati in capitoli successivi. La Figura 3.2 riassume i tipi di chiamate del sistema forniti normalmente da un sistema operativo.

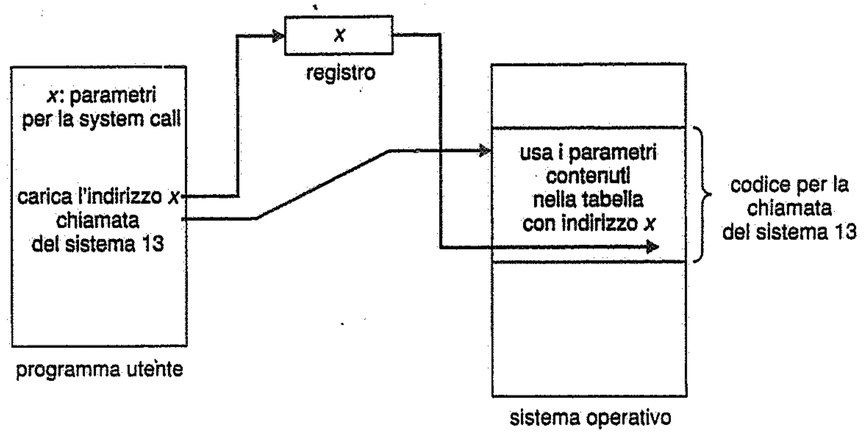


Figura 3.1 Passaggio di parametri in forma di tabella.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- Controllo dei processi
  - termine normale e anormale
  - caricamento, esecuzione
  - creazione e arresto di un processo
  - esame e impostazione degli attributi di un processo
  - attesa per il tempo indicato
  - attesa e segnalazione di un evento
  - assegnazione e rilascio di memoria
- Gestione dei file
  - creazione e cancellazione di file
  - apertura, chiusura
  - lettura, scrittura, posizionamento
  - esame e impostazione degli attributi di un file
- Gestione dei dispositivi
  - richiesta e rilascio di un dispositivo
  - lettura, scrittura, posizionamento
  - esame e impostazione degli attributi di un dispositivo
  - inserimento logico ed esclusione logica di un dispositivo
- Gestione delle informazioni
  - esame e impostazione dell'ora e della data
  - esame e impostazione dei dati del sistema
  - esame e impostazione degli attributi dei processi, file e dispositivi
- Comunicazione
  - creazione e chiusura di una connessione
  - invio e ricezione di messaggi
  - informazioni sullo stato di un trasferimento
  - inserimento ed esclusione di dispositivi remoti

Figura 3.2 Tipi di chiamate del sistema.

3.3.1 Controllo dei processi

Un programma in esecuzione deve potersi fermare sia normalmente (in tal caso si parla di end) sia anormalmente (`abort`). Se per terminare in modo anormale un programma in esecuzione s'impiega una chiamata del sistema, oppure se il programma incontra difficoltà e causa l'emissione di un segnale di eccezione, talvolta si ha la registrazione in un file di un'immagine del contenuto della memoria (*dump*) e l'emissione di un messaggio

d'errore. Uno specifico programma di ricerca e correzione degli errori (*debugger*) può esaminare tali informazioni per determinare le cause del problema. Sia in condizioni normali sia anormali il sistema operativo deve trasferire il controllo all'interprete dei comandi che legge il comando successivo. In un sistema interattivo l'interprete dei comandi continua semplicemente a interpretare il comando successivo; si suppone che l'utente invii un comando idoneo per rispondere a qualsiasi errore. In un sistema a lotti l'interprete dei comandi generalmente termina il lavoro corrente e prosegue con il successivo. Quando si presenta un errore, alcuni sistemi permettono alle schede di controllo di indicare le specifiche azioni di recupero da intraprendere. Se il programma scopre un errore nei dati ricevuti e desidera terminare in modo anormale, può anche definire un livello d'errore. Più grave è l'errore, più alto è il livello del parametro che lo individua. Sono quindi possibili una terminazione normale e una terminazione anormale, definendo la terminazione normale come errore di livello 0. L'interprete dei comandi o un programma successivo può usare questo livello d'errore per determinare l'azione da intraprendere.

Un processo che esegue un programma può richiedere di caricare (`load`) ed eseguire (`execute`) un altro programma. In questo caso l'interprete dei comandi esegue un programma come se tale richiesta fosse stata impartita, ad esempio, da un comando d'utente, oppure dal clic di un mouse. È interessante chiedersi dove si debba restituire il controllo una volta terminato il programma caricato. La questione è legata alle eventualità che il programma attuale sia andato perso, salvato oppure che abbia continuato l'esecuzione in modo concorrente con il nuovo programma.

Se al termine del nuovo programma il controllo rientra nel programma esistente, si deve salvare l'immagine della memoria del programma attuale, creando così effettivamente un meccanismo con cui un programma può chiamare un altro programma. Se entrambi i programmi continuano l'esecuzione in modo concorrente, si è creato un nuovo processo da sottoporre a multiprogrammazione. A questo scopo spesso si fornisce una chiamata del sistema specifica, e precisamente `create process`.

Quando si crea un nuovo processo, o anche un gruppo di processi, è necessario mantenerne il controllo; ciò richiede la capacità di determinare e reimpostare gli attributi di un processo, compresi la sua priorità, il suo tempo massimo d'esecuzione e così via (`get process attributes` e `set process attributes`). Inoltre, può essere necessario terminare un processo creato, se si riscontra che non è corretto o se non serve (`terminate process`).

Una volta che sono stati creati, può essere necessario attendere che i processi terminino la loro esecuzione. Quest'attesa si può impostare per un certo periodo di tempo (`wait` *`tempo`*), ma è più probabile che si preferisca attendere che si verifichi un dato evento (`wait` *`evento`*). I processi devono quindi segnalare il verificarsi di quell'evento (`signal` *`evento`*). Chiamate del sistema di questo tipo, che trattano cioè il coordinamento di processi concorrenti, sono esaminate in profondità nel Capitolo 7.

Un altro gruppo di chiamate del sistema è utile per la messa a punto dei programmi: molti sistemi forniscono chiamate del sistema per la registrazione dell'immagine della memoria (`dump`); alcuni offrono la possibilità di elencare ogni istruzione nel momento in cui viene eseguita (`trace`). Anche le CPU dispongono di un modo d'esecuzione noto come esecuzione a passo singolo (*single step*), nel quale la CPU emette un segnale di ec-

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

cezione dopo ogni istruzione. Tale eccezione è normalmente rilevata da un programma d'ausilio all'individuazione e correzione degli errori (*debugger*).

Molti sistemi offrono il profilo temporale di un programma, che indica la quantità di tempo impiegata dal programma per l'esecuzione in una locazione o in un gruppo di locazioni particolare. La creazione di un profilo temporale richiede una funzione specifica, detta funzione di tracciamento, o segnali d'interruzione regolari del temporizzatore. Ogni volta che si presenta un segnale d'interruzione del temporizzatore si registra il valore del contatore di programma. Se i segnali d'interruzione del temporizzatore sono abbastanza frequenti, è possibile ottenere un quadro statistico del tempo impiegato per le diverse parti del programma.

Il controllo dei processi presenta così tanti aspetti e varianti, che per chiarire questi concetti conviene ricorrere ad alcuni esempi. Il sistema operativo MS-DOS è un sistema che dispone di un interprete di comandi, attivato all'avviamento del calcolatore, e che esegue un solo programma alla volta (Figura 3.3.a), impiegando un metodo semplice e senza creare alcun nuovo processo; carica il programma nella memoria, riscrivendo anche la maggior parte della memoria che esso stesso occupa, in modo da lasciare al programma quanta più memoria è possibile (Figura 3.3.b); quindi imposta il contatore di programma alla prima istruzione del programma da eseguire. A questo punto si esegue il programma e si possono verificare due situazioni: un errore causa un segnale di eccezione, oppure il programma esegue una chiamata del sistema per terminare la propria esecuzione. In entrambi i casi si registra il codice d'errore nella memoria di sistema per un eventuale uso successivo, quindi quella piccola parte dell'interprete che non era stata sovrascritta riprende l'esecuzione e il suo primo compito consiste nel ricaricare dal disco la parte rimanente dell'interprete stesso. Eseguito questo compito, quest'ultimo mette a disposizione dell'utente, o del programma successivo, il codice d'errore registrato.

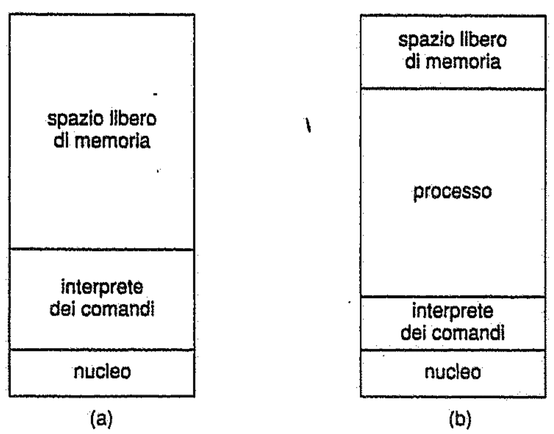


Figura 3.3 Esecuzione nell'MS-DOS. a) All'avviamento del sistema. b) Durante l'esecuzione di un programma.

Benché il sistema operativo MS-DOS non possa eseguire più compiti contemporaneamente, fornisce un metodo per una sia pur limitata esecuzione concorrente. Un programma TSR è un programma che si attiva 'intercettando' un segnale d'interruzione, quindi termina la propria esecuzione con la chiamata del sistema `terminate and stay resident`. Può ad esempio intercettare le interruzioni del temporizzatore mettendo l'indirizzo di una delle proprie procedure nella lista delle procedure di servizio delle interruzioni da eseguire quando si attiva il temporizzatore del sistema; in questo modo si esegue la procedura TSR diverse volte al secondo. La chiamata del sistema `terminate and stay resident` fa sì che l'MS-DOS riservi l'area della memoria occupata dal TSR affinché non sia sovrascritto quando si carica nuovamente l'interprete dei comandi.

Nel FreeBSD quando un utente inizia una sessione di lavoro, il sistema esegue un interprete dei comandi (in quest'ambiente chiamato *shell*) scelto dall'utente. Quest'interprete è simile a quello dell'MS-DOS nell'accettare i comandi e nell'eseguire programmi richiesti dall'utente. Comunque, poiché il FreeBSD è un sistema a partizione del tempo, può eseguire più programmi contemporaneamente, l'interprete dei comandi può continuare l'esecuzione mentre si esegue un altro programma (Figura 3.4).

Per avviare un nuovo processo, l'interprete dei comandi esegue la chiamata del sistema `fork`; si carica il programma selezionato nella memoria tramite la chiamata del sistema `exec` e infine si esegue il programma. Secondo come il comando è stato impartito, l'interprete dei comandi attende che il processo finisca, oppure esegue il processo 'in sottofondo' (*background*). In quest'ultimo caso l'interprete dei comandi richiede immediatamente un altro comando. Se un processo è eseguito in sottofondo, non può ricevere dati direttamente dalla tastiera, giacché anche l'interprete dei comandi sta usando tale risorsa. L'eventuale operazione di I/O è dunque eseguita tramite un file o tramite un'interfaccia basata su finestre e menu. Nel frattempo l'utente è libero di chiedere all'inter-

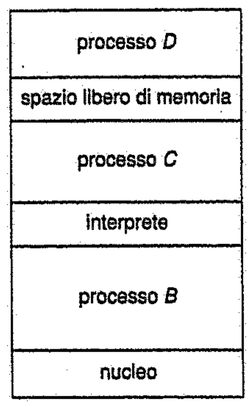


Figura 3.4 Esecuzione di più programmi nel sistema operativo UNIX.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

prete dei comandi di eseguire altri programmi, di controllare lo svolgimento del processo in esecuzione, di modificare la priorità di quel programma e così via. Terminato il proprio compito, il processo esegue una chiamata del sistema `exit` per terminare la propria esecuzione, riportando al processo chiamante un codice di stato 0 oppure un codice d'errore diverso da 0. Questo codice di stato (o d'errore) rimane disponibile per l'interprete dei comandi o per altri programmi. I processi sono trattati nel Capitolo 4.

3.3.2 Gestione dei file

Il file system è esaminato più in profomdità nei Capitoli 11 e 12, tuttavia si possono già identificare parecchie chiamate del sistema riguardanti i file.

Innanzi tutto è necessario poter creare (`create`) e cancellare (`delete`) i file. Ogni chiamata del sistema richiede il nome del file e probabilmente anche altri attributi. Una volta creato il file è necessario aprirlo (`open`) e usarlo. Si può anche leggere (`read`), scrivere (`write`) o riposizionare (`reposition`), ad esempio riavvolgendo e saltando alla fine del file. Si deve infine poter chiudere (`close`) un file per indicare che non è più in uso.

Queste stesse operazioni possono essere necessarie anche per le directory, nel caso in cui il file system sia strutturato in directory. È inoltre necessario poter determinare i valori degli attributi dei file o delle directory ed eventualmente modificarli. Tra gli attributi dei file figurano: il suo nome, il tipo, i codici di protezione, le informazioni di contabilizzazione, e così via. Per questa funzione sono richieste due chiamate del sistema, e precisamente `get file attribute` e `set file attribute`. Alcuni sistemi operativi forniscono molte più chiamate del sistema.

3.3.3 Gestione dei dispositivi

Durante l'esecuzione di un programma può essere necessario richiedere ulteriori risorse, che possono essere ulteriore spazio di memoria, unità a nastro, accesso a file, e così via. Se le risorse sono disponibili, si possono concedere, e il controllo può ritornare al programma utente, altrimenti il programma deve attendere fino a che non siano disponibili risorse sufficienti.

Poiché i file si possono immaginare come dispositivi astratti o virtuali, molte chiamate del sistema necessarie per i file si usano anche per i dispositivi; tuttavia, se un sistema ha più utenti, occorre prima richiedere (`request`) il dispositivo, per assicurarsene l'esclusiva. Terminato l'uso del dispositivo, occorre rilasciarlo (`release`). Queste funzioni sono simili alle chiamate del sistema `open` e `close` dei file.

Una volta richiesto e assegnato il dispositivo, è possibile leggervi (`read`), scrivervi (`write`) ed eventualmente procedere a un riposizionamento (`reposition`), esattamente come nei file ordinari; la somiglianza tra file e dispositivi di I/O è infatti tale che molti sistemi operativi, tra cui lo UNIX e l'MS-DOS, li fondono in un'unica **struttura file/dispositivo**. In questo caso i dispositivi di I/O sono identificati da nomi di file speciali.

3.3.4 Gestione delle informazioni

Molte chiamate del sistema hanno semplicemente lo scopo di trasferire le informazioni tra il programma utente e il sistema operativo. La maggior parte dei sistemi, ad esempio, ha una chiamata del sistema per ottenere l'ora (`time`) e la data attuali (`date`). Altre chiamate del sistema possono ottenere informazioni sul sistema, come il numero degli utenti collegati, il numero della versione del sistema operativo, la quantità di memoria disponibile o di spazio nei dischi, e così via.

Il sistema operativo contiene inoltre informazioni su tutti i propri processi; a queste informazioni si può accedere tramite alcune chiamate del sistema. In genere esistono anche chiamate del sistema per modificare le informazioni sui processi (`get process attributes` e `set process attributes`). Nel Paragrafo 4.1.3 si spiega quali sono tali informazioni.

3.3.5 Comunicazione

Esistono due modelli molto diffusi di comunicazione. Nel **modello a scambio di messaggi** le informazioni si scambiano per mezzo di una funzione di comunicazione tra processi, messa a disposizione dal sistema operativo. Prima di effettuare una comunicazione occorre aprire un collegamento. Il nome dell'altro comunicante deve essere noto, sia che si tratti di un altro processo nello stesso calcolatore, sia che si tratti di un processo in un altro calcolatore collegato attraverso una rete di comunicazione. Tutti i calcolatori di una rete hanno un **nome di macchina** (*host name*), ad esempio un nome IP, con il quale sono individuati. Analogamente, ogni processo ha un **nome di processo**, che si converte in un identificatore equivalente che il sistema operativo impiega per farvi riferimento. La conversione nell'identificatore si compie con le chiamate del sistema `get hostid` e `get processid`. Questi identificatori sono quindi passati alle chiamate del sistema d'uso generale `open` e `close` messe a disposizione dal file system, oppure, secondo il modello di comunicazione del sistema, alle chiamate del sistema specifiche `open connection` e `close connection`. Generalmente il processo ricevente deve acconsentire alla comunicazione con una chiamata del sistema `accept connection`. Nella maggior parte dei casi i processi che gestiscono la comunicazione sono **demoni** specifici, cioè programmi di sistema realizzati esplicitamente per questo scopo. Questi programmi eseguono una chiamata del sistema `wait for connection` e sono chiamati in causa quando si stabilisce un collegamento. L'origine della comunicazione, nota come *client*, e il demone ricevente, noto come *server*, possono quindi scambiarsi i messaggi per mezzo delle chiamate del sistema `read message` e `write message`; la chiamata del sistema `close connection` pone fine alla comunicazione.

Nel **modello a memoria condivisa**, invece, per accedere alle aree di memoria possedute da altri processi, i processi usano chiamate del sistema `map memory`. Occorre ricordare che, normalmente, il sistema operativo tenta di impedire a un processo l'accesso alla memoria di un altro processo. Il modello a memoria condivisa richiede che più processi concordino nel superare tale limite; a questo punto tali processi possono scambiarsi le informazioni leggendo e scrivendo i dati nelle aree di memoria condivise. La forma e la posizione dei dati sono determinate esclusivamente da questi processi e non sono

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sotto il controllo del sistema operativo. I processi sono dunque responsabili del rispetto della condizione di non scrivere contemporaneamente nella stessa posizione. Questi meccanismi sono discussi nel Capitolo 7. In questo testo, precisamente nel Capitolo 5, si tratta anche una variante del modello di processo, detto *thread*, che prevede che la condivisione della memoria sia predefinita.

Entrambi i metodi sono assai comuni e in certi sistemi operativi sono presenti contemporaneamente. Lo scambio di messaggi è utile soprattutto quando è necessario trasferire una piccola quantità di dati, poiché, in questo caso, non sussiste la necessità di evitare conflitti; è inoltre più facile da realizzare rispetto alla condivisione della memoria per la comunicazione tra calcolatori diversi.

La condivisione della memoria permette la massima velocità e convenienza nelle comunicazioni, poiché queste ultime, se avvengono all'interno del calcolatore, si possono svolgere alla velocità della memoria. Sussistono, in ogni caso, problemi per quel che riguarda la protezione e la sincronizzazione. I due modelli di comunicazione sono messi a confronto nella Figura 3.5.

3.4 Programmi di sistema

Un'altra caratteristica importante di un sistema moderno è quella che riguarda la serie di programmi di sistema. Facendo riferimento alla Figura 1.1, nella quale s'illustra la gerarchia logica di un calcolatore, si può notare che il livello più basso è occupato dai dispositivi fisici. Seguono, nell'ordine, il sistema operativo, i programmi di sistema e i programmi d'applicazione. I programmi di sistema offrono un ambiente più conveniente per lo sviluppo e l'esecuzione dei programmi; alcuni sono semplici interfacce per le chiamate del sistema, altri sono considerevolmente più complessi; in generale si possono classificare nelle seguenti categorie:

- **Gestione dei file.** Questi programmi creano, cancellano, copiano, ridenominano, stampano, elencano e in genere compiono operazioni sui file e le directory.
- **Informazioni di stato.** Alcuni programmi richiedono semplicemente al sistema di indicare la data, l'ora, la quantità di memoria disponibile o lo spazio nei dischi, il numero degli utenti o le informazioni di stato. Le informazioni fornite sono poi formattate e scritte sullo schermo di un terminale, in un altro dispositivo o in un file.
- **Modifica dei file.** Possono essere disponibili diversi editor per creare e modificare il contenuto di file memorizzati in dischi o nastri.
- **Ambienti d'ausilio alla programmazione.** I compilatori, gli assemblatori e gli interpreti dei linguaggi di programmazione comuni, come il Linguaggio C, C++, PASCAL, Java, Visual Basic, Perl, sono talvolta forniti all'utente insieme col sistema operativo, oppure venduti separatamente.

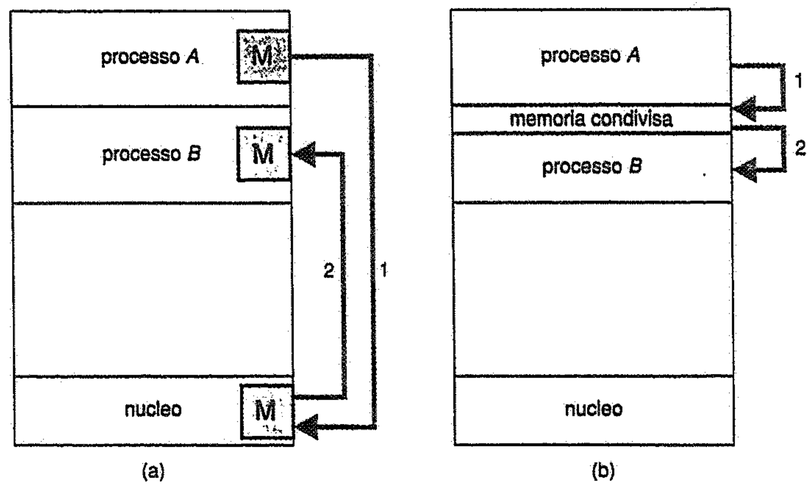


Figura 3.5 Modelli di comunicazione. a) Scambio di messaggi. b) Memoria condivisa.

- **Caricamento ed esecuzione dei programmi.** Una volta assemblato o compilato, per essere eseguito, un programma deve essere caricato nella memoria. Il sistema può mettere a disposizione caricatori assoluti, caricatori rilocabili, editor dei collegamenti (*linkage editor*) e caricatori di sezioni sovrapponibili di programmi (*overlay*). Sono necessari anche i sistemi d'ausilio all'individuazione e correzione degli errori (*debugger*) per i linguaggi d'alto livello o per il linguaggio di macchina.
- **Comunicazioni.** Questi programmi offrono i meccanismi con cui si possono creare collegamenti virtuali tra processi, utenti e calcolatori diversi. Permettono agli utenti d'inviare messaggi agli schermi d'altri utenti, di consultare il Web, d'inviare messaggi di posta elettronica, di accedere a calcolatori remoti, di trasferire file da un calcolatore a un altro.

Con la maggior parte dei sistemi operativi sono forniti programmi che risolvono problemi comuni, o che eseguono operazioni comuni. Tali programmi, noti come **programmi d'applicazione**, comprendono programmi di consultazione del Web, elaboratori di testi, fogli di calcolo, sistemi di basi di dati, compilatori, programmi di disegno, programmi per analisi statistiche e giochi.

Probabilmente il programma di sistema più importante è l'interprete dei comandi, la cui funzione principale consiste nel ricevere i comandi impartiti dagli utenti ed eseguirli.

Molti comandi impartiti a questo livello si usano per la gestione dei file, vale a dire per creare, cancellare, elencare, stampare, copiare, far eseguire e così via. Questi comandi si possono realizzare in due modi; in uno di questi lo stesso interprete dei comandi contiene il codice per l'esecuzione del comando. Il comando di cancellazione di un file,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ad esempio, può causare un salto dell'interprete dei comandi a una sezione del suo stesso codice che imposta i parametri e invoca le idonee chiamate del sistema; in questo caso il numero dei comandi che si possono impartire determina le dimensioni dell'interprete dei comandi, ciò poiché ogni comando richiede il proprio segmento di codice.

L'altro metodo, usato ad esempio nel sistema operativo UNIX, realizza la maggior parte dei comandi per mezzo di programmi di sistema speciali; in questo caso l'interprete dei comandi non 'capisce' il comando, ma ne impiega semplicemente il nome per identificare un file da caricare nella memoria per essere eseguito. Quindi, il comando dello UNIX

```
rm g
```

cerca un file chiamato `rm`, lo carica nella memoria e lo esegue con il parametro *g*. La funzione corrispondente al comando `rm` è interamente definita dal codice del file `rm`. In questo modo i programmatori possono aggiungere nuovi comandi al sistema, semplicemente creando nuovi file con il nome appropriato. Il programma dell'interprete dei comandi, che può quindi essere abbastanza piccolo, non necessita di alcuna modifica quando s'introducono nuovi comandi.

Questo metodo crea alcuni problemi nella progettazione di un interprete dei comandi. Innanzi tutto, siccome il codice d'esecuzione di un comando è un programma di sistema separato, il sistema operativo deve disporre di un meccanismo che permetta il passaggio dei parametri dall'interprete dei comandi al programma di sistema. Questo compito risulta spesso difficile, poiché il programma di sistema e l'interprete dei comandi possono non trovarsi contemporaneamente nella memoria e la lista dei parametri può essere lunga; quindi caricare ed eseguire un programma può essere più lento del semplice salto a un'altra sezione di codice all'interno del programma corrente.

Un altro problema è dovuto al fatto che l'interpretazione dei parametri è lasciata al programmatore del programma di sistema: i parametri potrebbero essere forniti in modo incoerente a programmi apparentemente simili che però sono stati scritti in momenti diversi o da programmatori diversi.

Per la maggior parte degli utenti, l'interfaccia col sistema operativo è dunque definita dai programmi di sistema piuttosto che dalle effettive chiamate del sistema. Si consideri l'uso di un PC col sistema operativo Microsoft Windows; per le proprie attività l'utente può usare l'interprete dei comandi a riga di comando dell'MS-DOS o l'interfaccia grafica con finestre e menu: entrambe fanno uso delle stesse chiamate del sistema ma si presentano e agiscono in maniera differente; di conseguenza la visione dell'utente può essere molto distante dalla reale struttura del sistema. Perciò, la progettazione di un'interfaccia d'utente utile e d'uso agevole e intuitivo non è una funzione diretta del sistema operativo. In questo testo l'attenzione si concentra sul problema fondamentale di fornire servizi adeguati ai programmi utenti; dal punto di vista del sistema operativo non si fa alcuna distinzione tra programmi utenti e programmi di sistema.

3.5 Struttura del sistema

Affinché possa funzionare correttamente ed essere facilmente modificato, un sistema vasto e complesso come un sistema operativo moderno deve essere progettato con estrema attenzione. Anziché progettare un sistema monolitico, un orientamento diffuso prevede la sua suddivisione in piccoli componenti; ciascuno deve essere un modulo ben definito del sistema, con interfacce e funzioni definite con precisione. Nel Paragrafo 3.1 sono trattati brevemente i componenti comuni ai sistemi operativi, in questo paragrafo si discute come questi componenti sono interconnessi e fusi in un nucleo.

3.5.1 Struttura semplice

Molti sistemi commerciali non hanno una struttura ben definita; spesso sono nati come sistemi piccoli, semplici e limitati, e solo in un secondo tempo si sono accresciuti superando il loro scopo originale. Un sistema di questo tipo è l'MS-DOS, originariamente progettato e realizzato da persone che non avrebbero mai immaginato una simile diffusione. Non fu suddiviso attentamente in moduli poiché, a causa dei limiti dell'architettura su cui era eseguito, lo scopo prioritario era fornire la massima funzionalità nel minimo spazio. La Figura 3.6 riporta la sua struttura.

Lo UNIX è un altro esempio di strutturazione che inizialmente era limitata dalle funzioni dell'architettura sottostante. È formato da due parti: il nucleo e i programmi di sistema. Il nucleo è a sua volta composto da una serie di interfacce e driver di dispositivi, che sono stati aggiunti ed estesi con l'evolversi dello stesso UNIX. Il sistema operativo UNIX

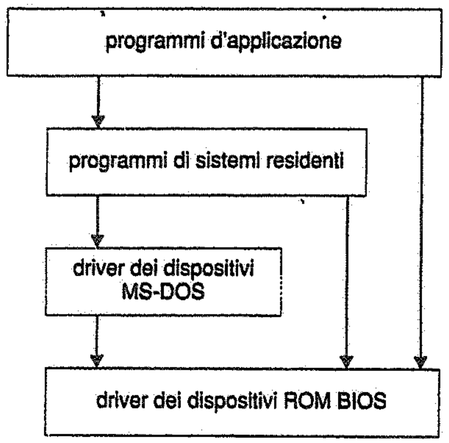


Figura 3.6 Struttura degli strati dell'MS-DOS.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

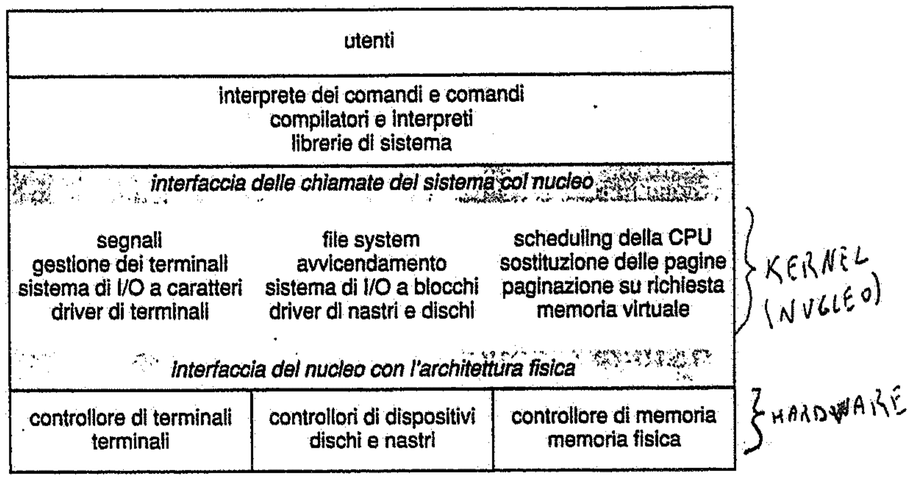


Figura 3.7 Struttura del sistema UNIX.

tradizionale si può considerare stratificato (Figura 3.7); tutto quel che si trova sotto l'interfaccia delle chiamate del sistema e sopra i dispositivi fisici è il nucleo. Per mezzo di una serie di chiamate del sistema, il nucleo offre il file system, lo scheduling della CPU, la gestione della memoria e altre funzioni riguardanti il sistema operativo. In un solo livello occorre combinare un'enorme quantità di funzioni. Ciò rende lo UNIX difficile da migliorare poiché le modifiche di una sua parte possono avere effetti negativi in altre sue parti.

Le chiamate del sistema definiscono l'API per lo UNIX. I programmi di sistema disponibili definiscono, invece, l'interfaccia d'utente. Le interfacce per il programmatore e per l'utente definiscono il contesto che il nucleo deve realizzare.

Le nuove versioni dello UNIX sono state progettate per impiegare architetture più progredite. La disponibilità di adeguate caratteristiche dell'architettura permette di organizzare i sistemi operativi in moduli più piccoli e più appropriati rispetto a quelli dei sistemi MS-DOS e UNIX originari. Ciò consente al sistema operativo di mantenere un controllo maggiore sul calcolatore e sulle applicazioni che fanno uso di tale calcolatore, e offre una maggiore libertà a chi desideri modificare il funzionamento interno del sistema e creare sistemi operativi modulari. Seguendo un metodo *top-down* s'individuano le funzioni e le caratteristiche generali che si definiscono come componenti. Questa separazione permette ai programmatori di nascondere le informazioni; sono perciò liberi di codificare come desiderano le procedure di basso livello, purché le interfacce esterne delle procedure restino immutate e le stesse procedure, anche se codificate in modo diverso, eseguano gli stessi compiti.

3.5.2 Metodo stratificato

L'organizzazione in moduli di un sistema si ottiene in vari modi; con il **metodo stratificato** si suddivide il sistema operativo in un certo numero di strati (o livelli), ciascuno costruito sopra gli strati inferiori. Lo strato più basso (strato 0) è lo strato fisico; lo strato più alto (strato *n*) è l'interfaccia d'utente. HARDWARE

Lo strato di un sistema operativo è una realizzazione di un oggetto astratto, che incapsula i dati e le operazioni che trattano tali dati. La Figura 3.8 illustra un tipico strato di sistema operativo; lo strato *m* ad esempio è composto da un insieme di strutture di dati e da un insieme di procedure che si possono chiamare dagli strati di livello superiore e, a sua volta, può richiedere l'esecuzione di operazioni sugli strati di livello inferiore.

Il vantaggio principale offerto da questo metodo è la **modularità**. Gli strati sono composti in modo che ciascuno di essi usi solo funzioni (o operazioni) e servizi che appartengono a strati di livello inferiore. Questo metodo semplifica la messa a punto e la verifica del sistema. Il primo strato si può mettere a punto senza intaccare il resto del sistema, poiché per realizzare le proprie funzioni usa, per definizione, solo lo strato fisico, che si presuppone sia corretto. Passando alla lavorazione del secondo strato si presume, dopo la messa a punto, la correttezza del primo. Il procedimento si ripete per ogni strato. Se durante la messa a punto di un particolare strato si riscontra un errore, questo deve essere in quello strato, poiché gli strati inferiori sono già stati corretti; quindi la suddivisione in strati semplifica la progettazione e la realizzazione di un sistema.

Ogni strato si realizza impiegando unicamente le operazioni messe a disposizione dagli strati inferiori; considerando soltanto le azioni che compiono, senza entrare nel merito di come queste sono realizzate. Di conseguenza ogni strato nasconde a quelli superiori l'esistenza di determinate strutture di dati, operazioni ed elementi fisici.

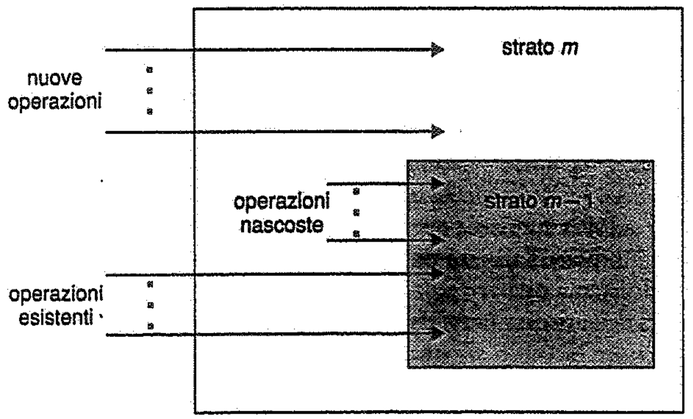


Figura 3.8 Uno strato di sistema operativo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Poiché uno strato può usare solo strati che si trovano a un livello inferiore, la principale difficoltà connessa al metodo stratificato riguarda l'accurata definizione degli strati. Ad esempio, i driver delle unità a disco impiegati dalle procedure di gestione della memoria virtuale devono essere in un livello inferiore rispetto alle procedure di gestione della memoria, poiché la gestione della memoria richiede la capacità d'uso dello spazio dei dischi.

Altri requisiti possono non essere così ovvi. Normalmente il driver della memoria ausiliaria (*backing store*) dovrebbe trovarsi sopra lo scheduler della CPU, poiché può accadere che il driver debba attendere un'istruzione di I/O, e in questo periodo la CPU si può sottoporre a scheduling. Tuttavia, in un grande sistema, lo scheduler della CPU può avere più informazioni su tutti i processi attivi di quante se ne possano contenere nella memoria; perciò è probabile che queste informazioni si debbano caricare e scaricare dalla memoria, quindi il driver della memoria ausiliaria dovrebbe trovarsi sotto lo scheduler della CPU.

Un ulteriore problema che si pone con la struttura stratificata è che essa tende a essere meno efficiente delle altre; ad esempio, un programma utente per eseguire un'operazione di I/O invoca una chiamata del sistema che è intercettata dallo strato di I/O che, a sua volta, esegue una chiamata allo strato di gestione della memoria, che a sua volta chiama lo strato di scheduling della CPU e che quindi è passata all'opportuno dispositivo di I/O. In ciascuno strato i parametri possono essere modificati, può essere necessario il passaggio di dati, e così via; ciascuno strato aggiunge un carico alla chiamata del sistema. Ne risulta una chiamata del sistema che richiede molto più tempo di una chiamata del sistema corrispondente in un sistema non stratificato.

Negli ultimi anni questi limiti hanno causato una piccola battuta d'arresto allo sviluppo della stratificazione. Attualmente si progettano sistemi basati su un numero inferiore di strati con più funzioni, che offrono la maggior parte dei vantaggi del codice modulare, evitando i difficili problemi connessi alla definizione e all'interazione degli strati. L'OS/2, un discendente dell'MS-DOS, è un sistema che, grazie alla partizione del tempo d'elaborazione della CPU, consente l'esecuzione di più processi in modo concorrente (*multitasking*), che impiega il duplice modo di funzionamento, e offre altre nuove funzioni. Considerando la maggiore complessità e potenza dell'architettura per cui è stato progettato, l'OS/2 è stato realizzato in modo più stratificato. Confrontando la struttura dell'MS-DOS (Figura 3.6) con quella illustrata nella Figura 3.9, sia dal punto di vista della progettazione del sistema sia da quello della realizzazione, l'OS/2 è più vantaggioso. Ad esempio, non si ammette l'accesso diretto da parte degli utenti alle funzioni di basso livello, dotando il sistema operativo di un maggiore controllo dei dispositivi e di una più vasta conoscenza delle risorse usate da ciascun programma utente.

Come ulteriore esempio si consideri l'evoluzione del sistema operativo Windows NT: le prime versioni erano fortemente stratificate, e ciò produsse basse prestazioni rispetto, ad esempio, a quelle offerte dal sistema Windows 95. Questi problemi sono stati parzialmente risolti spostando alcuni strati dallo spazio d'utente allo spazio del nucleo e integrandoli più strettamente.

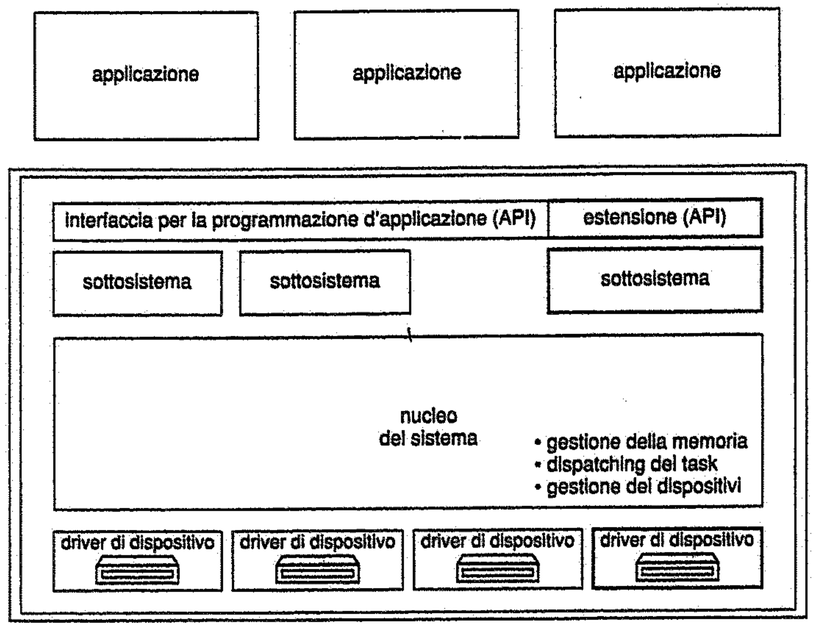


Figura 3.9 Struttura stratificata dell'OS/2.

Quasi tutte le funzionalità del kernel sono spostate nel micronucleo

3.5.3 Micronucleo

A mano a mano che il sistema operativo UNIX è stato esteso, il nucleo è cresciuto notevolmente ed è diventato sempre più difficile da gestire. Verso la metà degli anni Ottanta, un gruppo di ricercatori della Carnegie Mellon University progettò e realizzò un sistema operativo, Mach, col nucleo strutturato in moduli secondo il cosiddetto orientamento a micronucleo.

Seguendo questo orientamento si progetta il sistema operativo rimuovendo dal nucleo tutti i componenti non essenziali, realizzandoli come programmi del livello d'utente e di sistema. Ne risulta un nucleo di dimensioni assai inferiori. Non c'è un'opinione comune su quali servizi debbano rimanere nel nucleo e quali si debbano realizzare nello spazio d'utente. Tuttavia, in generale, un micronucleo offre i servizi minimi di gestione dei processi, della memoria e di comunicazione.

Lo scopo principale del micronucleo è fornire funzioni di comunicazione tra i programmi client e i vari servizi, anch'essi in esecuzione nello spazio d'utente. La comunicazione si realizza secondo il modello a scambio di messaggi, descritto nel Paragrafo 3.3.5. Per accedere a un file, ad esempio, un programma client deve interagire con il file server; ciò non avviene mai in modo diretto, ma tramite uno scambio di messaggi con il micronucleo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Uno dei vantaggi del micronucleo è la facilità di estensione del sistema operativo: i nuovi servizi si aggiungono allo spazio d'utente e non comportano modifiche al nucleo. Poiché è ridotto all'essenziale, se il nucleo deve essere modificato, i cambiamenti da fare sono ben circoscritti, e il sistema operativo risultante è più semplice da adattare alle diverse architetture. Inoltre offre maggiori garanzie di sicurezza e affidabilità poiché i servizi si eseguono in gran parte come processi utenti, e non come processi del nucleo: se un servizio è compromesso, il resto del sistema operativo rimane intatto.

L'orientamento a micronucleo è stato adottato per molti sistemi operativi moderni: il sistema Tru64 UNIX (in origine Digital UNIX), ad esempio, offre all'utente un'interfaccia di tipo UNIX, ma il suo nucleo è il Mach (il nucleo traduce le chiamate del sistema dello UNIX in messaggi ai corrispondenti servizi del livello d'utente); anche il sistema operativo Apple MacOS X Server è basato sul nucleo Mach.

Il QNX è un sistema operativo per elaborazioni in tempo reale, anch'esso basato su un micronucleo che offre i servizi per lo scambio dei messaggi, lo scheduling dei processi, la comunicazione di rete a basso livello e la gestione dei segnali d'interruzione dei dispositivi; tutti gli altri servizi sono realizzati da processi ordinari che si eseguono fuori del nucleo nel modo d'utente.

Il sistema operativo Windows NT adotta una struttura ibrida. Si è accennato al fatto che una parte della sua architettura è strutturata a strati. È stato progettato per eseguire vari tipi di applicazioni, incluse le applicazioni Windows (Win32), quelle per il sistema operativo OS/2, e quelle conformi allo standard POSIX. Per ciascun tipo d'applicazione, fornisce un server che si esegue nello spazio d'utente. Anche i programmi client, per ciascun tipo d'applicazione, si eseguono nello spazio utente. Il nucleo coordina lo scambio di messaggi tra applicazioni client e server. La Figura 3.10 illustra la struttura client-server del sistema Windows NT.

3.6 Macchine virtuali

Concettualmente, un sistema di calcolo è costituito da strati: la sua architettura fisica è il livello più basso; il nucleo, operante al livello successivo, sfrutta le istruzioni offerte dall'architettura fisica (istruzioni di macchina) per fornire un insieme di chiamate del sistema che sono usate dagli strati superiori. Quindi, i programmi di sistema che si trovano sopra il nucleo possono usare, indifferentemente, le chiamate del sistema e le istruzioni di macchina; sebbene vi si acceda in modo diverso, contribuiscono entrambe a fornire funzioni che i programmi di sistema possono usare per creare altre funzioni più complesse, trattando, a loro volta, le istruzioni di macchina e le chiamate del sistema come se fossero entrambe allo stesso livello.

Alcuni sistemi estendono questo schema, permettendo ai programmi d'applicazione di chiamare i programmi di sistema. Anche in questo caso, i programmi d'applicazione possono considerare quel che si trova a un livello gerarchico inferiore come se fosse parte della macchina stessa, sebbene i programmi di sistema si trovino a un livello superiore. Questo metodo stratificato conduce in modo naturale al concetto di **macchina virtuale**. Il sistema operativo VM dell'IBM è il miglior esempio del concetto di macchina virtuale, poiché è stata proprio l'IBM ad avventurarsi per prima in questo settore.

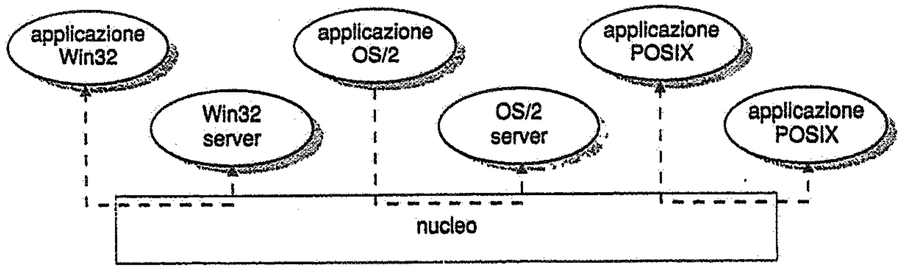


Figura 3.10 Struttura client-server del sistema operativo Windows NT.

Servendosi della partizione del tempo d'uso della CPU (con opportuni algoritmi di scheduling, descritti nel Capitolo 6), e delle tecniche di memoria virtuale, trattate nel Capitolo 10, un sistema operativo può creare l'illusione che un processo disponga della propria CPU con la propria memoria (virtuale). Normalmente il processo dispone di ulteriori caratteristiche, come chiamate del sistema e un file system, che non sono offerte dalla mera architettura fisica. D'altra parte il metodo basato sulle macchine virtuali non offre alcuna funzione aggiuntiva, ma è piuttosto un'interfaccia *identica* all'architettura sottostante. Ciascun processo dispone di una copia (virtuale) del calcolatore sottostante (Figura 3.11).

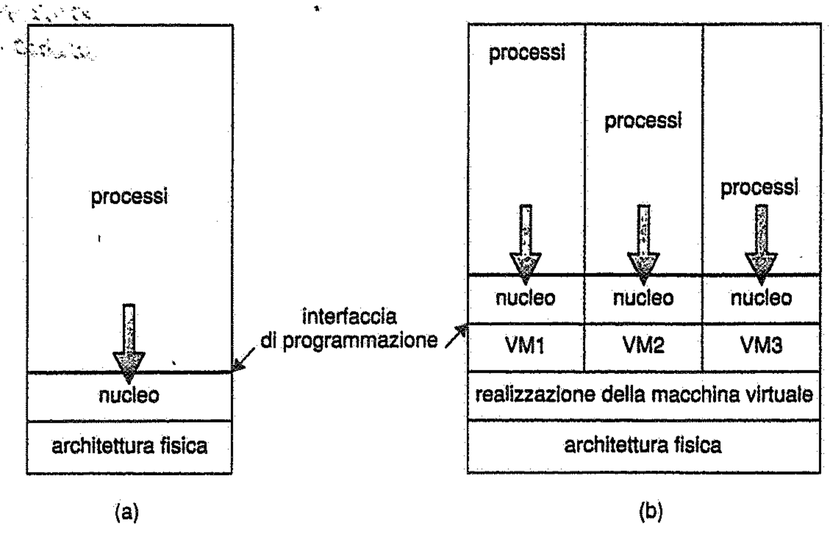


Figura 3.11 Modelli di sistema: a) Semplice; b) Macchina virtuale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il calcolatore fisico condivide le risorse in modo da creare macchine virtuali. La partizione del tempo d'uso della CPU si può usare sia per condividere la CPU sia per dare l'illusione che gli utenti dispongano ciascuno di una propria CPU. La gestione asincrona delle operazioni di I/O e dell'esecuzione di più processi, unita a un file system, consente di creare lettori di schede e stampanti virtuali. Un normale terminale di un sistema a partizione del tempo funziona da console d'operatore della macchina virtuale.

Una notevole difficoltà legata al metodo delle macchine virtuali s'incontra nella gestione dei dischi; si supponga, ad esempio, che la macchina fisica disponga di tre unità a disco, e che si vogliano realizzare sette macchine virtuali. Chiaramente, la macchina non può assegnare un'unità a disco a ciascuna macchina virtuale. Occorre ricordare che, per offrire le funzioni di memoria virtuale, i programmi che realizzano le macchine virtuali richiedono una parte rilevante dello spazio dei dischi. La soluzione è offerta dai dischi virtuali, che sono identici sotto tutti gli aspetti, escluse le dimensioni, a quelli fisici. Nel sistema operativo VM dell'IBM i dischi virtuali si chiamano *minidischi*. Il sistema realizza ogni minidisco riservando nei dischi fisici tante tracce quante ne richiede il minidisco. Naturalmente la somma delle dimensioni di tutti i minidischi deve essere minore della dimensione dello spazio fisico complessivamente disponibile nei dischi.

Ogni utente dispone quindi della propria macchina virtuale, da cui può far eseguire qualsiasi sistema operativo o programma progettato per la macchina fisica sottostante. Con il sistema VM dell'IBM gli utenti usano generalmente il CMS, un sistema operativo interattivo per singolo utente. I programmi che realizzano il sistema di macchine virtuali si occupano della multiprogrammazione di più macchine virtuali su una macchina fisica, senza offrire alcun altro tipo di servizio all'utente. Questo schema offre la possibilità di dividere la progettazione di un sistema interattivo multiutente in due parti più piccole.

3.6.1 Realizzazione

Benché utile, il concetto di macchina virtuale è difficile da realizzare; è difficile ottenere un *esatto* duplicato della macchina sottostante. Si ricordi che la macchina sottostante ha due modi di funzionamento, il modo d'utente e il modo di sistema. I programmi che realizzano il sistema di macchine virtuali si possono eseguire nel modo di sistema, poiché costituiscono il sistema operativo, mentre ciascuna macchina virtuale può funzionare solo nel modo d'utente. Quindi, proprio come la macchina fisica, anche quella virtuale deve avere due modi; di conseguenza si deve avere un modo d'utente virtuale e un modo di sistema virtuale, entrambi operanti su un modo d'utente fisico. Le azioni che causano un trasferimento dal modo d'utente al modo di sistema su una macchina reale, come una chiamata del sistema o un tentativo d'esecuzione di un'istruzione privilegiata, in una macchina virtuale devono causare un analogo trasferimento dal modo d'utente virtuale al modo di sistema virtuale.

Generalmente questo trasferimento si può eseguire in modo abbastanza semplice. Se, ad esempio, un programma in esecuzione su una macchina virtuale nel modo d'utente virtuale esegue una chiamata del sistema, si passa al modo di sistema della macchina virtuale all'interno della macchina reale. Quando il sistema della macchina virtuale acquisisce il controllo, può modificare il contenuto dei registri e il contatore di programma della macchina virtuale, in modo da simulare l'effetto della chiamata del sistema; quindi può riavviare la macchina virtuale, che si trova nel modo di sistema virtuale. Se, ad esempio, la macchina virtuale tenta di leggere dal proprio lettore di schede virtuale, esegue un'istruzione di I/O privilegiata, e poiché la macchina virtuale sta lavorando nel modo d'utente fisico, questa istruzione causa l'emissione di un segnale di eccezione rilevato dal sistema della macchina virtuale. Quest'ultimo deve quindi simulare l'effetto dell'istruzione di I/O: dapprima cerca il file che simula il lettore di schede, quindi converte l'operazione di lettura (read) dal lettore di schede virtuale in una operazione di lettura (read) dal file, e trasferisce il contenuto della successiva 'scheda virtuale' nella memoria virtuale della macchina virtuale; infine, riavvia la macchina virtuale. Lo stato della macchina virtuale risulta così modificato, esattamente come se l'istruzione di I/O fosse stata eseguita con un lettore di schede reale su una macchina reale operante nel modo di sistema reale.

La differenza più rilevante consiste, naturalmente, nel tempo. Mentre l'I/O reale potrebbe richiedere 100 millisecondi, quello virtuale potrebbe richiedere un tempo inferiore (poiché le operazioni di I/O avvengono su file contenuti in dischi, che simulano i dispositivi periferici di I/O), o superiore, poiché è interpretato. Inoltre la condivisione della CPU, rallenta ulteriormente in modo imprevedibile le macchine virtuali. In un caso limite, per offrire una vera macchina virtuale, può essere necessario simulare tutte le istruzioni. Il sistema VM funziona su calcolatori dell'IBM, poiché le normali istruzioni per le macchine virtuali possono essere eseguite direttamente dalla macchina reale. Soltanto le istruzioni privilegiate, richieste soprattutto per le operazioni di I/O, devono essere simulate e quindi eseguite più lentamente.

3.6.2 Vantaggi

L'uso delle macchine virtuali ha due vantaggi principali: proteggendo completamente le risorse del sistema, fornisce un efficace livello di sicurezza; inoltre permette lo sviluppo del sistema senza turbarne l'ordinario funzionamento. Ogni macchina virtuale è completamente isolata dalle altre, quindi non esistono problemi di sicurezza, giacché tutte le risorse del sistema sono completamente protette. Le applicazioni sospette prelevate dalla rete Internet, ad esempio, si possono far eseguire da una macchina virtuale separata. Uno svantaggio di questo tipo d'ambiente è che non c'è una condivisione *diretta* delle risorse. Per ottenere tale condivisione si sono seguiti due metodi. Il primo prevede la possibilità di condividere un minidisco, progettato rifacendosi a un disco fisico condiviso, ma simulato; in questo modo si possono condividere i file. Il secondo metodo prevede la definizione di una rete di macchine virtuali, ciascuna delle quali può inviare informazioni tramite la rete di comunicazione virtuale. Anche in questo caso la rete segue il modello delle reti di comunicazione fisiche, ma è simulata.

Un sistema di macchine virtuali di questo tipo è un perfetto mezzo di ricerca e sviluppo dei sistemi operativi. Normalmente è molto difficile modificare un sistema operativo, trattandosi di programmi vasti e complessi, una modifica in un suo punto può causare oscuri errori di programmazione in altri punti. I poteri di cui dispone il sistema

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

operativo rendono questa situazione particolarmente pericolosa: poiché il sistema operativo lavora nel modo di sistema, una modifica errata a un puntatore potrebbe, ad esempio, rendere inutilizzabile l'intero file system. Quindi, è necessario controllare con estrema attenzione tutte le modifiche fatte al sistema operativo.

L'esecuzione del sistema operativo, inoltre, riguarda tutta la macchina. Quindi, mentre si apportano e si controllano le modifiche, il sistema attuale deve essere fermato e tenuto fuori servizio. In questo intervallo di tempo, che comunemente si chiama **tempo di sviluppo del sistema**, gli utenti ordinari non possono usare il sistema; per questo motivo il tempo di sviluppo del sistema è spesso programmato per le ore notturne o per i giorni festivi, quando il carico del sistema è basso.

Un sistema di macchine virtuali consente di eliminare una gran parte di questi problemi. I programmatori di sistemi dispongono di una macchina virtuale sulla quale lavorare allo sviluppo del sistema. Raramente occorre interrompere il normale funzionamento del sistema. Nonostante questi vantaggi, negli ultimi tempi non si sono fatti molti progressi in questa tecnica.

Le macchine virtuali stanno aumentando la loro diffusione come mezzo per risolvere i problemi di compatibilità dei sistemi. Ci sono, ad esempio, migliaia di programmi disponibili per il sistema operativo Microsoft Windows in sistemi basati su CPU Intel; società che producono calcolatori, come la Sun Microsystems, impiegano altre CPU più veloci, ma vorrebbero offrire ai propri clienti la possibilità di usare i programmi per il sistema Microsoft Windows. La soluzione di tale problema consiste nel creare una macchina virtuale Intel sopra la CPU in esame: quando in questo ambiente si esegue un programma per il Microsoft Windows, la macchina virtuale traduce le istruzioni della CPU Intel nella serie di istruzioni della macchina sottostante; lo stesso Microsoft Windows si può eseguire in tale macchina virtuale, quindi il programma può eseguire, come al solito, le proprie chiamate del sistema. Il programma appare, quindi, come se fosse eseguito in un sistema basato su CPU Intel anche se, in realtà, è eseguito da una CPU assai diversa. Se la CPU è abbastanza veloce, il programma scritto per il sistema operativo Microsoft Windows viene eseguito abbastanza velocemente anche se ciascuna istruzione, per essere eseguita, deve essere tradotta in più istruzioni della macchina sottostante. Allo stesso modo, gli Apple Macintosh basati su architettura PowerPC sono dotati di una macchina virtuale della CPU Motorola 68000 che consente l'esecuzione dei programmi scritti per i vecchi Macintosh. Sfortunatamente, a una maggiore complessità della macchina da emulare corrisponde una maggiore difficoltà di realizzazione di un'accurata macchina virtuale, e una maggiore lentezza nell'esecuzione.

Un esempio più recente, è offerto dal sistema operativo LINUX. Esistono macchine virtuali che permettono alle applicazioni per il sistema Microsoft Windows d'essere eseguite da calcolatori col sistema LINUX. La macchina virtuale esegue sia il sistema operativo Windows sia le applicazioni per lo stesso Windows.

Una delle caratteristiche chiave del linguaggio Java è che i programmi scritti in tale linguaggio sono eseguiti tramite una macchina virtuale, ciò consente l'esecuzione di un programma scritto in Java su ogni calcolatore che dispone di una macchina virtuale per tale linguaggio.

I programmi scritti in Java sono più lenti di C/C++ si utilizza JIT per migliorare le prestazioni

3.6.3 Java

Il linguaggio di programmazione Java, introdotto dalla Sun Microsystems alla fine del 1995, è un linguaggio orientato agli oggetti molto diffuso e fornisce, oltre a una descrizione analitica della sintassi del linguaggio e a una vasta libreria API, anche la definizione della **macchina virtuale Java** (*Java virtual machine* — JVM).

Gli oggetti si specificano con il costrutto `class` e un programma consiste di una o più classi. Per ognuna di queste, il compilatore produce un file (`*.class`) contenente il cosiddetto *bytecode*; si tratta di codice nel linguaggio di macchina della JVM, indipendente dall'architettura soggiacente, che viene per l'appunto eseguito dalla JVM.

La JVM è un calcolatore astratto che consiste di un caricatore delle classi, di un verificatore delle classi e di un interprete del linguaggio che esegue il bytecode. Il caricatore delle classi carica i file `*.class`, sia del programma scritto in Java sia dalla libreria API, affinché l'interprete possa eseguirli. Dopo che una classe è stata caricata, il verificatore delle classi controlla la correttezza sintattica del codice bytecode, che il codice non produca accessi oltre i limiti della pila e che non esegua operazioni aritmetiche sui puntatori, che potrebbero generare accessi illegali alla memoria. Se il controllo ha un esito positivo, la classe viene eseguita dall'interprete (Figura 3.12). La JVM gestisce la memoria in modo automatico procedendo alla sua 'ripulitura' (*garbage collection*) — che consiste nel recupero delle aree della memoria assegnate a oggetti non più in uso per restituirla al sistema. Per l'incremento delle prestazioni dei programmi eseguiti dalla macchina virtuale si concentra una notevole attività di ricerca nello studio degli algoritmi di ripulitura della memoria.

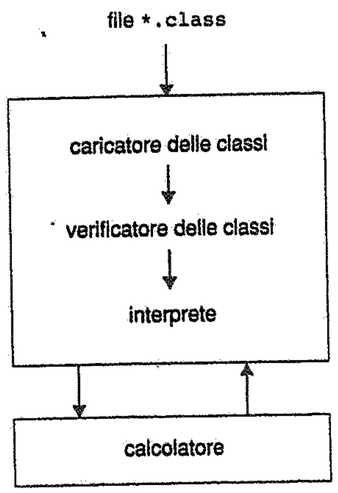


Figura 3.12 Macchina virtuale dell'ambiente Java.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'interprete del linguaggio Java può essere un programma che interpreta gli elementi del bytecode uno alla volta, o un compilatore istantaneo (*just-in-time* — JIT) che traduce il codice bytecode, indipendente dall'architettura, nel linguaggio di macchina dello specifico calcolatore. La maggior parte delle realizzazioni della JVM usa un compilatore JIT per migliorare le prestazioni. In altri casi, l'interprete si può incorporare nell'architettura del sistema, la JVM è un'unità d'elaborazione reale che esegue direttamente il bytecode.

La JVM rende possibile lo sviluppo di programmi indipendenti dall'architettura, è realizzata in modo specifico per ciascun sistema — come Windows o UNIX — e offre ai programmi scritti in Java un'astrazione uniforme del sistema. Fornisce un'interfaccia elegante e indipendente dall'architettura che permette a un file `*.class` di essere eseguito da qualsiasi sistema che dispone di una JVM.

Il linguaggio Java trae vantaggio dall'ambiente complessivo creato dalla macchina virtuale; la sua struttura definita dalla macchina virtuale offre una piattaforma sicura, efficiente, orientata agli oggetti, facilmente adattabile e indipendente dall'architettura su cui eseguire i programmi.

3.7 Progettazione e realizzazione di un sistema

Nei seguenti paragrafi si trattano i problemi riguardanti la progettazione e la realizzazione di un sistema. Naturalmente non si dispone di perfette soluzioni, ma alcuni metodi si sono dimostrati efficaci.

3.7.1 Scopi della progettazione

Il primo problema che s'incontra nella progettazione di un sistema riguarda la definizione degli scopi e delle caratteristiche del sistema stesso. Al più alto livello, la progettazione del sistema è influenzata in modo decisivo dalla scelta dell'architettura fisica e del tipo di sistema: a lotti o a partizione del tempo, mono o multiutente, distribuito, per elaborazioni in tempo reale o d'uso generale.

D'altra parte, oltre questo livello di progettazione, i requisiti possono essere molto difficili da specificare; anche se in generale si possono distinguere in due gruppi fondamentali: scopi degli *utenti* e scopi del *sistema*.

Gli utenti desiderano che un sistema abbia alcune caratteristiche ovvie: deve essere utile, facile da imparare e usare, affidabile, sicuro ed efficiente; queste caratteristiche non sono particolarmente utili nella progettazione di un sistema, poiché non tutti concordano sui metodi da applicare per raggiungere questi scopi.

Requisiti analoghi sono richiesti da chi deve progettare, creare e operare con il sistema: il sistema operativo deve essere di facile progettazione, realizzazione e manutenzione; deve essere flessibile, affidabile, senza errori ed efficiente. Anche in questo caso si tratta di requisiti vaghi, senza soluzione generale.

Non esiste una soluzione unica al problema della definizione dei requisiti di un sistema operativo. L'ampia gamma di sistemi mostra che da requisiti diversi possono risultare le soluzioni più varie per ambienti diversi. Ad esempio, i requisiti dell'MS-DOS, che com'è noto è un sistema per microcalcolatori a singolo utente, devono essere alquanto diversi da quelli dell'MVS, il sistema operativo multiaccesso e multiutente per mainframe IBM.

3.7.2 Meccanismi e criteri

Stabilire le caratteristiche e progettare un sistema operativo è, dunque, un compito estremamente creativo. Benché nessun libro di testo possa indicare come si possa fare tutto questo, esistono alcuni princìpi generali di pianificazione e progettazione applicabili soprattutto ai sistemi operativi.

Un principio molto importante è quello che riguarda la distinzione tra meccanismi e criteri. I meccanismi determinano *come* eseguire qualcosa; i criteri, invece, stabiliscono *cosa* si debba fare. Il temporizzatore del sistema (Paragrafo 2.5), ad esempio, è un meccanismo che assicura la protezione della CPU, ma la decisione riguardante la quantità di tempo da impostare nel temporizzatore per un utente specifico riguarda i criteri.

La distinzione tra meccanismi e criteri è molto importante ai fini della flessibilità. I criteri sono soggetti a cambiamenti rispetto alle situazioni o ai momenti. Nei casi peggiori il cambiamento di un criterio può richiedere il cambiamento del meccanismo sottostante. Sarebbe preferibile disporre di meccanismi generali: in questo caso un cambiamento di criterio implicherebbe solo la ridefinizione di alcuni parametri del sistema. Ad esempio, se in un calcolatore si decide che i programmi che impiegano intensamente le operazioni di I/O debbano avere priorità su quelli che impiegano intensamente le operazioni della CPU, ma si dispone di un meccanismo adeguatamente separato e indipendente dal criterio, si potrebbe facilmente istituire, in un secondo momento, un criterio opposto.

I sistemi operativi basati su un micronucleo portano alle estreme conseguenze la separazione dei meccanismi dai criteri, fornendo un insieme di funzioni fondamentali da impiegare come elementi di base; tali funzioni, quasi indipendenti dai criteri, consentono l'aggiunta di meccanismi e criteri più complessi tramite moduli del nucleo creati dagli utenti o anche tramite programmi utenti. All'estremo opposto si trovano i sistemi come le vecchie versioni dell'Apple Macintosh, in cui meccanismi e criteri sono codificati in modo da ottenere un ambiente omogeneo per l'intero sistema. Tutte le applicazioni hanno interfacce molto simili perché lo stesso sistema di interfaccia è incorporato nel nucleo.

Le decisioni relative ai criteri sono importanti per tutti i problemi di assegnazione delle risorse e di scheduling. Invece, ogni volta che un problema riguarda il *come* piuttosto che il *cosa* occorre definire un meccanismo.

http://www.adobepdf.com
Created by TIFF to PDF trial version, to remove this mark, please register this software.

3.7.3 Realizzazione

Una volta progettato, un sistema operativo deve essere realizzato. Tradizionalmente i sistemi operativi si scrivevano in un linguaggio assemblativo, attualmente si scrivono spesso in linguaggi di alto livello come il Linguaggio C o il C++.

Il primo sistema scritto in un linguaggio di alto livello fu probabilmente il Master Control Program (MCP) per i calcolatori Burroughs; l'MCP fu scritto infatti in una variante del linguaggio ALGOL; il MULTICS, sviluppato al MIT, fu scritto prevalentemente in PL/1; il sistema operativo Primos, per i calcolatori Prime, è scritto in un dialetto del FORTRAN; i sistemi operativi UNIX, OS/2 e Windows NT sono scritti prevalentemente in Linguaggio C. Nello UNIX originale soltanto 900 righe di codice erano scritte in linguaggio assemblativo; la maggior parte di queste riguardava il *dispatcher* e i driver dei dispositivi.

I vantaggi derivanti dall'uso di un linguaggio di alto livello, o perlomeno di un linguaggio orientato in modo specifico allo sviluppo di sistemi, sono gli stessi che si ottengono quando il linguaggio si usa per i programmi d'applicazione: il codice si scrive più rapidamente, è più compatto ed è più facile da capire e mettere a punto. Inoltre il perfezionamento delle tecniche di compilazione consente di migliorare il codice generato per l'intero sistema operativo con una semplice ricompilazione. Infine, un sistema operativo scritto in un linguaggio di alto livello è più facile da adattare a un'altra architettura (*porting*). L'MS-DOS, ad esempio, fu scritto nel linguaggio assemblativo dell'Intel 8088, quindi è disponibile solo per la famiglia di CPU Intel. Il sistema operativo UNIX, scritto prevalentemente in Linguaggio C, è invece disponibile per diversi tipi di CPU, tra cui le Intel 80*x*86 e Pentium, Motorola 680*x*0, Ultra SPARC, Compaq Alpha e Mips R*x*000.

I detrattori dell'uso dei linguaggi di alto livello per la scrittura dei sistemi operativi sostengono che i principali svantaggi sono una minore velocità d'esecuzione e una maggiore occupazione di spazio di memoria. D'altra parte, benché un programmatore esperto di un linguaggio assemblativo possa produrre piccole procedure molto efficienti, per quel che riguarda i programmi molto estesi, un moderno compilatore può eseguire complesse analisi e applicare raffinate ottimizzazioni che producono un codice eccellente. Le moderne CPU sono organizzate in numerose fasi d'elaborazione concatenate (*pipelining*) e parecchie unità, le cui complesse interdipendenze possono sopraffare la limitata capacità della mente umana di seguire i dettagli.

Come in altri sistemi, i miglioramenti principali nel rendimento sono dovuti più a strutture di dati e algoritmi migliori che a un ottimo codice in linguaggio assemblativo. Inoltre, sebbene i sistemi operativi siano molto grandi, solo una piccola parte del codice assume un'importanza critica riguardo al rendimento: il gestore della memoria e lo scheduler della CPU sono probabilmente le procedure più critiche. Una volta scritto il sistema e verificato il suo corretto funzionamento, è possibile identificare le procedure che possono costituire 'strozzature' e sostituirle con procedure equivalenti scritte in linguaggio assemblativo.

Per identificare le strozzature occorre controllare le prestazioni del sistema. Per calcolare e riportare le misurazioni riguardanti il comportamento del sistema è necessario aggiungere del codice. In molti sistemi, il sistema operativo esegue questo compito producendo documenti diagnostici del comportamento del sistema. Si registrano in un file

tutti gli avvenimenti interessanti specificando il momento in cui si sono verificati e i relativi parametri fondamentali. In seguito, un programma d'analisi può stabilire, secondo le informazioni registrate in tale file, il rendimento del sistema e identificarne i punti critici e le inefficienze. Queste stesse informazioni si possono usare anche come parametri per una simulazione che verifichi la validità dei miglioramenti suggeriti per tale sistema; e possono anche essere utili per individuare gli errori nel funzionamento del sistema operativo.

Un'altra possibilità è rappresentata dal calcolo e dalla comunicazione in tempo reale all'utente dei parametri delle prestazioni. Un temporizzatore può ad esempio attivare una procedura che memorizza il valore corrente del puntatore d'istruzione; da tali valori si può trarre una rappresentazione statistica delle locazioni del programma usate più di frequente all'interno del programma. Questo metodo consente agli operatori di acquisire familiarità con il comportamento del sistema e di modificarne il funzionamento in tempo reale.

3.8 Generazione di sistemi

Un sistema operativo si può progettare, codificare e realizzare specificamente per una singola macchina; tuttavia, è più diffusa la pratica di progettare sistemi operativi da impiegare in macchine di una stessa classe con configurazioni diverse. Il sistema si deve quindi configurare o generare per ciascuna situazione specifica; un processo talvolta noto come **generazione di sistemi** (SYSGEN). Il sistema operativo generalmente si distribuisce per mezzo di nastri o dischi. Per generare un sistema è necessario usare un programma speciale che può leggere da un file o chiedere all'operatore le informazioni riguardanti la configurazione specifica del sistema; o anche esplorare il sistema di calcolo per determinarne i componenti. Sono necessarie le seguenti informazioni:

- La CPU che si deve impiegare e le opzioni installate (serie di istruzioni estese, aritmetica in virgola mobile, e così via). Nel caso di sistemi con più unità d'elaborazione occorre anche descrivere ciascuna di esse.
- La quantità di memoria disponibile. Alcuni sistemi determinano autonomamente questi valori, accedendo a tutte le locazioni della memoria, fino a che viene generato un errore di indirizzo illegale. Questa procedura definisce l'indirizzo legale finale e quindi la quantità di memoria disponibile.
- I dispositivi disponibili. Il sistema deve conoscere gli indirizzi di ogni dispositivo e sapere come farvi riferimento (numero di dispositivo), deve conoscere il numero del segnale d'interruzione del dispositivo, il tipo e il modello del dispositivo e tutte le sue caratteristiche specifiche.
- Le opzioni del sistema operativo richieste o i valori dei parametri che è necessario usare. Queste informazioni possono contenere il numero delle aree della memoria da usare per le operazioni di I/O e la loro dimensione, l'algoritmo di scheduling della CPU richiesto, il numero massimo di processi da sostenere, e così via.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Una volta ottenute, queste informazioni si possono impiegare in modi diversi. In un caso limite, un amministratore di sistema le potrebbe usare per modificare una copia del codice sorgente del sistema operativo, che si dovrebbe poi ricompilare. Dichiarazioni di dati, inizializzazioni e costanti, insieme con una compilazione condizionale, produrrebbero una versione in codice di macchina del sistema operativo specifica per il sistema desiderato.

Per ottenere una versione meno specifica ma comunque adatta al sistema desiderato, la descrizione del sistema può determinare la creazione di tabelle e la selezione di moduli da una libreria precompilata, che si collegano per formare il sistema operativo richiesto. La selezione permette alla libreria di contenere i driver di tutti i dispositivi di I/O previsti, sebbene solo quelli effettivamente necessari siano inclusi nel sistema operativo richiesto. Poiché non si ricompila il sistema, la sua generazione risulta più rapida, ma il sistema ottenuto può essere inutilmente generale.

Un altro caso limite è rappresentato dalla costruzione di un sistema completamente controllato mediante tabelle. In questo caso tutto il codice è sempre parte del sistema e la selezione si effettua al momento dell'esecuzione del sistema stesso, anziché nella fase della compilazione o del collegamento. La generazione del sistema implica semplicemente la creazione di tabelle idonee a descrivere il sistema stesso. La maggior parte dei sistemi operativi moderni è costruita in questo modo. Il Solaris esegue qualche indagine sulla configurazione del sistema nella fase d'installazione e altre nella fase d'avviamento del sistema. L'amministratore del sistema può impiegare un file di configurazione per mettere a punto le variabili del sistema, ma la gestione dei dispositivi fisici è configurata automaticamente dal nucleo. In modo simile, il sistema Windows 2000 non richiede interventi diretti durante le fasi d'installazione e d'avviamento: una volta che si è risposto alle richieste fondamentali riguardanti il tipo di struttura desiderata per i dischi e la configurazione del sistema di rete, il programma d'installazione rileva in modo automatico i dispositivi che compongono il calcolatore e installa un sistema operativo adeguatamente generato.

Le differenze più rilevanti tra questi metodi riguardano la dimensione e la generalità del sistema ottenuto, oltre alla facilità di apportarvi modifiche in seguito a cambiamenti della sua struttura fisica. Si consideri, ad esempio, il costo delle modifiche da apportare al sistema per consentirgli la gestione di un nuovo terminale grafico o di un'altra unità a disco. I costi vanno ovviamente bilanciati con la frequenza delle modifiche.

Dopo che un sistema operativo è stato generato, deve essere eseguito dal calcolatore. Ma come fa la macchina fisica a 'sapere' dove si trova il nucleo e a caricarlo nella memoria? La procedura d'avviamento di un calcolatore attraverso il caricamento del nucleo è nota come avviamento (*booting*) del sistema: nella maggior parte dei sistemi di calcolo c'è un piccolo segmento di codice — memorizzato in una ROM e noto come programma d'avviamento (*bootstrap program*), o caricatore d'avviamento (*bootstrap loader*) — che individua il nucleo, lo carica nella memoria e ne avvia l'esecuzione. Alcuni sistemi, come i PC, eseguono tale compito in due passi: un caricatore d'avviamento molto semplice preleva dal disco un più complesso programma d'avviamento, che a sua volta carica il nucleo. La procedura d'avviamento di un sistema è trattata ulteriormente nel Paragrafo 14.3.2 e nell'Appendice A.

3.9 Sommario

I sistemi operativi offrono diversi servizi: al livello più basso, le chiamate del sistema permettono al programma in esecuzione di fare richieste direttamente al sistema operativo; a un livello superiore, l'interprete dei comandi (in alcuni ambiti noto come *shell*) mette a disposizione un meccanismo che consente a un utente di impartire una richiesta senza scrivere un programma. I comandi possono provenire da file, in un'esecuzione a lotti (*batch*); oppure direttamente da una tastiera, in modo interattivo o a partizione del tempo. I programmi di sistema offrono agli utenti i servizi più comuni.

I tipi di richieste variano secondo il livello delle stesse richieste. Il livello cui appartengono le chiamate del sistema deve offrire le funzioni di base, come quelle di controllo dei processi e gestione di file e dispositivi. Le richieste di livello superiore, soddisfatte dall'interprete dei comandi o dai programmi di sistema, sono tradotte in una sequenza di chiamate del sistema. I servizi di sistema si possono classificare in diverse categorie: controllo dei programmi, richieste di stato e richieste di I/O. Gli errori dei programmi si possono considerare richieste di servizio implicite.

Una volta definiti i servizi del sistema è possibile passare allo sviluppo della struttura del sistema operativo. Per registrare le informazioni che definiscono lo stato del calcolatore e lo stato dei processi del sistema occorrono diverse tabelle.

La progettazione di un nuovo sistema operativo è un compito molto difficile. Gli scopi del sistema si devono definire chiaramente prima di iniziare la progettazione; costituiscono la base da cui partire per poter scegliere tra le varie strategie e i vari algoritmi necessari.

Poiché un sistema operativo è di grandi dimensioni, la modularità è un altro fattore importante. La progettazione di un sistema come una sequenza di strati o l'uso di un micronucleo sono considerate buone tecniche. Il concetto di macchina virtuale tiene in grande considerazione il metodo basato sulla stratificazione e tratta il nucleo del sistema operativo come se facesse parte della macchina fisica. Su questa macchina virtuale si possono caricare persino altri sistemi operativi.

Ogni sistema operativo che dispone di una JVM può eseguire qualsiasi programma scritto in Java; la JVM astrae il sistema sottostante in un'interfaccia indipendente dall'architettura che offre ai programmi scritti in Java.

In tutto il ciclo di progettazione del sistema operativo, occorre prestare attenzione alla distinzione tra la scelta dei criteri e i dettagli dei meccanismi adottati. In questo modo si ottiene la massima flessibilità, che all'occorrenza consente di modificare più facilmente i criteri adottati.

Ormai i sistemi operativi sono quasi tutti scritti in un linguaggio per lo sviluppo di sistemi o in un linguaggio di alto livello; questa caratteristica facilita la realizzazione, la manutenzione e l'adattabilità a sistemi diversi. Per creare un sistema operativo per una macchina con una configurazione particolare occorre eseguire una generazione del sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

3.10 Esercizi

3.1 Dite quali sono le cinque attività principali di un sistema operativo relative alla gestione dei processi.

3.2 Dite quali sono le tre attività principali di un sistema operativo relative alla gestione della memoria.

3.3 Dite quali sono le tre attività principali di un sistema operativo relative alla gestione della memoria secondaria.

3.4 Dite quali sono le cinque attività principali di un sistema operativo relative alla gestione dei file.

3.5 Dite qual è lo scopo dell'interprete dei comandi, e perché di solito è separato dal nucleo.

3.6 Elencate cinque servizi offerti da un sistema operativo. Spiegate quali sono i vantaggi che ciascuno di essi offre all'operatore. Spiegate in quali casi i programmi utenti non possono offrire tali servizi.

3.7 Dite qual è lo scopo delle chiamate del sistema.

3.8 Usando le opportune chiamate del sistema, scrivete un programma in Linguaggio C o C++ che legge i dati contenuti in un file e li scrive in un altro file. Tale programma è descritto nel Paragrafo 3.3.

3.9 Spiegate perché il linguaggio Java non consente di invocare da un programma scritto in Java metodi nativi scritti, ad esempio, in Linguaggio C o C++. Esibite un esempio in cui è utile un metodo nativo.

3.10 Dite qual è lo scopo dei programmi di sistema.

3.11 Dite qual è il vantaggio principale offerto dal metodo stratificato alla progettazione dei sistemi.

3.12 Dite qual è il vantaggio principale del metodo basato su micronucleo alla progettazione di un sistema operativo.

3.13 Dite qual è il vantaggio principale che ottengono i progettisti di un sistema operativo che impiegano un'architettura a macchine virtuali, e qual è il vantaggio principale per gli utenti.

3.14 Spiegate perché un compilatore istantaneo (*just-in-time*) è utile per l'esecuzione dei programmi scritti in Java.

3.15 Spiegate perché la distinzione tra meccanismi e criteri è un principio auspicabile.

3.16 Il sistema operativo sperimentale Synthesis ha un assemblatore incorporato nel nucleo. Per ottimizzare le prestazioni delle chiamate del sistema, il nucleo assembla le procedure nello spazio del nucleo al fine di ridurre al minimo il percorso che le chiamate del sistema devono seguire attraverso il nucleo. Tale metodo è in antitesi al metodo stratificato che, pur rendendo più semplice la costruzione di un sistema operativo, determina un prolungamento del percorso delle chiamate del sistema attraverso il nucleo. Discutete i pro e i contro di tale metodo nella progettazione di un nucleo e nell'ottimizzazione delle prestazioni di un sistema.

3.11 Note bibliografiche

L'orientamento stratificato alla progettazione dei sistemi operativi è stato sostenuto da [Dijkstra 1968]. [Brinch Hansen 1970] è stato un primo sostenitore della costruzione di un sistema operativo come nucleo sul quale costruire sistemi più completi. Informazioni sul micronucleo QNX si possono trovare all'indirizzo `http://www.qnx.com`.

Il primo sistema operativo a offrire una macchina virtuale è stato il CP/67 su un IBM 360/67. Il sistema operativo IBM VM/370 disponibile in commercio deriva dal CP/67. Discussioni generali riguardanti le macchine virtuali si trovano in [Hendricks e Hartmann 1979], [MacKinnon 1979] e [Schultz 1988]. Una macchina virtuale che permette l'esecuzione nel sistema operativo LINUX delle applicazioni scritte per il sistema Microsoft Windows è reperibile all'indirizzo `http://www.vmware.com`. [Cheung e Loong 1995] esplora la questione della strutturazione di un sistema operativo da un micronucleo a un sistema estendibile. [Back et al. 2000] tratta la progettazione dei sistemi operativi per il linguaggio Java.

L'MS-DOS, Versione 3.1, è descritto in [Microsoft 1986], il sistema Windows NT è descritto in [Solomon 1998]. Il sistema operativo Apple Macintosh è descritto in [Apple 1987]. Il Berkeley UNIX (BSD) è descritto in [McKusick et al. 1996]. Lo standard AT&T dello UNIX system V è descritto in [Earhart 1986]. Una buona descrizione dell'OS/2 si trova in [Iacobucci 1988]. Il sistema Mach è presentato in [Accetta et al. 1986] e l'AIX in [Loucks e Sauer 1987]. Il sistema sperimentale Synthesis è trattato in [Massalin e Pu 1989]. [Bar 2000] offre una descrizione dettagliata del nucleo del sistema operativo LINUX.

La definizione del linguaggio Java e della relativa macchina virtuale è presentata in [Gosling et al. 1996] e in [Lindholm e Yellin 1998], rispettivamente. Il funzionamento interno della JVM è descritto in [Venners 1998] e in [Meyer e Downing 1997]. [Jones e Lin 1996] offre un'esauriente trattazione degli algoritmi di recupero dei blocchi della memoria non più in uso (*garbage collection*). Ulteriori informazioni sul linguaggio Java sono disponibili all'indirizzo `http://www.javasoft.com`.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Parte seconda

Gestione dei processi

Processi
Thread
Scheduling della CPU
Sincronizzazione dei processi
Stallo dei processi

Un *processo* si può pensare come un programma in esecuzione. Per svolgere il proprio compito, un processo richiede determinate risorse, come tempo d'elaborazione della CPU, memoria, file e dispositivi di I/O. Queste risorse si assegnano al processo al momento della sua creazione o durante l'esecuzione.

Il processo è l'unità di lavoro nella maggior parte dei sistemi. Un sistema di questo tipo è formato da processi del sistema operativo, che eseguono il codice di sistema, e da processi utenti, che eseguono il codice utente. Tutti questi processi si possono eseguire in modo concorrente.

Benché tradizionalmente un processo contenga un solo thread di controllo dell'esecuzione, la maggior parte dei sistemi operativi moderni attualmente gestisce processi con più thread.

Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei processi e dei thread di sistema e utenti: creazione e cancellazione, scheduling, offerta dei meccanismi di sincronizzazione e comunicazione, gestione delle situazioni di stallo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I sistemi multiprogrammati si basano sul concetto di PROCESSO ⇒ infatti avendo + processi in esecuzione si ottimizza l'uso della CPU.

Capitolo 4

Processi

= viene inteso come programma che si esegue seguendo un unico percorso d'esecuzione → 1 processo può svolgere 1 unico compito alla volta. Nei sistemi moderni si è esteso il concetto di processo in modo d'avere + percorsi d'esecuzione (MULTITHREAD)

I primi sistemi di calcolo consentivano l'esecuzione di un solo programma alla volta, che aveva il completo controllo del sistema e accesso a tutte le sue risorse. Gli attuali sistemi di calcolo consentono, invece, che più programmi siano caricati nella memoria ed eseguiti in modo concorrente. Tale evoluzione richiede un più severo controllo e una maggiore compartimentazione dei vari programmi. Da tali necessità deriva la nozione di **processo d'elaborazione** — o, più brevemente, **processo** —, che in prima istanza si può definire come un programma in esecuzione, e costituisce l'unità di lavoro dei moderni sistemi a partizione del tempo d'elaborazione.

Maggiore è la complessità di un sistema operativo, maggiori sono i servizi che si suppone esso fornisca ai propri utenti. Benché il suo compito principale sia l'esecuzione dei programmi utenti, deve anche occuparsi dei vari compiti di sistema che è più conveniente lasciare fuori del nucleo. Un sistema è quindi costituito da un insieme di processi: del sistema operativo, che eseguono il codice di sistema; utenti, che eseguono il codice d'utente. Tutti questi processi si possono eseguire potenzialmente in modo concorrente e l'uso della CPU (o di più unità d'elaborazione) è commutato tra i vari processi. Il sistema operativo può rendere il calcolatore più produttivo avvicendando i diversi processi nell'uso della CPU.

→ con multiprogrammazione fornisce 1 ambiente in cui le risorse del sistema sono impiegate in modo efficiente ma non fornisce 1 sistema d'interazione con l'utente

Sistemi a lotti	Sistemi a partizione del tempo
utente ≠ operatore: gli utenti affidavano i loro lavori su schede perforate ad operatori che li dividevano in gruppi a seconda delle loro caratteristiche	time sharing o multitasking: può eseguire + lavori commutando le loro esecuzioni con 1 frequenza, ma da permettere ad ogni utente con il suo programma durante la sua esecuzione.

4.1 Concetto di processo

La questione riguardante i nomi da attribuire a tutte le attività della CPU è un ostacolo alla discussione dei sistemi operativi. Un sistema a lotti (*batch*) esegue *lavori* (*job*), mentre un sistema a partizione del tempo esegue *programmi utenti* (o *task*). Persino in un sistema monoutente, come il Microsoft Windows o il Macintosh OS, un utente può far eseguire parecchi programmi contemporaneamente: un elaboratore di testi, un programma di consultazione del Web, un programma per la posta elettronica. Anche se l'utente esegue un solo programma alla volta, il sistema operativo deve svolgere le proprie attività interne, ad esempio la gestione della memoria. Queste attività sono simili per molti aspetti, perciò sono denominate *processi*. In questo testo i termini *lavoro* e *processo* sono

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

usati in modo quasi intercambiabile. Sebbene si preferisca il termine *processo*, occorre ricordare che la maggior parte della terminologia e della teoria dei sistemi operativi si è sviluppata in un periodo in cui l'attività principale dei sistemi operativi riguardava la gestione dei lavori d'elaborazione in sistemi a lotti. Sarebbe fuorviante evitare di usare termini comunemente accettati che contengono la parola *lavoro* o *job*, ad esempio *job scheduling*, solo perché il termine *processo* ha ormai soppiantato il termine *lavoro*.

4.1.1 Processo

Informalmente, un *processo* è un programma in esecuzione. È qualcosa di più del codice di un programma, talvolta noto anche come sezione di testo: comprende l'attività corrente, rappresentata dal valore del contatore di programma e dal contenuto dei registri della CPU; normalmente comprende anche la propria pila (*stack*), che contiene a sua volta i dati temporanei, come i parametri di un metodo, gli indirizzi di rientro e le variabili locali, e una sezione di dati contenente le variabili globali. Si sottolinea che un programma di per sé non è un processo; un programma è un'entità *passiva*, come il contenuto di un file memorizzato in un disco, mentre un processo è un'entità *attiva*, con un contatore di programma che specifica qual è la prossima istruzione da eseguire e un insieme di risorse associate.

Sebbene due processi possano essere associati allo stesso programma, sono tuttavia da considerare due sequenze d'esecuzione distinte. Parecchi utenti possono, ad esempio, far eseguire diverse istanze dello stesso programma di posta elettronica, così come un utente può invocare più istanze dello stesso elaboratore di testi. Ciascuna di queste è un diverso processo, e benché le sezioni di testo siano equivalenti, le sezioni dei dati sono diverse. È inoltre usuale che durante la propria esecuzione un processo generi altri processi. Questo argomento è trattato nel Paragrafo 4.4.

4.1.2 Stato del processo

Mentre un processo è in esecuzione è soggetto a cambiamenti di *stato*, definiti in parte dall'attività corrente del processo stesso. Ogni processo può trovarsi in uno tra i seguenti stati:

- **Nuovo.** Si crea il processo.
- **Esecuzione.** Un'unità d'elaborazione esegue le istruzioni del relativo programma.
- **Attesa.** Il processo attende che si verifichi qualche evento (come il completamento di un'operazione di I/O o la ricezione di un segnale).
- **Pronto.** Il processo attende di essere assegnato a un'unità d'elaborazione.
- **Terminato.** Il processo ha terminato l'esecuzione.

Queste definizioni sono piuttosto arbitrarie, e variano secondo il sistema operativo. Gli stati che rappresentano sono in ogni modo presenti in tutti i sistemi, anche se alcuni sistemi operativi introducono ulteriori distinzioni tra gli stati dei processi. In ciascuna unità d'elaborazione può essere in *esecuzione* solo un processo per volta, sebbene molti processi possano essere *pronti* o nello stato di *attesa*. Il diagramma di transizione corrispondente a questi stati è riportato nella Figura 4.1.

4.1.3 Descrittore di processo

Ogni processo è rappresentato nel sistema operativo da un descrittore di processo (*process descriptor* — PD), detto anche blocco di controllo di un processo (*process control block* — PCB, o *task control block* — TCB). Un descrittore di processo (Figura 4.2) contiene molte informazioni connesse a un processo specifico, tra cui le seguenti:

1. **Stato del processo.** Lo stato può essere: nuovo, pronto, esecuzione, attesa, arresto, ecc.
2. **Contatore di programma.** Il contatore di programma contiene l'indirizzo della successiva istruzione da eseguire per tale processo.
3. **Registri di CPU.** I registri variano in numero e tipo secondo l'architettura del calcolatore. Essi comprendono accumulatori, registri d'indice, puntatori alla cima delle strutture a pila (*stack pointer*), registri d'uso generale e registri contenenti informazioni relative ai codici di condizione. Quando si verifica un'interruzione della CPU, si devono salvare tutte queste informazioni insieme con il contatore di programma, in modo da permettere la corretta esecuzione del processo in un momento successivo (Figura 4.3).

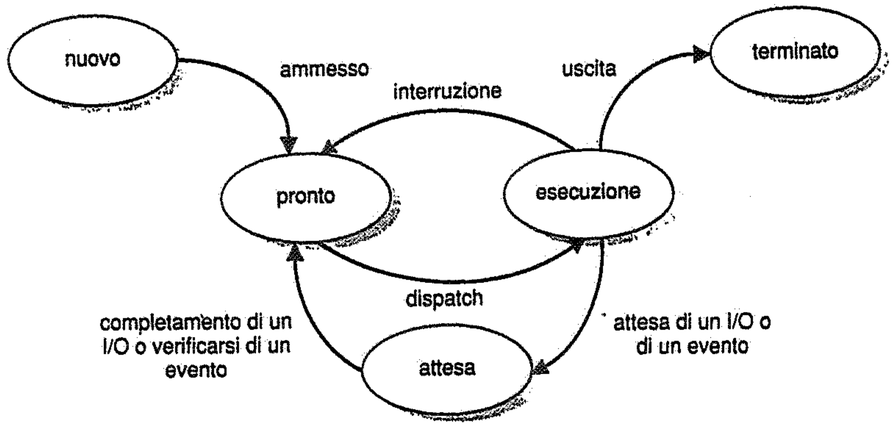


Figura 4.1 Diagramma di transizione degli stati di un processo.

http://www.pdftpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

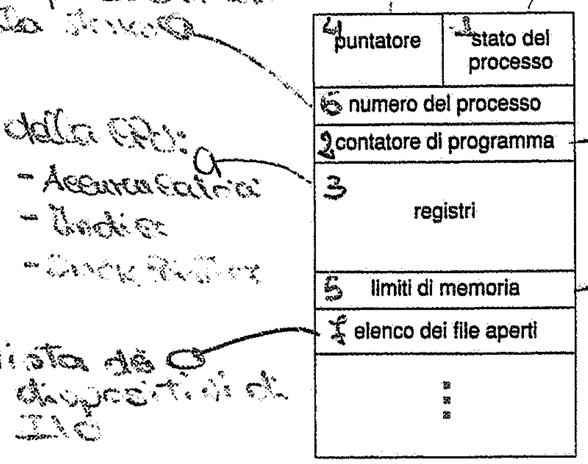


Figura 4.2 Descrittore di processo (PCB).

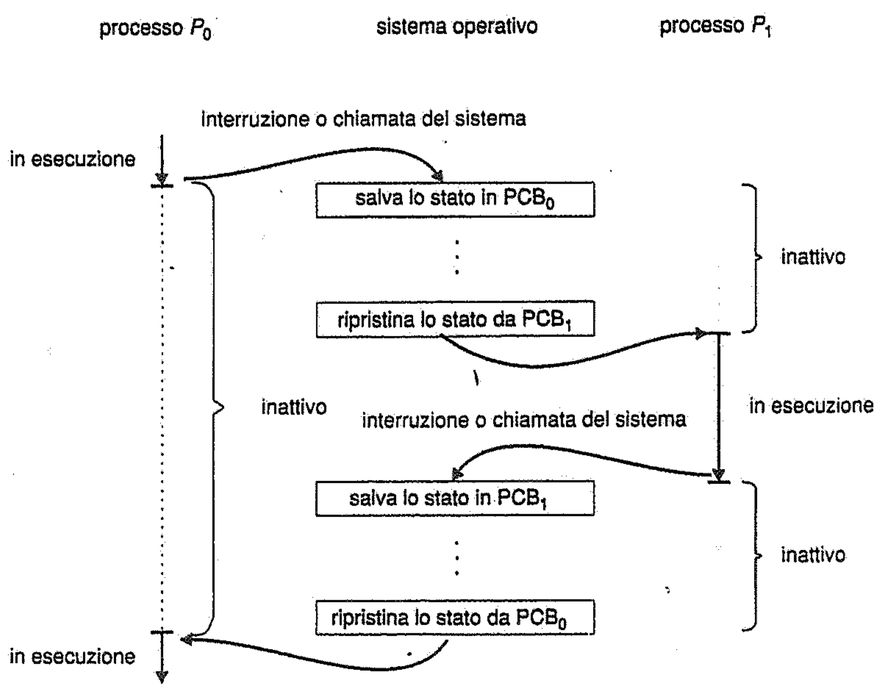


Figura 4.3 La CPU può essere commutata tra i processi.

- **Informazioni sullo scheduling di CPU.** Queste informazioni comprendono la priorità del processo, i puntatori alle code di scheduling e tutti gli altri parametri di scheduling (nel Capitolo 6 si descrive lo scheduling dei processi).
- **Informazioni sulla gestione della memoria.** Queste informazioni si possono esprimere attraverso i valori dei registri di base e di limite, le tabelle delle pagine, o le tabelle dei segmenti, secondo il sistema di gestione della memoria usato dal sistema operativo (Capitolo 9).
- **Informazioni di contabilizzazione delle risorse.** Queste informazioni comprendono il tempo d'uso della CPU e il tempo reale d'utilizzo della stessa, i limiti di tempo, i numeri dei processi, e così via.
- **Informazioni sullo stato dell'I/O.** Queste informazioni comprendono la lista dei dispositivi di I/O assegnati a un determinato processo, l'elenco dei file aperti, e così via. Il PCB si usa semplicemente come deposito per tutte le informazioni relative ai vari processi.

4.1.4 Thread

Il modello dei processi discusso fino a questo punto sottintende che un processo è un programma che si esegue seguendo un unico percorso d'esecuzione. Se un processo sta, ad esempio, eseguendo un programma di elaborazione di testi, l'esecuzione avviene seguendo una singola sequenza di istruzioni; quindi il processo può svolgere un solo compito alla volta. Tramite lo stesso processo, ad esempio, un utente non può contemporaneamente inserire caratteri e verificare la correttezza ortografica di quel che sta scrivendo. In molti sistemi operativi moderni si è esteso il concetto di processo introducendo la possibilità d'avere più percorsi d'esecuzione, in modo da permettere che un processo possa svolgere più di un compito alla volta. Il Capitolo 5 è dedicato ai processi multithread.

4.2 Scheduling dei processi

L'obiettivo della multiprogrammazione consiste nel disporre dell'esecuzione contemporanea di alcuni processi in modo da massimizzare l'utilizzo della CPU. L'obiettivo della partizione del tempo è di commutare l'uso della CPU tra i vari processi così frequentemente che gli utenti possano interagire con ciascun programma mentre esso è in esecuzione. In un sistema con singola CPU si può avere in esecuzione un solo processo alla volta. Se esistono più processi, quelli che non sono in esecuzione devono aspettare che la CPU sia libera e la loro esecuzione possa quindi essere ripresa.

http://www.adultpdf.com
Created by TIFF To PDF [illegible] mark, please register this software.

• Coda dei job → Insieme di tutti i processi presenti nel sistema
• Ready queue → Insieme di tutti i processi presenti nella memoria centrale, pronti ed in attesa di essere messi in esecuzione
• Code dei dispositivi → Insieme dei processi in attesa x 1 dispositivo di I/O

4.2.1 Code di scheduling

Ogni processo è inserito in una coda di processi, composta da tutti i processi del sistema. I processi presenti nella memoria centrale, che sono pronti e nell'attesa d'essere eseguiti si trovano in una lista detta **coda dei processi pronti** (*ready queue*). Questa coda generalmente si memorizza come una lista concatenata: un'intestazione della coda dei processi pronti contiene i puntatori al primo e all'ultimo PCB dell'elenco, e ciascun PCB è esteso con un campo puntatore che indica il successivo processo contenuto nella coda dei processi pronti.

Il sistema operativo ha anche altre code. Quando si assegna la CPU a un processo, quest'ultimo rimane in esecuzione per un certo tempo e prima o poi termina, viene interrotto, oppure si ferma nell'attesa di un evento particolare, come il completamento di una richiesta di I/O. Una richiesta di I/O può essere diretta a un'unità a nastri oppure a un dispositivo condiviso, come un disco. Poiché nel sistema esistono molti processi, il disco può essere occupato con una richiesta di I/O di qualche altro processo, quindi il processo deve attendere che il disco sia disponibile. L'elenco dei processi che attendono la disponibilità di un particolare dispositivo di I/O si chiama **coda del dispositivo**; ogni dispositivo ha la propria coda (Figura 4.4).

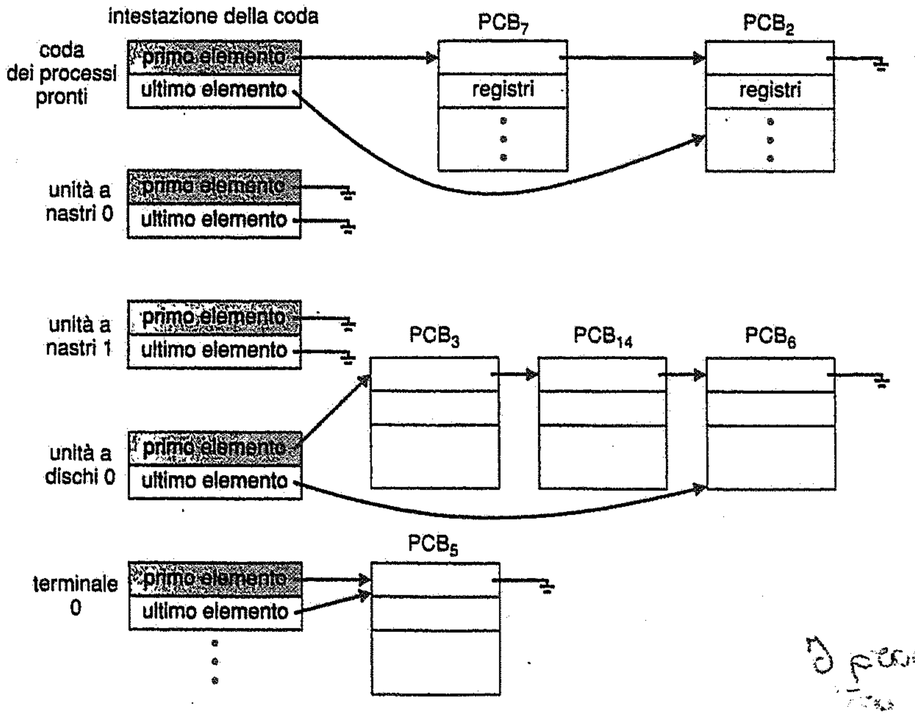


Figura 4.4 Coda dei processi pronti e diverse code di dispositivi I/O.

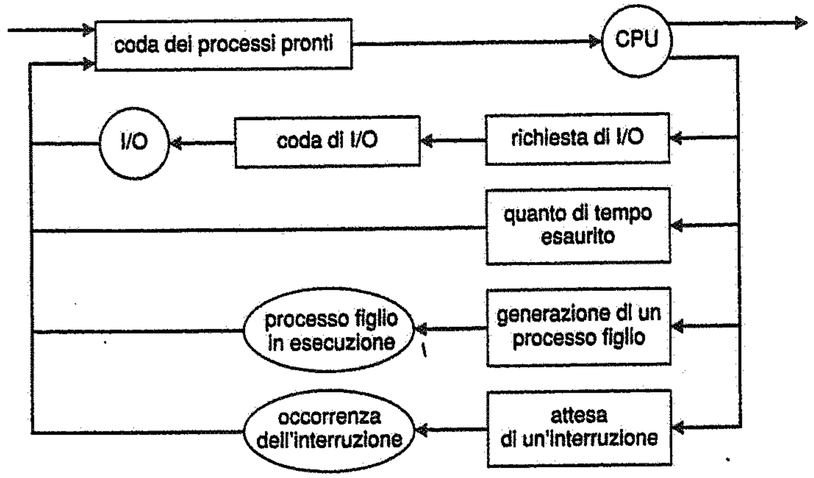


Figura 4.5 Diagramma di accodamento per lo scheduling dei processi.

Una comune rappresentazione utile alla discussione dello scheduling dei processi è data da un *diagramma di accodamento* come quello illustrato nella Figura 4.5. Ogni riquadro rappresenta una coda. Sono presenti due tipi di coda: la coda dei processi pronti e un insieme di code di dispositivi. I cerchi rappresentano le risorse che servono le code, le frecce indicano il flusso di processi nel sistema.

Un nuovo processo si colloca inizialmente nella coda dei processi pronti; dove attende finché non è selezionato per essere eseguito. Una volta che il processo è assegnato alla CPU ed è nella fase d'esecuzione, si può verificare uno dei seguenti eventi:

- il processo può emettere una richiesta di I/O e quindi essere inserito in una coda di I/O;
- il processo può creare un nuovo processo e attenderne la terminazione;
- il processo può essere rimosso forzosamente dalla CPU a causa di un'interruzione, ed essere reinserito nella coda dei processi pronti.

Nei primi due casi, al completamento della richiesta di I/O o al termine del processo figlio, il processo passa dallo stato d'attesa allo stato pronto ed è nuovamente inserito nella coda dei processi pronti. Un processo continua questo ciclo finché la sua esecuzione termina, cioè fino alla sua rimozione da tutte le code, si rimuove il suo PCB e si revocano le varie risorse.

→ processo [illegible] coda

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

4.2.2 Scheduler

Nel corso della sua esistenza, un processo si trova in varie code di scheduling. Il sistema operativo, che è incaricato di selezionare i processi dalle suddette code, compie la selezione per mezzo di un opportuno scheduler.

In un sistema a lotti, accade spesso che si sottopongano più lavori d'elaborazione di quanti se ne possano eseguire immediatamente. Questi lavori si trasferiscono in dispositivi di memoria secondaria, generalmente dischi, dove si tengono fino al momento dell'esecuzione (*spooling*). Lo **scheduler a lungo termine** (o *job scheduler*), sceglie i lavori da questo insieme e li carica nella memoria affinché siano eseguiti. Lo scheduler a breve termine, o **scheduler di CPU**, fa la selezione tra i lavori pronti per essere eseguiti e assegna la CPU a uno di essi.

Questi due scheduler si differenziano principalmente per la frequenza con la quale sono eseguiti. Lo scheduler a breve termine seleziona frequentemente un nuovo processo per la CPU. Il processo può essere in esecuzione solo per pochi millisecondi prima di passare ad attendere una richiesta di I/O. Poiché spesso si esegue almeno una volta ogni 100 millisecondi, lo scheduler a breve termine deve essere molto rapido: se impiegasse 10 millisecondi per decidere quale processo eseguire nei prossimi 100 millisecondi, si userebbe, o per meglio dire si sprecherebbe, il 10/(100 + 10) = 9 per cento del tempo di CPU per il solo scheduling.

Lo scheduler a lungo termine, invece, si esegue con una frequenza molto inferiore; tra la creazione di nuovi processi possono trascorrere diversi minuti. Lo scheduler a lungo termine controlla il **grado di multiprogrammazione**, cioè il numero di processi presenti nella memoria. Se il grado di multiprogrammazione è stabile, la velocità media di creazione dei processi deve essere uguale alla velocità media con cui i processi abbandonano il sistema; quindi lo scheduler a lungo termine si può chiamare solo quando un processo abbandona il sistema. A causa del maggiore intervallo che intercorre tra le esecuzioni, lo scheduler a lungo termine dispone di più tempo per scegliere un processo per l'esecuzione.

Lo scheduler a lungo termine deve fare un'accurata selezione dei processi. In generale, la maggior parte dei processi si può caratterizzare come avente una prevalenza di I/O, o come avente una prevalenza d'elaborazione. Un **processo con prevalenza di I/O** (*I/O bound*) impiega la maggior parte del proprio tempo nell'esecuzione di operazioni di I/O. Un **processo con prevalenza d'elaborazione** (*CPU bound*), viceversa, richiede poche operazioni di I/O e impiega la maggior parte del proprio tempo nelle elaborazioni. Lo scheduler a lungo termine dovrebbe scegliere una buona **combinazione di processi** con prevalenza di I/O e con prevalenza d'elaborazione. Se tutti i processi fossero con prevalenza di I/O, la coda dei processi pronti sarebbe quasi sempre vuota e lo scheduler a breve termine resterebbe pressoché inattivo. Se tutti i processi fossero con prevalenza d'elaborazione, la coda d'attesa per l'I/O sarebbe quasi sempre vuota, i dispositivi non sarebbero utilizzati, e il sistema sarebbe nuovamente sbilanciato. Le prestazioni migliori sono date da una combinazione equilibrata di processi con prevalenza di I/O e processi con prevalenza d'elaborazione.

Esistono sistemi nei quali lo scheduler a lungo termine può essere assente o minimo. Ad esempio, i sistemi a partizione del tempo, come lo UNIX, sono spesso privi di scheduler a lungo termine, e si limitano a caricare nella memoria tutti i nuovi processi, che sono gestiti dallo scheduler a breve termine. La stabilità di questi sistemi dipende dai limiti fisici degli stessi, come un numero limitato di terminali disponibili, oppure dall'autoregolamentazione insita nella natura degli utenti umani: quando ci si accorge che il rendimento della macchina scende a livelli inaccettabili si possono semplicemente chiudere alcune attività o la sessione di lavoro.

In alcuni sistemi operativi, come i sistemi a partizione del tempo, si può introdurre un livello di scheduling intermedio; in questo modo uno **scheduler a medio termine** (Figura 4.6) può rimuovere processi dalla memoria (e dalla contesa attiva per la CPU) e così ridurre il grado di multiprogrammazione, successivamente può reintrodurre il processo nella memoria e riprendere la sua esecuzione dal punto in cui era stata abbandonata. Questo schema si chiama **avvicendamento dei processi nella memoria** — o, più in breve, **avvicendamento** (*swapping*) — ed è anche noto come **scambio**. L'avvicendamento dei processi nella memoria può servire a migliorare la combinazione di processi, oppure a liberare una parte della memoria se un cambiamento dei requisiti di memoria ha impegnato eccessivamente la memoria disponibile. L'avvicendamento dei processi nella memoria è trattato nel Capitolo 9.

4.2.3 Cambio di contesto

Il passaggio della CPU a un nuovo processo implica la registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo. Questa procedura è nota col nome di **cambio di contesto** (*context switch*). Il contesto di un processo è descritto nel PCB di tale processo: include i valori dei registri della CPU, lo stato del processo (Figura 4.1) e le informazioni di gestione della memoria. Quando si ve-

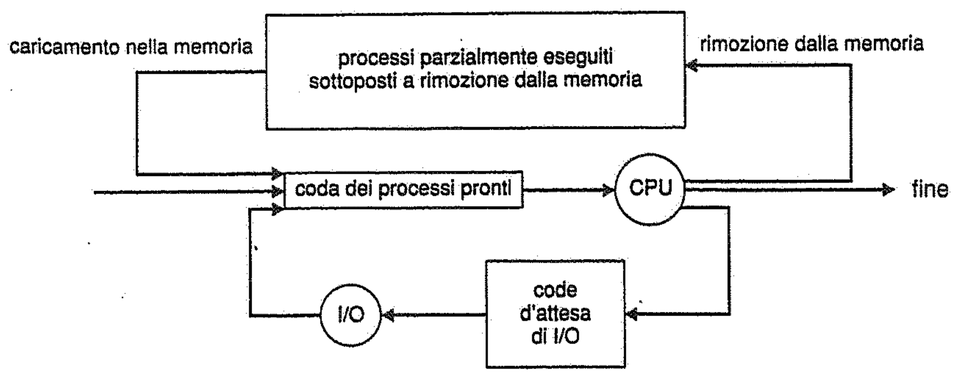


Figura 4.6 Aggiunta di scheduling a medio termine al diagramma di accodamento.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

rifica un cambio di contesto, il nucleo registra il contesto del vecchio processo nel suo PCB e carica il contesto precedentemente registrato del nuovo processo scelto per l'esecuzione. Il tempo necessario al cambio di contesto è puro sovraccarico, infatti il sistema non compie nessun lavoro utile durante la commutazione, e varia da calcolatore a calcolatore secondo la velocità della memoria, il numero dei registri che si devono copiare e l'esistenza di istruzioni speciali (come un'istruzione singola di caricamento o memorizzazione di tutti i registri). Generalmente, questo tempo è compreso tra 1 e 1000 microsecondi.

La durata del cambio di contesto dipende molto dall'architettura; ad esempio, alcune CPU (come la Sun UltraSPARC) offrono più gruppi di registri, quindi il cambio di contesto prevede la semplice modifica di un puntatore al gruppo di registri corrente. Naturalmente, se il numero dei processi attivi è maggiore di quello dei gruppi di registri disponibili, il sistema rimedia copiando i dati dei registri nella e dalla memoria, come prima. Quindi, più complesso è il sistema operativo, più lavoro si deve svolgere durante un cambio di contesto. Come si discute nel Capitolo 9, l'uso di tecniche complesse di gestione della memoria può richiedere lo spostamento di ulteriori dati a ogni cambio di contesto. Ad esempio, si deve preservare lo spazio d'indirizzi del processo corrente mentre si prepara lo spazio per il processo successivo. Il modo in cui si preserva tale spazio e la relativa quantità di lavoro dipendono dal metodo di gestione della memoria del sistema operativo. Come si vede nel Capitolo 5, l'uso del cambio di contesto tende a diventare una tale 'strozzatura' per le prestazioni che i programmatori impiegano nuove strutture, i thread, per evitarlo ogniqualvolta è possibile.

4.3 Operazioni sui processi

I processi in un sistema si possono eseguire in modo concorrente, e si devono creare e cancellare dinamicamente; a tal fine il sistema operativo deve offrire un meccanismo che permetta di creare e terminare un processo.

4.3.1 Creazione di un processo

Durante la propria esecuzione, un processo può creare numerosi nuovi processi tramite un'apposita chiamata del sistema (`create_process`). Il processo creante si chiama processo **genitore**, mentre il nuovo processo si chiama processo **figlio**. Ciascuno di questi nuovi processi può creare a sua volta altri processi, formando un *albero* di processi (Figura 4.7).

In generale, per eseguire il proprio compito, un processo necessita di alcune risorse (tempo d'elaborazione, memoria, file, dispositivi di I/O). Quando un processo crea un sottoprocesso, quest'ultimo può essere in grado di ottenere le proprie risorse direttamente dal sistema operativo, oppure può essere vincolato a un sottoinsieme delle risorse del processo genitore. Il processo genitore può avere la necessità di spartire le proprie risorse tra i suoi processi figli, oppure può essere in grado di condividerne alcune, come la memoria o i file, tra più processi figli. Limitando le risorse di un processo figlio a un sottoinsieme di risorse del processo genitore, si può evitare che un processo sovraccarichi il sistema creando troppi sottoprocessi.

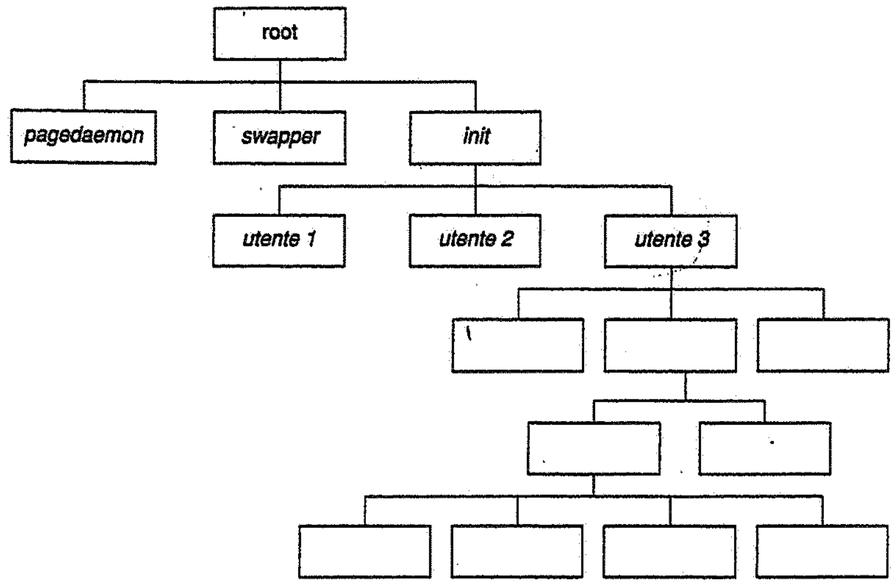


Figura 4.7 Albero di processi in un tipico sistema UNIX.

Quando si crea un processo, esso ottiene, oltre alle varie risorse fisiche e logiche, i dati di inizializzazione che il processo genitore può passare al processo figlio. Si consideri, ad esempio, un processo la cui funzione sia riportare lo stato del file f_1 sullo schermo di un terminale. Al momento della sua creazione, questo processo riceve dal proprio processo genitore il nome del file f_1; questo dato si usa durante l'esecuzione del processo per ottenere le informazioni desiderate. Il processo può anche ricevere il nome del dispositivo di emissione dei dati. Alcuni sistemi operativi passano le risorse ai processi figli. In tali sistemi il nuovo processo può ricevere due file aperti, f_1 e il file che rappresenta il dispositivo terminale, quindi deve semplicemente trasferire i dati tra i due.

Quando un processo crea un nuovo processo, per quel che riguarda l'esecuzione ci sono due possibilità:

- il processo genitore continua l'esecuzione in modo concorrente con i propri processi figli;
- il processo genitore attende che alcuni o tutti i suoi processi figli terminino.

Ci sono due possibilità anche per quel che riguarda lo spazio d'indirizzi del nuovo processo:

- il processo figlio è un duplicato del processo genitore;
- nel processo figlio si carica un programma.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Nel sistema operativo UNIX ad esempio ogni processo è identificato dal proprio **identificatore di processo**; un intero unico. Un nuovo processo si crea per mezzo della chiamata del sistema `fork`, ed è composto di una copia dello spazio degli indirizzi del processo genitore. Questo meccanismo permette al processo genitore di comunicare senza difficoltà con il proprio processo figlio. Entrambi i processi (genitore e figlio) continuano l'esecuzione all'istruzione successiva alla chiamata del sistema `fork`; con una differenza: la chiamata del sistema `fork` riporta il valore zero nel nuovo processo (il figlio), ma riporta l'identificatore del processo figlio (il PID diverso da zero) nel processo genitore. Tramite il valore riportato del PID, i due processi possono procedere nell'esecuzione 'sapendo' qual è il processo padre e qual è il processo figlio.

Generalmente, dopo una chiamata del sistema `fork`, uno dei due processi impiega una chiamata del sistema `execlp` per sostituire lo spazio di memoria del processo con un nuovo programma. La chiamata del sistema `execlp` carica nella memoria un file binario, cancellando l'immagine di memoria del programma contenente la stessa chiamata del sistema `execlp`, quindi avvia la sua esecuzione. In questo modo i due processi possono comunicare e quindi procedere in modo diverso. Il processo genitore può anche generare più processi figli, oppure, se durante l'esecuzione del processo figlio non ha nient'altro da fare, può invocare la chiamata del sistema `wait` per rimuovere se stesso dalla coda dei processi pronti fino alla terminazione del figlio. Il programma scritto nel Linguaggio C della Figura 4.8 illustra le chiamate del sistema dello UNIX descritte precedentemente. Il processo genitore crea un processo figlio con la chiamata del sistema `fork`. Si ottengono due processi distinti ciascuno dei quali è un'istanza d'esecuzione dello stesso programma. Il valore assegnato alla variabile `pid` (riportato dalla chiamata del sistema `fork`) è zero nel processo figlio, e un numero intero maggiore di zero nel processo genitore. Usando la chiamata del sistema `execlp`, il processo figlio sovrappone il proprio spazio d'indirizzi con il comando `/bin/ls` dello UNIX (che si usa per ottenere l'elenco del contenuto di una directory). Impiegando la chiamata del sistema `wait`, il processo genitore attende che il processo figlio termini. Quando ciò accade, il processo genitore chiude la propria fase d'attesa dovuta alla chiamata del sistema `wait` e termina usando la chiamata del sistema `exit`.

Il sistema operativo DEC VMS, al contrario, crea un nuovo processo, carica il programma specificato in tale processo e ne avvia l'esecuzione. Il sistema operativo Microsoft Windows 2000 prevede entrambi i modelli: si può duplicare lo spazio d'indirizzi del processo genitore, oppure il processo genitore può specificare il nome di un programma che il sistema operativo provvederà a caricare nello spazio d'indirizzi del nuovo processo.

4.3.2 Terminazione di un processo

Un processo termina quando finisce l'esecuzione della sua ultima istruzione e chiede al sistema operativo di essere cancellato usando la chiamata del sistema `exit`; a questo punto, il processo figlio può riportare alcuni dati al processo genitore, che li riceve attraverso la chiamata del sistema `wait`. Tutte le risorse del processo, incluse la memoria fisica e virtuale, i file aperti e le aree della memoria per l'I/O, sono liberate dal sistema operativo.

La terminazione di un processo si può avere anche in altri casi. Un processo può causare la terminazione di un altro processo per mezzo di un'opportuna chiamata del sistema,


```
#include <stdio.h>

void main(int argc, char *argv[])
{
int pid;

    /* genera un nuovo processo */
    pid = fork();

    if (pid < 0) { /* errore */
      fprintf(stderr, "generazione del nuovo processo fallita")
      exit(-1);
    }
    else if (pid == 0) { /* processo figlio */
      execlp("/bin/ls", "ls", NULL);
    }
    else { /* processo genitore */
      /* il genitore attende il completamento del figlio */
      wait(NULL);
      printf("il processo figlio ha terminato");
      exit(0);
    }
}
```

Figura 4.8 Programma scritto in Linguaggio C che genera un nuovo processo.

ad esempio `abort`. Generalmente solo il genitore del processo che si vuole terminare può invocare una chiamata del sistema di questo tipo, altrimenti gli utenti potrebbero causare arbitrariamente la terminazione forzata di processi di chiunque. Occorre notare che un genitore deve conoscere le identità dei propri figli, perciò quando un processo crea un nuovo processo, l'identità del nuovo processo viene passata al processo genitore.

Un processo genitore può porre termine all'esecuzione di uno dei suoi processi figli per diversi motivi, tra i quali i seguenti:

- Il processo figlio ha ecceduto nell'uso di alcune tra le risorse che gli sono state assegnate. Ciò richiede che il processo genitore disponga di un sistema che esamini lo stato dei propri processi figli.
- Il compito assegnato al processo figlio non è più richiesto.
- Il processo genitore termina e il sistema operativo non consente a un processo figlio di continuare l'esecuzione in tale circostanza. In tali sistemi, se un processo termina, sia normalmente sia anormalmente, si devono fermare anche i suoi figli. Questo fenomeno, che si chiama **terminazione a cascata**, è di solito avviato dal sistema operativo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per avere un esempio di esecuzione e terminazione di un processo, si consideri che nel sistema operativo UNIX un processo può terminare per mezzo della chiamata del sistema `exit`, e il suo processo genitore può attendere l'evento per mezzo della chiamata del sistema `wait`. Quest'ultima riporta l'identificatore di un processo figlio che ha terminato l'esecuzione, sicché il genitore può stabilire quale tra i suoi processi figli ha terminato l'esecuzione. Se un processo genitore termina, si affidano tutti i suoi processi figli al processo `init`, che assume il ruolo di nuovo genitore, cui i processi figli possono riportare i risultati delle proprie attività.

4.4 Processi cooperanti

I processi concorrenti in esecuzione nel sistema operativo possono essere indipendenti o cooperanti. Un processo è **indipendente** se non può influire su altri processi nel sistema o subirne l'influsso. Chiaramente, un processo che non condivide dati (temporanei o permanenti) con altri processi è indipendente. D'altra parte un processo è cooperante se influenza o può essere influenzato da altri processi in esecuzione nel sistema. Ovviamente, qualsiasi processo che condivide dati con altri processi è un processo cooperante.

Un ambiente che consente la cooperazione tra processi può essere utile per diverse ragioni:

- **Condivisione d'informazioni.** Poiché più utenti possono essere interessati alle stesse informazioni (ad esempio un file condiviso), è necessario offrire un ambiente che consenta un accesso concorrente a tali risorse.
- **Accelerazione del calcolo.** Alcune attività d'elaborazione si possono eseguire più rapidamente se si suddividono in sottoattività che si possono eseguire in parallelo. Un'accelerazione di questo tipo si può ottenere solo se il calcolatore dispone di più elementi capaci di attività d'elaborazione (come più unità d'elaborazione o canali di I/O).
- **Modularità.** Può essere necessaria la costruzione di un sistema modulare, che divida le funzioni di sistema in processi o thread distinti (si veda il Capitolo 3).
- **Convenienza.** Anche un solo utente può avere la necessità di compiere più attività contemporaneamente; ad esempio, può eseguire in parallelo le operazioni di scrittura, stampa e compilazione.

Un'esecuzione concorrente di processi cooperanti richiede meccanismi che consentano ai processi di comunicare tra loro (Paragrafo 4.5) e di sincronizzare le proprie azioni (Capitolo 7).

Per illustrare il concetto di cooperazione tra processi, si consideri il problema del produttore e del consumatore; tale problema è un usuale paradigma per processi cooperanti. Un processo **produttore** produce informazioni che sono consumate da un processo **consumatore**. Un programma di stampa, ad esempio, produce caratteri che sono consumati dal driver della stampante. Un compilatore può produrre del codice assemblativo

consumato da un assemblatore; l'assemblatore a sua volta può produrre moduli oggetto che sono consumati dal caricatore.

Per permettere un'esecuzione concorrente dei processi produttore e consumatore, occorre disporre di un vettore di elementi che possano essere inseriti dal produttore e prelevati dal consumatore. Un produttore può produrre un elemento mentre il consumatore ne sta consumando un altro. Il produttore e il consumatore devono essere sincronizzati in modo che il consumatore non tenti di consumare un elemento che non è stato ancora prodotto; in questo caso il consumatore deve attendere la produzione di un elemento.

Il problema dei processi produttore e consumatore con memoria illimitata non pone limiti alla dimensione del vettore. Il consumatore può trovarsi ad attendere nuovi oggetti, ma il produttore può sempre produrne. Il problema del produttore e del consumatore con memoria limitata presuppone l'esistenza di una dimensione fissata del vettore. In questo caso, il consumatore deve attendere se il vettore è vuoto, viceversa, il produttore deve attendere se il vettore è pieno. Il vettore può essere fornito dal sistema operativo attraverso l'uso di una funzione di comunicazione tra processi (*interprocess communication* — IPC, si veda il Paragrafo 4.5), oppure codificato esplicitamente, facendo uso di memoria condivisa, dal programmatore dell'applicazione. Di seguito si presenta una soluzione, facente uso di memoria condivisa, del problema con memoria limitata. Il produttore e il consumatore condividono le seguenti variabili:

```
#define DIM_VETTORE 10
typedef struct {
      . . .
} elemento;
elemento vettore[DIM_VETTORE]
int inserisci = 0;
int preleva = 0;
```

Il vettore condiviso è realizzato come un vettore circolare con due puntatori logici: `inserisci` e `preleva`. La variabile `inserisci` indica la successiva posizione libera nel vettore; `preleva` indica la prima posizione piena nel vettore. Il vettore è vuoto se `inserisci == preleva`; il vettore è pieno se `(inserisci + 1) % DIM_VETTORE == preleva`.

Di seguito si riporta il codice per i processi produttore e consumatore. Il processo produttore ha una variabile locale `appena_Prodotto` che contiene il nuovo elemento da produrre:

```
while (1) {
    /* produce un elemento in appena_Prodotto */
    while (((inserisci + 1) % DIM_VETTORE) == preleva)
        ; /* non fa niente */
    vettore[inserisci] = appena_Prodotto;
    inserisci = (inserisci + 1) % DIM_VETTORE;
}
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il processo consumatore ha una variabile locale `da_Consumare` in cui si memorizza l'elemento da consumare:

```
While (1) {
    While (inserisci == preleva)
        ; /* non fa niente */
    da_Consumare = vettore[preleva];
    preleva = (preleva + 1) % DIM_VETTORE;
    /* consuma l'elemento in da_Consumare */
}
```

Questo schema permette di avere al massimo `DIM_VETTORE - 1` elementi contemporaneamente nel vettore. Come esercizio, il lettore può trovare una soluzione che consenta di avere `DIM_VETTORE` elementi contemporaneamente nel vettore.

Nel Capitolo 7 si discute nei dettagli come la sincronizzazione tra processi cooperanti si possa realizzare efficacemente in un ambiente a memoria condivisa.

4.5 Comunicazione tra processi (IPC)

Nel Paragrafo 4.4 si è parlato di come può avvenire la comunicazione tra processi cooperanti in un ambiente a memoria condivisa. Per essere applicato, lo schema proposto richiede che tali processi condividano l'accesso a un vettore di memoria e che il codice per la realizzazione del vettore sia scritto esplicitamente dal programmatore che crea l'applicazione. Un altro modo in cui il sistema operativo può ottenere i medesimi risultati consiste nel fornire gli strumenti per la comunicazione tra processi, realizzando ciò che comunemente prende il nome di sistema di comunicazione tra processi (IPC).

Attraverso le funzioni di IPC, i processi possono comunicare tra loro e sincronizzare le proprie azioni senza condividere lo stesso spazio d'indirizzi. L'IPC è particolarmente utile in un ambiente distribuito dove i processi comunicanti possono risiedere in diversi calcolatori connessi da una rete; un esempio sono i programmi per le chiacchierate in rete (*chat*) che si usano nel World Wide Web.

Le funzioni di comunicazione tra processi sono fornite nel migliore dei modi da un sistema a scambio di messaggi, che a sua volta si può definire in molti modi. In questo paragrafo si considerano diverse questioni riguardanti la progettazione di un sistema di scambio di messaggi.

4.5.1 Sistemi di scambio di messaggi

Un sistema di scambio di messaggi permette ai processi di comunicare tra loro senza ricorrere a dati condivisi. Nel Paragrafo 3.5.3 si è visto lo scambio di messaggi come metodo di comunicazione nei micronuclei. In questo schema i servizi sono forniti sotto forma di ordinari processi utenti; i servizi operano cioè fuori del nucleo. La comunicazione

fra i processi utenti avviene tramite lo scambio di messaggi. Un sistema di IPC fornisce almeno le due operazioni `send(messaggio)` e `receive(messaggio)`.

I messaggi inviati da un processo possono avere dimensione fissa o variabile. Se si possono inviare solo messaggi di dimensione fissa, la realizzazione al livello del sistema è semplice. Tuttavia, questa limitazione rende più difficile il compito della programmazione. D'altra parte, i messaggi di dimensione variabile richiedono una realizzazione più complessa al livello del sistema, ma semplificano la programmazione.

Se i processi *P* e *Q* vogliono comunicare, devono inviare e ricevere messaggi tra loro; deve, dunque, esistere un canale di comunicazione, che si può realizzare in molti modi. In questo paragrafo non si tratta la realizzazione fisica del canale (come la memoria condivisa, i bus o le reti, che sono trattati nel Capitolo 15) ma si tratta la sua realizzazione logica. Ci sono diversi metodi per realizzare al livello logico un canale di comunicazione e le operazioni `send` e `receive`:

- comunicazione diretta o indiretta;
- comunicazione simmetrica o asimmetrica;
- capacità zero, limitata o illimitata delle code di messaggi;
- invio per copiatura o per riferimento;
- messaggi di dimensione fissa o di dimensione variabile.

Nel seguito si discutono questi tipi di messaggi.

4.5.2 Nominazione

Per comunicare, i processi devono disporre di un modo con cui riferirsi agli altri processi; a tale scopo è possibile servirsi di una comunicazione diretta oppure di una comunicazione indiretta.

4.5.2.1 Comunicazione diretta

Con la comunicazione diretta, ogni processo che intenda comunicare deve nominare esplicitamente il ricevente o il trasmittente della comunicazione. In questo schema le funzioni primitive `send` e `receive` si definiscono come segue:

- `send(P, messaggio)`, invia `messaggio` al processo `P`;
- `receive(Q, messaggio)`, riceve, in `messaggio`, un messaggio dal processo `Q`.

All'interno di questo schema, un canale di comunicazione ha le seguenti caratteristiche:

- tra ogni coppia di processi che intendono comunicare si stabilisce automaticamente un canale, i processi devono conoscere solo la reciproca identità;
- un canale è associato esattamente a due processi;
- esiste esattamente un canale tra ciascuna coppia di processi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Questo schema ha una simmetria nell'indirizzamento, vale a dire che per poter comunicare, il trasmittente e il ricevente devono nominarsi a vicenda. Una variante di questo schema si avvale dell'asimmetria dell'indirizzamento: soltanto il trasmittente nomina il ricevente, mentre il ricevente non deve nominare il trasmittente. In questo schema le primitive send e receive si definiscono come segue:

- `send(P, messaggio)`, invia *messaggio* al processo *P*;
- `receive(id, messaggio)`, riceve, in *messaggio*, un messaggio da qualsiasi processo, nella variabile *id* si riporta il nome del processo con cui è avvenuta la comunicazione.

Entrambi gli schemi, simmetrico e asimmetrico, hanno lo svantaggio di una limitata modularità delle risultanti definizioni dei processi. La modificazione del nome di un processo può infatti implicare la necessità di un riesame di tutte le altre definizioni dei processi, per trovare tutti i riferimenti al vecchio nome allo scopo di sostituirli con il nuovo nome. Se si fa una compilazione separata questa situazione non è certo auspicabile.

4.5.2.2 Comunicazione indiretta

Con la **comunicazione indiretta**, i messaggi s'inviano a delle porte (dette anche cassette postali — *mailbox*), che li ricevono. Una porta si può considerare in modo astratto come un oggetto nel quale i processi possono introdurre e prelevare messaggi, ed è identificata in modo unico. In questo schema un processo può comunicare con altri processi tramite un certo numero di porte. Due processi possono comunicare solo se condividono una porta. Le primitive send e receive si definiscono come segue:

- `send(A, messaggio)`, invia *messaggio* alla porta *A*;
- `receive(A, messaggio)`, riceve, in *messaggio*, un messaggio dalla porta *A*.

In questo schema un canale di comunicazione ha le seguenti caratteristiche:

- tra una coppia di processi si stabilisce un canale solo se entrambi i processi della coppia condividono una stessa porta;
- un canale può essere associato a più di due processi;
- tra ogni coppia di processi comunicanti possono esserci più canali diversi, ciascuno corrispondente a una porta.

A questo punto, si supponga che i processi P_1, P_2 e P_3 condividano la porta A. Il processo P_1 invia un messaggio ad A, mentre sia P_2 sia P_3 eseguono una receive da A. Sorge il problema di sapere quale processo riceverà il messaggio. La soluzione dipende dallo schema che si sceglie:

- si può fare in modo che un canale sia associato al massimo a due processi;

- si può consentire a un solo processo alla volta di eseguire un'operazione receive;
- si può consentire al sistema di decidere arbitrariamente quale processo riceverà il messaggio (il messaggio sarà ricevuto da P_2 o da P_3, ma non da entrambi), il sistema può comunicare l'identità del ricevente al trasmittente.

Una porta può appartenere al processo o al sistema. Se appartiene a un processo, cioè fa parte del suo spazio d'indirizzi, occorre distinguere ulteriormente tra il proprietario, che può soltanto ricevere messaggi tramite la porta, e l'utente, che può solo inviare messaggi alla porta. Poiché ogni porta ha un unico proprietario, non può sorgere confusione su chi debba ricevere un messaggio inviato a una determinata porta. Quando un processo che possiede una porta termina, questa scompare, e qualsiasi processo che invii un messaggio alla porta di un processo già terminato deve essere informato della sua scomparsa.

D'altra parte, una porta posseduta dal sistema operativo è indipendente e non è legata ad alcun processo particolare. Il sistema operativo offre un meccanismo che permette a un processo le seguenti operazioni:

- creare una nuova porta;
- inviare e ricevere messaggi tramite la porta;
- rimuovere una porta.

Il processo che crea una nuova porta è il proprietario predefinito della porta, inizialmente è l'unico processo che può ricevere messaggi attraverso questa porta. Tuttavia, il diritto di proprietà e il diritto di ricezione si possono passare ad altri processi per mezzo di idonee chiamate del sistema. Naturalmente questa disposizione potrebbe dare luogo alla creazione di più riceventi per ciascuna porta.

4.5.3 Sincronizzazione

La comunicazione fra processi avviene attraverso chiamate delle primitive send e receive. Ci sono diverse possibilità nella definizione di ciascuna primitiva. Lo scambio di messaggi può essere **sincrono** (o **bloccante**) oppure **asincrono** (o **non bloccante**).

- **Invio sincrono:** il processo che invia il messaggio si blocca nell'attesa che il processo ricevente, o la porta, riceva il messaggio.
- **Invio asincrono:** il processo invia il messaggio e riprende la propria esecuzione.
- **Ricezione sincrona:** il ricevente si blocca nell'attesa dell'arrivo di un messaggio.
- **Ricezione asincrona:** il ricevente riceve un messaggio valido oppure un valore nullo.

È possibile anche avere diverse combinazioni di send e receive tra quelle illustrate sopra. Se le primitive send e receive sono entrambe bloccanti si parla di *rendezvous* tra mittente e ricevente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- Terminazione dei processi
- Messaggi perduti
- Messaggi Alterati

4.5.4 Code di messaggi

Se la comunicazione è diretta o indiretta, i messaggi scambiati tra processi comunicanti risiedono in code temporanee. Fondamentalmente esistono tre modi per realizzare queste code:

- **Capacità zero.** La coda ha lunghezza massima 0, quindi il canale non può avere messaggi in attesa al suo interno. In questo caso il trasmittente deve fermarsi finché il ricevente prende in consegna il messaggio.
- **Capacità limitata.** La coda ha lunghezza finita *n*, quindi al suo interno possono risiedere al massimo *n* messaggi. Se la coda non è piena, quando s'invia un nuovo messaggio, quest'ultimo è posto in fondo alla coda, il messaggio viene copiato oppure si tiene un puntatore a quel messaggio. Il trasmittente può proseguire la propria esecuzione senza essere costretto ad attendere. Ma il canale ha comunque una capacità limitata; se è pieno, il trasmittente deve fermarsi nell'attesa che ci sia spazio disponibile nella coda.
- **Capacità illimitata.** La coda ha una lunghezza potenzialmente infinita, quindi al suo interno può attendere un numero indefinito di messaggi. Il trasmittente non si ferma mai.

Il caso con capacità zero è talvolta chiamato sistema a scambio di messaggi senza memorizzazione transitoria (*no buffering*); gli altri due, con memorizzazione transitoria automatica (*automatic buffering*).

4.5.5 Un esempio: Mach

Come esempio di sistema operativo basato sullo scambio di messaggi, si considera il sistema operativo Mach, sviluppato alla Carnegie Mellon University. Il suo nucleo consente la creazione e la soppressione di più *task*, che sono simili ai processi ma hanno più thread di controllo. La maggior parte delle comunicazioni — compresa una gran parte delle chiamate del sistema e tutte le informazioni tra task — si compie per mezzo di *messaggi*. I messaggi s'inviano e si ricevono attraverso *porte*.

Anche le chiamate del sistema s'invocano per mezzo di messaggi. Al momento della creazione di ogni task si creano due porte speciali: la porta **Kernel** e la porta **Notify**. Il nucleo usa la porta Kernel per comunicare con il task e notifica l'occorrenza di un evento alla porta Notify. Per il trasferimento dei messaggi sono necessarie solo tre chiamate del sistema: `msg_send`, che invia un messaggio a una porta; `msg_receive`, per ricevere un messaggio; `msg_rpc`, per le **chiamate di procedure remote** (*remote procedure call* — RPC), invia un messaggio e ne attende esattamente uno di risposta per il trasmittente (in questo modo la RPC riproduce l'usuale chiamata di procedura, ma può operare tra sistemi diversi).

La chiamata del sistema `port_allocate` crea una nuova porta e assegna lo spazio per la sua coda di messaggi. La dimensione massima predefinita di tale coda è di otto messaggi. Il task che crea la porta è il proprietario della stessa, e può accedervi per la ricezione dei messaggi. Solo un task alla volta può possedere una porta o ricevere da una porta ma, all'occorrenza, questi diritti si possono trasmettere anche ad altri task.

La porta ha inizialmente una coda di messaggi vuota e i messaggi si copiano nella porta nell'ordine in cui sono ricevuti: tutti i messaggi hanno la stessa priorità. Il Mach garantisce che più messaggi in arrivo dallo stesso trasmittente siano accodati nell'ordine d'arrivo (*first-in, first-out* — FIFO), ma non garantisce un ordinamento assoluto. Ad esempio, i messaggi inviati da due trasmittenti possono essere accodati in un ordine qualsiasi.

Gli stessi messaggi sono composti da un'intestazione di lunghezza fissa, seguita da una porzione di dati di lunghezza variabile. L'intestazione contiene la lunghezza del messaggio e due nomi di porte, uno dei quali è il nome della porta cui s'invia il messaggio. Normalmente il thread trasmittente attende una risposta; il nome della porta del trasmittente è passato al task ricevente, che lo impiega come un 'indirizzo del mittente' per inviare messaggi al trasmittente.

La parte variabile del messaggio è composta da una lista di dati tipizzati. Ogni elemento della lista ha un tipo, una dimensione e un valore. Il tipo degli oggetti specificati nel messaggio è importante, poiché oggetti definiti dal sistema operativo, come diritti di proprietà o di ricezione, stati del task e segmenti di memoria, si possono inviare all'interno dei messaggi.

Anche le operazioni di trasmissione e ricezione sono piuttosto flessibili. Ad esempio, quando s'invia un messaggio a una porta che non è già piena, si copia il messaggio al suo interno e il thread trasmittente prosegue la sua esecuzione. Se la porta è piena, il thread trasmittente ha le seguenti possibilità:

1. Attendere indefinitamente che nella porta ci sia spazio.
2. Attendere al massimo *n* millisecondi.
3. Non attendere, ma rientrare immediatamente.
4. Memorizzare temporaneamente il messaggio. Un messaggio si può consegnare al sistema operativo anche se la porta che dovrebbe riceverlo è piena. Quando il messaggio può effettivamente essere messo nella porta, s'invia un avviso al trasmittente; in qualunque momento per un dato thread trasmittente, per una porta piena può restare in sospeso un solo messaggio di questo tipo.

L'ultima possibilità si usa per i task che svolgono servizi (*server task*), come i driver delle stampanti. Dopo aver portato a termine una richiesta, questi task possono aver bisogno d'inviare un'unica risposta al task che aveva richiesto il servizio, ma devono anche proseguire con altre richieste di servizi, anche se la porta di risposta per un client è piena.

Nell'operazione `receive` occorre specificare da quale porta o insieme di porte debba provenire il messaggio. Un **insieme di porte**, dichiarato dal task, si tratta come se fosse un'unica porta. I thread di un task possono ricevere solo da un insieme di porte (o da una porta) per il quale tale task ha il diritto di ricezione. Una chiamata del sistema `port_status` riporta il numero dei messaggi in una data porta. L'operazione di ricezione tenta di ricevere nei modi seguenti:

1. da una qualsiasi tra le porte di un insieme;
2. da una porta specifica (nominata).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Se non è atteso alcun messaggio, il thread ricevente può attendere, attendere al massimo *n* millisecondi o non attendere.

Il sistema Mach è stato progettato per i sistemi distribuiti (trattati nei Capitoli dal 15 al 17), ma è adatto anche a sistemi con singola CPU. I problemi più gravi dei sistemi a scambio di messaggi sono generalmente dovuti alle scarse prestazioni; ciò a causa dell'operazione di copiatura dei messaggi dal trasmittente alla porta e quindi dalla porta al ricevente. Il sistema di messaggi del Mach cerca di evitare doppie operazioni di copiatura impiegando tecniche di gestione della memoria virtuale (Capitolo 10). Fondamentalmente, il Mach associa lo spazio d'indirizzi contenente il messaggio del trasmittente allo spazio d'indirizzi del ricevente. Il messaggio stesso non è mai effettivamente copiato, quindi le prestazioni del sistema migliorano notevolmente anche se solo nel caso di messaggi all'interno dello stesso sistema. Il sistema operativo Mach è descritto nell'Appendice B.

4.5.6 Un esempio: Windows 2000

Il sistema operativo Windows 2000 è un esempio di moderno progetto che impiega la modularità per aumentare la funzionalità e diminuire il tempo necessario alla realizzazione di nuove caratteristiche. Gestisce più ambienti operativi o *sottosistemi* coi quali i programmi d'applicazione comunicano attraverso un meccanismo a scambio di messaggi; i programmi d'applicazione si possono considerare *client* del *server* costituito dal sottosistema.

La funzione di scambio di messaggi del Windows 2000 è detta chiamata di procedura locale (*local procedure call* — LPC) e si usa per la comunicazione tra due processi presenti nello stesso calcolatore. È simile al meccanismo standard della chiamata di procedura remota ma è ottimizzata per questo sistema. Come nel Mach, nel sistema Windows 2000 s'impiega un oggetto porta per stabilire e mantenere una connessione tra due processi. Ogni client che invoca un sottosistema necessita di un canale di comunicazione che è fornito da un oggetto porta e che non è mai ereditato. Esistono due tipi di porte: le porte di connessione e le porte di comunicazione; sono essenzialmente identiche ma ricevono nomi diversi secondo il modo in cui si usano. Le porte di connessione sono *oggetti con nome* visibili a tutti i processi e forniscono alle applicazioni un modo per stabilire un canale di comunicazione (Capitolo 21). Tale comunicazione funziona come segue:

- il client apre una maniglia (*handle*) per l'oggetto porta di connessione del sottosistema;
- il client invia una richiesta di connessione;
- il server crea due porte di comunicazione private e riporta la maniglia per una di esse al client;
- il client e il server impiegano la corrispondente maniglia di porta per inviare messaggi o *callback* e ascoltare le risposte.

Il sistema Windows 2000 si serve di tre tipi di tecniche di scambio di messaggi su una porta che il client specifica quando stabilisce il canale. La più semplice si usa per brevi messaggi (fino a 256 byte) e consiste nell'impiegare la coda dei messaggi della porta come una memoria intermedia per copiare il messaggio da un processo all'altro. Se il client deve inviare un messaggio più lungo, il trasferimento avviene tramite un *oggetto sezione* (memoria condivisa); quando istituisce il canale, il client stabilisce se deve inviare messaggi lunghi, in tal caso richiede la creazione di un oggetto sezione. Allo stesso modo, se il server stabilisce che le risposte sono lunghe, provvede alla creazione di un oggetto sezione. Per usare gli oggetti sezione s'invia un breve messaggio che contiene un puntatore e le informazioni sulle dimensioni dell'oggetto sezione. Questo metodo è un po' più complicato del primo ma evita la copiatura dei dati. In entrambi i casi, se il client o il server non può rispondere immediatamente alla richiesta, si può impiegare un meccanismo di *callback*; tale meccanismo consente di gestire i messaggi in modo asincrono.

4.6 Comunicazione nei sistemi client-server

Si consideri un utente che deve accedere a dati presenti in qualche altro server; potrebbe, ad esempio, voler determinare il numero totale di righe, parole e caratteri di un file presente in un altro server *A*. Questa richiesta d'esecuzione è gestita da un server remoto nel sito *A*, che accede al file, calcola i dati richiesti e invia i risultati all'utente.

4.6.1 Socket

Una *socket* è definita come l'estremità di un canale di comunicazione. Una coppia di processi che comunica attraverso una rete usa una coppia di socket, una per ogni processo, e ogni socket è identificata da un indirizzo IP concatenato a un numero di porta. In generale, le socket impiegano un'architettura client-server: il server attende le richieste dei client, stando in ascolto a una porta specificata; quando il server riceve una richiesta, se accetta la connessione proveniente dalla socket del client, si stabilisce la comunicazione.

I server che svolgono servizi specifici (come telnet, ftp, e http) stanno in ascolto a porte note (i server telnet alla porta 23, i server ftp alla porta 21, e i Web server (http) alla porta 80). Tutte le porte sotto il valore 1024 sono considerate note e si usano per realizzare servizi standard.

Quando un processo client richiede una connessione, il calcolatore che lo esegue assegna una porta specifica, che consiste di un numero arbitrario maggiore di 1024. Si supponga ad esempio che un processo client presente nel calcolatore *x* con indirizzo IP 146.86.5.20 voglia stabilire una connessione con un Web server (in ascolto alla porta 80) all'indirizzo 161.25.19.8; il calcolatore *x* potrebbe assegnare al client, ad esempio, la porta 1625. La connessione sarebbe composta di una coppia di socket: (146.86.5.20:1625) nel calcolatore *x* e (161.25.19.8:80) nel Web server. La Figura 4.9 mostra questa situazione. La consegna dei pacchetti al processo giusto avviene secondo il numero della porta di destinazione.

Tutte le connessioni devono essere uniche; quindi, se un altro processo, nel calcolatore *x*, vuole stabilire un'altra connessione con lo stesso Web server, riceve un numero di porta maggiore di 1024 e diverso da 1625. Ciò assicura che ciascuna connessione sia identificata da una distinta coppia di socket.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

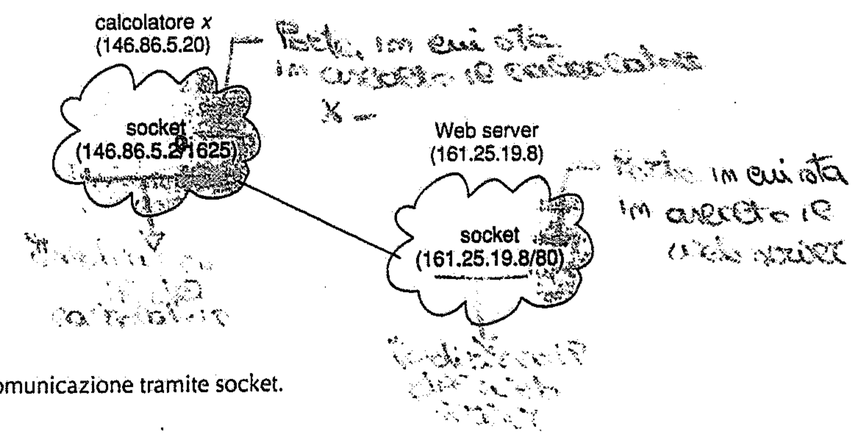


Figura 4.9 Comunicazione tramite socket.

Sebbene la maggior parte degli esempi di programmazione di questo testo sia scritta nel Linguaggio C, le socket sono illustrate usando il linguaggio Java poiché questo offre un'interfaccia alle socket più semplice e dispone di una ricca libreria di strumenti di rete. Il lettore interessato alla programmazione con le socket nel Linguaggio C o C++ può consultare le Note bibliografiche.

Il linguaggio Java prevede tre tipi differenti di socket: quelle **orientate alla connessione** (TCP) sono realizzate con la classe `Socket`; quelle prive di connessione (UDP) usano la classe `DatagramSocket`; il terzo tipo di socket è basato sulla classe `MulticastSocket`; si tratta di una sottoclasse della classe `DatagramSocket` che permette l'invio simultaneo dei dati a diversi destinatari (*multicast*).

Come esempio di socket nell'ambiente Java, si consideri una classe che realizza un server che riporta la data e l'ora correnti (server Orario). Il server sta in ascolto alla porta 5155 — sebbene si possa usare una qualsiasi porta con numero maggiore di 1024 —; quando si stabilisce una connessione, il server riporta al client la data e l'ora correnti.

Il codice del server Orario è riportato nella Figura 4.10. Il server crea una `ServerSocket` che indica che la propria porta d'ascolto è la 5155, quindi si mette in ascolto a tale porta con il metodo `accept`. Il server resta bloccato al metodo `accept` nell'attesa che un client richieda una connessione. Quando si riceve una richiesta di connessione, il metodo `accept` riporta una socket che il server può usare per comunicare con il client.

Si considerino i dettagli della comunicazione del server attraverso la socket. Il server inizialmente crea un oggetto `PrintWriter` per comunicare con il client — tale oggetto permette al server di scrivere in una socket usando i normali metodi `print` e `println` —, quindi invia la data e l'ora al client invocando il metodo `println`. Dopo aver scritto la data e l'ora nella socket, il server chiude la socket e attende una nuova richiesta.

Un client comunica con il server creando una socket e connettendosi alla porta alla quale il server sta in ascolto. Si consideri la codifica del client nel programma scritto in Java della Figura 4.11. Il client crea una socket e richiede una connessione con il server all'indirizzo IP 127.0.0.1 e alla porta 5155. Una volta che la connessione si è stabilita, il


```
import java.net.*;
import java.io.*;

public class Server
{
public static void main(String[] args) throws IOException {
      Socket client = null;
      PrintWriter pout = null;
      ServerSocket sock = null;

      try {
        sock = new ServerSocket(5155);
        // attende richieste di connessione

        while (true) {
          client = sock.accept();

          // c'è una connessione
          pout = new PrintWriter(client.getOutputStream(), true);

          // scrive data e ora nella socket
          pout.println(new java.util.Date().toString());

          pout.close();
          client.close();
        }
      }
      catch (IOException ioe) {
        System.err.println(ioe);
      }
      finally {
        if (client != null)
          client.close();
        if (sock != null)
          sock.close();
      }
    }
}
```

Figura 4.10 Server Orario.

client può leggere dalla socket usando le normali istruzioni per il flusso di I/O. Ricevute la data e l'ora dal server, il client chiude la socket e termina la sua esecuzione. L'indirizzo IP 127.0.0.1 è un indirizzo speciale noto come *localhost*; quando un calcolatore si riferisce all'indirizzo IP 127.0.0.1, si riferisce a se stesso. Questo meccanismo permette ai client e ai server dello stesso calcolatore di comunicare usando il protocollo TCP/IP. L'in-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
import java.net.*;
import java.io.*;

public class Client
{
    public static void main(String[] args) throws IOException {
       InputStream in = null;
       BufferedReader bin = null;
       Socket sock = null;

       try {
          // chiede la connessione alla socket del server
          sock = new Socket("127.0.0.1",5155);

          in = sock.getInputStream();
          bin = new BufferedReader(new InputStreamReader(in));

          String line;
          while ( (line = bin.readLine()) != null)
             System.out.println(line);

       }
       catch (IOException ioe) {
          System.err.println(ioe);

       }
       finally {
          if (sock != null)
             sock.close();
       }
    }
}
```

Figura 4.11 Client.

dirizzo IP 127.0.0.1 si potrebbe sostituire con l'indirizzo IP di un altro calcolatore che esegue il server Orario.

La comunicazione tramite socket, sebbene diffusa ed efficiente, è considerata una forma di comunicazione a basso livello tra processi distribuiti. Una delle ragioni è che le socket permettono lo scambio fra thread comunicanti di una semplicemente sequenza di byte non strutturata, lasciando all'applicazione client o server il compito di dare una struttura ai dati. Nei Paragrafi 4.6.2 e 4.6.3 si presentano due metodi di comunicazione ad alto livello: le chiamate di procedure remote (*remote procedure call* — RPC) e l'invocazione di metodi remoti (*remote method invocation* — RMI).

4.6.2 Chiamate di procedure remote

Uno tra i più diffusi tipi di servizio remoto è il paradigma della RPC. La RPC è stata progettata per astrarre il meccanismo della chiamata di procedura affinché si possa usare tra sistemi collegati tramite una rete. Per molti aspetti è simile al meccanismo IPC descritto nel Paragrafo 4.5, ed è generalmente costruita su un sistema di questo tipo. Poiché in un sistema distribuito si eseguono i processi su sistemi distinti, per offrire un servizio remoto occorre impiegare uno schema di comunicazione basato sullo scambio di messaggi. Contrariamente a quel che accade nella funzione IPC, i messaggi scambiati per la comunicazione RPC sono strutturati e non semplici pacchetti di dati. Si indirizzano a un demone di RPC, in ascolto a una porta del sistema remoto, e contengono un identificatore della funzione da eseguire e i parametri da inviare a tale funzione. Nel sistema remoto si esegue questa funzione e s'invia ogni risultato al richiedente in un messaggio distinto.

La *porta* è semplicemente un numero inserito all'inizio dei pacchetti di messaggi. Mentre un singolo sistema ha normalmente un solo indirizzo di rete, all'interno dell'indirizzo può avere molte porte che servono a distinguere i molti servizi di rete che può fornire. Se un processo remoto richiede un servizio, indirizza i propri messaggi alla porta corrispondente; ad esempio, affinché un sistema permetta ad altri sistemi di ottenere un elenco dei suoi attuali utenti, deve possedere un demone in ascolto a una porta, ad esempio la porta 3027, che realizzi una siffatta RPC. Qualsiasi sistema remoto può ottenere l'informazione richiesta, vale a dire l'elenco degli utenti, inviando un messaggio RPC alla porta 3027 del server; i dati si ricevono in un messaggio di risposta.

La semantica delle RPC permette a un client di invocare una procedura presente in un sistema remoto nello stesso modo in cui esso invocherebbe una procedura locale. Il sistema delle RPC nasconde i dettagli necessari che consentono che la comunicazione avvenga. Il sistema delle RPC ottiene questo risultato assegnando un segmento di codice di riferimento (*stub*) alla parte client. Tipicamente esiste un segmento di codice di riferimento per ogni diversa procedura remota. Quando il client invoca una procedura remota, il sistema delle RPC chiama l'appropriato segmento di codice di riferimento, passando i parametri della procedura remota. Il segmento di codice di riferimento individua la porta del server e struttura i parametri; la strutturazione dei parametri (*marshalling*) implica l'assemblaggio dei parametri in una forma che si può trasmettere tramite una rete. Il segmento di codice di riferimento quindi trasmette un messaggio al server usando lo scambio di messaggi. Un analogo segmento di codice di riferimento nel server riceve questo messaggio e invoca la procedura nel server; se è necessario, riporta i risultati al client usando la stessa tecnica.

Una questione che si deve affrontare riguarda le differenze nella rappresentazione dei dati nel client e nel server. Si consideri la rappresentazione a 32 bit dei numeri interi; alcuni sistemi (ad esempio i mainframe dell'IBM e le CPU Motorola 680x0) usano l'indirizzo di memoria maggiore per contenere il byte *più* significativo (questo sistema di rappresentazione è noto come *big-endian*), altri sistemi (ad esempio i VAX e le CPU Intel 80x86) usano l'indirizzo di memoria maggiore per contenere il byte *meno* significativo

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

(*little-endian*). [Questa terminologia deriva dall'analogia di Cohen (Cohen, D., "On Holy Wars and a Plea for Peace", *IEEE Computer Magazine*, vol. 14, pagg. 48-54, ottobre 1981) con la controversia raccontata nei *Viaggi di Gulliver*, Capitolo IV, riguardante l'estremità — larga o stretta — cui si dovessero rompere le uova per berle.] Per risolvere questo problema, molti sistemi di RPC definiscono una rappresentazione dei dati indipendente dalla macchina. Uno di questi sistemi di rappresentazione è noto come **rappresentazione esterna dei dati** (*external data representation* — XDR). Nel client la strutturazione dei parametri riguarda la conversione dei dati dal formato della specifica macchina nel formato XDR prima di inviarli al server; nel server, i dati nel formato XDR si convertono nel formato della macchina server.

Il meccanismo RPC è comune nei sistemi connessi a una rete di comunicazione; ci sono quindi diversi argomenti da trattare a proposito del suo funzionamento, uno tra i più importanti riguarda la semantica di una chiamata. Mentre le procedure locali non funzionano correttamente solo in circostanze particolarmente gravi, le RPC possono non funzionare correttamente o essere duplicate ed eseguite più volte a causa dei frequenti errori della rete di comunicazione. Poiché si trasferiscono messaggi per mezzo di linee di comunicazione non affidabili, per un sistema operativo è molto più facile assicurare che un messaggio ha avuto effetto almeno una volta, anziché assicurare che il messaggio ha avuto effetto esattamente una volta. Poiché le chiamate di procedure locali hanno proprio il secondo significato, la maggior parte dei sistemi tenta di riprodurlo unendo a ogni messaggio una marca temporale (*timestamp*). Il server tiene una lista cronologica di tutte le marche temporali dei messaggi che ha già elaborato o almeno una lista cronologica abbastanza lunga da assicurare l'individuazione dei messaggi ripetuti; s'ignorano i messaggi in arrivo le cui marche temporali sono già presenti nella lista cronologica. La generazione delle marche temporali è trattata nel Paragrafo 17.1.2.

Un altro argomento importante riguarda la comunicazione tra server e client. Con le ordinarie chiamate di procedure, durante la fase di collegamento, di caricamento o d'esecuzione di un programma (Capitolo 9), ha luogo una forma di associazione che sostituisce il nome della procedura chiamata con l'indirizzo di memoria della procedura stessa. Lo schema delle RPC richiede una corrispondenza di questo genere tra il client e la porta del server; ma c'è il problema di come il client possa conoscere i numeri delle porte del server. Nessun sistema dispone d'informazioni complete sugli altri sistemi poiché essi non condividono memoria. Per risolvere questo problema s'impiegano per lo più due metodi. Nel primo, l'informazione sulla corrispondenza tra il client e la porta del server si può predeterminare fissando gli indirizzi delle porte: una RPC si associa nella fase di compilazione a un numero di porta fisso, il server non può modificare il numero di porta del servizio richiesto. Nel secondo metodo la corrispondenza si può effettuare dinamicamente tramite un meccanismo di *rendezvous*. Generalmente il sistema operativo fornisce un demone di rendezvous (*matchmaker*) a una porta di RPC fissata. Un client invia un messaggio, contenente il nome della RPC, al demone di rendezvous per richiedere l'indirizzo della porta della RPC che si deve eseguire. Il demone risponde col numero di porta, e la richiesta d'esecuzione della RPC si può inviare a quella porta fino al termine del processo (o fino alla caduta del server). Questo metodo richiede un ulteriore carico a

causa della richiesta iniziale, ma è più flessibile del primo metodo. La Figura 4.12 illustra un esempio d'interazione.

Lo schema della RPC è utile nella realizzazione di un file system distribuito, DFS (Capitolo 16); un sistema di questo tipo si può realizzare come un insieme di demoni e client di RPC. I messaggi s'indirizzano alla porta del DFS di un server nel quale deve av-

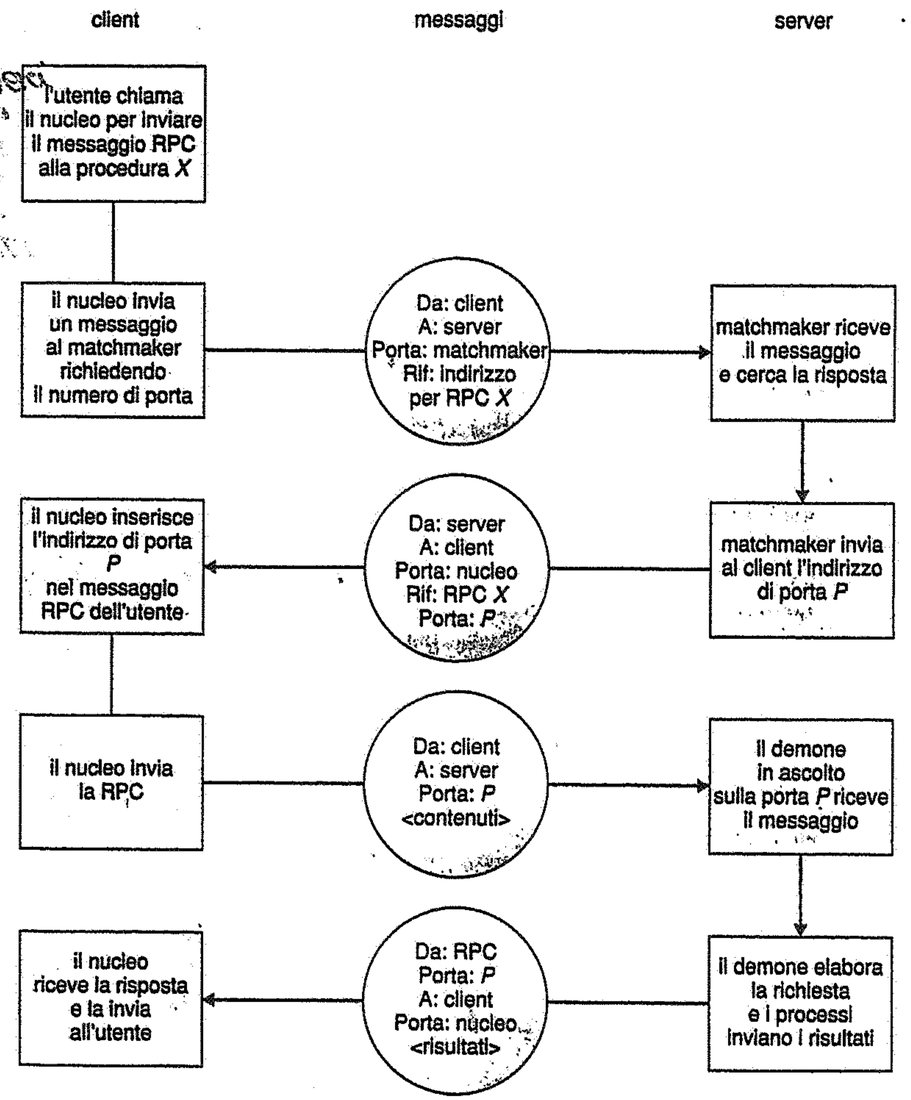


Figura 4.12 Esecuzione di una chiamata di procedura remota (RPC).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

venire l'operazione sui file. I messaggi contengono le operazioni da svolgere nei dischi: `read`, `write`, `rename`, `delete`, `status`, corrispondenti alle normali chiamate del sistema che si usano per i file. Il messaggio di risposta contiene i dati risultanti da quella chiamata, che il demone del DFS esegue su incarico del client. Un messaggio può, ad esempio, contenere una richiesta di trasferimento di un intero file a un client, oppure semplici richieste di blocchi. Nel secondo caso per trasferire un intero file possono essere necessarie parecchie richieste di questo tipo.

→ Metodo di comunicazione nel caso di oggetto

4.6.3 Invocazione di metodi remoti

⇒ Sia variamo object-oriented dell'RPC ⇒ è RMI

L'invocazione di metodi remoti (*remote method invocation* — RMI) è una funzione del linguaggio Java simile alla RPC, permette a un thread di invocare un metodo di un oggetto remoto, dove per remoto s'intende un oggetto residente in una diversa macchina virtuale (JVM) — l'oggetto remoto potrebbe essere in una diversa macchina virtuale presente nello stesso calcolatore oppure in un calcolatore remoto connesso tramite una rete — (Figura 4.13). RMI e RPC differiscono per due motivi fondamentali: innanzi tutto le RPC seguono il paradigma della programmazione procedurale, secondo cui si possono chiamare solo procedure o funzioni remote; le RMI sono orientate agli oggetti e consentono di invocare metodi su oggetti remoti; in secondo luogo, i parametri per le procedure remote sono normali strutture di dati nelle RPC, mentre con le RMI si possono passare oggetti come parametri ai metodi remoti. Permettendo a un programma scritto in Java di chiamare metodi su oggetti remoti, le RMI rendono possibile lo sviluppo di applicazioni distribuite su una rete di calcolatori.

Per rendere i metodi remoti trasparenti sia al client sia al server, le RMI realizzano un oggetto remoto usando un segmento di codice di riferimento (*stub*) nel client e il cosiddetto scheletro (*skeleton*) nel server. Il codice di riferimento è un intermediario (*proxy*) per l'oggetto remoto e risiede nel client. Quando un client invoca un metodo remoto, in realtà viene chiamato il codice di riferimento del client, che struttura (*marshall*) i parametri del metodo e li inserisce in un pacchetto (*parcel*) insieme con il nome del metodo che deve essere invocato nel server. Il codice di riferimento invia il pacchetto al server, dove viene ricevuto dallo scheletro dell'oggetto remoto che destruttura (*unmarshall*) i parametri e invoca nel server il metodo richiesto. In seguito, lo scheletro struttura il valore di ritorno (o dell'eccezione, se si verifica) in un pacchetto che riporta al client. Il codice di riferimento nel client destruttura il valore di ritorno e lo passa al client.

Figura 4.13 Invocazione di metodo remoto.

Nella pratica, questo meccanismo funziona come segue: si assuma che un client voglia invocare, su un oggetto remoto `Server`, un metodo `nomeMetodo(Oggetto, Oggetto)` che riporta un valore booleano. Il client esegue l'istruzione

```
boolean val = Server.nomeMetodo(A, B);
```

La chiamata a `nomeMetodo` con i parametri `A` e `B` invoca il codice di riferimento per l'oggetto remoto, che struttura in un pacchetto i parametri `A` e `B` e il nome del metodo che deve essere invocato nel server, e invia il pacchetto al server. Nel server, lo scheletro destruttura i parametri e invoca il metodo `nomeMetodo`; l'effettivo codice di `nomeMetodo` risiede nel server. Quando il metodo ha completato la sua esecuzione, lo scheletro struttura il valore booleano riportato da `nomeMetodo` e lo invia al client, dove il codice di riferimento lo destruttura e lo passa al client vero e proprio. Il processo è schematizzato nella Figura 4.14.

Fortunatamente, il livello d'astrazione offerto dalle RMI rende i codici di riferimento e gli scheletri trasparenti, permettendo di scrivere programmi che invocano metodi distribuiti come se invocassero metodi locali. Tuttavia, è cruciale che si capiscano alcune regole riguardanti il passaggio dei parametri.

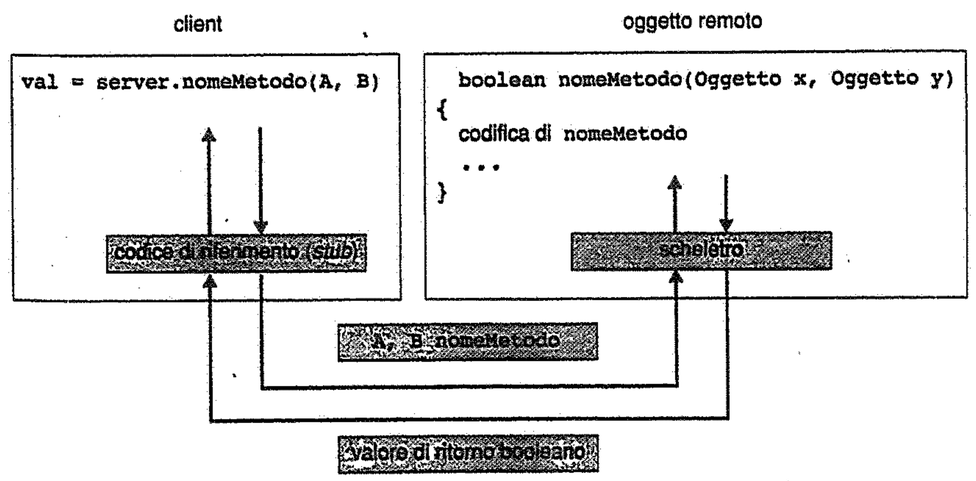


Figura 4.14 Strutturazione dei parametri.

http://www.chinapdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- Se i parametri strutturati sono oggetti locali (o non remoti), sono passati per copiatura, secondo una tecnica detta serializzazione dell'oggetto. I parametri che sono oggetti remoti, sono invece passati per riferimento. Nel nostro esempio, se `A` è un oggetto locale e `B` un oggetto remoto, `A` viene serializzato e passato per copiatura, mentre `B` viene passato per riferimento. Questo permetterebbe a sua volta al server di invocare metodi su `B` in modo remoto.
- Se devono essere passati come parametri agli oggetti remoti, gli oggetti locali devono realizzare l'interfaccia `java.io.Serializable`. Molti oggetti nel nucleo principale dell'API del linguaggio Java realizzano `Serializable`, rendendosi così disponibili per l'uso con le RMI. La serializzazione degli oggetti permette che lo stato di un oggetto si possa scrivere in un flusso di byte.

4.7 Sommario

Un processo è un programma in esecuzione. Nel corso delle sue attività, un processo cambia stato, e tale stato è definito dall'attività corrente del processo stesso. Ogni processo può trovarsi in uno tra i seguenti stati: nuovo, pronto, esecuzione, attesa o arresto. In un sistema operativo ogni processo è rappresentato dal proprio descrittore di processo (PCB).

Un processo, quando non è in esecuzione, è inserito in una coda d'attesa. Le due classi principali di code in un sistema operativo sono le code di richieste di I/O e la coda dei processi pronti per l'esecuzione, quest'ultima contiene tutti i processi pronti per l'esecuzione che si trovino nell'attesa della CPU. Ogni processo è rappresentato da un PCB, e i PCB si possono collegare tra loro in modo da formare una coda dei processi pronti. Lo scheduling a lungo termine (o *job scheduling*) consiste nella scelta dei processi che si contenderanno la CPU. Normalmente lo scheduling a lungo termine è influenzato in modo consistente da considerazioni riguardanti l'assegnazione delle risorse, in particolar modo quelle concernenti la gestione della memoria. Lo scheduling a breve termine (o scheduling della CPU) consiste nella selezione di un processo dalla coda dei processi pronti.

I processi del sistema si possono eseguire in modo concorrente. Esistono parecchie ragioni per permettere tale tipo d'esecuzione: condivisione d'informazioni, accelerazione del calcolo, modularità e convenienza. L'esecuzione concorrente richiede un meccanismo che crei e cancelli i processi.

I processi in esecuzione nel sistema operativo possono essere indipendenti o cooperanti. I processi cooperanti devono avere i mezzi per comunicare tra loro. Fondamentalmente esistono due schemi complementari di comunicazione: memoria condivisa e scambio di messaggi.

Nel metodo con memoria condivisa i processi in comunicazione devono condividere alcune variabili, per mezzo delle quali i processi scambiano informazioni; il compito della comunicazione è lasciato ai programmatori di applicazioni, il sistema operativo deve semplicemente offrire la memoria condivisa. Il metodo con scambio di messaggi

permette di compiere uno scambio di messaggi tra i processi; in questo caso il compito di attuare la comunicazione è del sistema operativo. Questi due schemi non sono mutuamente esclusivi, e si possono impiegare insieme in uno stesso sistema operativo.

Una *socket* si definisce come un punto terminale di comunicazione. Una connessione tra una coppia di applicazioni consiste di una coppia di socket, ciascuna a un'estremità del canale di comunicazione. Le RPC sono un'altra forma di comunicazione distribuita: una RPC si verifica quando un processo (o un thread) invoca una procedura in un'applicazione remota. Le RMI sono la versione del linguaggio Java delle RPC; consentono a un thread di invocare un metodo su un oggetto remoto esattamente come se si trattasse di oggetto locale. La differenza principale tra le RPC e le RMI è che i dati passati a una procedura remota hanno la forma di un'ordinaria struttura di dati, le RMI consentono anche il passaggio di oggetti.

4.8 Esercizi

4.1 L'MS-DOS non offre strumenti di elaborazione concorrente. Descrivete le complicazioni principali che intervengono in un sistema operativo a causa dell'elaborazione concorrente.

4.2 Descrivete le differenze tra scheduling a breve termine, a medio termine e a lungo termine.

4.3 Il calcolatore DECSYSTEM-20 ha insiemi di registri multipli. Descrivete le azioni necessarie all'esecuzione di un cambio di contesto nel caso in cui il nuovo contesto sia già stato caricato in uno di tali insiemi di registri. Descrivete, inoltre, le ulteriori operazioni da compiersi qualora tutti gli insiemi di registri siano già in uso e il nuovo contesto si trovi nella memoria.

4.4 Descrivete le azioni intraprese dal nucleo nell'esecuzione di un cambio di contesto tra processi.

4.5 Considerando sia il livello dei sistemi sia il livello dei programmatori, dite quali sono i vantaggi e quali sono gli svantaggi dei seguenti metodi:

a) comunicazione diretta o indiretta;
b) comunicazione simmetrica o asimmetrica;
c) capacità zero, limitata o illimitata;
d) invio per copiatura o per riferimento;
e) messaggi di lunghezza fissa o di lunghezza variabile.

4.6 L'algoritmo del produttore e del consumatore, illustrato nel Paragrafo 4.4, permette di avere contemporaneamente nel vettore solo $n - 1$ elementi. Modificate l'algoritmo per permettere che il vettore sia pienamente utilizzato.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

4.7 Considerate lo schema di comunicazione tra processi che fa uso di porte.

a) Supponete che un processo *P* attenda due messaggi, uno dalla porta *A* e uno dalla porta *B*. Specificate la sequenza di `send` e `receive` che dovrebbe eseguire.

b) Specificate la sequenza di `send` e `receive` che *P* dovrebbe eseguire se volesse attendere un messaggio dalla porta *A* o dalla porta *B* (o da entrambe).

c) Un'operazione `receive` fa attendere un processo finché la porta è non vuota. Indicate uno schema che permetta a un processo di attendere finché una porta è vuota o spiegate perché uno schema di questo tipo non può esistere.

4.8 Scrivete un server *Indovino*, basato sulle socket, che sta in ascolto a una porta specificata. Quando un client stabilisce una connessione, il server risponde con una profezia scelta a caso dalla sua base di dati di profezie.

4.9 Note bibliografiche

[Brinch Hansen 1970] ha analizzato l'argomento della comunicazione fra processi con riferimento al sistema RC 4000. [Schlichting e Schneider 1982] tratta le primitive per lo scambio di messaggi asincrono. [Bershad et al. 1990] descrive la realizzazione al livello d'utente delle funzioni di IPC.

[Gray 1997] presenta in modo dettagliato la comunicazione tra processi del sistema UNIX. [Barrera 1991] e [Vahalia 1996] trattano la comunicazione tra processi del sistema Mach. [Solomon e Russinovich 2000] descrive la comunicazione tra processi del sistema Windows 2000. In [Stevens 1997] si trova un'approfondita discussione della programmazione con le socket.

[Birrell e Nelson 1984] si occupa della realizzazione delle RPC. [Shrivastava e Panzieri 1982] presenta un progetto per la realizzazione di un meccanismo affidabile di RPC. [Tay e Ananda 1990] presenta una rassegna delle RPC. Infine, [Stankovic 1982] e [Staunstrup 1982] mettono a confronto gli aspetti di comunicazione attraverso le chiamate di procedure e le comunicazioni a scambio di messaggi.

[Tanenbaum 1996] descrive le socket e le RPC. [Waldo 1988] e [Farley 1998] trattano le RPC e le RMI. [Niemeyer e Peck 1997] e [Horstmann e Cornell 1998] offrono una buona introduzione all'uso delle RMI. Risorse aggiornate sulle RMI si trovano all'indirizzo `http://www.javasoft.com/products/jdk/rmi/`.

PAG. 103

Capitolo 5

Thread

Nel modello introdotto nel Capitolo 4, un processo è considerato come un programma in esecuzione con un unico percorso di controllo. Molti sistemi operativi moderni permettono che un processo possa avere più percorsi di controllo che comunemente si chiamano *thread*. Questo capitolo introduce diversi concetti associati ai sistemi di calcolo *multithread*, tra i quali una discussione dell'API **Pthreads** e dei thread nel linguaggio Java. Si esaminano molti aspetti legati alla programmazione multithread, e il modo in cui essa influenza la progettazione dei sistemi operativi; infine, si discute il modo in cui alcuni sistemi operativi moderni gestiscono i thread al livello del nucleo.

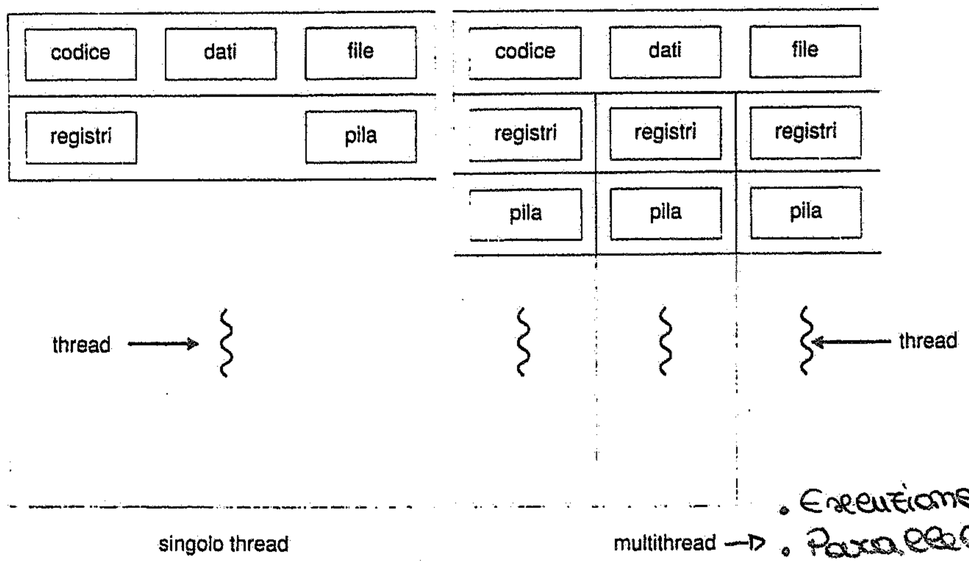
5.1 Introduzione

Un thread, talvolta chiamato **processo leggero** (*lightweight process* — LWP), è l'unità di base d'uso della CPU e comprende un identificatore di thread (ID), un contatore di programma, un insieme di registri, e una pila (*stack*). Condivide con gli altri thread che appartengono allo stesso processo la sezione del codice, la sezione dei dati e altre risorse di sistema, come i file aperti e i segnali. Un processo tradizionale, chiamato anche **processo pesante** (*heavyweight process*), è composto da un solo thread. Un processo multithread è in grado di lavorare a più compiti in modo concorrente. La Figura 5.1 mostra la differenza tra un processo tradizionale, a singolo thread, e uno multithread.

5.1.1 Motivazioni

Molti programmi per i moderni PC sono predisposti per essere eseguiti da processi **multithread**. Di solito, un'applicazione si codifica come un processo a sé stante comprendente più thread di controllo: un programma di consultazione del Web potrebbe avere un thread per la rappresentazione sullo schermo delle immagini e del testo, mentre un altro thread potrebbe occuparsi del reperimento dei dati nella rete; un elaboratore di testi potrebbe avere un thread per la rappresentazione grafica, uno per la lettura dei dati immessi con la tastiera e uno per la correzione ortografica e grammaticale eseguita in sottofondo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.



• Esecuzione concorrente.
• Parallelismo.

Figura 5.1 Processi a singolo thread e multithread.

In alcune situazioni, una singola applicazione deve poter gestire molti compiti simili tra loro. Ad esempio, un Web server accetta dai client richieste di pagine Web, immagini, suoni, e altro. Per un noto Web server potrebbero esservi molti (forse centinaia) client che vi accedono in modo concorrente; se il Web server fosse eseguito come un processo tradizionale a **singolo thread**, sarebbe in grado di soddisfare un solo client alla volta. Il tempo che un client potrebbe dover aspettare prima che la propria richiesta sia soddisfatta potrebbe essere enorme.

Una soluzione è eseguire il server come un singolo processo che accetta richieste. Quando riceve una richiesta, il server crea un processo separato per eseguirla. In effetti, questo metodo di creazione di processi era molto usato prima che si diffondesse la possibilità di gestione dei thread. Come si rileva nel Capitolo 4, la creazione dei processi è molto onerosa; se il nuovo processo si deve occupare degli stessi compiti del processo corrente, non c'è alcuna ragione di accettare tutto il carico che la sua creazione comporta. Generalmente, per raggiungere lo stesso obiettivo è più conveniente impiegare un processo multithread. Nel caso del Web server, l'adozione di questo metodo porterebbe alla scelta di un processo multithread: il server genererebbe un thread distinto per ricevere eventuali richieste dei client; alla presenza di una richiesta, anziché creare un altro processo, si creerebbe un altro thread per soddisfarla.

• Il thread è all'interno del contesto di un processo
↓
Possiede gli stessi stati del processo e così
come il processo può essere eseguito in modo atomico
o in modo di sub-task.

I thread hanno anche un ruolo primario nei sistemi che impiegano le RPC (*remote procedure call*); si tratta di un sistema che permette la comunicazione tra processi, fornendo un meccanismo di comunicazione simile alle normali chiamate di funzione o procedura (Capitolo 4). Di solito, i server RPC sono multithread; quando riceve un messaggio, il server delega la gestione a un thread separato, in questo modo può gestire diverse richieste in modo concorrente.

5.1.2 Vantaggi

I vantaggi della programmazione multithread si possono classificare rispetto a quattro fattori principali:

1. **Tempo di risposta.** Rendere multithread un'applicazione interattiva può permettere a un programma di continuare la sua esecuzione, anche se una parte di esso è bloccata o sta eseguendo un'operazione particolarmente lunga, riducendo il tempo di risposta medio all'utente. Ad esempio, un programma di consultazione del Web multithread potrebbe permettere l'interazione con l'utente tramite un thread mentre un'immagine sarebbe caricata da un altro thread.
2. **Condivisione delle risorse.** Normalmente, i thread condividono la memoria e le risorse del processo cui appartengono. Il vantaggio della condivisione del codice consiste nel fatto che un'applicazione può avere molti thread di attività diverse, tutti nello stesso spazio d'indirizzi.
3. **Economia.** Assegnare memoria e risorse per la creazione di nuovi processi è oneroso; poiché i thread condividono le risorse del processo cui appartengono, è molto più conveniente creare thread e gestirne i cambiamenti di contesto. È difficile misurare empiricamente la differenza del carico richiesto per creare e gestire un processo invece che un thread, tuttavia la creazione e gestione dei processi richiede in generale molto più tempo. Nel Solaris 2, la creazione di un processo richiede un tempo trenta volte maggiore di quello richiesto per la creazione di un thread, un cambio di contesto per un processo richiede un tempo pari a circa cinque volte quello richiesto per un thread.
4. **Uso di più unità d'elaborazione.** I vantaggi della programmazione multithread aumentano notevolmente nelle architetture con più unità d'elaborazione, dove i thread si possono eseguire in parallelo (uno per ciascuna unità d'elaborazione). Un processo a singolo thread è eseguito da una singola unità d'elaborazione, indipendentemente dal numero di unità d'elaborazione disponibili. L'impiego della programmazione multithread in un sistema con più unità d'elaborazione fa aumentare il grado di parallelismo. In un'architettura con una singola unità d'elaborazione i thread si avvicendano rapidamente nell'uso della CPU creando l'illusione di un'esecuzione parallela, in realtà si esegue un unico thread alla volta.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

5.1.3 Thread al livello d'utente e thread al livello del nucleo

La gestione dei thread può avvenire sia al livello d'utente sia al livello del nucleo.

- I **thread al livello d'utente** sono gestiti come uno strato separato sopra il nucleo del sistema operativo, e sono realizzati tramite una libreria di funzioni per la creazione, lo scheduling e la gestione senza alcun intervento diretto del nucleo. Poiché il nucleo del sistema operativo non vede i thread del livello d'utente, la creazione dei thread e lo scheduling avvengono nello spazio d'utente senza richiedere l'intervento del nucleo, sicché la creazione e la gestione di questi thread richiede generalmente poco tempo. Ci sono tuttavia degli svantaggi: se il nucleo è a singolo thread, ogni thread d'utente che esegue una chiamata del sistema bloccante causa il blocco dell'intero sistema, anche se altri thread dell'applicazione sono disponibili per l'esecuzione. Tra le librerie di thread al livello d'utente ci sono la libreria POSIX **Pthreads**, la **C-threads** del sistema Mach e la **UI-threads** del Solaris 2.
- I **thread al livello del nucleo** sono gestiti direttamente dal sistema operativo: il nucleo si occupa della creazione, scheduling e gestione nello spazio d'indirizzi del nucleo. Poiché sono gestiti dal sistema operativo, i thread al livello del nucleo sono generalmente più lenti da creare e gestire dei thread al livello d'utente. Tuttavia, poiché è il nucleo che gestisce i thread, se un thread esegue una chiamata del sistema bloccante, il nucleo può attivare l'esecuzione di un altro thread dell'applicazione. Inoltre, in un ambiente con più unità d'elaborazione, il nucleo può far eseguire i thread da unità d'elaborazione diverse. La maggior parte dei sistemi operativi moderni, tra i quali Windows NT, Windows 2000, Solaris 2, BeOS e Tru64 UNIX, gestiscono i thread al livello del nucleo.

La libreria Pthreads è trattata nel Paragrafo 5.4 come esempio di libreria di thread al livello d'utente. Come esempi di sistemi operativi che gestiscono i thread al livello del nucleo sono trattati il sistema Windows 2000 (Paragrafo 5.6), il Solaris 2 (Paragrafo 5.5), e (sebbene non li consideri esattamente dei *thread*) il sistema LINUX (Paragrafo 5.7). Anche il linguaggio Java gestisce i thread; tuttavia, poiché i thread del linguaggio Java sono creati e gestiti dalla macchina virtuale (JVM), non si possono considerare né al livello del nucleo, né al livello d'utente (Paragrafo 5.8).

5.2 Modelli di programmazione multithread

La pluralità di sistemi che gestiscono i thread, sia al livello d'utente sia al livello del nucleo, ha portato alla definizione di diversi modelli di programmazione multithread; di seguito si analizzano tre fra i più diffusi.

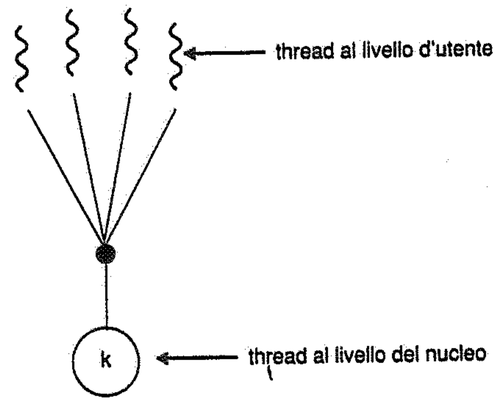


Figura 5.2 Modello da molti a uno.

5.2.1 Modello da molti a uno

Il modello da molti a uno (Figura 5.2) fa corrispondere molti thread al livello d'utente a un singolo thread al livello del nucleo. La gestione dei thread è efficiente poiché si svolge nello spazio d'utente, ma l'intero processo rimane bloccato se un thread invoca una chiamata del sistema di tipo bloccante. Inoltre, poiché un solo thread alla volta può accedere al nucleo, è impossibile eseguire thread multipli in parallelo in calcolatori dotati di più unità d'elaborazione; la libreria **green threads**, disponibile per il Solaris 2, usa questo modello. Le librerie di thread al livello d'utente, realizzate per sistemi operativi che non gestiscono i thread al livello del nucleo adottano il modello da molti a uno.

5.2.2 Modello da uno a uno

Il modello da uno a uno (Figura 5.3) mette in corrispondenza ciascun thread del livello d'utente con un thread del livello del nucleo. Questo modello offre un grado di concorrenza maggiore, poiché anche se un thread invoca una chiamata del sistema bloccante, è possibile eseguire un altro thread; il modello permette anche l'esecuzione dei thread in parallelo nei sistemi con più unità d'elaborazione. L'unico svantaggio di questo modello è che la creazione di ogni thread al livello d'utente comporta la creazione del corrispondente thread al livello del nucleo. Poiché il carico dovuto alla creazione di un thread al livello del nucleo può compromettere le prestazioni di un'applicazione, la maggior parte delle realizzazioni di questo modello limita il numero di thread gestibili dal sistema. I sistemi operativi Windows NT, Windows 2000, e OS/2 adottano il modello da uno a uno.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

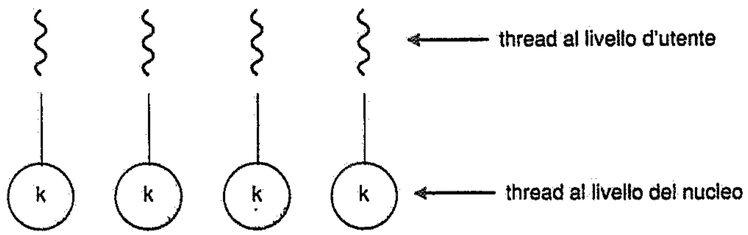


Figura 5.3 Modello da uno a uno.

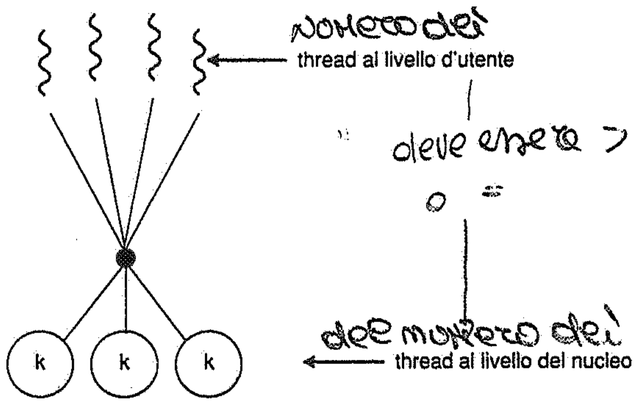


Figura 5.4 Modello da molti a molti.

5.2.3 Modello da molti a molti

Il modello da molti a molti (Figura 5.4) mette in corrispondenza più thread del livello d'utente con un numero minore o uguale di thread del livello del nucleo; quest'ultimo può essere specifico per una certa applicazione o per un particolare calcolatore (un'applicazione potrebbe assegnare un numero maggiore di thread del livello del nucleo a un'architettura con più unità d'elaborazione rispetto quanti ne assegnerebbe a una con singola CPU). Nonostante il modello da molti a uno permetta ai programmatori di creare tanti thread del livello d'utente quanti ne desiderino, non si realizzata una concorrenza reale poiché il meccanismo di scheduling del nucleo può scegliere un solo thread alla volta. Il modello da uno a uno permette una maggiore concorrenza, ma i programmatori devono stare attenti a non creare troppi thread all'interno di un'applicazione (in qualche caso si possono avere specifiche limitazioni sul numero di thread che si possono creare). Il modello da molti a molti non ha nessuno di questi difetti: i programmatori possono creare liberamente i thread che ritengono necessari, e i corrispondenti thread del livello del nucleo si possono eseguire in parallelo nelle architetture dotate di più unità d'elaborazione. Inoltre, se un thread impiega una chiamata del sistema bloccante, il nucleo può fare in modo che si esegua un altro thread. I sistemi operativi Solaris 2, IRIX, HP-UX, e Tru64 UNIX impiegano questo modello.

5.3 Questioni di programmazione multithread

In questo paragrafo si affrontano alcune questioni legate ai programmi multithread.

5.3.1 Chiamate del sistema fork ed exec

Nel Capitolo 4 è descritto l'uso della chiamata del sistema fork per la creazione di un nuovo processo tramite la duplicazione di un processo esistente. In un programma multithread la semantica delle chiamate del sistema fork ed exec cambia: se un thread in un programma invoca la chiamata del sistema fork, il nuovo processo potrebbe, in generale, contenere un duplicato di tutti i thread oppure del solo thread invocante. Alcuni sistemi UNIX includono entrambe le versioni. La chiamata del sistema exec di solito funziona nello stesso modo descritto nel Capitolo 4: se un thread invoca la chiamata del sistema exec, il programma specificato come parametro della exec sostituisce l'intero processo, inclusi tutti i thread. L'uso delle due versioni della fork dipende dall'applicazione. Se s'invoca la exec immediatamente dopo la fork, la duplicazione dei thread non è necessaria, poiché il programma specificato nei parametri della exec sostituirà il processo. In questo caso conviene duplicare il solo thread chiamante. Tuttavia, se la exec non segue immediatamente la fork, potrebbe essere utile una duplicazione di tutti i thread del processo genitore.

5.3.2 Cancellazione

La **cancellazione dei thread** è l'operazione che permette di terminare un thread prima che completi il suo compito. Ad esempio, se più thread stanno facendo una ricerca in modo concorrente in una base di dati e un thread riporta il risultato, gli altri thread possono essere annullati. Una situazione analoga potrebbe verificarsi quando un utente preme il pulsante di terminazione di un programma di consultazione del Web per interrompere il caricamento di una pagina. Spesso il caricamento di una pagina è gestito da un thread distinto; quando l'utente preme il pulsante di terminazione, il thread che sta caricando la pagina viene cancellato. Un thread da cancellare è spesso chiamato **thread bersaglio**. La cancellazione di un thread bersaglio può avvenire in due modi diversi:

1. cancellazione asincrona: un thread fa immediatamente terminare il thread bersaglio;
2. cancellazione differita: il thread bersaglio può periodicamente controllare se deve terminare, in modo da riuscirvi in modo opportuno.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Si presentano difficoltà con la cancellazione nei casi in cui ci siano risorse assegnate a un thread cancellato, o se si cancella un thread mentre sta aggiornando dei dati che condivide con altri thread. Quest'ultimo caso è particolarmente problematico se si tratta di cancellazione asincrona. Il sistema operativo di solito si riappropria delle risorse di sistema usate da un thread cancellato, ma spesso non si riappropria di tutte le risorse. Quindi, la cancellazione di un thread in modo asincrono potrebbe non liberare una risorsa necessaria per tutto il sistema.

La cancellazione differita invece funziona tramite un thread che segnala la necessità di cancellare un certo thread bersaglio; la cancellazione avviene soltanto quando il thread bersaglio verifica se deve essere cancellato oppure no. Questo metodo permette di programmare la verifica in un punto dell'esecuzione in cui il thread può essere cancellato senza problemi. Nella libreria Pthreads questi punti si chiamano **punti di cancellazione** (*cancellation point*).

La maggior parte dei sistemi operativi permette la cancellazione asincrona dei processi o dei thread. Tuttavia, poiché l'API Pthreads permette la cancellazione differita, ogni sistema operativo che l'impiega permette implicitamente questo metodo di cancellazione.

5.3.3 Gestione dei segnali

Nei sistemi UNIX si usano i segnali per comunicare ai processi il verificarsi di determinati eventi. Un segnale si può ricevere in modo sincrono o asincrono, secondo la sorgente e la ragione della segnalazione dell'evento. Indipendentemente dal modo di ricezione sincrono o asincrono, tutti i segnali seguono lo stesso schema:

1. all'occorrenza di un particolare evento si genera un segnale;
2. s'invia il segnale a un processo;
3. una volta ricevuto, il segnale deve essere gestito.

segnali intercettati dall'hardware

Un accesso illegale alla memoria o una divisione per zero generano segnali sincroni. In questi casi, se un programma in esecuzione compie le suddette azioni, viene generato un segnale. I segnali sincroni s'inviano allo stesso processo che ha eseguito l'operazione causa del segnale (perciò si chiamano sincroni).

Quando un segnale è causato da un evento esterno al processo in esecuzione, tale processo riceve il segnale in modo asincrono. Esempi di segnali di questo tipo sono la terminazione di un processo richiesta con specifiche combinazioni di tasti (come Control + C) oppure la scadenza di un temporizzatore. Di solito un segnale asincrono s'invia a un altro processo.

segnali generati dall'utente

Ogni segnale si può gestire in due modi:

1. tramite un gestore predefinito di segnali;
2. tramite un gestore di segnali definito dall'utente.

Per ogni segnale esiste un **gestore predefinito del segnale** che il nucleo esegue quando deve gestire il segnale. La gestione predefinita può essere sostituita da una funzione di **gestione del segnale definita dall'utente**.

In questo caso, per gestire il segnale si chiama la funzione definita dall'utente al posto di quella predefinita. Sia i segnali sincroni sia quelli asincroni si possono gestire in modi diversi: alcuni si possono semplicemente ignorare (ad esempio, il ridimensionamento di una finestra); altri si possono gestire terminando l'esecuzione del programma (ad esempio, un accesso illegale alla memoria).

Per i processi a singolo thread la gestione dei segnali è semplice poiché nell'inviare un segnale è sufficiente fare riferimento al processo interessato. Per i processi multithread si pone il problema del thread cui si deve inviare il segnale. In generale esistono le seguenti possibilità:

1. inviare il segnale al thread cui il segnale si riferisce;
2. inviare il segnale a ogni thread del processo;
3. inviare il segnale a specifici thread del processo;
4. definire un thread specifico per ricevere tutti i segnali diretti al processo.

Il metodo per recapitare un segnale dipende dal tipo di segnale. I segnali sincroni, ad esempio, si devono inviare al thread che ha generato l'evento causa del segnale e non ad altri thread nel processo. Se si tratta di segnali asincroni la situazione non è invece così chiara; alcuni segnali asincroni — come il segnale che termina un processo (Control + C, ad esempio) — si devono inviare a tutti i thread. Alcune versioni multithread del sistema operativo UNIX permettono che per ciascun thread si indichino i segnali da accettare e i segnali da bloccare. Quindi, alcuni segnali asincroni si potrebbero recapitare soltanto ai thread che non li bloccano. Tuttavia, poiché ogni segnale deve essere gestito una sola volta, di solito raggiunge solo il primo fra tutti i thread del processo che non bloccano il segnale.

Il sistema Solaris 2 impiega la quarta alternativa: crea per ciascun processo un thread specifico per la gestione dei segnali. Quando s'invia un segnale asincrono a un processo, si fa riferimento a questo specifico thread, che a sua volta passa il segnale al primo thread che non lo blocca.

Il sistema operativo Windows 2000 non prevede la gestione esplicita dei segnali, ma questi si possono emulare con le **chiamate di procedure asincrone** (*asynchronous procedure call* — APC). Le funzioni APC permettono a un thread del livello d'utente di specificare la funzione da chiamare quando il thread riceve la comunicazione di un particolare evento. Come s'intuisce dal nome, una APC è grosso modo equivalente a un segnale asincrono dello UNIX. Ma mentre in un ambiente multithread lo UNIX necessita di un criterio di gestione dei segnali, il sistema delle APC è più semplice poiché una APC è rivolta a un particolare thread e non a un processo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

5.3.4 Gruppi di thread

Nel Paragrafo 5.1, è descritto lo scenario di un Web server multithread, in cui per ogni richiesta ricevuta, il server crea un thread distinto per fornire il servizio richiesto. Nonostante la creazione di un thread distinto sia molto più vantaggiosa della creazione di un nuovo processo, un server multithread presenta diversi problemi. Il primo riguarda il tempo richiesto per la creazione del thread prima di poter soddisfare la richiesta, considerando anche il fatto che questo thread sarà terminato non appena avrà completato il proprio lavoro.

La seconda questione è più problematica: se si permette che tutte le richieste concorrenti siano servite da un nuovo thread, non si è posto un limite al numero di thread attivi in modo concorrente nel sistema. Un numero illimitato di thread potrebbe esaurire le risorse del sistema, come il tempo di CPU o la memoria.

L'impiego dei **gruppi di thread** è una possibile soluzione a questo problema. L'idea generale è quella di creare un certo numero di thread alla creazione del processo, e organizzarli in un *gruppo* nel quale attendano il lavoro che gli sarà richiesto. Quando un server riceve una richiesta, attiva un thread del gruppo — se ce ne è uno disponibile — e gli passa la richiesta; dopo aver completato il suo lavoro, il thread rientra nel gruppo d'attesa. Se il gruppo non contiene alcun thread disponibile, il server attende fino al rientro di un thread. In particolare, si hanno i seguenti vantaggi:

1. di solito il servizio di una richiesta tramite un thread esistente è più rapido poiché elimina l'attesa della creazione di un nuovo thread;
2. un gruppo di thread limita il numero di thread esistenti a un certo istante; ciò è particolarmente rilevante per sistemi che non possono sostenere un elevato numero di thread concorrenti.

Il numero di thread di un gruppo si può determinare tramite euristiche che considerano fattori come il numero di unità d'elaborazione nel sistema, la quantità di memoria fisica e il numero atteso di richieste concorrenti da parte dei client. Architetture più raffinate per la gestione dei gruppi di thread possono correggere dinamicamente il numero di thread di un gruppo secondo schemi d'uso. Queste architetture hanno l'ulteriore vantaggio di avere gruppi più piccoli — quindi un minore impegno della memoria — quando il carico del sistema è basso.

5.3.5 Dati specifici dei thread

Uno dei vantaggi principali della programmazione multithread è dato dal fatto che i thread che appartengono allo stesso processo ne condividono i dati. Tuttavia, in particolari circostanze, ogni thread può necessitare di una copia privata di certi dati, chiamati **dati specifici di thread**. Ad esempio, in un sistema per transazioni si può svolgere ciascuna transazione tramite un thread distinto e un identificatore unico per ogni transazione. Per associare ciascun thread al relativo identificatore si possono usare dati specifici dei thread. La maggior parte delle librerie di thread — incluse la Win32 e la Pthreads — e l'ambiente del linguaggio Java consentono l'impiego dei dati specifici di thread.

www.IEEE.org

Da considerarsi come insieme di libreria a più thread a livello utente.

5.4 Pthreads

(cosa è permette la programmazione multithread)

Col termine Pthreads ci si riferisce allo standard POSIX (IEEE 1003.1c) che definisce l'API per la creazione e la sincronizzazione dei thread. Non si tratta di una *realizzazione*, ma di una *definizione* del comportamento dei thread; i progettisti di sistemi operativi possono realizzare le API così definite come meglio credono. Di solito le librerie Pthreads sono limitate ai sistemi UNIX, come il Solaris 2. I sistemi operativi della famiglia Windows generalmente non dispongono della libreria Pthreads, sebbene siano disponibili versioni di pubblico dominio.

interfaccia

Come esempio di una libreria di thread al livello d'utente, in questo paragrafo sono descritte alcune API Pthreads. Si considerano al livello d'utente poiché non esiste alcuna relazione definita fra i thread creati usando le API Pthreads e i corrispondenti thread al livello del nucleo. Il programma scritto in Linguaggio C nella Figura 5.5 mostra l'uso delle API Pthreads di base per la codifica di un programma multithread. I lettori interessati a maggiori dettagli sulle API Pthreads possono consultare le Note bibliografiche.

Il programma riportato nella Figura 5.5 crea un thread distinto che fa la somma di tutti gli interi positivi fino all'intero non negativo fornito come argomento. In un programma che impiega la libreria Pthreads, i thread cominciano la loro esecuzione in una specifica funzione. Nel programma in esame si tratta della funzione `runner`. Quando comincia l'esecuzione del programma, c'è un unico thread di controllo che parte da `main`; dopo una fase d'inizializzazione, `main` crea un secondo thread che comincia l'esecuzione dalla funzione `runner`.

Tutti i programmi che impiegano la libreria Pthreads devono includere il file d'intestazione `pthread.h`. La dichiarazione di variabili `pthread_t tid` specifica l'identificatore per il thread da creare. Ogni thread ha un insieme di attributi che includono la dimensione della pila e informazioni di scheduling. La dichiarazione `pthread_attr_t attr` riguarda la struttura di dati per gli attributi del thread, i cui valori si assegnano con la chiamata di funzione `pthread_attr_init(&attr)`. Poiché non sono stati esplicitamente forniti valori per gli attributi, si usano quelli predefiniti. La chiamata di funzione `pthread_create` crea un nuovo thread. Oltre all'identificatore del thread e ai suoi attributi, si passa anche il nome della funzione dalla quale il nuovo thread comincerà l'esecuzione, in questo caso la funzione `runner`, e il numero intero fornito come parametro alla riga di comando e individuato da `argv[1]`. A questo punto il programma ha due thread: il thread iniziale, in `main`; e il thread che esegue la somma, in `runner`. Dopo aver creato il secondo, il primo thread attende il completamento del secondo chiamando la funzione `pthread_join`. Il secondo thread termina quando s'invoca la funzione `pthread_exit`.

PTHREADS → definisce le funzioni e le interfacce di programmazione alle applicazioni (API) x la gestione e sincronizzazione dei thread.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
#include <pthread.h>
#include <stdio.h>

int somma; /* questo dato è condiviso dal/dai thread */
void *runner(void *param); /* il thread */

main(int argc, char *argv[])
{
    pthread_t tid; /* identificatore del thread */
    pthread_attr_t attr; /* insieme di attributi del thread */
    if (argc != 2) {
       fprintf(stderr,"uso: a.out <valore intero>\n");
       exit();
    }
    if (atoi(argv[1]) < 0) {
       fprintf(stderr,"%d deve essere >= 0\n", atoi(argv[1]));
       exit();
    }
    /* reperisce gli attributi predefiniti */
    pthread_attr_init(&attr);
    /* crea il thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* attende la terminazione del thread */
    pthread_join(tid,NULL);
    printf("somma = %d\n",somma);
}

/* Il thread assume il controllo da questa funzione */
void *runner(void *param)
{
    int sup = atoi(param);
    int i;
    somma = 0;
    if (sup > 0) {
       for (i = 1; i <= sup; i++)
          somma += i;
    }
    pthread_exit(0);
}
```

Figura 5.5 Un programma multithread in Linguaggio C che impiega l'API Pthreads.

5.5 Thread del sistema Solaris 2

Il sistema operativo Solaris 2 è una versione del sistema UNIX che prevede i thread del livello d'utente e del livello del nucleo, la SMP e lo scheduling per elaborazioni in tempo reale. Per quel che riguarda i thread al livello d'utente, dispone della libreria API Pthreads, presentata nel Paragrafo 5.4, e di una libreria, nota come UI-threads. Le differenze tra queste due librerie sono minime, sebbene la libreria Pthreads sia attualmente la preferita dalla maggior parte dei programmatori. Il sistema Solaris 2 definisce anche un livello intermedio di thread; fra i thread del livello d'utente e quelli del livello del nucleo ci sono i cosiddetti processi leggeri o LWP. Ogni processo contiene almeno un LWP. La libreria dei thread mette dinamicamente in corrispondenza i thread del livello d'utente col gruppo di LWP per quel processo; possono essere in esecuzione i soli thread del livello d'utente che sono correntemente associati a un LWP. Gli altri sono bloccati oppure attendono gli LWP che possano eseguirli.

I thread ordinari del livello del nucleo eseguono tutte le operazioni dentro il nucleo. Ogni LWP ha un corrispondente thread del livello del nucleo; alcuni thread del livello del nucleo sono invece eseguiti per conto dello stesso nucleo e non hanno alcun LWP a essi associato (ad esempio un thread che serve richieste d'accesso a un disco). I thread del livello del nucleo sono gli unici oggetti cui il sistema applica l'algoritmo di scheduling (Capitolo 6). Il sistema Solaris 2 impiega il modello di gestione da molti a molti; il suo completo sistema di thread è riassunto nella Figura 5.6.

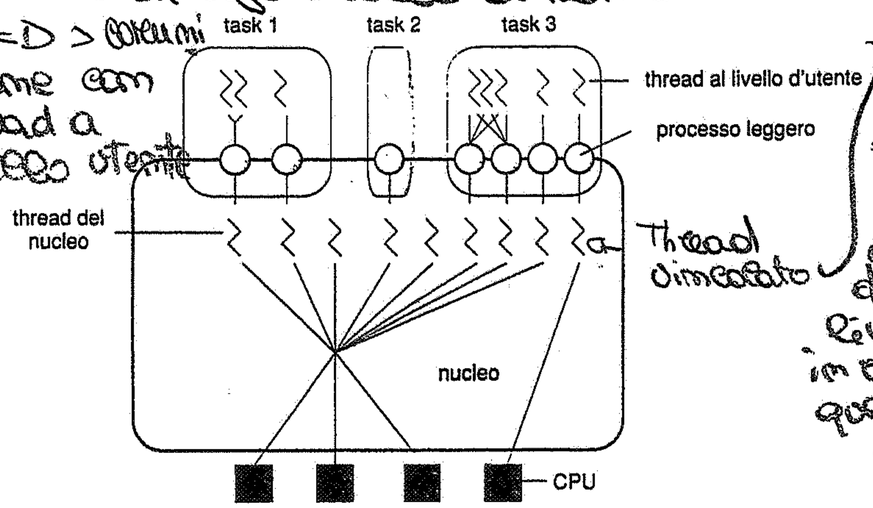
I thread del livello d'utente possono essere vincolati o svincolati. Un thread del livello d'utente vincolato è associato in modo permanente a un LWP. Soltanto quel thread può essere eseguito sul LWP e, a richiesta, il LWP può essere riservato a una specifica unità d'elaborazione (si veda il thread più a destra della Figura 5.6). Il vincolo di un thread può essere utile nei casi che richiedono rapide risposte, come nelle applicazioni d'elaborazione in tempo reale. I thread svincolati non sono associati in modo permanente ad alcun LWP, ma si associano dinamicamente a un gruppo di LWP disponibili per l'applicazione. La modalità predefinita per i thread è quella di essere svincolati. Il sistema Solaris 8 fornisce una libreria alternativa di thread che, per convenzione, vincola tutti i thread a un corrispondente LWP.

Si consideri ciò che avviene nel sistema: ciascun processo può avere molti thread del livello d'utente che possono essere selezionati dallo scheduler e quindi associati agli LWP dalla libreria dei thread senza l'intervento del nucleo. I thread del livello d'utente sono estremamente efficienti perché non richiedono l'intervento del nucleo per la creazione e l'eliminazione dei thread; né per conto della libreria dei thread per i cambi di contesto fra thread del livello d'utente.

Ciascun LWP è associato esattamente a un thread del livello del nucleo, mentre ogni thread del livello d'utente è indipendente dal nucleo. Molti LWP potrebbero essere parte di un processo, ma sono richiesti solo quando il thread deve comunicare col nucleo; ad esempio, è necessario un LWP per ciascun thread che potrebbe bloccarsi nell'invocare una

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

·All'interno di ogni nucleo ci sono cosa LWP => > colleghi corrisponde con i thread a livello utente



Thread dimenticato

VANTAGGIO e permette => di ottenere di gestione + di nucleo dei thread a livello nucleo in relazione con quelli a livello utente

Figura 5.6 Thread nel sistema operativo Solaris 2.

chiamata del sistema. Per cinque diverse richieste simultanee di letture di file sono richiesti cinque LWP, poiché potrebbero essere tutte nell'attesa del termine di operazioni di I/O nel nucleo. Se un processo ha soltanto quattro LWP, la quinta richiesta deve attendere che uno degli LWP sia restituito dal nucleo; d'altra parte, l'aggiunta di un sesto LWP non porta alcun vantaggio se c'è lavoro da svolgere soltanto per cinque.

I thread del livello del nucleo sono scelti per l'esecuzione dallo scheduler. Se un thread del livello del nucleo si blocca (come nel caso di un'attesa per il completamento di un'operazione di I/O), la CPU può nel frattempo eseguire un altro thread del livello del nucleo. Se il thread bloccato era in esecuzione per conto di un LWP, il LWP è anch'esso bloccato. In cima alla catena, il thread del livello d'utente correntemente associato a tale LWP risulta bloccato. Se un processo impiega più di un LWP, il nucleo può fare in modo che un altro di questi LWP sia scelto dallo scheduler.

Al fine di assicurare le migliori prestazioni, la libreria dei thread può modificare dinamicamente il numero dei thread nel gruppo. Se, ad esempio, tutti gli LWP in un processo sono bloccati e altri thread hanno la possibilità di essere eseguiti, la libreria dei thread crea automaticamente un altro LWP da assegnare a un thread in attesa. In questo modo si evita il problema del blocco dell'esecuzione di un programma per mancanza di LWP liberi. Inoltre, gli LWP sono risorse del nucleo costose da mantenere se restano inutilizzate. La libreria dei thread tiene conto per ogni LWP del tempo in cui questo rimane inutilizzato e lo elimina se supera una certa soglia, tipicamente 5 minuti. Per realizzare i thread nel Solaris 2, i progettisti hanno usato le seguenti strutture di dati:

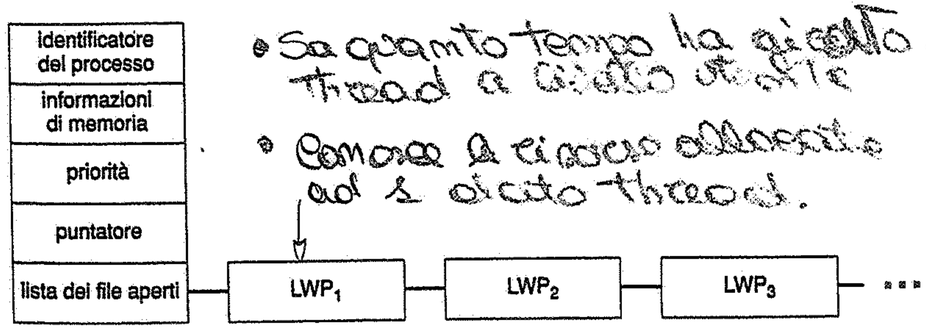
- Ogni thread del livello d'utente contiene un identificatore di thread, un insieme di registri (incluso un contatore di programma e un puntatore alla cima della pila), la pila, la priorità (usata dalla libreria dei thread per lo scheduling). Nessuna di queste strutture di dati è una risorsa del nucleo; tutte fanno parte dello spazio d'utente.
- Ogni LWP ha un insieme di registri per il thread del livello d'utente che sta eseguendo, oltre alle risorse di memoria e alle informazioni per la contabilizzazione. Un LWP è una struttura di dati del nucleo e risiede nello spazio del nucleo.
- Ogni thread del livello del nucleo ha soltanto una piccola struttura di dati e una pila. La struttura di dati contiene una copia dei registri del nucleo, un puntatore al LWP cui è associato e informazioni su priorità e scheduling.

Ogni processo del sistema Solaris 2 contiene molte informazioni contenute nel descrittore di processo (PCB) descritto nel Paragrafo 4.1.3. In particolare, contiene un identificatore del processo (PID), informazioni sull'associazione degli indirizzi di memoria, l'elenco dei file aperti, la priorità e un puntatore all'elenco dei thread del livello del nucleo associati al processo (Figura 5.7).

5.6 Thread nel sistema Windows 2000

Il sistema operativo Windows 2000 offre l'API Win32; si tratta dell'API principale della famiglia dei sistemi operativi della Microsoft (Windows 95/98/*Me*/ e Windows NT/2000/XP). La maggior parte del contenuto questo paragrafo si applica all'intera famiglia di tali sistemi operativi.

Un'applicazione per l'ambiente Windows si esegue come un processo separato e ogni processo può contenere uno o più thread. Il sistema Windows 2000 impiega il modello da uno a uno, descritto nel Paragrafo 5.2.2, secondo cui ogni thread del livello d'utente si associa a un thread del livello del nucleo. Tuttavia è disponibile anche la libreria *fiber*, che offre le funzioni che realizzano il modello da molti a molti (Paragrafo 5.2.3).



• Sa quanto tempo ha già speso il thread a livello utente

• Conosce la risorsa allocata ad il dato thread.

Figura 5.7 Processo in Solaris 2.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Ogni thread che appartiene a un processo può accedere allo spazio d'indirizzi virtuali di quel processo. I componenti generali di un thread includono:

- un identificatore di thread (ID), che identifica il thread in modo unico;
- un insieme di registri che rappresentano lo stato della CPU;
- una pila d'utente, usata quando il thread è eseguito nel modo d'utente; analogamente ogni thread ha anche una pila del nucleo, usata quando il thread è eseguito nel modo del nucleo.
- un'area di memoria privata, usata da diverse librerie di fase d'esecuzione e dinamiche (DLL).

L'insieme di registri, le pile e la memoria privata si chiamano *contesto* del thread e sono specifici dell'architettura sulla quale si esegue il sistema operativo. Le strutture di dati principali di un thread includono:

- ETHREAD (*executive thread block*);
- KTHREAD (*kernel thread block*);
- TEB (*thread environment block*).

I componenti chiave dell'ETHREAD sono un puntatore al processo al quale il thread appartiene e l'indirizzo della funzione nella quale il thread assume il controllo. La struttura ETHREAD contiene anche un puntatore alla corrispondente struttura KTHREAD.

Quest'ultima include informazioni per il thread relative allo scheduling e alla sincronizzazione. Oltre queste, KTHREAD contiene la pila del nucleo (usata quando il thread viene eseguito nel modo del nucleo) e un puntatore alla struttura TEB.

Le strutture ETHREAD e KTHREAD risiedono interamente nello spazio del nucleo; ciò implica che solo il nucleo vi può accedere. La struttura di dati TEB appartiene invece allo spazio d'utente e vi si accede quando il thread è eseguito nel modo d'utente. Tra gli altri campi, il TEB contiene una pila per il modo d'utente e un vettore per dati privati del thread che in questo contesto si chiama **memoria locale del thread** (*thread-local storage*).

5.7 Thread nel sistema LINUX

I thread sono stati introdotti nel LINUX dalla versione 2.2 del nucleo. Tale sistema operativo offre la chiamata del sistema `fork`, con la funzione ordinaria di duplicazione di un processo, e la chiamata del sistema `clone`, che è l'analogo della `fork` per la creazione di un thread: funziona in modo corrispondente, ma anziché creare una copia del processo chiamante, crea un processo distinto che condivide lo spazio d'indirizzi del processo chiamante. Attraverso questa condivisione dello spazio d'indirizzi del processo genitore, un task clonato si comporta in modo molto simile a un thread distinto.

La condivisione è permessa dal modo in cui un processo è rappresentato nel nucleo. Per ogni processo, nel sistema esiste un'unica struttura di dati nel nucleo. Tuttavia, invece di memorizzare i dati di ogni processo in questa struttura di dati, la struttura contiene dei puntatori ad altre strutture dove i dati sono effettivamente contenuti. Ad esempio, la struttura di dati di ciascun processo contiene puntatori ad altre strutture di dati che rappresentano l'elenco dei file aperti, informazioni per la gestione dei segnali e la memoria virtuale. Quando s'invoca la `fork`, si crea un nuovo processo insieme con una *copia* di tutte le strutture di dati del processo genitore.

Anche quando s'invoca la `clone` si crea un nuovo processo, ma anziché ricevere una copia di tutte le strutture di dati, il nuovo processo riceve un *puntatore* alle strutture di dati del processo genitore, che permette al processo figlio di condividerne la memoria e le altre risorse. La chiamata del sistema `clone` riceve come parametro un insieme di indicatori (*flag*), che specifica quanto del processo genitore deve essere condiviso col figlio. Se nessuno degli indicatori è attivato, non si ha alcuna condivisione e la `clone` si comporta esattamente come la `fork`; se sono attivati tutti e cinque, il processo genitore condivide tutto col figlio. Tra questi due estremi si possono avere vari livelli di condivisione intermedi.

È interessante notare che il sistema operativo LINUX non distingue tra processi e thread; infatti quando ci si riferisce al percorso di controllo di un programma, generalmente si usa il termine *task*, anziché processo o thread. Oltre la clonazione dei processi, il sistema operativo non dispone di alcuna funzione per la programmazione multithread, strutture di dati separate, o procedure del nucleo. Sono tuttavia disponibili varie versioni della libreria Pthreads per la programmazione multithread al livello d'utente.

5.8 Thread nel linguaggio Java Leggere

La gestione dei thread al livello d'utente si può realizzare tramite una libreria come la Pthreads; la maggior parte dei sistemi operativi ne consente la gestione al livello del nucleo; il linguaggio Java è uno dei pochi che permettono la gestione e la creazione dei thread al livello del linguaggio di programmazione. Ciononostante, poiché sono gestiti dalla macchina virtuale (JVM) e non da una libreria al livello d'utente o del nucleo, è difficile classificare i thread del linguaggio Java come thread al livello del nucleo o al livello d'utente. In questo paragrafo s'introducono i thread del linguaggio Java come alternativa ai rigidi modelli di livello del nucleo o d'utente. Nel seguito del paragrafo si trattano le possibilità di associazione di un thread del linguaggio Java a un thread sottostante del livello del nucleo.

Tutti i programmi scritti in Java prevedono almeno un thread di controllo; anche un semplice programma formato da un unico metodo `main`, è eseguito dalla JVM come un unico thread. Sono inoltre previsti comandi che permettono ai programmatori di creare e manipolare nuovi thread di controllo all'interno dei programmi.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

5.8.1 Creazione dei thread

Un modo per creare esplicitamente un thread consiste nel creare una nuova classe derivata dalla classe `Thread` e di sovrascrivere il metodo `run` di quella classe. Questo metodo è esemplificato dalla versione scritta in Java di un programma multithread che determina la somma dei primi *n* numeri naturali (Figura 5.8).

```
class Somma_primi_interi extends Thread
{
  public Somma_primi_interi(int n) {
    sup = n;
  }

  public void run () {
    int somma = 0;

    if (sup > 0) {
      for (int i = 1; i <= sup; i++)
        somma += i;
    }

    System.out.println("La somma dei primi "+sup+"
      interi è "+somma);
  }

  private int sup;
}

public class ThreadTester
{
  public static void main(string[] args) { ← MAIN
    if (args.length > 0) {
      if (Integer.parseInt(args[0]) < 0)
        System.err.println(args[0] + "deve essere >= 0.");
    else {
      Somma_primi_interi thrd =
        new Somma_primi_interi(Integer.parseInt(args[0]));
      thrd.start();
    }
  }
  else
    System.err.println("Uso: Somma_primi_interi
      <valore intero>");
  }
}
```

Figura 5.8 Programma scritto in Java per la somma dei primi *n* numeri naturali.

Un oggetto di questa classe derivata è eseguito come thread di controllo distinto nella JVM. Tuttavia, la creazione di un oggetto derivato dalla classe `Thread` non determina la creazione di un nuovo thread; è il metodo `start` che crea effettivamente il nuovo thread. La chiamata del metodo per il nuovo oggetto determina due azioni:

1. l'assegnazione della memoria e l'inizializzazione di un nuovo thread nella JVM;
2. la chiamata del metodo `run`, che rende il thread disponibile a essere eseguito dalla macchina virtuale (si noti che il metodo `run` non si chiama mai esplicitamente, ma tramite il metodo `start`).

Quando si esegue il programma per la somma dei primi *n* numeri naturali, la JVM crea due thread: il primo è associato all'applicazione, il thread che comincia l'esecuzione al metodo `main`; il secondo è il thread `Somma_primi_interi`, che si crea esplicitamente col metodo `start`, comincia la propria esecuzione nel suo metodo `start` e termina all'uscita del metodo `run`.

5.8.2 Macchina virtuale e sistema operativo

La macchina virtuale del linguaggio Java (JVM) di solito funziona su un sistema operativo sottostante; ciò permette alla JVM di nascondere i dettagli del sistema operativo e di offrire un ambiente astratto e coerente che permette ai programmi scritti in Java di essere eseguiti su qualsiasi piattaforma che disponga di una JVM. La definizione della JVM non prescrive il modo in cui i thread dell'ambiente Java si devono far corrispondere ai servizi del sistema operativo sottostante; tale funzione dipende dalla specifica versione della JVM di ciascun sistema operativo. I sistemi operativi Windows 95/98 e Windows NT/2000 adottano il modello da uno a uno; quindi a ogni thread dell'ambiente Java di una JVM che opera in questi sistemi si fa corrispondere un thread del livello del nucleo. La JVM per il Solaris 2 impiegava inizialmente il modello da molti a uno (chiamato *green threads*); a partire dalla versione 1.1 della JVM per il Solaris 2.6, è stato adottato il modello da molti a molti.

5.9 Sommario

Un thread è un percorso di controllo d'esecuzione all'interno di un processo. Un processo multithread contiene più percorsi di controllo tra loro diversi ma che condividono lo stesso spazio d'indirizzi. I vantaggi della programmazione multithread includono un miglioramento del tempo di risposta, la condivisione di risorse all'interno del processo, il risparmio e la capacità di sfruttare le architetture dotate di più unità d'elaborazione.

I thread del livello d'utente sono thread visibili al programmatore e sconosciuti al nucleo. Una libreria di thread nello spazio d'utente di solito gestisce thread del livello d'utente. Il nucleo del sistema operativo gestisce thread del livello del nucleo. In generale, i thread del livello d'utente richiedono minor tempo per essere creati e gestiti rispetto a quelli del livello del nucleo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Ci sono tre tipi diversi di modelli che descrivono le relazioni fra thread del livello d'utente e del livello del nucleo: il modello da molti a uno associa più thread del livello d'utente a un singolo thread del livello del nucleo; il modello da uno a uno associa ciascun thread del livello d'utente a un corrispondente thread del livello del nucleo; il modello da molti a molti associa dinamicamente più thread del livello d'utente a un numero minore o uguale di thread del livello del nucleo.

I programmi multithread introducono molti aspetti critici per il programmatore, tra i quali la semantica delle chiamate del sistema fork e exec; altri aspetti sono ad esempio la cancellazione, la gestione dei segnali e i dati privati dei thread. Molti sistemi operativi moderni prevedono la gestione dei thread al livello del nucleo; ad esempio i sistemi Windows NT e Windows 2000, Solaris 2 e LINUX. L'API Pthreads offre un insieme di funzioni per creare e gestire thread al livello d'utente. Il linguaggio Java prevede un'API simile per i thread; tuttavia, poiché sono gestiti dalla JVM e non da una libreria di thread al livello d'utente o del nucleo, i thread del linguaggio Java non si possono considerare né thread del livello d'utente né del livello del nucleo.

5.10 Esercizi

5.1 Descrivete due esempi di programmazione multithread che offrono prestazioni migliori rispetto alla corrispondente soluzione a singolo thread.

5.2 Descrivete due esempi di programmazione multithread che *non* offrono prestazioni migliori rispetto alla corrispondente soluzione a singolo thread.

5.3 Enunciate due differenze fra i thread del livello d'utente e quelli del livello del nucleo, e spiegate in quali circostanze un tipo è meglio dell'altro.

5.4 Descrivete le azioni intraprese da un nucleo per un cambio di contesto fra thread del livello del nucleo.

5.5 Descrivete le azioni intraprese da una libreria di thread per un cambio di contesto fra thread del livello d'utente.

5.6 Dite quali risorse s'impiegano nella creazione di un thread, e in che cosa differiscono da quelle che s'impiegano nella creazione di un processo.

5.7 Si consideri un sistema operativo che fa corrispondere ai thread del nucleo i thread del livello d'utente secondo il modello da molti a molti in cui tale corrispondenza avviene tramite gli LWP. Inoltre, il sistema permette ai programmatori di creare thread per elaborazioni in tempo reale. Dite se è necessario associare un thread per elaborazioni in tempo reale a un LWP. Spiegate la risposta.

5.8 Scrivete un programma multithread in Java o che impiega la libreria Pthreads per la generazione della successione di Fibonacci. Il programma deve avere il seguente comportamento: l'utente avvia l'esecuzione del programma e inserisce alla riga di comando la quantità di numeri di Fibonacci che il programma deve generare; il programma crea un thread separato per la generazione dei numeri di Fibonacci.

5.9 Scrivete un programma multithread in Java o che impiega la libreria Pthreads per la generazione di numeri primi. Il programma deve avere il seguente comportamento: l'utente avvia l'esecuzione del programma e inserisce un numero alla riga di comando; il programma crea un thread distinto che riporta tutti i numeri primi minori o uguali al numero inserito dall'utente.

5.11 Note bibliografiche

Gli aspetti relativi alle prestazioni dei thread sono affrontati in [Anderson et al. 1989], con una continuazione del lavoro in [Anderson et al. 1991] per la valutazione delle prestazioni dei thread del livello d'utente con l'ausilio del nucleo. [Marsh et al. 1991] presenta una discussione sui thread di prima classe del livello d'utente. [Bershad et al. 1990] descrive la combinazione di thread e RPC. [Draves et al. 1991] illustra l'uso delle continuazioni per realizzare la gestione dei thread e la comunicazione tra di essi nei sistemi operativi. [Engelschall 2000] presenta una tecnica di gestione dei thread del livello d'utente. Un'analisi della dimensione ottimale di un gruppo di thread si può trovare in [Ling et al. 2000].

Il sistema operativo IBM OS/2 è un sistema operativo multithread per PC, [Kogan e Rawson 1988]. [Peacock 1992] discute l'uso della programmazione multithread nel file system del sistema operativo Solaris 2. [Vahalia 1996] tratta l'uso dei thread in diverse versioni dello UNIX. [Mauro e McDougall 2001] descrive i recenti sviluppi relativi all'uso dei thread nel nucleo del Solaris 2. [Zabatta e Young 1998] confronta i thread del sistema Windows NT e del Solaris 2 relativamente alle architetture SMP.

Informazioni sulla programmazione multithread si trovano in [Lewis e Berg 1988], [Sun 1995], e [Kleiman et al. 1996], sebbene tutti questi lavori tendano a prediligere l'uso della libreria Pthreads. [Oaks e Wong 1999], [Lea 2000], e [Hartley 1998] discutono la programmazione multithread nel linguaggio Java. [Solomon 1998] e [Solomon e Russinovich 2000] descrivono la realizzazione dei thread nei sistemi operativi Windows NT e Windows 2000, rispettivamente. [Beveridge e Wiener 1997] affronta il tema della programmazione multithread con Win32, e [Pham e Garg 1996] descrive la programmazione multithread con Windows NT.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 6

Scheduling della CPU

Lo scheduling della CPU è alla base dei sistemi operativi multiprogrammati: attraverso la commutazione del controllo della CPU tra i vari processi, il sistema operativo può rendere più produttivo il calcolatore. In questo capitolo s'introducono i concetti fondamentali dello scheduling e si descrivono vari algoritmi di scheduling della CPU. Si affronta inoltre il problema della scelta dell'algoritmo da impiegare per un dato sistema.

6.1 Concetti fondamentali

L'obiettivo della multiprogrammazione è avere sempre processi in esecuzione al fine di massimizzare l'utilizzo della CPU. In un sistema con una sola unità d'elaborazione si può eseguire al più un processo alla volta; gli altri processi, se ci sono, devono attendere che la CPU sia libera e possa essere nuovamente sottoposta a scheduling.

L'idea della multiprogrammazione è relativamente semplice. Un processo è in esecuzione finché non deve attendere un evento, generalmente il completamento di qualche richiesta di I/O; durante l'attesa, in un sistema di calcolo semplice, la CPU resterebbe inattiva, e tutto il tempo d'attesa sarebbe sprecato. Con la multiprogrammazione si cerca d'impiegare questi tempi d'attesa in modo produttivo: si tengono contemporaneamente più processi nella memoria, e quando un processo deve attendere un evento, il sistema operativo gli sottrae il controllo della CPU per cederlo a un altro processo.

Lo scheduling è una funzione fondamentale dei sistemi operativi; si sottopongono a scheduling quasi tutte le risorse di un calcolatore. Naturalmente la CPU è una delle risorse principali, e il suo scheduling è alla base della progettazione dei sistemi operativi.

Il max impiego della CPU è ottenuto con la multiprogrammazione.

6.1.1 Ciclicità delle fasi d'elaborazione e di I/O

Il successo dello scheduling della CPU dipende dall'osservazione della seguente proprietà dei processi: l'esecuzione del processo consiste in un ciclo d'elaborazione (svolta dalla CPU) e d'attesa del completamento delle operazioni di I/O. I processi si alternano tra questi due stati. L'esecuzione di un processo comincia con una sequenza (una 'raffica') di operazioni d'elaborazione svolte dalla CPU (*CPU burst*), seguita da una sequenza di opera-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

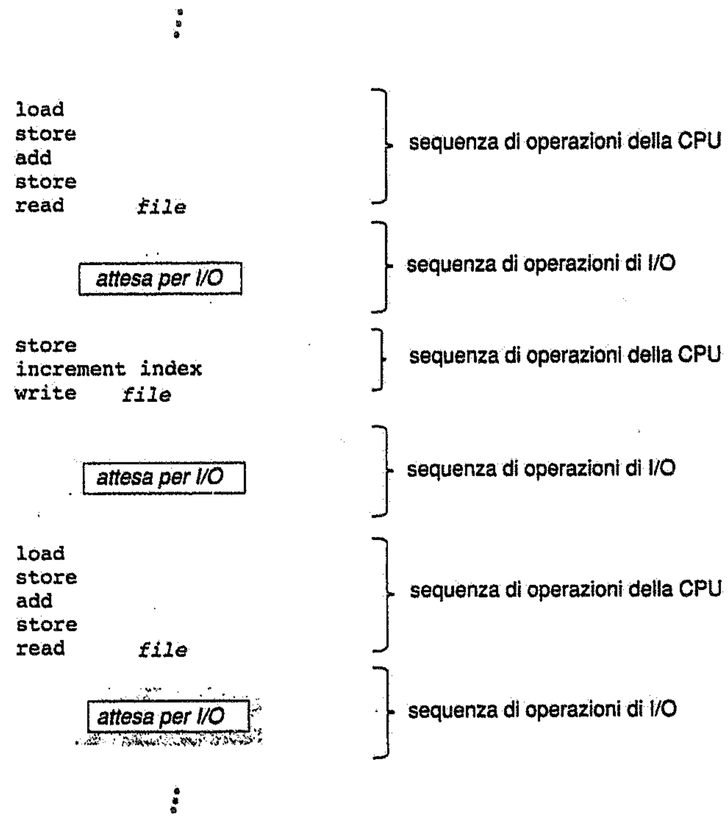


Figura 6.1 Serie alternata di sequenze di operazioni della CPU e di sequenze di operazioni di I/O.

zioni di I/O (*I/O burst*), quindi un'altra sequenza di operazioni della CPU, di nuovo una sequenza di operazioni di I/O, e così via. L'ultima sequenza di operazioni della CPU si conclude con una richiesta al sistema di terminare l'esecuzione. La Figura 6.1 mostra una serie alternata di sequenze di operazioni della CPU e di sequenze di operazioni di I/O.

Le durate delle sequenze di operazioni della CPU sono state misurate, e sebbene varino molto secondo il processo e secondo il calcolatore, la loro curva di frequenza è simile a quella illustrata nella Figura 6.2. La curva è generalmente di tipo esponenziale o iperesponenziale, con molte brevi sequenze di operazioni della CPU, e poche sequenze di operazioni della CPU molto lunghe. Un programma con prevalenza di I/O (*I/O bound*) produce generalmente molte sequenze di operazioni della CPU di breve durata. Un programma con prevalenza d'elaborazione (*CPU bound*), invece, produce poche sequenze di operazioni della CPU molto lunghe. Queste caratteristiche possono essere utili nella scelta di un appropriato algoritmo di scheduling della CPU.

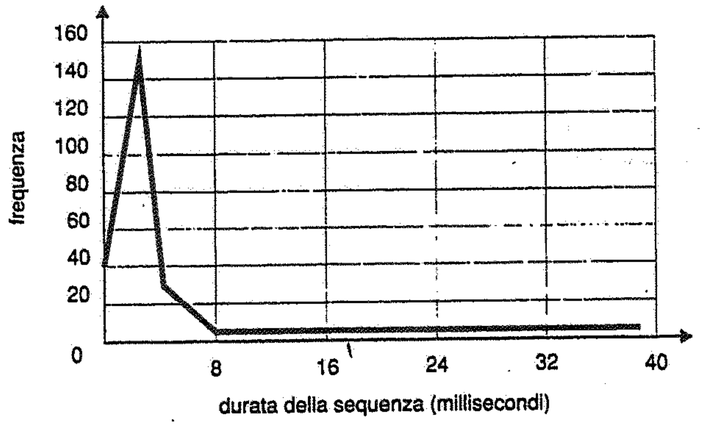


Figura 6.2 Diagramma delle durate delle sequenze di operazioni della CPU.

6.1.2 Scheduler della CPU

Ogniqualvolta la CPU passa nello stato d'inattività, il sistema operativo sceglie per l'esecuzione uno dei processi presenti nella coda dei processi pronti. In particolare, è lo scheduler a breve termine, o scheduler della CPU che, tra i processi nella memoria pronti per l'esecuzione, sceglie quello cui assegnare la CPU.

La coda dei processi pronti non è necessariamente una coda in ordine d'arrivo (*first-in, first-out* — FIFO). Come si nota analizzando i diversi algoritmi di scheduling, una coda dei processi pronti si può realizzare come una coda FIFO, una coda con priorità, un albero o semplicemente una lista concatenata non ordinata. Tuttavia, concettualmente tutti i processi della coda dei processi pronti sono posti nella lista d'attesa per accedere alla CPU. Generalmente gli elementi delle code sono i descrittori dei processi (*process control block* — PCB).

6.1.3 Scheduling con diritto di prelazione

Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:

1. un processo passa dallo stato di esecuzione allo stato di attesa (ad esempio, richiesta di I/O o richiesta di attesa per la terminazione di uno dei processi figli);
2. un processo passa dallo stato di esecuzione allo stato pronto (ad esempio, quando si verifica un segnale d'interruzione);
3. un processo passa dallo stato di attesa allo stato pronto (ad esempio, al completamento di un'operazione di I/O);
4. un processo termina.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I casi 1 e 4 in quanto tali non comportano alcuna scelta di scheduling; a essi segue la scelta di un nuovo processo da eseguire, sempre che ce ne sia almeno uno nella coda dei processi pronti per l'esecuzione. Una scelta si deve invece fare nei casi 2 e 3.

Quando lo scheduling interviene solo nelle condizioni 1 e 4, si dice che lo schema di scheduling è **senza diritto di prelazione** (*nonpreemptive*); altrimenti, lo schema di scheduling è **con diritto di prelazione** (*preemptive*). Nel caso dello scheduling senza diritto di prelazione, quando si assegna la CPU a un processo, questo rimane in possesso della CPU fino al momento del suo rilascio, dovuto al termine dell'esecuzione o al passaggio nello stato di attesa. Questo metodo di scheduling, impiegato nell'ambiente Microsoft Windows, è l'unico che si può usare in certe architetture perché non richiede particolari caratteristiche, come la presenza di un temporizzatore, necessarie allo scheduling con diritto di prelazione.

Lo scheduling con diritto di prelazione ha un inconveniente. Si consideri il caso in cui due processi condividono dati; mentre uno di questi aggiorna i dati si ha la sua prelazione in favore dell'esecuzione dell'altro. Il secondo processo può, a questo punto, tentare di leggere i dati che sono stati lasciati in uno stato incoerente dal primo processo. Sono quindi necessari nuovi meccanismi per coordinare l'accesso ai dati condivisi; questo argomento è trattato nel Capitolo 7.

La capacità di prelazione si ripercuote anche sulla progettazione del nucleo del sistema operativo. Durante l'elaborazione di una chiamata del sistema, il nucleo può essere impegnato in attività in favore di un processo; tali attività possono comportare la necessità di modifiche a importanti dati del nucleo, come le code di I/O. Se si ha la prelazione del processo durante tali modifiche e il nucleo (o un driver di dispositivo) deve leggere o modificare gli stessi dati, si può avere il caos. Alcuni sistemi operativi, tra cui la maggior parte delle versioni dello UNIX, affrontano questo problema attendendo il completamento della chiamata del sistema o che si abbia il blocco dell'I/O prima di eseguire un cambio di contesto. Quindi, il nucleo non può esercitare la prelazione su un processo mentre le strutture di dati del nucleo si trovano in uno stato incoerente. Tale schema assicura una semplice struttura del nucleo. Sfortunatamente, questo modello d'esecuzione del nucleo non è adeguato alle computazioni in tempo reale e alla multielaborazione. Questi problemi e le relative soluzioni sono descritti nei Paragrafi 6.4 e 6.5.

Per quel che riguarda lo UNIX, sono ancora presenti sezioni di codice a rischio. Poiché le interruzioni si possono, per definizione, verificare in ogni istante e il nucleo non può sempre ignorale, le sezioni di codice eseguite per effetto delle interruzioni devono essere protette da un uso simultaneo. Il sistema operativo deve ignorare raramente le interruzioni, altrimenti si potrebbero perdere dati in ingresso, o si potrebbero sovrascrivere dati in uscita. Per evitare che più processi accedano in modo concorrente a tali sezioni di codice, queste disattivano le interruzioni al loro inizio e le riattivano alla fine. Sfortunatamente la disattivazione e l'attivazione delle interruzioni richiedono tempo, in modo particolare nei sistemi con più unità d'elaborazione. Per gestire in modo efficiente sistemi con sempre più numerose unità d'elaborazione, si devono ridurre al minimo le modifiche allo stato del sistema delle interruzioni, e si deve aumentare al massimo la selettività dei meccanismi di bloccaggio. Questa è ad esempio una sfida per la 'scalabilità' del sistema LINUX.

Mary & Cheese 29/6/03

MODULO ALLOCATORE

6.1.4 Dispatcher

Un altro elemento coinvolto nella funzione di scheduling della CPU è il **dispatcher**; si tratta del modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine. Questa funzione riguarda quel che segue:

- il cambio di contesto;
- il passaggio al modo d'utente;
- il salto alla giusta posizione del programma d'utente per riavviarne l'esecuzione.

Poiché si attiva a ogni cambio di contesto, il dispatcher dovrebbe essere quanto più rapido è possibile. Il tempo richiesto dal dispatcher per fermare un processo e avviare l'esecuzione di un altro è nota come **latenza di dispatch**.

6.2 Criteri di scheduling

Diversi algoritmi di scheduling della CPU hanno proprietà differenti e possono favorire una particolare classe di processi. Prima di scegliere l'algoritmo da usare in una specifica situazione occorre considerare le caratteristiche dei diversi algoritmi.

Per il confronto tra gli algoritmi di scheduling della CPU sono stati suggeriti molti criteri. Le caratteristiche usate per il confronto possono incidere in modo rilevante sulla scelta dell'algoritmo migliore. Di seguito si riportano alcuni criteri:

MAX UTILIZZO DELLA CPU

Utilizzo della CPU. La CPU deve essere più attiva possibile. Teoricamente, l'utilizzo della CPU può variare dallo 0 al 100 per cento. In un sistema reale può variare dal 40 per cento, per un sistema con poco carico, al 90 per cento per un sistema con utilizzo intenso.

MAX PRODUTTIVITÀ

Produttività. La CPU è attiva quando si svolge del lavoro. Una misura del lavoro svolto è data dal numero dei processi completati nell'unità di tempo: tale misura è detta **produttività** (*throughput*). Per processi di lunga durata questo rapporto può essere di un processo all'ora, mentre per brevi transazioni è possibile avere una produttività di 10 processi al secondo.

MINIMO TEMPO DI COMPLETAMENTO

Tempo di completamento. Considerando un processo particolare, un criterio importante può essere relativo al tempo necessario per eseguire il processo stesso. L'intervallo che intercorre tra la sottomissione del processo e il completamento dell'esecuzione è chiamato **tempo di completamento** (*turnaround time*), ed è la somma dei tempi passati nell'attesa dell'ingresso nella memoria, nella coda dei processi pronti, durante l'esecuzione nella CPU e nel compiere operazioni di I/O.

MINIMO TEMPO DI ATTESA

Tempo d'attesa. L'algoritmo di scheduling della CPU non influisce sul tempo impiegato per l'esecuzione di un processo o di un'operazione di I/O; influisce solo sul tempo d'attesa nella coda dei processi pronti. Il **tempo d'attesa** è la somma degli intervalli d'attesa passati nella coda dei processi pronti.

http://www.adultpdf.com
Created by TIFF Combine - (trial version) To remove this mark, please register this software.

MINIMOTEMPO DI RISPOSTA

- **Tempo di risposta.** In un sistema interattivo il tempo di completamento può non essere il miglior criterio di valutazione: spesso accade che un processo emetta dati abbastanza presto, e continui a calcolare i nuovi risultati mentre quelli precedenti sono in fase d'emissione. Quindi, un'altra misura di confronto è data dal tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta. Questa misura è chiamata **tempo di risposta**, ed è data dal tempo necessario per iniziare la risposta, ma non dal suo tempo d'emissione. Generalmente il tempo di completamento è limitato dalla velocità del dispositivo d'emissione dei dati.

Utilizzo e produttività della CPU si devono aumentare al massimo, mentre il tempo di completamento, il tempo d'attesa e il tempo di risposta si devono ridurre al minimo. Nella maggior parte dei casi si ottimizzano i valori medi; in alcune circostanze è più opportuno ottimizzare i valori minimi o massimi, anziché i valori medi; ad esempio, per garantire che tutti gli utenti ottengano un buon servizio, può essere utile ridurre il massimo tempo di risposta.

Per i sistemi interattivi, come i sistemi a partizione del tempo, alcuni analisti suggeriscono che sia più importante ridurre al minimo la varianza del tempo di risposta anziché il tempo medio di risposta. Un sistema il cui tempo di risposta sia ragionevole e prevedibile può essere considerato migliore di un sistema mediamente più rapido, ma molto variabile. Tuttavia, è stato fatto poco sugli algoritmi di scheduling della CPU al fine di ridurre al minimo la varianza. (di tali parametri)

Poiché si analizzano diversi algoritmi di scheduling della CPU, è opportuno che se ne illustri anche il funzionamento. Una spiegazione approfondita richiederebbe l'uso di molti processi, ognuno dei quali costituito da parecchie centinaia di sequenze di operazioni della CPU e di sequenze di operazioni di I/O. Per motivi di semplicità, negli esempi si considera una sola sequenza di operazioni della CPU (la cui durata è espressa in millisecondi) per ogni processo. La misura di confronto adottata è il tempo d'attesa medio. Meccanismi di valutazione più raffinati sono trattati nel Paragrafo 6.6.

6.3 Algoritmi di scheduling

IMP

Lo scheduling della CPU riguarda la scelta dei processi presenti nella coda dei processi pronti cui assegnare la CPU. In questo paragrafo si descrivono alcuni fra i tanti algoritmi di scheduling della CPU.

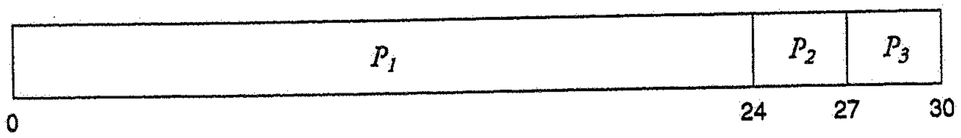
6.3.1 Scheduling in ordine d'arrivo

Il più semplice algoritmo di scheduling della CPU è l'algoritmo di **scheduling in ordine d'arrivo** (*scheduling first-come, first-served* — FCFS). Con questo schema la CPU si assegna al processo che la richiede per primo. La realizzazione del criterio FCFS si fonda su una coda FIFO. Quando un processo entra nella coda dei processi pronti, si collega il suo PCB all'ultimo elemento della coda. Quando è libera, si assegna la CPU al processo che si trova alla testa della coda dei processi pronti, rimuovendolo da essa. Il codice per lo scheduling FCFS è semplice sia da scrivere sia da capire.

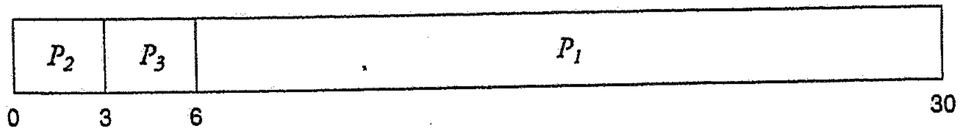
Il tempo medio d'attesa per l'algoritmo FCFS è spesso abbastanza lungo. Si consideri il seguente insieme di processi, che si presenta al momento 0, con la durata della sequenza di operazioni della CPU espressa in millisecondi:

Processo	Durata della sequenza
P_1	24
P_2	3
P_3	3

Se i processi arrivano nell'ordine P_1, P_2, P_3 e sono serviti in ordine FCFS, si ottiene il risultato illustrato dal seguente **schema di Gantt**:



Il tempo d'attesa è 0 millisecondi per il processo P_1, 24 millisecondi per il processo P_2 e 27 millisecondi per il processo P_3. Quindi, il tempo d'attesa medio è (0 + 24 + 27)/3 = 17 millisecondi. Se i processi arrivassero nell'ordine P_2, P_3, P_1, i risultati sarebbero quelli illustrati nel seguente schema di Gantt:



Il tempo di attesa medio è ora di (6 + 0 + 3)/3 = 3 millisecondi. Si tratta di una notevole riduzione. Quindi, il tempo medio d'attesa in condizioni di FCFS non è in genere minimo, e può variare sostanzialmente al variare della durata delle sequenze di operazioni della CPU dei processi.

Si considerino inoltre le prestazioni dello scheduling FCFS in una situazione dinamica. Si supponga di avere un processo con prevalenza d'elaborazione e molti processi con prevalenza di I/O. Via via che i processi fluiscono nel sistema si può riscontrare come il processo con prevalenza d'elaborazione occupi la CPU. Durante questo periodo tutti gli altri processi terminano le proprie operazioni di I/O e si spostano nella coda dei processi pronti, nell'attesa della CPU. Mentre i processi si trovano nella coda dei processi pronti, i dispositivi di I/O sono inattivi. Successivamente il processo con prevalenza d'elaborazione termina la propria sequenza di operazioni della CPU e passa a una fase di I/O. Tutti i

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

→ EFFETTO CONVOGLIO → effetto che si verifica quando processi di breve durata devono attendere che il processo ricevuto lungo libera la CPU.

processi con prevalenza di I/O, caratterizzati da sequenze di operazioni della CPU molto brevi, sono eseguiti rapidamente e tornano alle code di I/O, lasciando inattiva la CPU. Il processo con prevalenza d'elaborazione torna nella coda dei processi pronti e riceve il controllo della CPU; così, finché non termina l'esecuzione del processo con prevalenza d'elaborazione, tutti i processi con prevalenza di I/O si trovano nuovamente ad attendere nella coda dei processi pronti. Si ha un effetto convoglio, tutti i processi attendono che un lungo processo liberi la CPU, che causa una riduzione dell'utilizzo della CPU e dei dispositivi rispetto a quella che si avrebbe se si eseguissero per primi i processi più brevi.

L'algoritmo di scheduling FCFS è senza prelazione; una volta che la CPU è stata assegnata a un processo, questo la trattiene fino al momento del rilascio, che può essere dovuto al termine dell'esecuzione o alla richiesta di un'operazione di I/O. L'algoritmo FCFS risulta particolarmente problematico nei sistemi a partizione del tempo, dove è importante che ogni utente disponga della CPU a intervalli regolari. Permettere a un solo processo di occupare la CPU per un lungo periodo condurrebbe a risultati disastrosi.

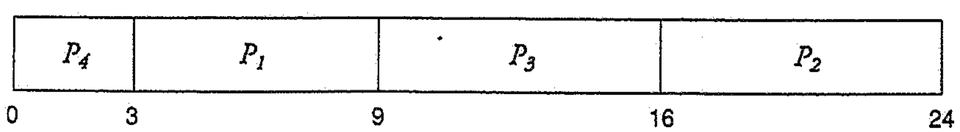
6.3.2 Scheduling per brevità

Un criterio diverso di scheduling della CPU si può ottenere con l'algoritmo di scheduling per brevità (*shortest-job-first* — SJF). Questo algoritmo associa a ogni processo la lunghezza della successiva sequenza di operazioni della CPU. Quando è disponibile, si assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU. Se due processi hanno le successive sequenze di operazioni della CPU della stessa lunghezza, si applica lo scheduling FCFS. Si noti che sarebbe più appropriato il termine *shortest next CPU burst*, infatti lo scheduling si esegue esaminando la lunghezza della successiva sequenza di operazioni della CPU del processo e non la sua lunghezza totale. Tuttavia, poiché è usato nella maggior parte dei libri di testo, anche qui si fa uso del termine SJF.

Come esempio si consideri il seguente insieme di processi, con la durata della sequenza di operazioni della CPU espressa in millisecondi:

Processo	Durata della sequenza
P_1	6
P_2	8
P_3	7
P_4	3

Con lo scheduling SJF questi processi si ordinerebbero secondo il seguente diagramma di Gantt:



Il tempo d'attesa è di 3 millisecondi per il processo P_1, di 16 millisecondi per il processo P_2, di 9 millisecondi per il processo P_3 e di 0 millisecondi per il processo P_4. Quindi, il tempo d'attesa medio è di $(3 + 16 + 9 + 0)/4 = 7$ millisecondi. Usando lo scheduling FCFS, il tempo d'attesa medio sarebbe di 10,25 millisecondi.

Si può dimostrare che l'algoritmo di scheduling SJF è *ottimale*, nel senso che rende minimo il tempo d'attesa medio per un dato insieme di processi. Spostando un processo breve prima di un processo lungo, il tempo d'attesa per il processo breve diminuisce più di quanto aumenti il tempo d'attesa per il processo lungo. Di conseguenza, il tempo d'attesa *medio* diminuisce.

La difficoltà reale implicita nell'algoritmo SJF consiste nel conoscere la durata della successiva richiesta della CPU. Per lo scheduling a lungo termine di un sistema a lotti (*job scheduling*) si può usare come durata il tempo limite d'elaborazione che gli utenti specificano nel sottoporre il processo. Gli utenti sono quindi motivati a stabilire con precisione tale limite, poiché un valore inferiore può significare una risposta più rapida. Occorre però notare che un valore troppo basso causa un errore di superamento del tempo limite e richiede una nuova esecuzione. Lo scheduling SJF si usa spesso nello scheduling a lungo termine.

Sebbene sia ottimale, l'algoritmo SJF non si può realizzare al livello dello scheduling della CPU a breve termine, poiché non esiste alcun modo per conoscere la lunghezza della successiva sequenza di operazioni della CPU. Un possibile metodo consiste nel tentare di approssimare lo scheduling SJF: se non è possibile *conoscere* la lunghezza della prossima sequenza di operazioni della CPU, si può cercare di *predire* il suo valore; è probabile, infatti, che sia simile ai precedenti. Quindi, calcolando un valore approssimato della lunghezza, si può scegliere il processo con la più breve fra tali lunghezze previste.

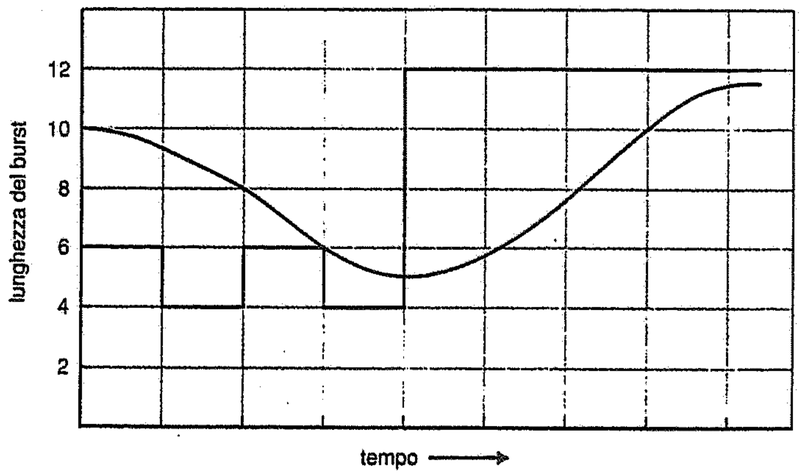
La lunghezza della successiva sequenza di operazioni della CPU generalmente si ottiene calcolando la media esponenziale delle effettive lunghezze delle precedenti sequenze di operazioni della CPU. Denotando con t_n la lunghezza dell'n-esima sequenza di operazioni della CPU e con τ_{n+1} il valore previsto per la successiva sequenza di operazioni della CPU, con α tale che $0 \le \alpha \le 1$, si definisce

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

Questa formula definisce una media esponenziale. Il valore di t_n contiene le informazioni più recenti; τ_n registra la storia passata. Il parametro α controlla il peso relativo sulla predizione della storia recente e di quella passata. Se $\alpha = 0$, allora, $\tau_{n+1} = \tau_n$, e la storia recente non ha effetto; si suppone, cioè, che le condizioni attuali siano transitorie; se $\alpha = 1$, allora $\tau_{n+1} = t_n$ e ha significato solo la più recente sequenza di operazioni della CPU: si suppone, cioè, che la storia sia vecchia e irrilevante. Più comune è la condizione in cui $\alpha = 1/2$, valore che indica che la storia recente e la storia passata hanno lo stesso peso.

Nella Figura 6.3 è illustrata una media esponenziale con $\alpha = 1/2$. Il τ_0 iniziale si può definire come una costante o come una media complessiva del sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.



CPU burst (t_i)		6	4	6	4	13	13	13	...
predizione (τ_i)	10	8	6	6	5	9	11	12	...

Figura 6.3 Predizione della lunghezza della successiva sequenza di operazioni della CPU.

Per comprendere il comportamento della media esponenziale, si può sviluppare la formula per τ_{n+1} sostituendo in τ_n, in modo da ottenere

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1}\tau_0$$

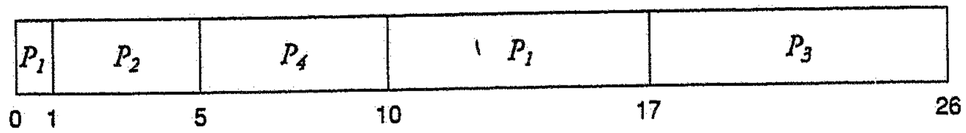
Siccome sia α sia $(1-\alpha)$ sono minori o uguali a 1, ogni termine ha peso inferiore a quello del suo predecessore.

L'algoritmo SJF può essere sia *con prelazione* sia *senza prelazione*. La scelta si presenta quando alla coda dei processi pronti arriva un nuovo processo mentre un altro processo è ancora in esecuzione. Il nuovo processo può avere una successiva sequenza di operazioni della CPU più breve di quella che resta al processo correntemente in esecuzione. Un algoritmo SJF con prelazione sostituisce il processo attualmente in esecuzione, mentre un algoritmo SJF senza prelazione permette al processo correntemente in esecuzione di portare a termine la propria sequenza di operazioni della CPU. (Lo scheduling SJF con prelazione è talvolta chiamato scheduling *shortest-remaining-time-first*.)

Come esempio, si considerino i quattro processi seguenti, dove la durata delle sequenze di operazioni della CPU è data in millisecondi:

Processo	Istante d'arrivo	Durata della sequenza
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Se i processi arrivano alla coda dei processi pronti nei momenti indicati e richiedono le durate delle sequenze di operazioni della CPU indicate, allora dallo scheduling SJF con prelazione risulta quel che è illustrato dal seguente schema di Gantt:



All'istante 0 si avvia il processo P_1, poiché è l'unico che si trova nella coda. All'istante 1 arriva il processo P_2. Il tempo necessario per completare il processo P_1 (7 millisecondi) è maggiore del tempo richiesto dal processo P_2 (4 millisecondi), perciò si ha la prelazione sul processo P_1 sostituendolo col processo P_2. Il tempo d'attesa medio per questo esempio è $((10-1)+(1-1)+(17-2)+(5-3))/4 = 26/4 = 6{,}5$ millisecondi. Con uno scheduling SJF senza prelazione si otterrebbe un tempo d'attesa medio di 8,75 millisecondi.

6.3.3 Scheduling per priorità

L'algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità: si associa una priorità a ogni processo e si assegna la CPU al processo con priorità più alta; i processi con priorità uguali si ordinano secondo uno schema FCFS.

Un algoritmo SJF è semplicemente un algoritmo con priorità in cui la priorità (p) è l'inverso della lunghezza (prevista) della successiva sequenza di operazioni della CPU. A una maggiore lunghezza corrisponde una minore priorità, e viceversa. Occorre notare che la discussione si svolge nei termini di priorità *alta* e priorità *bassa*. Generalmente le priorità sono un intervallo di numeri fissato, come da 0 a 7, oppure da 0 a 4095. Tuttavia, non si è ancora stabilito se attribuire allo 0 la priorità più alta o quella più bassa; alcuni sistemi usano numeri bassi per rappresentare priorità basse, altri usano numeri bassi per priorità alte. In questo testo i numeri bassi indicano priorità alte.

Come esempio, si consideri il seguente insieme di processi, che si suppone siano arrivati al tempo 0, nell'ordine P_1, P_2 ... P_5, e dove la durata delle sequenze di operazioni della CPU è espressa in millisecondi:

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Processo	Durata della sequenza	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Usando lo scheduling per priorità, questi processi sarebbero ordinati secondo il seguente schema di Gantt:

Il tempo d'attesa medio è di 8,2 millisecondi.

Le priorità si possono definire sia internamente sia esternamente. Le priorità definite internamente usano una o più quantità misurabili per calcolare la priorità del processo; ad esempio, i limiti di tempo, i requisiti di memoria, il numero dei file aperti e il rapporto tra la lunghezza media delle sequenze di operazioni di I/O e la lunghezza media delle sequenze di operazioni della CPU. Le priorità esterne si definiscono secondo criteri esterni al sistema operativo, come l'importanza del processo, il tipo e la quantità dei fondi pagati per l'uso del calcolatore, il dipartimento che promuove il lavoro e altri fattori, spesso di ordine politico.

Lo scheduling per priorità può essere sia con prelazione sia senza prelazione. Quando un processo arriva alla coda dei processi pronti, si confronta la sua priorità con la priorità del processo attualmente in esecuzione. Un algoritmo di scheduling per priorità e con diritto di prelazione sottrae la CPU al processo attualmente in esecuzione se la priorità dell'ultimo processo arrivato è superiore. Un algoritmo di scheduling senza diritto di prelazione si limita a porre l'ultimo processo arrivato alla testa della coda dei processi pronti.

Un problema importante relativo agli algoritmi di scheduling per priorità è l'**attesa indefinita** (*starvation*, letteralmente, inedia). Un processo pronto per l'esecuzione ma che non dispone della CPU, si può considerare bloccato nell'attesa della CPU. Un algoritmo di scheduling per priorità può lasciare processi con bassa priorità nell'attesa indefinita della CPU. Un flusso costante di processi con priorità maggiore può impedire a un processo con bassa priorità di accedere alla CPU. Generalmente accade una delle cose seguenti: o il processo è eseguito, alle ore 2 a. m. della domenica, quando il sistema ha finalmente ridotto il proprio carico, oppure il calcolatore si sovraccarica al punto di perdere tutti i processi con bassa priorità non terminati. Corre voce che, quando fu fermato l'IBM 7094 al MIT, nel 1973, si scoprì che un processo con bassa priorità sottoposto nel 1967 non era ancora stato eseguito.

Una soluzione al problema dell'attesa indefinita dei processi con bassa priorità è costituita dall'**invecchiamento** (*aging*); si tratta di una tecnica di aumento graduale delle priorità dei processi che attendono nel sistema da parecchio tempo. Ad esempio, se le priorità variano da 127 (bassa) a 0 (alta), si potrebbe decrementare di 1 ogni 15 minuti la priorità di un processo in attesa. Anche un processo con priorità iniziale 127 può ottenere la priorità massima nel sistema e quindi essere eseguito: un processo con priorità 127 non impiega più di 32 ore per raggiungere la priorità 0.

6.3.4 Scheduling circolare

L'**algoritmo di scheduling circolare** (*round-robin* — RR) è stato progettato appositamente per i sistemi a partizione del tempo; è simile allo scheduling FCFS ma ha in più la capacità di prelazione per la commutazione dei processi. Ciascun processo riceve una piccola quantità fissata del tempo della CPU, chiamata **quanto di tempo** (o **porzione di tempo**), che varia generalmente da 10 a 100 millisecondi, e la coda dei processi pronti è trattata come una coda circolare. Lo scheduler della CPU scorre la coda dei processi pronti, assegnando la CPU a ciascun processo per un intervallo di tempo della durata massima di un quanto di tempo. dopo di che il processo è costretto a rientrare la CPU e viene ricaricato nella Ready queue

Per realizzare lo scheduling RR si gestisce la coda dei processi pronti come una coda FIFO. I nuovi processi si aggiungono alla fine della coda dei processi pronti. Lo scheduler della CPU individua il primo processo dalla coda dei processi pronti, imposta un temporizzatore in modo che invii un segnale d'interruzione alla scadenza di un intervallo pari a un quanto di tempo, e attiva il dispatcher per l'effettiva esecuzione del processo.

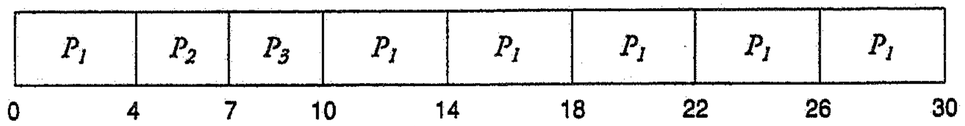
A questo punto si può verificare una delle seguenti situazioni: il processo ha una sequenza di operazioni della CPU di durata minore di un quanto di tempo, quindi il processo stesso rilascia volontariamente la CPU e lo scheduler passa al processo successivo della coda dei processi pronti; oppure la durata della sequenza di operazioni della CPU del processo attualmente in esecuzione è più lunga di un quanto di tempo, in questo caso si raggiunge la scadenza del quanto di tempo e il temporizzatore invia un segnale d'interruzione al sistema operativo, che esegue un cambio di contesto, aggiunge il processo alla fine della coda dei processi pronti e, tramite lo scheduler della CPU, seleziona il processo successivo nella coda dei processi pronti. Il tempo d'attesa medio per il criterio di scheduling RR è spesso abbastanza lungo. Si consideri il seguente insieme di processi che si presenta al tempo 0, con la durata delle sequenze di operazioni della CPU espressa in millisecondi:

Processo	Durata della sequenza
P_1	24
P_2	3
P_3	3

Se si usa un quanto di tempo di 4 millisecondi, il processo P_1 ottiene i primi 4 millisecondi ma, poiché richiede altri 20 millisecondi, è soggetto a prelazione dopo il primo

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

quanto di tempo e la CPU passa al processo successivo della coda, il processo P_2. Poiché il processo P_2 non necessita di 4 millisecondi, termina prima che il suo quanto di tempo si esaurisca, così si assegna immediatamente la CPU al processo successivo, il processo P_3. Una volta che tutti i processi hanno ricevuto un quanto di tempo, si assegna nuovamente la CPU al processo P_1 per un ulteriore quanto di tempo. Dallo scheduling RR risulta quanto segue:



Il tempo d'attesa medio è di 17/3 = 5,66 millisecondi.

Nell'algoritmo di scheduling RR la CPU si assegna a un processo per non più di un quanto di tempo per volta. Se la durata della sequenza di operazioni della CPU di un processo eccede il quanto di tempo, il processo viene sottoposto a *prelazione* e riportato nella coda dei processi pronti. L'algoritmo di scheduling RR è pertanto con prelazione.

Se nella coda dei processi pronti esistono n processi e il quanto di tempo è pari a q, ciascun processo ottiene $1/n$-esimo del tempo di elaborazione della CPU in frazioni di, al più, q unità di tempo, allora ogni processo non deve attendere per più di $(n - 1) \times q$ unità di tempo. Ad esempio, dati cinque processi e un quanto di tempo di 20 millisecondi, ogni processo usa 20 millisecondi ogni 100 millisecondi.

Le prestazioni dell'algoritmo RR dipendono molto dalla dimensione del quanto di tempo. Nel caso limite in cui il quanto di tempo è molto lungo (indefinito), il criterio di scheduling RR si riduce al criterio di scheduling FCFS. Se il quanto di tempo è molto breve (ad esempio, un microsecondo), il criterio RR si chiama **condivisione della CPU** e teoricamente gli utenti hanno l'impressione che ciascuno degli n processi disponga di una propria CPU in esecuzione a $1/n$ della velocità della CPU reale. Questo metodo fu usato nell'architettura del sistema della Control Data Corporation (CDC) per simulare 10 unità d'elaborazione con 10 gruppi di registri e una sola CPU, che eseguiva un'istruzione per un gruppo di registri, quindi procedeva col successivo. Questo ciclo continuava, con il risultato che si avevano 10 unità d'elaborazione lente al posto di una CPU veloce (in effetti, poiché la CPU era molto più rapida della memoria e ogni istruzione faceva riferimento alla memoria, le unità d'elaborazione simulate non erano molto più lente di quanto sarebbero state dieci vere unità d'elaborazione).

Riguardo alle prestazioni dello scheduling RR, occorre tuttavia considerare l'effetto dei cambi di contesto. Dato un solo processo della durata di 10 unità di tempo, se il quanto di tempo è di 12 unità, il processo impiega meno di un quanto di tempo; se però il quanto di tempo è di 6 unità, il processo richiede 2 quanti di tempo e un cambio di contesto; e se il quanto di tempo è di una unità di tempo, occorrono nove cambi di contesto, con proporzionale rallentamento dell'esecuzione del processo. Tale situazione è visibile nello schema della Figura 6.4.

Da ciò che si è affermato segue che il quanto di tempo deve essere ampio rispetto alla durata del cambio di contesto; se, ad esempio, questa è pari al 10 per cento del quanto di tempo, allora s'impiega nel cambio di contesto circa il 10 per cento del tempo d'elaborazione della CPU.

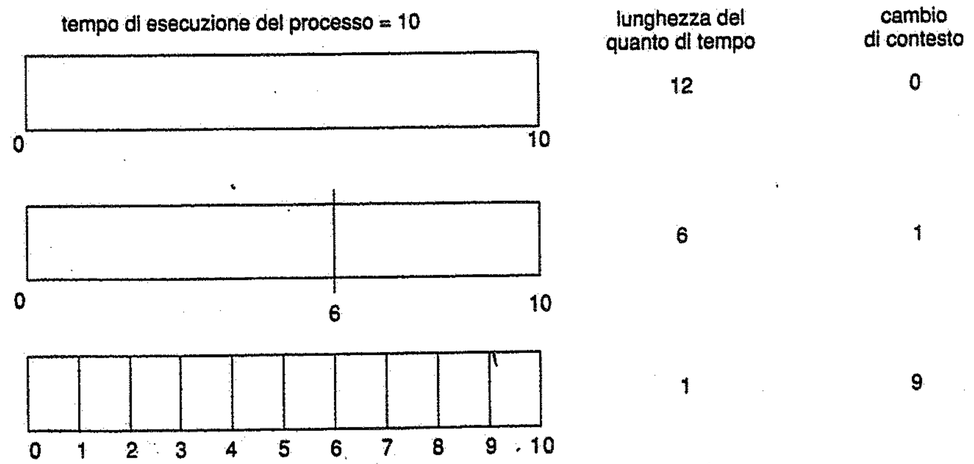


Figura 6.4 Un quanto di tempo minore incrementa il numero di cambi di contesto.

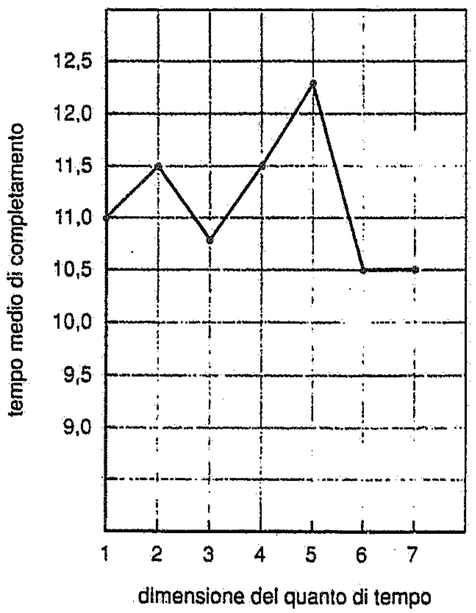
Anche il tempo di completamento dipende dalla dimensione del quanto di tempo: com'è evidenziato nella Figura 6.5, il tempo di completamento medio di un insieme di processi non migliora necessariamente con l'aumento della dimensione del quanto di tempo. In generale, il tempo di completamento medio può migliorare se la maggior parte dei processi termina la successiva sequenza di operazioni della CPU in un solo quanto di tempo. Ad esempio, dati tre processi della durata di 10 unità di tempo ciascuno e un quanto di una unità di tempo, il tempo di completamento medio è di 29 unità. Se però il quanto di tempo è di 10 unità, il tempo di completamento medio scende a 20 unità. Aggiungendo il tempo del cambio di contesto, con un piccolo quanto di tempo, il tempo di completamento medio aumenta poiché sono richiesti più cambi di contesto.

D'altra parte, se il quanto di tempo è molto ampio, il criterio di scheduling RR tende al criterio FCFS. Empiricamente si può stabilire che l'80 per cento delle sequenze di operazioni della CPU debba essere più breve del quanto di tempo.

6.3.5 Scheduling a code multiple

È stata creata una classe di algoritmi di scheduling adatta a situazioni in cui i processi si possono classificare facilmente in gruppi diversi. Una distinzione diffusa è ad esempio quella che si fa tra i processi che si eseguono in primo piano (*foreground*), o **interattivi**, e i processi che si eseguono in **sottofondo** (*background*), o a **lotti** (*batch*). Questi due tipi di processi hanno tempi di risposta diversi e possono quindi avere diverse necessità di scheduling. Inoltre, i processi che si eseguono in primo piano possono avere la priorità, definita esternamente, sui processi che si eseguono in sottofondo.

http://www.addtpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.



processo	tempo
P_1	6
P_2	3
P_3	1
P_4	7

Figura 6.5 Variazione del tempo di completamento rispetto alla dimensione del quanto di tempo.

L'algoritmo di scheduling a code multiple (*multilevel queue scheduling algorithm*) suddivide la coda dei processi pronti in code distinte (Figura 6.6). I processi si assegnano in modo permanente a una coda, generalmente secondo qualche caratteristica del processo, come la quantità di memoria richiesta, la priorità o il tipo di processo. Ogni coda ha il proprio algoritmo di scheduling. Ad esempio, per i processi in primo piano e i processi in sottofondo si possono usare code distinte. La coda dei processi in primo piano si può gestire con un algoritmo RR, mentre la coda dei processi in sottofondo si può gestire con un algoritmo FCFS.

In questa situazione è inoltre necessario avere uno scheduling tra le code; si tratta comunemente di uno scheduling per priorità fissa e con prelazione. Ad esempio, la coda dei processi in primo piano può avere la priorità assoluta sulla coda dei processi in sottofondo.

Si consideri il seguente algoritmo di scheduling a code multiple:

1. processi di sistema;
2. processi interattivi;
3. processi interattivi di *editing*;
4. processi eseguiti in sottofondo;
5. processi degli studenti.

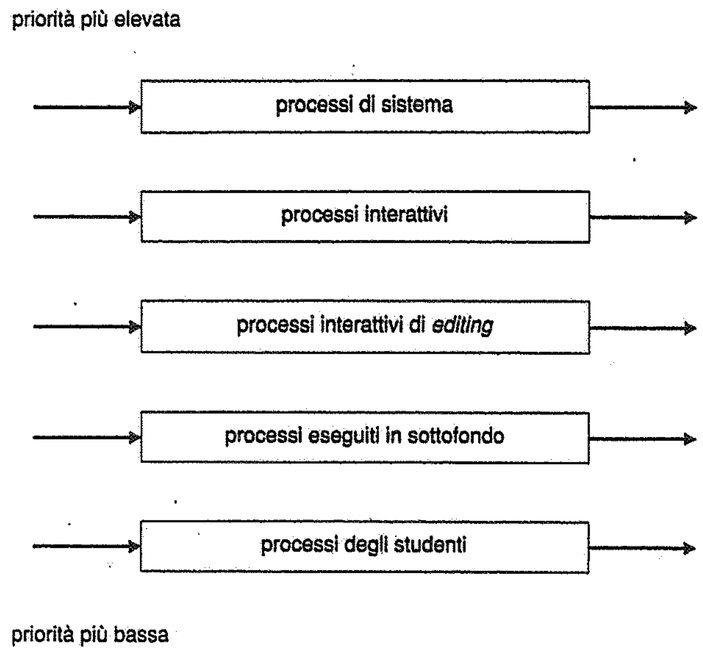


Figura 6.6 Scheduling a code multiple.

Ogni coda ha la priorità assoluta sulle code di priorità più bassa; nessun processo della coda dei processi in sottofondo può iniziare l'esecuzione finché le code per i processi di sistema, i processi interattivi e i processi interattivi di *editing* non sono tutte vuote. Se un processo interattivo di *editing* entrasse nella coda dei processi pronti durante l'esecuzione di un processo in sottofondo, si avrebbe la prelazione su quest'ultimo. Il sistema operativo Solaris 2 usa un algoritmo di questo tipo.

Esiste anche la possibilità di impostare i quanti di tempo per le code. Per ogni coda si stabilisce una parte del tempo d'elaborazione della CPU, che si può a sua volta suddividere tra i processi che la costituiscono. Nell'esempio precedente, si può assegnare l'80 per cento del tempo d'elaborazione della CPU alla coda dei processi in primo piano, per lo scheduling RR tra i suoi processi; mentre per la coda dei processi in sottofondo si riserva il 20 per cento del tempo d'elaborazione della CPU, da assegnare ai suoi processi secondo il criterio FCFS.

6.3.6 Scheduling a code multiple con retroazione

Di solito, in un algoritmo di scheduling a code multiple, i processi si assegnano in modo permanente a una coda all'entrata nel sistema, e non si possono spostare tra le code. Se, ad esempio, esistono code distinte per i processi che si eseguono in primo piano e i pro-

http://www.adultpdf.com
Created by TIFF Tools 4.1 - Trial version, to remove this mark, please register this software.

cessi che si eseguono in sottofondo, i processi non possono passare da una coda all'altra, poiché non possono cambiare la loro natura di processi in primo piano o in sottofondo. Quest'impostazione è rigida ma ha il vantaggio di avere un basso carico di scheduling.

Lo **scheduling a code multiple con retroazione** (*multilevel feedback queue scheduling*), invece, permette ai processi di spostarsi fra le code. L'idea consiste nel separare i processi che hanno caratteristiche diverse nelle sequenze delle operazioni della CPU. Se un processo usa troppo tempo di elaborazione della CPU, viene spostato in una coda con priorità più bassa. Questo schema mantiene i processi con prevalenza di I/O e i processi interattivi nelle code con priorità più elevata. Analogamente, si può spostare in una coda con priorità più elevata un processo che attende troppo a lungo. In questo modo si attua una forma d'invecchiamento che impedisce il verificarsi di un'attesa indefinita.

Si consideri, ad esempio, uno scheduler a code multiple con retroazione con tre code, numerate da 0 a 2, come nella Figura 6.7. Lo scheduler fa eseguire tutti i processi presenti nella coda 0; quando la coda 0 è vuota, si eseguono i processi nella coda 1; analogamente, i processi nella coda 2 si eseguono solo se le code 0 e 1 sono vuote. Un processo in ingresso nella coda 1 ha la prelazione sui processi della coda 2; un processo in ingresso nella coda 0, a sua volta, ha la prelazione sui processi della coda 1.

All'ingresso nella coda dei processi pronti, i processi vengono assegnati alla coda 0 e ottengono un quanto di tempo di 8 millisecondi; i processi che non terminano entro tale quanto di tempo, vengono spostati alla fine della coda 1. Se la coda 0 è vuota, si assegna un quanto di tempo di 16 millisecondi al processo alla testa della coda 1, ma se questo non riesce a completare la propria esecuzione, viene sottoposto a prelazione e messo nella coda 2. Se le code 0 e 1 sono vuote, si eseguono i processi della coda 2 secondo il criterio FCFS.

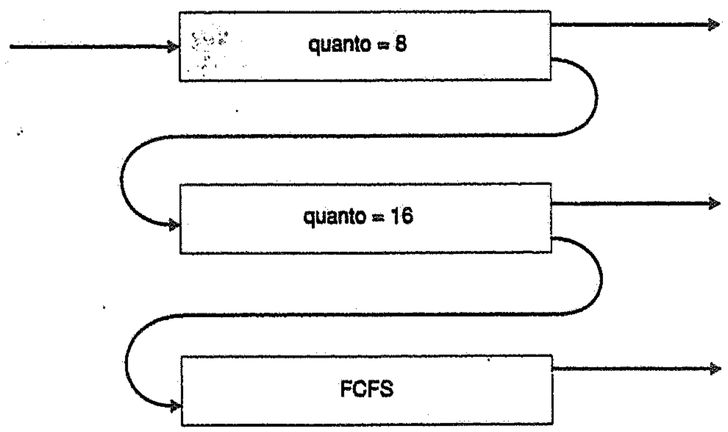


Figura 6.7 Code multiple con retroazione.

Questo algoritmo di scheduling dà la massima priorità ai processi con una sequenza di operazioni della CPU della durata di non più di 8 millisecondi. I processi di questo tipo ottengono rapidamente la CPU, terminano la propria sequenza di operazioni della CPU e passano alla successiva sequenza di operazioni di I/O; anche i processi che necessitano di più di 8 ma di non più di 24 millisecondi (coda 1) vengono serviti rapidamente. I processi più lunghi finiscono nella coda 2 e sono serviti secondo il criterio FCFS all'interno dei cicli di CPU lasciati liberi dai processi delle code 0 e 1.

Generalmente uno scheduler a code multiple con retroazione è caratterizzato dai seguenti parametri:

- numero di code;
- algoritmo di scheduling per ciascuna coda;
- metodo usato per determinare quando spostare un processo in una coda con priorità maggiore;
- metodo usato per determinare quando spostare un processo in una coda con priorità minore;
- metodo usato per determinare in quale coda si deve mettere un processo quando richiede un servizio.

La definizione di uno scheduler a code multiple con retroazione costituisce il più generale criterio di scheduling della CPU, che nella fase di progettazione si può adeguare a un sistema specifico. Sfortunatamente necessita anche di alcuni mezzi per scegliere i valori di tutti i parametri che definiscono lo scheduler migliore. Il criterio di scheduling che si avvale delle code multiple con retroazione è il più generale ma anche il più complesso.

6.4 Scheduling per sistemi con più unità d'elaborazione

Fin qui la trattazione ha riguardato i problemi inerenti lo scheduling della CPU in un sistema con una sola CPU; se sono disponibili più unità d'elaborazione, anche il problema dello scheduling è proporzionalmente più complesso. Si sono sperimentate diverse possibilità e, come s'è visto nella trattazione dello scheduling di una sola CPU, non esiste 'la soluzione migliore'. Nel seguito si analizzano brevemente alcuni argomenti attinenti lo scheduling per più unità d'elaborazione (una trattatazione completa esula dallo scopo di questo testo; per un approfondimento della materia si vedano le Note bibliografiche). Si considerano i sistemi nei quali le unità d'elaborazione sono, in relazione alle loro funzioni, identiche (**sistemi omogenei**): si può usare qualunque unità d'elaborazione disponibile per eseguire qualsiasi processo presente nella coda. Si assume inoltre un sistema d'accesso uniforme alla memoria (*uniform memory access* — UMA). Nei Capitoli dal 15 al 17 si trattano i sistemi con unità d'elaborazione diverse (**sistemi eterogenei**), una specifica unità d'elaborazione può eseguire solo i programmi compilati per la propria serie di

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

istruzioni di macchina. Possono in ogni modo esserci limitazioni relative allo scheduling anche nel caso di sistemi omogenei. Si consideri, ad esempio, un sistema con un dispositivo di I/O collegato a un bus privato di un'unità d'elaborazione: i processi che devono usare tale dispositivo devono essere eseguiti da tale unità d'elaborazione, altrimenti il dispositivo non sarebbe disponibile.

Se sono disponibili più unità d'elaborazione identiche, invece, si può avere una condivisione del carico. Si potrebbe impiegare una coda distinta per ciascuna unità d'elaborazione, ma in questo caso un'unità d'elaborazione potrebbe essere inattiva, con una coda vuota, mentre un'altra unità d'elaborazione potrebbe essere molto impegnata. Per impedire il verificarsi di tale situazione, si usa una coda dei processi pronti comune: tutti i processi fanno parte di un'unica coda e si assegnano a una qualsiasi unità d'elaborazione disponibile.

In uno schema di questo tipo si possono seguire due criteri di scheduling. Un criterio prevede che ogni unità d'elaborazione esamini la coda dei processi pronti comune e scelga un processo da eseguire. Come si descrive nel Capitolo 7, se più unità d'elaborazione tentano di accedere e aggiornare una struttura di dati comune, occorre che ciascuna unità d'elaborazione sia programmata con la massima attenzione: si deve essere certi che due unità d'elaborazione non scelgano lo stesso processo e che i processi non vadano perduti dalla coda. L'altro criterio evita questo problema fissando un'unità d'elaborazione per lo scheduling di tutte le altre unità d'elaborazione, creando così una struttura gerarchica.

In alcuni sistemi si fa un ulteriore passo nella stessa direzione: tutte le decisioni di scheduling, l'elaborazione delle operazioni di I/O e le altre attività di sistema sono gestite da un'unica unità d'elaborazione, che quindi assume il ruolo di unità centrale d'elaborazione (il cosiddetto *master server*); le altre unità d'elaborazione eseguono soltanto il codice d'utente. Tale schema, detto **multielaborazione asimmetrica**, è assai più semplice della multielaborazione simmetrica — essendoci una sola unità d'elaborazione che può accedere alle strutture di dati di sistema, diminuisce la condivisione dei dati — ma non è altrettanto efficiente: i processi con prevalenza di I/O possono produrre strozzature in una CPU che esegue tutte le operazioni. La multielaborazione asimmetrica s'include comunemente in sistemi operativi che poi evolveranno verso la multielaborazione simmetrica.

In alcuni sistemi la CPU ha il compito di effettuare non solo lo scheduling ma anche l'elaborazione delle I/O e le altre attività di sistema

6.5 Scheduling per sistemi d'elaborazione in tempo reale

Nel Capitolo 1 si è parlato dei sistemi operativi per le elaborazioni in tempo reale e della loro crescente importanza; in questo paragrafo si descrivono le caratteristiche che un algoritmo di scheduling deve avere affinché un sistema progettato per scopi generali possa gestire le elaborazioni in tempo reale.

Esistono due categorie di elaborazioni in tempo reale. Con il termine **tempo reale stretto** (*hard real-time*) s'intendono i sistemi capaci di garantire il completamento delle funzioni critiche entro un tempo definito. In generale, in questi sistemi un processo si

Checked by 16/05/03

presenta insieme con una dichiarazione del tempo entro cui deve completare le proprie funzioni o eseguire delle operazioni di I/O. Se è possibile garantirne il completamento entro i termini di tempo dichiarati, lo scheduler accetta il processo, altrimenti rifiuta la richiesta, poiché è impossibile soddisfarla. Questo metodo prende il nome di **prenotazione delle risorse**. Per poter garantire il rispetto delle scadenze, lo scheduler deve conoscere esattamente la durata dell'esecuzione di qualsiasi tipo di funzione del sistema, quindi è necessario che l'esecuzione di ogni operazione non duri di più del tempo massimo specificato. Questa garanzia non si può rispettare nei sistemi con memoria virtuale o con memorie secondarie, poiché queste caratteristiche implicano un'inevitabile quanto imprevedibile variabilità del tempo necessario all'esecuzione di un processo. Per questo motivo i sistemi per le elaborazioni in tempo reale stretto sono composti da programmi specifici, eseguiti da architetture progettate per la gestione dei processi critici. Per contro, questi sistemi non offrono le funzioni dei moderni calcolatori e dei sistemi operativi più progrediti.

I vincoli imposti per le elaborazioni in tempo reale debole (*soft real-time*) sono meno restrittivi, giacché si limitano a richiedere che i processi critici abbiano una priorità maggiore dei processi ordinari. Si possono quindi estendere in questa direzione le funzioni dei sistemi d'elaborazione in tempo reale, anche se ciò può significare un criterio d'assegnazione delle risorse iniquo che può ritardare l'esecuzione di alcuni processi, e condurre perfino a situazioni d'attesa indefinita. Si ottiene un sistema d'uso generale capace di offrire, oltre le funzioni tradizionali, un ambiente adatto all'esecuzione delle applicazioni multimediali, alla grafica interattiva ad alte prestazioni e a una varietà di compiti che non potrebbero funzionare in maniera accettabile in un ambiente ordinario.

La realizzazione delle funzioni di elaborazione in tempo reale debole richiede un accurato lavoro di progettazione dello scheduler e degli altri componenti del sistema operativo a esso correlati. Innanzi tutto il sistema deve disporre di un algoritmo di scheduling per priorità, in modo da poter associare le priorità più elevate ai processi d'elaborazione in tempo reale. Inoltre, la priorità dei processi d'elaborazione in tempo reale non deve diminuire con il trascorrere del tempo, anche se ciò può accadere per i processi tradizionali. Infine, la latenza di dispatch deve essere bassa: quanto più breve è quest'intervallo, tanto più celermente un processo d'elaborazione in tempo reale pronto per essere eseguito può iniziare le proprie elaborazioni. Garantire il soddisfacimento della seconda proprietà è relativamente semplice. Ad esempio, si potrebbe pensare di non consentire l'invecchiamento dei processi d'elaborazione in tempo reale, garantendo in questo modo l'inalterabilità delle priorità di questi processi. Viceversa, garantire il soddisfacimento della terza proprietà è assai più complicato. Il problema risiede nel fatto che molti sistemi operativi, compresa la maggior parte delle versioni dello UNIX, prima di poter eseguire un cambio di contesto devono attendere il completamento della chiamata del sistema attualmente in esecuzione o il verificarsi di un blocco di I/O. Quindi, vista la complessità di alcune chiamate del sistema e la lentezza di certi dispositivi di I/O, la latenza di dispatch in questi sistemi può diventare piuttosto lunga.

Per mantenere bassa la latenza di dispatch è necessario consentire che anche le chiamate del sistema possano essere soggette a prelazione. Questo risultato si può ottenere in diversi modi. Uno consiste nell'inserire dei **punti di prelazione** all'interno delle chiama-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

te del sistema di durata particolarmente lunga, in corrispondenza dei quali verificare la presenza di processi ad alta priorità che devono essere eseguiti. Alla presenza di uno di tali processi, il sistema esegue il cambio di contesto e, una volta terminata l'esecuzione del processo ad alta priorità, si riprende l'esecuzione della chiamata del sistema sottoposta a prelazione. Questi punti si possono inserire solamente nelle locazioni 'sicure' del codice del nucleo, cioè dove non si modificano le strutture di dati dello stesso nucleo. Anche questo metodo, comunque, non garantisce la soluzione del problema, poiché il numero di punti di prelazione applicabili al nucleo è, nella pratica, assai esiguo.

Un'alternativa consiste nel far sì che l'intero nucleo possa essere soggetto a prelazione. Al fine di assicurare un corretto funzionamento, tutte le strutture di dati del nucleo si devono proteggere attraverso l'uso dei metodi di sincronizzazione discussi nel Capitolo 7. Questo metodo consente di avere la prelazione sul nucleo in qualsiasi istante, poiché garantisce l'integrità delle strutture di dati di sistema contro eventuali modifiche da parte di processi ad alta priorità. Tra i metodi più diffusi, è il più efficace (e complesso) ed è usato nel Solaris 2.

Si consideri cosa accadrebbe se un processo ad alta priorità richiedesse di leggere o modificare strutture di dati del nucleo cui ha correntemente accesso un altro processo con minore priorità. Il primo dovrebbe attendere il completamento del secondo, determinando una situazione nota come **inversione delle priorità**. Ci si potrebbe trovare in una situazione in cui esiste una catena di processi che accedono alle risorse richieste dal processo ad alta priorità. Il problema si può risolvere usando un **protocollo di ereditarietà delle priorità**, grazie al quale i processi che accedono alle risorse richieste dal processo a priorità più elevata ereditano temporaneamente l'alto grado di priorità, fino al rilascio della risorsa contesa. Dopodiché, i processi riprendono la loro ordinaria priorità.

Nella Figura 6.8 è illustrata la composizione della latenza di dispatch. La fase conflittuale della latenza di dispatch consiste in due parti:

1. prelazione di tutti i processi correntemente in esecuzione all'interno del nucleo;
2. rilascio da parte dei processi a bassa priorità delle risorse richieste dal processo ad alta priorità.

Nel Solaris 2, ad esempio, la latenza di dispatch con la prelazione disabilitata è di oltre 100 millisecondi; con la prelazione abilitata si riduce, di solito, a 2 millisecondi.

6.6 Valutazione degli algoritmi

Ci si può chiedere come si deve scegliere un algoritmo di scheduling della CPU per un sistema particolare. Come si discute nel Paragrafo 6.3, esistono molti algoritmi di scheduling, ciascuno dotato dei propri parametri; quindi, la scelta di un algoritmo può essere abbastanza difficile.

Il primo problema da affrontare riguarda la definizione dei criteri da usare per scegliere l'algoritmo. Nel Paragrafo 6.2 si spiega che i criteri si definiscono spesso nei termini dell'utilizzo della CPU, del tempo di risposta o della produttività. Per scegliere un al-

sistema specifico bisogna utilizzare dei criteri di scelta:
- utilizzo della CPU
- Produttività
- Tempo di completamento
- Tempo di attesa
- Tempo di risposta

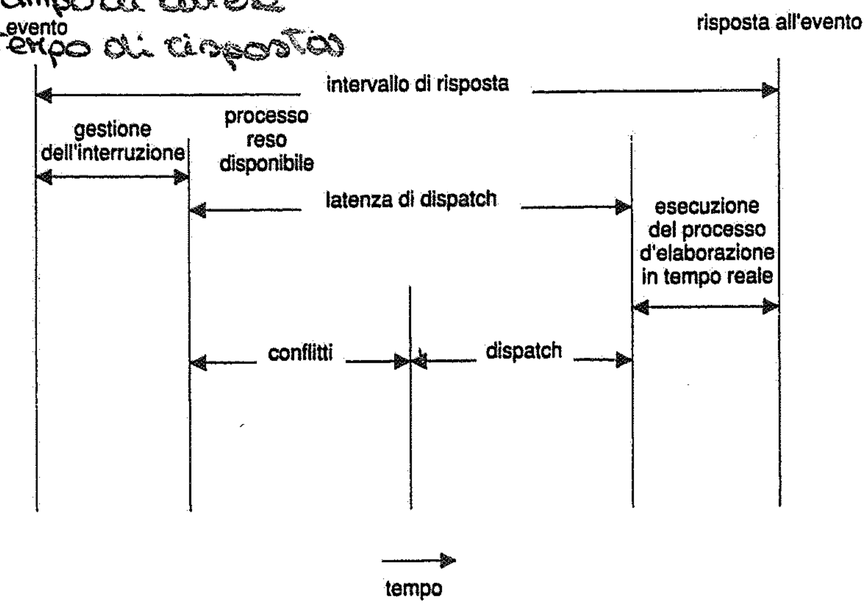


Figura 6.8 Latenza di dispatch.

goritmo occorre innanzi tutto stabilire l'importanza relativa di queste misure. Tra i criteri suggeriti si possono inserire diverse misure, ad esempio le seguenti:

- rendere massimo l'utilizzo della CPU con il vincolo che il massimo tempo di risposta debba essere 1 secondo;
- rendere massima la produttività in modo che il tempo di completamento sia (in media) linearmente proporzionale al tempo d'esecuzione totale.

Una volta definiti i criteri di selezione, è necessario valutare gli algoritmi considerati. Nei Paragrafi dal 6.6.1 al 6.6.4 si descrivono i vari metodi di valutazione.

6.6.1 Modelli deterministici

Fra i metodi di valutazione sono di grande importanza quelli che rientrano nella classe della valutazione analitica. La valutazione analitica, secondo l'algoritmo dato e il carico di lavoro del sistema, fornisce una formula o un numero che valuta le prestazioni dell'algoritmo per quel carico di lavoro.

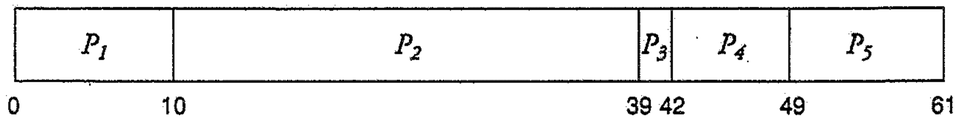
La definizione e lo studio di un **modello deterministico** è un tipo di valutazione analitica, che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Si supponga, ad esempio, di avere il carico di lavoro illustrato di seguito; i cinque processi si presentano al tempo 0, nell'ordine dato, e la durata delle sequenze di operazioni della CPU è espressa in millisecondi:

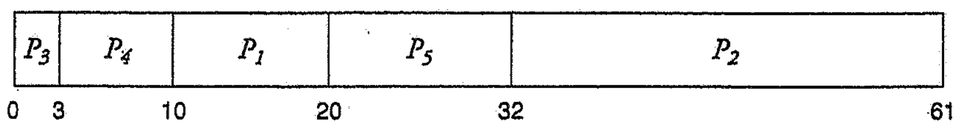
Processo	Durata della sequenza
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Si può stabilire con quale fra gli algoritmi di scheduling FCFS, SJF e RR (quanto di tempo = 10 millisecondi) per questo insieme di processi si ottiene il minimo tempo medio d'attesa. Con l'algoritmo FCFS i processi si eseguono secondo lo schema seguente:



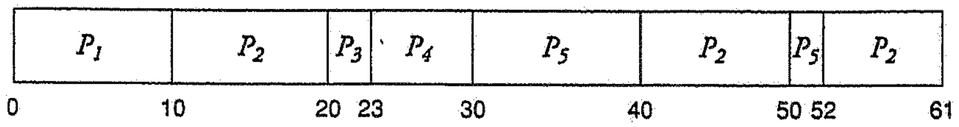
Il tempo d'attesa è di 0 millisecondi per il processo P_1, di 10 millisecondi per il processo P_2, di 39 millisecondi per il processo P_3, di 42 millisecondi per il processo P_4 e di 49 millisecondi per il processo P_5. Quindi, il tempo d'attesa medio è di (0 + 10 + 39 + 42 + 49)/5 = 28 millisecondi.

Con l'algoritmo SJF senza prelazione i processi si eseguono come segue:



Il tempo d'attesa è di 10 millisecondi per il processo P_1, di 32 millisecondi per il processo P_2, di 0 millisecondi per il processo P_3, di 3 millisecondi per il processo P_4 e di 20 millisecondi per il processo P_5. Quindi, il tempo d'attesa medio è di (10 + 32 + 0 + 3 + 20)/5 = 13 millisecondi.

Con l'algoritmo RR il processo P_2 viene sottoposto a prelazione dopo 10 millisecondi d'esecuzione e posto in fondo alla coda:



Il tempo d'attesa è di 0 millisecondi per il processo P_1, di 32 millisecondi per il processo P_2, di 20 millisecondi per il processo P_3, di 23 millisecondi per il processo P_4 e di 40 millisecondi per il processo P_5. Quindi, il tempo d'attesa medio è di (0 + 32 + 20 + 23 + 40)/5 = 23 millisecondi.

È importante notare come, *in questo caso*, il criterio SJF fornisca come risultato un tempo medio d'attesa minore della metà del tempo corrispondente ottenuto con lo scheduling FCFS; l'algoritmo RR fornisce un risultato intermedio tra i precedenti.

La definizione e lo studio di un modello deterministico è semplice e rapida; i risultati sono numeri esatti che consentono il confronto tra gli algoritmi. Nondimeno, anche i parametri devono essere numeri esatti e i risultati sono applicabili solo a quei casi. Il suo impiego principale consiste nella descrizione degli algoritmi di scheduling e nella presentazione d'esempi. Nei casi in cui si possono eseguire ripetutamente gli stessi programmi e si possono misurare con precisione i requisiti d'elaborazione dei programmi, si possono usare i modelli deterministici per scegliere un algoritmo di scheduling. Lo studio dei modelli deterministici rispetto a un insieme d'esempi può indicare tendenze che si possono poi analizzare e verificare separatamente. Si può ad esempio mostrare che per l'ambiente descritto, vale a dire tutti i processi e i relativi tempi disponibili al tempo 0, con il criterio SJF si ottiene sempre il tempo d'attesa minimo.

Tuttavia l'analisi dei modelli deterministici è eccessivamente specifica, e per essere effettivamente utile richiede conoscenze troppo dettagliate.

6.6.2 Reti di code

I processi eseguiti in molti sistemi variano di giorno in giorno, quindi non esiste un insieme statico di processi (e di tempi) da usare nei modelli deterministici. Si possono però determinare le distribuzioni delle sequenze di operazioni della CPU e delle sequenze di operazioni di I/O, poiché si possono misurare e quindi approssimare, o più semplicemente stimare. Si ottiene una formula matematica che indica la probabilità di una determinata sequenza di operazioni della CPU. Comunemente questa distribuzione è di tipo esponenziale ed è descritta dalla sua media. Analogamente, è necessario fornire anche la distribuzione degli istanti d'arrivo dei processi nel sistema.

Il sistema di calcolo si descrive come una rete di unità serventi, ciascuna con una coda d'attesa. La CPU è un'unità servente con la propria coda dei processi pronti, e il sistema di I/O ha le sue code dei dispositivi. Se sono noti l'andamento degli arrivi e dei servizi, si possono calcolare l'utilizzo, la lunghezza media delle code, il tempo medio d'attesa e così via. Questo tipo di studio si chiama **analisi delle reti di code**.

Si consideri il seguente esempio: sia n la lunghezza media di una coda, escluso il processo correntemente servito, detti W il tempo medio d'attesa nella coda e λ l'andamento medio d'arrivo dei nuovi processi nella coda (ad esempio, 3 processi al secondo); si prevede che, nel tempo W durante il quale un processo attende nella coda, $\lambda \times W$ nuovi processi arrivino alla coda. Se il sistema è stabile, il numero dei processi che lasciano la coda deve essere uguale al numero dei processi che vi arrivano; quindi,

$$n = \lambda \times W$$

FORMULA DI LITTLE

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Quest'equazione è nota come formula di Little ed è utile soprattutto perché è valida per qualsiasi algoritmo di scheduling e distribuzione d'arrivi.

La formula di Little si può usare per calcolare una delle tre variabili, quando sono note le altre due. Ad esempio, sapendo che ogni secondo arrivano sette processi (in media), e che normalmente nella coda sono presenti 14 processi, si può calcolare che il tempo medio d'attesa per ogni processo è di 2 secondi.

L'analisi delle reti di code può essere utile per il confronto degli algoritmi di scheduling, ma ha comunque alcuni limiti. Attualmente le classi di algoritmi e distribuzioni trattabili sono piuttosto limitate. Inoltre, poiché può essere difficile lavorare con la matematica di distribuzioni e algoritmi complicati, spesso, affinché siano trattabili matematicamente, si definiscono in modo irrealistico le distribuzioni d'arrivo e servizio. Generalmente è necessario stabilire anche un numero di presupposti indipendenti che possono non essere precisi. Per poter calcolare una risposta, le reti di code spesso si limitano ad approssimare un sistema reale, rendendo discutibile la precisione dei risultati ottenuti.

6.6.3 Simulazioni

Per riuscire ad avere una valutazione più precisa degli algoritmi di scheduling ci si può servire di simulazioni. Le simulazioni implicano la programmazione di un modello del sistema di calcolo; le strutture di dati rappresentano gli elementi principali del sistema. Il simulatore dispone di una variabile che rappresenta un clock; con l'aumentare del valore di questa variabile, il simulatore modifica lo stato del sistema in modo da descrivere le attività dei dispositivi, dei processi e dello scheduler. Durante l'esecuzione della simulazione si raccolgono e si stampano statistiche che indicano le prestazioni degli algoritmi.

I dati necessari per condurre la simulazione si possono ottenere in vari modi. Il metodo più diffuso impiega un generatore di numeri casuali, programmato per generare processi, durata delle sequenze di operazioni della CPU, arrivi, conclusioni e così via; in modo conforme alle rispettive distribuzioni di probabilità, che si possono definire matematicamente (esponenziali, uniformi, di Poisson) oppure in modo empirico. Se la distribuzione deve essere definita in modo empirico, si fanno misure sul sistema reale in esame, e si usano i risultati per definire la distribuzione effettiva degli eventi nel sistema reale.

Tuttavia, una simulazione condotta secondo la distribuzione può non essere precisa, a causa delle relazioni esistenti tra eventi successivi nel sistema reale. La distribuzione relativa alle frequenze, infatti, si limita a indicare quanti eventi di una data categoria si verificano, senza fornire informazioni sul loro ordine. Per rimediare a questo problema si può sottoporre il sistema reale a un controllo continuo, con la registrazione della sequenza degli eventi effettivi — in questo modo si ottiene un cosiddetto *trace tape* — (Figura 6.9), che poi si usa per condurre la simulazione. Si tratta di uno strumento eccellente che permette di confrontare gli algoritmi rispetto allo stesso insieme di dati reali, e con cui si possono ottenere risultati molto precisi.

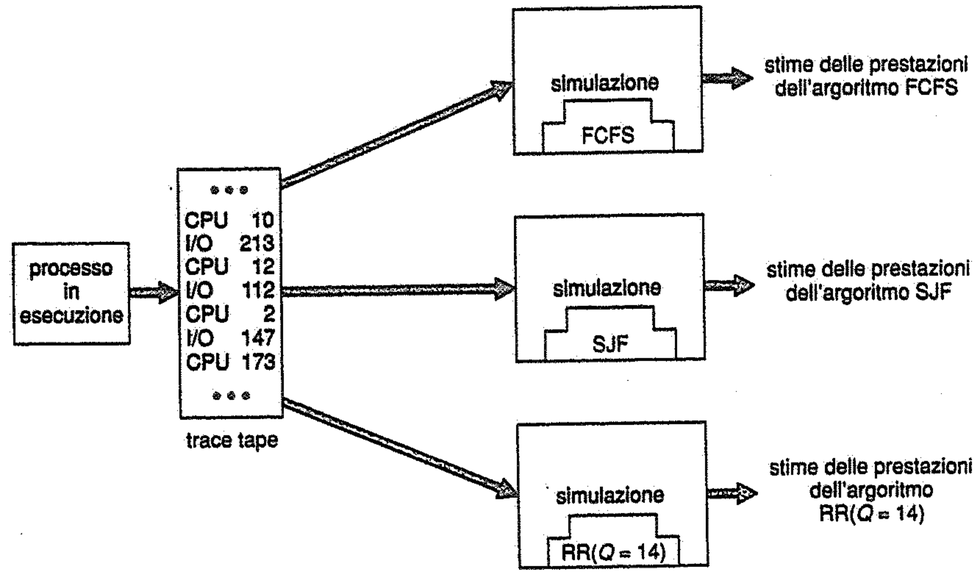


Figura 6.9 Valutazione di algoritmi di scheduling della CPU tramite una simulazione.

Poiché spesso richiedono diverse ore del tempo d'elaborazione, le simulazioni possono tuttavia essere molto onerose. Una simulazione molto dettagliata dà risultati molto precisi, ma richiede anche molto tempo, e molto spazio di memoria per la registrazione degli eventi. Inoltre, la progettazione, la codifica e la messa a punto di un simulatore possono essere un compito assai impegnativo.

6.6.4 Realizzazione

Persino una simulazione ha dei limiti per quel che riguarda la precisione. L'unico modo assolutamente sicuro per valutare un algoritmo di scheduling consiste nel codificarlo, inserirlo nel sistema operativo e osservarne il comportamento nelle reali condizioni di funzionamento del sistema.

Il problema principale di questo metodo è il suo costo; le spese non sono dovute solo alla codifica dell'algoritmo e alle modifiche da fare al sistema operativo affinché possa gestire l'algoritmo con le sue strutture di dati, ma anche alle reazioni degli utenti a fronte di costanti modifiche del sistema operativo. Alla maggior parte degli utenti non interessa la realizzazione di un sistema operativo migliore, ma soltanto eseguire i processi e usare i risultati. Un sistema operativo che si trovi costantemente in fasi di modifica e messa a punto non aiuta gli utenti a svolgere il loro lavoro. Un metodo di questo tipo si

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

usa comunemente nelle nuove installazioni dei calcolatori. Per una nuova funzione per il Web, ad esempio, si può simulare un carico d'utenza prima dell'effettiva pubblicazione, in modo da determinare le inefficienze presenti nella funzione e stimare il numero di utenti che il sistema può sostenere.

L'altra difficoltà da affrontare per fare qualsiasi valutazione di un algoritmo è dovuta al cambiamento dell'ambiente in cui lo si usa. L'ambiente non cambia solo nel modo consueto, cioè per la scrittura di nuovi programmi e il cambiamento dei tipi di problemi che si possono riscontrare, ma cambia anche in seguito alle prestazioni dello scheduler. Se si dà la priorità ai processi brevi, gli utenti possono suddividere i processi più lunghi in gruppi di processi brevi. Se si dà la priorità ai processi interattivi rispetto ai processi non interattivi, gli utenti possono passare all'uso interattivo.

Nel DEC TOPS-20, ad esempio, il sistema classificava automaticamente i processi interattivi e non interattivi secondo la quantità di I/O effettuati al terminale. Se un processo non eseguiva I/O al terminale a intervalli minori di un minuto, questo processo era classificato come non interattivo e spostato in una coda con priorità più bassa. Il programmatore reagiva a questo tipo di criterio modificando i suoi programmi, istruendoli a scrivere un carattere qualsiasi al terminale a intervalli regolari di durata inferiore a un minuto. Il sistema assegnava ai suoi programmi una priorità elevata, anche se l'I/O al terminale era totalmente privo di significato.

Gli algoritmi di scheduling più flessibili possono essere modificati dai gestori del sistema. Nella fase di realizzazione del sistema operativo, nella fase di avviamento, o nella fase d'esecuzione, si possono modificare le variabili usate dagli scheduler per riflettere l'utilizzo atteso del sistema. Per ottenere un sistema di scheduling flessibile è utile la separazione dei meccanismi dai criteri. Se ad esempio si devono elaborare e stampare immediatamente alcuni assegni, ma le stampe appartengono alla categoria delle elaborazioni a lotti a bassa priorità, si può assegnare temporaneamente una priorità più elevata alla coda delle elaborazioni a lotti. Sfortunatamente i sistemi operativi che permettono questo tipo di scheduling regolabile sono pochi.

6.7 Modelli di scheduling dei processi

Questo paragrafo tratta lo scheduling dei processi nei sistemi operativi Solaris 2, Windows 2000, e LINUX. Prima di procedere con la descrizione di questi diversi modelli di scheduling, è necessario però precisare la relazione fra thread e scheduling dei processi.

Nel Capitolo 5 si descrive l'introduzione dei thread nel modello dei processi, che permette a un singolo processo di avere più thread di controllo, distinguendo i thread del *livello d'utente* da quelli del *livello del nucleo*. I thread del livello d'utente sono gestiti da una libreria, mentre il nucleo non ne conosce l'esistenza. Per essere eseguiti da una CPU, ai thread del livello d'utente si fa corrispondere un thread del livello del nucleo, tale corrispondenza può essere indiretta tramite un processo leggero (LWP). Una differenza fra i thread del livello d'utente e quelli del livello del nucleo sta nel metodo di scheduling:

la libreria dei thread compie lo scheduling dei thread del livello d'utente per farli eseguire da un LWP disponibile, si tratta di uno schema chiamato **scheduling dei processi locale** (*process local scheduling*), in cui lo scheduling dei thread si svolge nell'ambito dell'applicazione; viceversa, il nucleo impiega il cosiddetto **scheduling di sistema globale** (*system global scheduling*) per decidere quale thread del livello del nucleo si debba eseguire. In questo testo non si considerano i dettagli riguardanti lo scheduling locale dei thread svolto dalle diverse librerie — si tratta di un problema delle librerie per la programmazione e non del sistema operativo — ma si considera lo scheduling globale poiché è una delle attività del sistema operativo.

6.7.1 Un esempio: Solaris 2

Il sistema operativo Solaris 2 adotta uno scheduling dei processi basato sulle priorità, e considera quattro classi di scheduling: *real time*, *system*, *time sharing* e *interactive*; in ordine di priorità. Sebbene le classi *time sharing* e *interactive* seguano gli stessi criteri di scheduling, ogni classe prevede diverse priorità e diversi algoritmi di scheduling (Figura 6.10).

Un processo comincia la propria esecuzione con un LWP e, secondo le necessità, può creare altri LWP che ereditano la classe di scheduling e la priorità del processo genitore. La classe di scheduling predefinita per un processo è la *time sharing*. Il criterio di scheduling per questa classe prevede che si modifichino dinamicamente le priorità e si assegnino quanti di tempo di lunghezza diversa servendosi di una coda multipla con retroazione. Normalmente, c'è una relazione inversa tra priorità e quanti di tempo: a una priorità alta corrisponde un quanto di tempo breve; a una priorità bassa corrisponde un quanto di tempo lungo. Di solito i processi interattivi hanno priorità alta e i processi con prevalenza d'elaborazione hanno priorità bassa. Questo criterio di scheduling offre un buon tempo di risposta per i processi interattivi e una buona produttività per i processi con prevalenza d'elaborazione. Con la versione 2.4 del sistema operativo è stata introdotta nel modello di scheduling dei processi la classe *interactive*, che segue gli stessi criteri di scheduling della classe *time sharing*, ma, per ottenere migliori prestazioni, assegna priorità più elevate alle applicazioni che fanno uso d'interfacce a finestre.

I processi del nucleo, come lo scheduler e il *demone* che si occupa della paginazione (*paging daemon*), appartengono alla classe *system*; i processi utenti eseguiti nel modo del nucleo non appartengono alla classe *system*. Una volta fissata, la priorità di un processo della classe *system* non cambia. Il criterio di scheduling per la classe system non è basato sui quanti di tempo; un thread di questa classe è eseguito fino a quando si blocca oppure viene sottoposto a prelazione da un thread a più alta priorità.

I thread della classe *real time* ottengono le priorità più alte fra tutte le classi, si eseguono prima dei thread appartenenti alle altre classi e di solito sono pochi. Ciò permette a un processo d'elaborazione in tempo reale di ottenere una risposta dal sistema entro un tempo limitato.

http://www.adultpdf.com
Created by TIFF To PDF Trial Version, to remove this mark, please register this software.

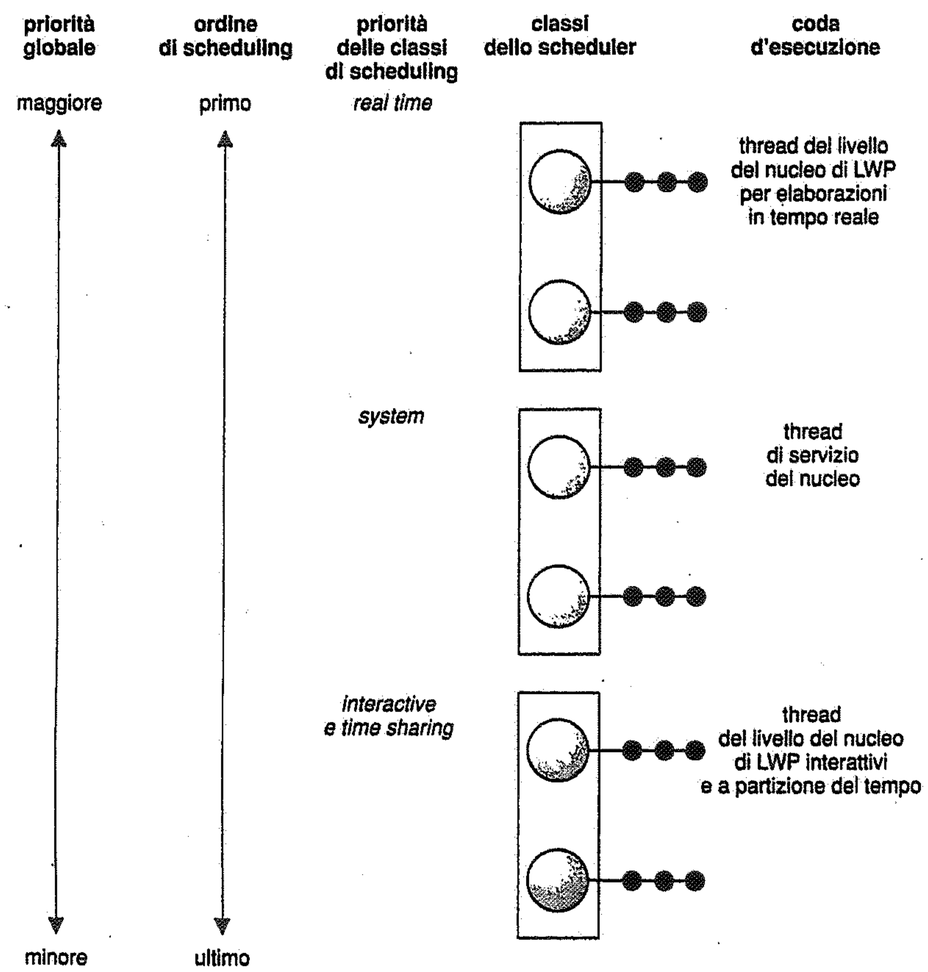


Figura 6.10 Scheduling nel sistema operativo Solaris 2.

Ciascuna classe di scheduling ha un proprio insieme di priorità; lo scheduler converte le priorità specifiche della classe in priorità globali e per l'esecuzione sceglie il thread con la priorità globale più elevata. La CPU esegue il thread prescelto fino all'occorrenza di uno dei seguenti eventi:

1. il thread si blocca;
2. il thread esaurisce il suo quanto di tempo (se non è un thread della classe *system*);
3. il thread viene sottoposto a prelazione da un thread con priorità più alta.

Se più thread hanno la stessa priorità, lo scheduler si serve di una coda RR.

6.7.2 Un esempio: Windows 2000

Il sistema operativo Windows 2000 compie lo scheduling dei thread servendosi di un algoritmo basato su priorità e prelazione. Lo scheduler assicura che si eseguano sempre i thread a più alta priorità. La porzione del nucleo che si occupa dello scheduling si chiama *dispatcher*. Una volta selezionato dal *dispatcher*, un thread viene eseguito fino a quando non è sottoposto a prelazione da un altro thread a priorità più alta oppure termina, esaurisce il suo quanto di tempo o esegue una chiamata del sistema bloccante, ad esempio un'operazione di I/O. Se un thread d'elaborazione in tempo reale, ad alta priorità, entra nella coda dei processi pronti per l'esecuzione mentre è in esecuzione un thread a bassa priorità, quest'ultimo viene sottoposto a prelazione. Ciò realizza un accesso preferenziale alla CPU per i thread d'elaborazione in tempo reale che ne hanno necessità. Tuttavia, il Windows 2000 non è un sistema operativo per elaborazioni in tempo reale stretto, poiché non è in grado di garantire un limite di tempo prefissato entro il quale un thread d'elaborazione in tempo reale comincia la sua esecuzione.

Il *dispatcher* impiega uno schema di priorità a 32 livelli per determinare l'ordine d'esecuzione dei thread. Le priorità sono divise in due classi: la classe *variable* raccoglie i thread con priorità da 1 a 15, mentre la classe *real-time* raccoglie i thread con priorità tra 16 e 31 (esiste anche un thread, per la gestione della memoria, che si esegue con priorità 0). Il *dispatcher* adopera una coda per ciascuna priorità di scheduling e percorre l'insieme delle code da quella a priorità più alta a quella a priorità più bassa, fino a quando trova un thread pronto per l'esecuzione. In assenza di tali thread, il *dispatcher* fa eseguire un thread speciale detto *idle thread*.

C'è una relazione tra le priorità numeriche del nucleo del sistema operativo Windows 2000 e quelle dell'API Win32. Secondo l'API Win32 un processo può appartenere a diverse classi di priorità, tra cui le seguenti:

- REALTIME_PRIORITY_CLASS;
- HIGH_PRIORITY_CLASS;
- ABOVE_NORMAL_PRIORITY_CLASS;
- NORMAL_PRIORITY_CLASS;
- BELOW_NORMAL_PRIORITY_CLASS;
- IDLE_PRIORITY_CLASS.

Tutte le classi di priorità eccetto REALTIME_PRIORITY_CLASS sono priorità di classe variabile, cioè la priorità di un thread che appartiene a queste classi può cambiare.

Ciascuna di queste classi prevede delle priorità relative, i cui valori comprendono i seguenti:

- TIME_CRITICAL;
- HIGHEST;
- ABOVE_NORMAL;

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- NORMAL;
- BELOW_NORMAL;
- LOWEST;
- IDLE.

La priorità di ciascun thread dipende dalla priorità della classe cui appartiene e dalla priorità relativa che il thread ha all'interno della stessa classe. Questa relazione è rappresentata nella Figura 6.11. I valori di ciascuna classe di priorità sono riportati nella prima riga in alto. La prima colonna a sinistra contiene i valori delle diverse priorità relative. Ad esempio, se la priorità relativa di un thread nella classe ABOVE_NORMAL_PRIORITY_CLASS è NORMAL, la priorità numerica di quel thread è 10.

Inoltre, ogni thread ha una priorità di base che rappresenta un valore nell'intervallo di priorità della classe di appartenenza. Il valore predefinito per la priorità di base in una classe è quello della priorità relativa NORMAL per quella classe. Le priorità di base per ciascuna classe di priorità sono le seguenti:

- REALTIME_PRIORITY_CLASS–24;
- HIGH_PRIORITY_CLASS–13;
- ABOVE_NORMAL_PRIORITY_CLASS–10;
- NORMAL_PRIORITY_CLASS–8;
- BELOW_NORMAL_PRIORITY_CLASS–6;
- IDLE_PRIORITY_CLASS–4.

	REALTIME	HIGH	ABOVE_ NORMAL	NORMAL	BELOW_ NORMAL	IDLE_ PRIORITY
TIME_CRITICAL	31	15	15	15	15	15
HIGHEST	26	15	12	10	8	6
ABOVE_NORMAL	25	14	11	9	7	5
NORMAL	24	13	10	8	6	4
BELOW_NORMAL	23	12	9	7	5	3
LOWEST	22	11	8	6	4	2
IDLE	16	1	1	1	1	1

Figura 6.11 Priorità nel sistema operativo Windows 2000.

Di solito, i processi fanno parte della classe NORMAL_PRIORITY_CLASS, sempre che il processo genitore non appartenga alla classe IDLE_PRIORITY_CLASS, o sia stata specificata un'altra classe alla creazione del processo. Di solito la priorità iniziale di un thread è la priorità di base del processo al quale il thread appartiene.

Quando il quanto di tempo di un thread si esaurisce, il thread viene interrotto e se il thread fa parte della classe a priorità variabile, la sua priorità viene ridotta. Tuttavia, la priorità non si abbassa mai oltre la priorità di base. L'abbassamento della priorità tende a limitare l'uso della CPU da parte dei thread con prevalenza d'elaborazione. Se un thread a priorità variabile è rilasciato da un'operazione d'attesa, il *dispatcher* aumenta la sua priorità. L'entità di questo aumento dipende dal tipo d'evento che il thread attendeva: un thread che attendeva dati dalla tastiera riceve un forte aumento di priorità, uno che attendeva operazioni relative a un disco riceve un aumento più moderato. Questa strategia mira a fornire buoni tempi di risposta per i thread interattivi, con interfacce basate su mouse e finestre, e permette ai thread con prevalenza di I/O di tenere occupati i dispositivi di I/O, e rende nel contempo possibile l'utilizzo con esecuzione in sottofondo dei cicli di CPU inutilizzati da parte dei thread con prevalenza d'elaborazione. Questa strategia si segue in molti sistemi operativi a partizione del tempo d'elaborazione, compreso lo UNIX. Inoltre, per migliorare il tempo di risposta, la finestra attraverso cui l'utente sta interagendo ottiene un incremento di priorità.

Quando un utente richiede l'esecuzione di un programma interattivo, il sistema deve fornire al relativo processo prestazioni particolarmente elevate. Per questa ragione, il sistema Windows 2000 segue una regola specifica di scheduling per i processi della classe NORMAL_PRIORITY_CLASS. Il sistema operativo Windows 2000 distingue tra il *processo in primo piano*, correntemente selezionato sullo schermo e i *processi in sottofondo*, che non sono attualmente selezionati. Quando un processo passa in primo piano, il sistema aumenta il suo quanto di tempo di un certo fattore, tipicamente pari a 3; ciò fa sì che il processo in primo piano possa continuare la propria esecuzione per un tempo tre volte più lungo, prima che si abbia una prelazione dovuta alla partizione del tempo d'elaborazione.

6.7.3 Un esempio: LINUX

Il sistema operativo LINUX prevede due diversi algoritmi di scheduling: uno è un algoritmo a partizione del tempo d'elaborazione, per uno scheduling con diritto di prelazione dei processi equo; l'altro è progettato per le elaborazioni in tempo reale in cui le priorità assolute sono più importanti dell'equità. Nel Paragrafo 6.5, è descritta una situazione in cui i sistemi d'elaborazione in tempo reale devono consentire la prelazione del nucleo per mantenere bassa la latenza di dispatch. Il LINUX permette la prelazione dei soli processi eseguiti nel modo d'utente; un processo del nucleo non si può sottoporre a prelazione anche se un processo in tempo reale, con una priorità più elevata, è pronto per l'esecuzione.

Una parte dell'identità di ogni processo è costituita dalla classe di scheduling, secondo la quale si sceglie l'algoritmo da applicare al processo; le classi adottate dal LINUX

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sono definite dalle estensioni dello standard POSIX all'elaborazione in tempo reale (POSIX.4, oggi noto come POSIX.1b).

La prima categoria di scheduling è quella dei processi a partizione del tempo; per i processi a partizione del tempo convenzionali il LINUX usa un algoritmo a priorità basato sui crediti: ogni processo possiede un certo numero di crediti di scheduling; quando occorre assegnare la CPU a un nuovo processo, si sceglie quello che ha il massimo numero di crediti. A ogni interruzione generata dal temporizzatore, il processo correntemente in esecuzione perde un credito; quando i suoi crediti sono ridotti a zero esso viene sospeso e si sceglie un altro processo per l'esecuzione.

Se tutti i processi eseguibili hanno esaurito i crediti, il sistema operativo procede a una riassegnazione dei crediti, aggiungendo crediti a *tutti* i processi del sistema, e non solo a quelli eseguibili, secondo la regola seguente:

$$crediti = \frac{crediti}{2} + priorità$$

Quest'algoritmo pondera due fattori: la storia e la priorità dei processi. Si riassegna al processo metà dei crediti già in suo possesso; si tratta di un modo di tener conto del suo comportamento recente: i processi che sono spesso in esecuzione esauriscono in fretta la scorta di crediti, ma quelli che rimangono sospesi per la maggior parte del tempo accumulano crediti grazie alle operazioni di riassegnazione dei crediti, e finiscono con l'avere un numero di crediti maggiore dopo ogni operazione di questo tipo. Si dà un'alta priorità ai processi interattivi o con prevalenza di I/O; in questi casi, infatti, è importante avere un basso tempo di risposta.

Usare le priorità dei processi per il calcolo dei nuovi crediti permette di regolare con precisione le priorità da assegnare a ciascun processo; ai processi che si eseguono in sottofondo si possono assegnare basse priorità; questi riceveranno automaticamente meno crediti dei processi interattivi degli utenti, usufruendo così di un tempo di CPU più breve rispetto a processi anche simili ma con maggiore priorità. Il sistema LINUX realizza in questo modo il meccanismo *nice* di gestione delle priorità dello UNIX.

Lo scheduling per elaborazioni in tempo reale è ancora più semplice; il LINUX dispone delle due classi di scheduling richieste dal documento POSIX.1b, la classe FCFS e la RR (Paragrafi 6.3.1 e 6.3.4, rispettivamente); in entrambi i casi si assegna una priorità a ogni processo. Nello scheduling a partizione del tempo, però, i processi con priorità diverse sono, almeno in una certa misura, in competizione; mentre nello scheduling in tempo reale lo scheduler esegue sempre il processo con priorità più alta, oppure, a parità di priorità, il processo che attende da più tempo. L'unica differenza fra i criteri FCFS e RR è che, nel primo, l'esecuzione di un processo continua fino alla terminazione o fino a quando esso stesso si blocca; nel secondo, un processo può essere sospeso e posto alla fine della coda di scheduling allo scadere del suo quanto di tempo; la conseguenza di ciò è che col criterio RR i processi con la stessa priorità si comportano come in un ordinario sistema a partizione del tempo d'elaborazione.

Si noti che lo scheduling per le elaborazioni in tempo reale del LINUX è uno scheduling in tempo reale debole e non in tempo reale stretto; offre rigide garanzie sulle priorità relative dei processi d'elaborazione in tempo reale, ma il nucleo non fornisce alcuna garanzia sulla rapidità con cui un processo d'elaborazione in tempo reale pronto per l'esecuzione sarà effettivamente eseguito una volta che sia stato scelto per l'esecuzione dallo scheduler: si ricordi a questo proposito che, nel LINUX, non si può mai sospendere l'esecuzione del codice del nucleo a vantaggio di un processo utente, perciò se un segnale d'interruzione dovesse rendere eseguibile un processo d'elaborazione in tempo reale mentre il nucleo sta eseguendo una chiamata del sistema per conto di un altro processo, il processo d'elaborazione in tempo reale dovrebbe attendere la terminazione o il blocco della chiamata del sistema correntemente in esecuzione.

6.8 Sommario

Lo scheduling della CPU consiste nella scelta di un processo dalla coda dei processi pronti cui assegnare la CPU. L'effettiva assegnazione della CPU al processo prescelto è eseguita dal dispatcher.

L'algoritmo di scheduling in ordine d'arrivo (FCFS) è il più semplice, ma può far sì che brevi processi attendano processi molto lunghi. Si dimostra che lo scheduling ottimale, che determina il minimo tempo medio d'attesa, è lo scheduling per brevità (SJF). Realizzare lo scheduling SJF è complicato, poiché è difficile prevedere la lunghezza della successiva sequenza di operazioni della CPU. L'algoritmo SJF è un caso particolare dell'algoritmo generale di scheduling per priorità, che si limita ad assegnare la CPU al processo con priorità più elevata. Sia lo scheduling per priorità sia lo scheduling SJF possono condurre a situazioni d'attesa indefinita. L'invecchiamento (*aging*) è una tecnica si usa per impedire che avvengano tali situazioni.

Lo scheduling circolare (RR) è il più appropriato per un sistema a partizione del tempo: si assegna la CPU al primo processo della coda dei processi pronti per q unità di tempo (quanto di tempo); dopodiché si ha la prelazione della CPU e si mette il processo in fondo alla coda dei processi pronti. Il problema principale è la scelta della durata del quanto di tempo; se è troppo lungo, lo scheduling RR si riduce a uno scheduling FCFS; se è troppo breve, il carico di scheduling dovuto al tempo dei cambi di contesto diventa eccessivo.

L'algoritmo FCFS è senza prelazione; l'algoritmo RR è con prelazione; gli algoritmi SJF e con priorità possono essere sia con prelazione sia senza prelazione.

Lo scheduling a code multiple permette l'uso di diversi algoritmi per diverse classi di processi. Le più comuni sono la coda dei processi interattivi, eseguiti in primo piano, con scheduling RR; e la coda per processi a lotti, eseguiti in sottofondo, con scheduling FCFS. Le code multiple con retroazione permettono ai processi di spostarsi da una coda all'altra.

Poiché esiste un'ampia varietà di algoritmi di scheduling, è necessario disporre di un metodo di selezione. I metodi analitici si servono di modelli matematici per determinare le prestazioni di un algoritmo; i metodi di simulazione determinano le prestazioni imitando l'algoritmo di scheduling su un campione rappresentativo di processi e calcolando le prestazioni risultanti.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I sistemi operativi che gestiscono i thread al livello del nucleo, devono occuparsi dello scheduling dei thread, e non dello scheduling dei processi. Tra questi vi sono il Solaris 2 e il Windows 2000; entrambi gestiscono lo scheduling dei thread impiegando un algoritmo di scheduling con diritto di prelazione, basato su priorità e che comprende i thread in tempo reale. Anche lo scheduler dei processi del LINUX impiega un algoritmo basato su priorità e che prevede la gestione dei processi in tempo reale. Gli algoritmi di scheduling di questi tre sistemi operativi favoriscono tipicamente i processi interattivi rispetto ai processi a lotti o con prevalenza d'elaborazione.

6.9 Esercizi

6.1 Un algoritmo di scheduling della CPU stabilisce un ordine d'esecuzione fra i processi sottoposti a scheduling. Dati n processi da sottoporre a scheduling per una CPU, calcolate il numero dei possibili ordinamenti diversi; fornite una formula in termini di n. $n!$

6.2 Definite la differenza tra scheduling con diritto di prelazione e scheduling senza diritto di prelazione. Spiegate perché è improbabile che uno scheduling strettamente senza prelazione sia usato in un centro di calcolo.

6.3 Considerate il seguente insieme di processi, con la durata della sequenza di operazioni della CPU espressa in millisecondi:

Processo	Durata della sequenza	Priorità
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Presumendo che i processi siano arrivati nell'ordine P_1, P_2, P_3, P_4, P_5, e siano tutti presenti al tempo 0,

a) disegnate quattro schemi di Gantt che illustrino l'esecuzione di questi processi con gli algoritmi di scheduling FCFS, SJF, con priorità senza prelazione (un numero di priorità più basso indica una priorità maggiore) e RR (quanto = 1);
b) calcolate il tempo di completamento di ciascun processo per ciascun algoritmo di scheduling di cui al punto a);
c) calcolate il tempo d'attesa di ciascun processo per ciascun algoritmo di scheduling di cui al punto a);
d) dite quale, fra le esecuzioni di cui al punto a), ha il minimo tempo medio d'attesa (per tutti i processi).

6.4 Supponete che i seguenti processi si presentino per l'esecuzione nei momenti indicati. Ogni processo sarà eseguito nella quantità di tempo elencata. Per rispondere alle domande, usate un criterio di scheduling senza diritto di prelazione e prendete le decisioni secondo le informazioni disponibili al momento della decisione.

Processo	Istante d'arrivo	Durata della sequenza
P_1	0,0	8
P_2	0,4	4
P_3	1,0	1

a) Calcolate il tempo di completamento medio per questi processi con l'algoritmo FCFS.
b) Calcolate il tempo di completamento medio per questi processi con l'algoritmo SJF.
c) Si suppone che l'algoritmo SJF migliori le prestazioni, ma occorre notare che si era scelto di eseguire il processo P_1 al tempo 0 poiché non era possibile sapere che presto sarebbero arrivati due processi più corti. Calcolate quale sarebbe il tempo di completamento medio se si lasciasse inattiva la CPU per la prima unità di tempo, e quindi se si usasse lo scheduling SJF. Ricordate che i processi P_1 e P_2 attendono per tutto questo periodo d'inattività, quindi il loro tempo d'attesa può aumentare. Questo algoritmo si potrebbe chiamare *scheduling con conoscenza del futuro.*

6.5 Data una variante dell'algoritmo di scheduling RR in cui gli elementi della coda dei processi pronti sono puntatori ai PCB,

a) descrivete l'effetto dell'inserimento di due puntatori allo stesso processo nella coda dei processi pronti;
b) descrivete principali vantaggi e svantaggi di questo schema;
c) ipotizzate una modifica all'algoritmo RR ordinario che consenta di ottenere lo stesso effetto senza duplicare i puntatori.

6.6 Spiegate il vantaggio offerto dal disporre di quanti di tempo di durata diversa ai vari livelli di un sistema con code multiple.

6.7 Considerate il seguente algoritmo di scheduling con diritto di prelazione, e basato su priorità variabili dinamicamente. I numeri di priorità maggiori indicano una maggiore priorità. Quando un processo attende la CPU (nella coda dei processi pronti), la sua priorità varia a un tasso α; quando è in esecuzione, la sua priorità varia a un tasso β. All'ingresso nella coda dei processi pronti, si attribuisce la priorità 0 a tutti i processi. I parametri α e β si possono impostare in modo da fornire algoritmi di scheduling diversi:

a) descrivete l'algoritmo risultante da $\beta > \alpha > 0$;
b) descrivete l'algoritmo risultante da $\alpha < \beta < 0$.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

6.8 Molti algoritmi di scheduling della CPU sono parametrici: l'algoritmo RR ad esempio richiede un parametro che indichi il quanto di tempo; l'algoritmo a code multiple con retroazione richiede parametri che definiscano il numero delle code, gli algoritmi di scheduling per ciascuna coda, i criteri usati per spostare i processi fra le code e così via. Quindi questi algoritmi sono di fatto insiemi di algoritmi (ad esempio, l'insieme degli algoritmi RR per tutti i quanti di tempo, e così via). Un insieme di algoritmi può contenerne un altro (ad esempio, l'algoritmo FCFS è l'algoritmo RR con un quanto di tempo infinito). Segnalate la relazione, se esiste, tra le seguenti coppie d'insiemi di algoritmi:

a) priorità e SJF; Sì
b) code multiple con retroazione e FCFS;
c) priorità e FCFS;
d) RR e SJF.

6.9 Supponete che un algoritmo di scheduling della CPU favorisca i processi che nel passato recente hanno usato il minor tempo d'elaborazione. Spiegate perché questo algoritmo favorisce i programmi con prevalenza di I/O e non causa l'attesa indefinita dei programmi con prevalenza d'elaborazione.

6.10 Spiegate le differenze tra i seguenti algoritmi di scheduling rispetto al livello di discriminazione in favore dei processi brevi:

a) FCFS;
b) RR;
c) code multiple con retroazione.

6.10 Note bibliografiche

Una trattazione generale dello scheduling è presente in [Lampson 1968]. Trattati più formali sulla teoria dello scheduling sono [Kleinrock 1975], [Sauer e Chandy 1981] e [Lazowska et al. 1984]. Un orientamento unificante allo scheduling è quello presentato in [Ruschitzka e Fabry 1977]. [Haldar e Subramanian 1991] discute il problema dell'equità (*fairness*) nello scheduling della CPU per i sistemi d'elaborazione in tempo reale.

Lo scheduling a code multiple con retroazione è stato realizzato originariamente sul sistema CTSS, descritto in [Corbato et al. 1962], ed è stato analizzato da [Schrage 1967]. Sue varianti sono state studiate in [Coffman e Kleinrock 1968]. Altri studi sono stati presentati in [Coffman e Denning 1973] e [Svobodova 1976]. Una struttura di dati per la manipolazione delle code con priorità è presente in [Vuillemin 1978].

[Anderson et al. 1989] discute lo scheduling dei thread. Lo scheduling per sistemi con più unità d'elaborazione è trattato in [Jones e Schwarz 1980], [Tucker e Gupta 1989], [Zahorjan e McCann 1990], [Feitelson e Rudolph 1990] e [Leutenegger e Vernon 1990].

Lo scheduling per i sistemi d'elaborazione in tempo reale è affrontato da [Liu e Layland 1973], [Abbot 1984], [Jensen et al. 1985], [Hong et al. 1989] e [Khanna et al. 1992]. [Zhao 1989] è un numero speciale sui sistemi operativi per elaborazioni in tempo reale. [Eykholt et al. 1992] descrive i componenti per l'elaborazione in tempo reale del Solaris 2.

Gli scheduler a equa condivisione (*fair share*) sono trattati in [Henry 1984], [Woodside 1986] e [Kay e Lauder 1988].

[Mauro e McDougall 2001], [Solomon e Russinovich 2000] e [Bovet e Cesati 2001] descrivono, rispettivamente, i metodi di scheduling dei sistemi Solaris 2, Windows 2000 e LINUX.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un'applicazione può utilizzare i vantaggi delle prestazioni offerte da processi e/o thread che cooperano in parallelo e/o in concorrenza

- Esecuzione concorrente → 2 o + processi o thread in progresso nello stesso istante
- Esecuzione parallela → 2 o + processi o thread in esecuzione nello stesso istante su + processori

La concorrenza ed il parallelismo introducono un sovrapporsi, dovuto alla sincronizzazione dei compiti dei processi e thread

Capitolo 7

Sincronizzazione dei processi

Un processo cooperante è un processo che può influenzare un altro processo in esecuzione nel sistema o anche subirne l'influenza. I processi cooperanti possono condividere direttamente uno spazio logico di indirizzi (cioè, codice e dati) oppure possono condividere dati soltanto attraverso i file. Nel primo caso si fa uso di processi leggeri o *thread*, trattati nel Capitolo 5. L'accesso concorrente a dati condivisi può causare situazioni di incoerenza degli stessi dati. In questo capitolo si trattano vari meccanismi atti ad assicurare un'ordinata esecuzione dei processi cooperanti, che condividono uno spazio logico di indirizzi, che preservi la coerenza dei dati.

7.1 Introduzione

Nel Capitolo 4 è descritto un modello di sistema formato di un certo numero di **processi sequenziali cooperanti**, tutti in esecuzione asincrona e con la possibilità di condividere dati. Tale modello è illustrato attraverso l'esempio dei produttori e dei consumatori con memoria limitata, che ben rappresenta molte situazioni che riguardano i sistemi operativi.

La soluzione del problema dei produttori e dei consumatori con memoria limitata, data nel Paragrafo 4.4 facendo uso di memoria condivisa, consente la presenza contemporanea nel vettore di non più di `DIM_VETTORE - 1` elementi. Si supponga di voler modificare l'algoritmo per rimediare a questa carenza. Una possibilità consiste nell'aggiungere una variabile intera, `contatore`, inizializzata a 0, che si incrementa ogniqualvolta s'inserisce un nuovo elemento nel vettore e si decrementa ogniqualvolta si preleva un elemento dal vettore. Il codice per il processo produttore si può modificare come segue:

```
while (1){
    /* produce un elemento in appena_prodotto */
    while (contatore == DIM_VETTORE)
        ; /* non fa niente */
    vettore[inserisci] = appena_prodotto;
    inserisci = (inserisci + 1) % DIM_VETTORE;
    contatore++;
}
```

http://www.adultpdf.com
Created by TIFF To PDF Trial Version, to remove this mark, please register this software.

eseguite ATOMICAMENTE
- un'operazione è ATOMICA quando viene completata senza subire interruzioni -

Il codice per il processo consumatore si può modificare come segue:

```
while (1){
    while (contatore == 0)
        ; /* non fa niente */
    da_consumare = vettore[preleva];
    preleva = (preleva + 1) % DIM_VETTORE;
    contatore--;
    /* consuma un elemento in da_consumare */
}
```

Sebbene siano corrette se si considerano separatamente, le procedure del produttore e del consumatore possono non funzionare altrettanto correttamente se si eseguono in modo concorrente. Si supponga ad esempio che il valore della variabile contatore sia attualmente 5, e che i processi produttore e consumatore eseguano le istruzioni contatore++ e contatore-- in modo concorrente. Terminata l'esecuzione delle due istruzioni, il valore della variabile contatore potrebbe essere 4, 5 o 6! Il solo risultato corretto è contatore == 5, che si ottiene se si eseguono separatamente il produttore e il consumatore.

Si può dimostrare che il valore di contatore può essere scorretto: l'istruzione contatore++ si può codificare in un tipico linguaggio di macchina, come

$$\texttt{registro}_1 \texttt{ := contatore;}$$
$$\texttt{registro}_1 \texttt{ := registro}_1 \texttt{ + 1;}$$
$$\texttt{contatore := registro}_1$$

dove $\texttt{registro}_1$ è un registro locale della CPU. Analogamente, l'istruzione contatore-- si può codificare come

$$\texttt{registro}_2 \texttt{ := contatore;}$$
$$\texttt{registro}_2 \texttt{ := registro}_2 \texttt{ - 1;}$$
$$\texttt{contatore := registro}_2$$

dove $\texttt{registro}_2$ è un registro locale della CPU. Anche se $\texttt{registro}_1$ e $\texttt{registro}_2$ possono essere lo stesso registro fisico, ad esempio un accumulatore, occorre ricordare che il contenuto di questo registro viene salvato e recuperato dal gestore dei segnali d'interruzione (Paragrafo 2.1).

L'esecuzione concorrente delle istruzioni contatore++ e contatore-- equivale a un'esecuzione sequenziale delle istruzioni del linguaggio di macchina introdotte precedentemente, intercalate (*interleaved*) in una qualunque sequenza che però conservi l'ordine interno di ogni singola istruzione di alto livello. Una di queste sequenze è

Se sia il produttore che il consumatore, tentano di accedere al buffer contemporaneamente, le istruzioni in linguaggio assembler possono essere intercalate

T_0:	*produttore*	esegue	$\texttt{registro}_1$:= contatore	{$\texttt{registro}_1$ = 5}
T_1:	*produttore*	esegue	$\texttt{registro}_1$:= $\texttt{registro}_1$ + 1	{$\texttt{registro}_1$ = 6}
T_2:	*consumatore*	esegue	$\texttt{registro}_2$:= contatore	{$\texttt{registro}_2$ = 5}
T_3:	*consumatore*	esegue	$\texttt{registro}_2$:= $\texttt{registro}_2$ − 1	{$\texttt{registro}_2$ = 4}
T_4:	*produttore*	esegue	contatore := $\texttt{registro}_1$	{contatore = 6}
T_5:	*consumatore*	esegue	contatore := $\texttt{registro}_2$	{contatore = 4}

e conduce al risultato errato in cui contatore == 4; si registra la presenza di 4 elementi nel vettore, mentre in realtà gli elementi nel vettore sono 5. Poiché entrambi i processi hanno la possibilità di modificare il valore della variabile contatore in modo concorrente, l'inversione dell'ordine delle istruzioni in T_4 e T_5 conduce allo stato errato in cui contatore == 6.

Per evitare le situazioni di questo tipo, in cui più processi accedono e modificano gli stessi dati in modo concorrente e i risultati dipendono dall'ordine degli accessi (le cosiddette *race condition*) occorre assicurare che un solo processo alla volta possa modificare la variabile contatore. Questa condizione richiede una forma di sincronizzazione dei processi. Tali situazioni capitano spesso nei sistemi operativi, nei quali diversi componenti compiono operazioni su risorse condivise, ma tali operazioni non devono interferire reciprocamente in modi indesiderati. La maggior parte di questo capitolo è dedicata ai problemi della **sincronizzazione e coordinazione dei processi**.

cioè quelle situazioni in cui + processi accedono in concorrenza e modificano dati condivisi

x evitarle occorre che processi concorrenti vengano SINCRONIZZATI in modo che nessuno che + di 1 processo alla stessa risorsa

7.2 Problema della sezione critica

Si consideri un sistema composto di n processi $\{P_0, P_1, \ldots, P_{n-1}\}$; ciascuno avente un segmento di codice, chiamato **sezione critica**, nel quale il processo può modificare variabili comuni, aggiornare una tabella, scrivere in un file e così via. Quando un processo è in esecuzione nella propria sezione critica, non si deve consentire a nessun altro processo di essere in esecuzione nella propria sezione critica. Quindi, l'esecuzione delle sezioni critiche da parte dei processi è *mutuamente esclusiva* nel tempo. Il problema della *sezione critica* si affronta progettando un protocollo che i processi possono usare per cooperare. Ogni processo deve chiedere il permesso per entrare nella propria sezione critica. La sezione di codice che realizza questa richiesta è la **sezione d'ingresso**. La sezione critica può essere seguita da una **sezione d'uscita**, e la restante parte del codice è detta **sezione non critica**.

Una soluzione del problema della sezione critica deve soddisfare i tre seguenti requisiti:

1. **Mutua esclusione.** Se il processo P_i è in esecuzione nella sua sezione critica, nessun altro processo può essere in esecuzione nella propria sezione critica.
2. **Progresso.** Se nessun processo è in esecuzione nella sua sezione critica e qualche processo desidera entrare nella propria sezione critica, solo i processi che si trovano fuori delle rispettive sezioni non critiche possono partecipare alla decisione riguardante la scelta del processo che può entrare per primo nella propria sezione critica; questa scelta non si può rimandare indefinitamente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

3. **Attesa limitata.** Se un processo ha già richiesto l'ingresso nella sua sezione critica, esiste un limite al numero di volte che si consente ad altri processi di entrare nelle rispettive sezioni critiche prima che si accordi la richiesta del primo processo.

Si suppone che ogni processo sia eseguito a una velocità diversa da zero. Tuttavia, non si può fare alcuna ipotesi sulla **velocità relativa** degli n processi.

Nei Paragrafi 7.2.1 e 7.2.2 si ricercano soluzioni al problema della sezione critica che soddisfino i tre requisiti sopra elencati. Le soluzioni non presuppongono alcun requisito particolare relativamente alle istruzioni offerte dall'architettura di macchina o al numero di unità d'elaborazione disponibili, tuttavia si suppone che le istruzioni di base del linguaggio di macchina, vale a dire le istruzioni fondamentali come `load`, `store` e `test`, siano eseguite in modo atomico. Ciò significa che l'esecuzione concorrente di due istruzioni di questo tipo equivale a un'esecuzione sequenziale in un ordine qualunque. Quindi, se si eseguono in modo concorrente un'istruzione `load` e un'istruzione `store`, l'istruzione `load` riceve il valore vecchio oppure quello nuovo, ma non una combinazione dei due.

Nella presentazione degli algoritmi si definiscono unicamente le variabili impiegate per la sincronizzazione e si descrive soltanto un tipico processo P_i, la cui struttura generale è mostrata nella Figura 7.1. La sezione d'ingresso e la sezione d'uscita sono racchiuse da un contorno affinché tali importanti segmenti di codice siano ben evidenziati.

7.2.1 Soluzione per due processi

In questo paragrafo la trattazione si limita agli algoritmi applicabili a due processi alla volta. I processi sono indicati con P_0 e P_1; per motivi di convenienza, una volta introdotto P_i l'altro processo si denota con P_j, dove $j = 1 - i$.

Struttura generale del processo P_i

```
do{
    [sezione d'ingresso]
        sezione critica
    [sezione d'uscita]
        sezione non critica
} while (1);
```

• I processi possono condividere alcune variabili x sincronizzare le loro azioni

Figura 7.1 Struttura generale di un tipico processo P_i.

Maccio & Checco

7.2.1.1 Algoritmo 1

Il primo tentativo di soluzione consiste nel far condividere ai processi una variabile intera, `turno`, inizializzata a `0` (oppure a `1`). Se `turno == i`, si permette al processo P_i di entrare nella propria sezione critica. La struttura del processo P_i è illustrata nella Figura 7.2.

Questa soluzione assicura che, in un dato momento, un solo processo può trovarsi nella propria sezione critica. Tuttavia la soluzione non soddisfa il requisito di progresso, poiché richiede una stretta alternanza dei processi nell'esecuzione della sezione critica. Se, ad esempio, `turno == 0`, P_1 non può entrare nella sua sezione critica, anche se P_0 si trova nella propria sezione non critica.

7.2.1.2 Algoritmo 2

L'Algoritmo 1 non possiede informazioni sufficienti sullo stato di ogni processo; ricorda solo il processo cui si permette l'ingresso nella propria sezione critica. Per rimediare a questo problema si può sostituire la variabile `turno` con il seguente vettore:

```
boolean pronto[2];
```

pronto[0] = pronto[1] = false

Gli elementi del vettore s'inizializzano a `false`. Se il valore di `pronto[i]` è `true`, significa che P_i *è pronto* per entrare nella sua sezione critica. La struttura del processo P_i è illustrata nella Figura 7.3.

In questo algoritmo il processo P_i assegna innanzi tutto il valore `true` a `pronto[i]`, segnalando che è pronto per entrare nella propria sezione critica, quindi verifica che il processo P_j non sia pronto per entrare nella sua sezione critica. Se P_j fosse pronto, P_i attenderebbe fino a che P_j indica che non intende più restare nella propria sezione critica, vale a dire fino a che `pronto[j]` è `false`. A questo punto P_i entrerebbe nella sezione critica, e all'uscita assegnerebbe il valore `false` a `pronto[i]`, permettendo a un altro processo, purché ce ne sia qualcuno in attesa, di entrare nella propria sezione critica. Questa soluzione soddisfa il requisito di mutua esclusione, ma non soddisfa il requisito di progresso.

Algoritmo (1)

```
do {
    while (turno != i);
        sezione critica
    turno = j;
        sezione non critica
} while (1);
```

Indica che fino a quando turno non è = i, il processo non entra nella sezione critica ma attende che turno = i ma resta –

Figura 7.2 Struttura del processo P_i nell'Algoritmo 1.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Algoritmo (2)

```
do {
    pronto[i] = true;
    while (pronto[j]);

        sezione critica

    pronto[i] = false;

        sezione non critica

} while (1);
```

Figura 7.3 Struttura del processo P_i nell'Algoritmo 2.

Per illustrare questo problema si consideri la seguente sequenza d'esecuzione:

T_0: P_0 assegna `pronto[0] = true`
T_1: P_1 assegna `pronto[1] = true`

P_0 e P_1 entrano in un ciclo infinito nelle rispettive istruzioni `while`.

Questo algoritmo dipende in modo decisivo dall'esatta temporizzazione dei due processi. La sequenza può derivare da un ambiente con più unità d'elaborazione, oppure da un ambiente in cui si presenta un'interruzione della CPU (ad esempio, un segnale d'interruzione proveniente dal temporizzatore) immediatamente dopo l'esecuzione del passo T_0, e la CPU passa all'altro processo.

Occorre notare che invertendo l'ordine delle istruzioni di assegnazione per `pronto[i]` e di controllo del valore di `pronto[j]` il problema non si risolve. L'inversione, infatti, determina a una situazione in cui entrambi i processi possono trovarsi contemporaneamente nelle rispettive sezioni critiche, violando il requisito di mutua esclusione.

7.2.1.3 Algoritmo 3 (PETERSON)

Combinando i concetti chiave dell'Algoritmo 1 e dell'Algoritmo 2 si può ottenere una soluzione corretta del problema della sezione critica, che soddisfa tutti e tre i requisiti. I processi condividono due variabili:

```
boolean pronto[2];
int turno;
```

Inizialmente `pronto[0] = pronto[1] = false`; il valore di `turno` è irrilevante (ma è 0 oppure 1). La struttura del processo P_i è illustrata nella Figura 7.4.

Algoritmo (3)

```
do {
    pronto[i] = true;
    turno = j;
    while (pronto[j] && turno == j);

        sezione critica

    pronto[i] = false;

        sezione non critica

} while (1);
```

x indicare che se il processo P_j non vuole entrare nella sua sezione critica.

Figura 7.4 Struttura del processo P_i nell'Algoritmo 3.

Per entrare nella sezione critica, il processo P_i prima assegna `true` a `pronto[i]`, e poi `j` a `turno` per asserire che l'altro processo, se vuole, può entrare nella sua sezione critica. Se entrambi i processi tentano di entrare contemporaneamente nella propria sezione critica, si assegna sia `i` sia `j` a `turno` pressoché nello stesso istante.

Ovviamente si conserva una sola assegnazione — l'altra avviene, ma è immediatamente sovrascritta — il cui valore consente di stabilire a quale dei due processi si debba permettere di entrare per primo nella propria sezione critica. Per dimostrare la correttezza di questa soluzione occorre verificare le seguenti proprietà:

1. mutua esclusione;
2. progresso;
3. attesa limitata.

Per quel che riguarda la proprietà 1 si nota che ogni P_i entra nella propria sezione critica solo se `pronto[j] == false` o `turno == i`. Inoltre se entrambi i processi fossero contemporaneamente in esecuzione nelle rispettive sezioni critiche, si avrebbe `pronto[0] == pronto[1] == true`. Queste due osservazioni implicano che P_0 e P_1 non possono avere eseguito contemporaneamente l'istruzione `while` poiché il valore di `turno` può essere 0 oppure 1, ma non entrambi. Quindi, uno dei processi, ad esempio P_j, ha eseguito con successo l'istruzione `while`, mentre P_i è stato costretto a eseguire almeno un'altra istruzione (`turno == j`). Tuttavia, poiché in quell'istante `pronto[j] == true` e `turno == j`, condizione che persiste fino a che P_j si trova nella propria sezione critica, ne consegue che la mutua esclusione è rispettata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per dimostrare le proprietà 2 e 3 si osserva che si può impedire a un processo P_i di entrare nella sezione critica solo se questo è bloccato nel ciclo `while` dalla condizione `pronto[j] == true` e `turno == j`; questo è l'unico ciclo. Se P_j non è pronto per entrare nella sua sezione critica, allora `pronto[j] == false`, e P_i può entrare nella propria sezione critica. Se P_j ha impostato `pronto[j]` al valore `true` e si trova anche in esecuzione nella sua istruzione `while`, allora `turno == i`, oppure `turno == j`. Se `turno == i`, P_i entra nella sua sezione critica; se `turno == j`, è P_j a entrare nella sezione critica. Comunque, una volta che P_j esce dalla propria sezione critica, reimposta `pronto[j]` al valore `false`, permettendo a P_i di entrare nella propria sezione critica. Se P_j imposta `pronto[j]` a `true`, deve impostare anche `turno` a `i`. Quindi, poiché P_i non cambia il valore della variabile `turno` durante l'esecuzione dell'istruzione `while`, P_i entra nella sezione critica (progresso) al massimo dopo un ingresso da parte di P_j (attesa limitata).

7.2.2 Soluzione per più processi

L'Algoritmo 3 risolve il problema della sezione critica per due processi. In questo paragrafo si descrive un algoritmo che ha lo scopo di risolvere il problema della sezione critica per n processi. Tale algoritmo, noto come **algoritmo del fornaio**, è basato su uno schema di servizio comunemente usato nelle panetterie, nelle gelaterie, nelle macellerie, nei pubblici registri automobilistici e in altri luoghi dove si deve evitare la confusione nei turni di servizio. Questo algoritmo fu sviluppato per un ambiente distribuito, anche se in questa sede si trattano gli aspetti che riguardano un ambiente centralizzato.

Al suo ingresso nel negozio, ogni cliente riceve un numero; si serve, progressivamente, il cliente con il numero più basso. Poiché l'algoritmo del fornaio non può assicurare che due processi (clienti) non ricevano lo stesso numero, a parità di numero si serve per primo il processo con il nome minore. Cioè, se P_i e P_j ricevono lo stesso numero e $i < j$, si serve prima P_i. Poiché i nomi dei processi sono unici e totalmente ordinati, l'algoritmo è del tutto deterministico.

Le strutture di dati comuni sono

```
boolean scelta[n];
int numero[n];
```

Queste strutture di dati s'inizializzano rispettivamente a `false` e 0. Per motivi di convenienza si definisce la seguente notazione:

- $(a, b) < (c, d)$ se $a < c$ oppure se $a = c$ e $b < d$;
- $\max(a_0, \ldots, a_{n-1})$ è un numero, k, tale che $k \geq a_i$ per $i = 0, \ldots, n-1$.

La struttura del processo P_i nell'algoritmo del fornaio è illustrata nella Figura 7.5.

Per dimostrare la correttezza dell'algoritmo del fornaio, occorre innanzitutto dimostrare che se P_i si trova nella propria sezione critica e P_k ($k \neq i$) ha già scelto il proprio `numero[k] != 0`, allora `(numero[i], i) < (numero[k], k)`. Nell'Esercizio 7.3 si richiede di dimostrare questa asserzione.


```
do {
    scelta[i] = true;
    numero[i] = max(numero[0], numero[1], ..., numero[n - 1]) + 1;
    scelta[i] = false;
    for (j = 0; j < n; j++) {
        while (scelta[j]);
        while ((numero[j] != 0)&&(numero[j,j] < numero[i,i]));
    }
    sezione critica
    numero[i] = 0;
    sezione non critica
} while (1);
```

Figura 7.5 Struttura del processo P_i nell'algoritmo del fornaio.

Dato tale risultato, è facile dimostrare che la mutua esclusione è soddisfatta. Si consideri ora di avere P_i nella sua sezione critica, mentre P_k tenta di entrare nella propria. Quando il processo P_k esegue la seconda istruzione `while` per `j == i`, trova che

- `numero[i] != 0;`
- `(numero[i], i) < (numero[k], k)`.

Quindi, continua il ciclo nell'istruzione `while` fino a che P_i lascia la propria sezione critica. Per dimostrare che i requisiti di progresso e d'attesa limitata sono rispettati, e che l'algoritmo assicura l'equità, è sufficiente osservare che i processi entrano nelle rispettive sezioni critiche secondo il criterio FCFS.

7.3 Architetture di sincronizzazione

Così come accade col sistema operativo, certe caratteristiche dell'architettura di macchina possono rendere più semplice la programmazione e migliorare l'efficienza del sistema. In questo paragrafo si descrivono alcune semplici istruzioni, disponibili in molte unità d'elaborazione, e si dimostra come si possono impiegare efficacemente per risolvere il problema della sezione critica.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

In un sistema dotato di una singola CPU tale problema si potrebbe risolvere semplicemente se si potessero interdire le interruzioni mentre si modificano le variabili condivise. In questo modo si assicurerebbe un'esecuzione ordinata e senza possibilità di prelazione della corrente sequenza di istruzioni; non si potrebbe eseguire nessun'altra istruzione, quindi non si potrebbe apportare nessuna modifica inaspettata alle variabili condivise.

Sfortunatamente questa soluzione non è sempre praticabile; la disabilitazione delle interruzioni nei sistemi con più unità d'elaborazione può comportare sprechi di tempo dovuti alla necessità di trasmettere la richiesta di disabilitazione delle interruzioni a tutte le unità d'elaborazione. Tale trasmissione ritarda l'accesso a ogni sezione critica determinando una diminuzione dell'efficienza. Si considerino, inoltre, gli effetti su un orologio di sistema aggiornato tramite le interruzioni.

Per questo motivo molte architetture offrono particolari istruzioni che permettono di controllare e modificare il contenuto di una parola di memoria, oppure di scambiare il contenuto di due parole di memoria, in modo atomico — cioè come un'unità non interrompibile. Queste speciali istruzioni si possono usare per risolvere il problema della sezione critica in modo relativamente semplice. Anziché discutere una specifica istruzione di una specifica architettura, è preferibile astrarre i concetti principali che stanno alla base di queste istruzioni. L'istruzione `TestAndSet` si può definire com'è illustrato nella Figura 7.6. Questa istruzione è eseguita in modo atomico, cioè come un'unità non soggetta a interruzioni, quindi se si eseguono contemporaneamente due istruzioni `TestAndSet`, ciascuna in un'unità d'elaborazione diversa, queste vengono eseguite in modo sequenziale in un ordine arbitrario.

Se si dispone dell'istruzione `TestAndSet`, si può realizzare la mutua esclusione dichiarando una variabile booleana globale `blocco`, inizializzata a `false`. La struttura del processo P_i è illustrata nella Figura 7.7.

L'istruzione `Swap`, definita nella Figura 7.8, agisce sul contenuto di due parole di memoria; come l'istruzione `TestAndSet`, è anch'essa eseguita in modo atomico.

Se si dispone dell'istruzione `Swap`, la mutua esclusione si garantisce dichiarando e inizializzando al valore `false` una variabile booleana globale `blocco`. Inoltre, ogni processo possiede anche una variabile booleana locale `chiave`. La struttura del processo P_i è illustrata nella Figura 7.9.

Controlla e modifica il contenuto di una parola di memoria

```
boolean TestAndSet(boolean &obiettivo) {
    boolean valore = obiettivo;
    obiettivo = true;
    return valore;
}
```

• L'intera operazione è eseguita in maniera atomica.

Figura 7.6 Definizione dell'istruzione `TestAndSet`.


```
do {
    while (TestAndSet(blocco));
        sezione critica
    blocco = false;
        sezione non critica
} while (1);
```

• Non modifica l'altra variabile

Figura 7.7 Realizzazione di mutua esclusione con `TestAndSet`.

```
void Swap(boolean &a, boolean &b) {
    boolean temp = a;
    a = b;
    b = temp;
}
```

Scambia il contenuto di 2 parole di memoria

• L'intera operazione è eseguita in maniera atomica

Figura 7.8 Definizione dell'istruzione `Swap`.

```
do {
    chiave = true;
    while (chiave == true)
        Swap(blocco,chiave);
        sezione critica
    blocco = false;
        sezione non critica
} while (1);
```

• Non modifica l'altra variabile

Figura 7.9 Realizzazione di mutua esclusione con `Swap`.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Questi algoritmi non soddisfano il requisito di attesa limitata. La Figura 7.10 illustra un algoritmo che impiega l'istruzione `TestAndSet` e soddisfa tutti i requisiti della sezione critica. Le strutture di dati comuni sono

```
boolean attesa[n];
boolean blocco;
```

e sono inizializzate al valore `false`. Per dimostrare che l'algoritmo soddisfa il requisito di mutua esclusione, si considera che il processo P_i può entrare nella propria sezione critica solo se `attesa[i] == false` oppure `chiave == false`. Il valore di `chiave` può diventare `false` solo se si esegue `TestAndSet`. Il primo processo che esegue `TestAndSet` trova `chiave == false`; tutti gli altri devono attendere. La variabile `attesa[i]` può diventare `false` solo se un altro processo esce dalla propria sezione critica; solo una variabile `attesa[i]` vale `false`, il che consente di rispettare il requisito di mutua esclusione.

Per dimostrare che l'algoritmo soddisfa il requisito di progresso, basta osservare che le argomentazioni fatte per la mutua esclusione valgono anche in questo caso, infatti un processo che esce dalla sezione critica o imposta `blocco` al valore `false` oppure `attesa[j]` al valore `false`; entrambe consentono a un processo che attende di entrare nella propria sezione critica di procedere.

```
do {

    attesa[i] = true;
    chiave = true
    while (attesa[i] && chiave)
        chiave = TestAndSet(blocco);
    attesa[i] = false;

        sezione critica

    j = (i + 1) % n;
    while ((j != i) && !attesa[i])
        j = (j + 1) % n;
    if (j == i)
        blocco = false
    else
        attesa[j] = false;

        sezione non critica

} while (1);
```

Figura 7.10 Mutua esclusione con attesa limitata con `TestAndSet`.

Per dimostrare che l'algoritmo soddisfa il requisito di attesa limitata occorre osservare che un processo, quando lascia la propria sezione critica, scandisce il vettore `attesa` nell'ordinamento ciclico ($i + 1, i + 2, \ldots, n - 1, 0, \ldots, i - 1$) e designa il primo processo in questo ordinamento presente nella sezione d'ingresso (`attesa[j] == true`) come il primo processo che deve entrare nella propria sezione critica. Qualsiasi processo che attende di entrare nella propria sezione critica può farlo entro $n - 1$ turni. Sfortunatamente la progettazione delle istruzioni atomiche `TestAndSet` per sistemi con più unità d'elaborazione non è un compito banale. Quest'argomento è trattato nei testi sulle architetture dei calcolatori.

7.4 Semafori

Le soluzioni al problema della sezione critica, introdotte nel Paragrafo 7.3, non si possono generalizzare facilmente a problemi più complessi. Per superare questa difficoltà si può usare uno strumento di sincronizzazione chiamato semaforo. Un semaforo `S` è una variabile intera cui si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche predefinite: `wait` e `signal`. Queste operazioni erano originariamente chiamate `P` (per `wait`; dall'olandese *proberen*, verificare) e `V` (per `signal`; da *verhogen*, incrementare). La definizione classica di `wait` in pseudocodice è la seguente:

```
wait(S) {
    while(S <= 0)
        ;//non-op;
    S--;
}
```

La definizione classica di `signal` in pseudocodice è la seguente:

```
signal(S) {
    S++;
}
```

Le modifiche al valore del semaforo contenute nelle operazioni `wait` e `signal` si devono eseguire in modo indivisibile: mentre un processo cambia il valore del semaforo, nessun altro processo può contemporaneamente modificare quello stesso valore. Inoltre, nel caso della `wait(S)` si devono eseguire senza interruzione anche la verifica del valore intero di `S` (`S <= 0`) e la sua possibile modifica (`S--`). Nel Paragrafo 7.4.2 si spiega come si possono realizzare queste operazioni.

7.4.1 Uso dei semafori

I semafori si possono usare per risolvere il problema della sezione critica con n processi. Gli n processi condividono un semaforo comune, `mutex` (da `mut`*ual* `ex`*clusion*), inizializzato a 1. Ogni processo P_i è strutturato com'è illustrato nella Figura 7.11.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
do {
    wait(mutex);
        sezione critica
    signal(mutex);
        sezione non critica
} while (1);
```

Figura 7.11 Realizzazione di mutua esclusione con semafori.

I semafori si possono usare anche per risolvere diversi problemi di sincronizzazione. Si considerino, ad esempio, due processi in esecuzione concorrente: P_1 con un'istruzione S_1 e P_2 con un'istruzione S_2, si supponga di voler eseguire S_2 solo dopo che S_1 è terminata. Questo schema si può prontamente realizzare facendo condividere a P_1 e P_2 un semaforo comune, `sincronizzazione`, inizializzato a 0, e inserendo nel processo P_1 le istruzioni

```
S1;
signal(sincronizzazione);
```

e nel processo P_2 le istruzioni

```
wait(sincronizzazione);
S2;
```

Poiché `sincronizzazione` è inizializzato a 0, P_2 esegue S_2 solo dopo che P_1 ha eseguito `signal(sincronizzazione)`, che si trova dopo S_1.

7.4.2 Realizzazione

Il principale svantaggio delle soluzioni del problema della mutua esclusione, illustrate nel Paragrafo 7.2, e della definizione di semaforo, fornita nel Paragrafo 7.4, è che richiedono una condizione di attesa attiva (*busy waiting*). Mentre un processo si trova nella propria sezione critica, qualsiasi altro processo che tenti di entrare nella sezione critica si trova sempre nel ciclo del codice della sezione d'ingresso. Chiaramente questa soluzione costituisce un problema per un sistema con multiprogrammazione, poiché la condizione d'attesa attiva spreca cicli della CPU che un altro processo potrebbe sfruttare in modo produttivo. Questo tipo di semaforo è anche detto *spinlock*, perché gira (*spin*) finché rimane bloccato (*lock*) durante l'attesa. I semafori ad attesa attiva sono utili nei sistemi con più unità d'elaborazione. Il vantaggio di un semaforo ad attesa attiva consiste nel fatto che non si deve compiere alcun cambio di contesto quando un processo attende per un

accesso; occorre ricordare che un cambio di contesto può richiedere molto tempo. Quindi, i semafori ad attesa attiva sono utili quando è probabile che l'attesa sia molto breve.

Per superare la necessità dell'attesa attiva, si possono modificare le definizioni delle operazioni `wait` e `signal`: quando un processo invoca l'operazione `wait` e trova che il valore del semaforo non è positivo, deve attendere, ma anziché restare nell'attesa attiva può *bloccare* se stesso. L'operazione di bloccaggio pone il processo in una coda d'attesa associata al semaforo e cambia lo stato del processo nello stato d'attesa. Quindi, si trasferisce il controllo allo scheduler della CPU che sceglie un altro processo pronto per l'esecuzione.

Un processo bloccato, che attende a un semaforo `S`, sarà riavviato in seguito all'esecuzione di un'operazione `signal` su `S` da parte di qualche altro processo. Il processo si riavvia tramite un'operazione wakeup, che modifica lo stato del processo da attesa a pronto, il processo entra nella coda dei processi pronti e, secondo il criterio di scheduling, l'uso della CPU può essere o non essere commutato dal processo in esecuzione al processo appena divenuto pronto.

Per realizzare i semafori secondo quel che s'è detto si può definire il semaforo come una struttura del Linguaggio C:

```
typedef struct {
    int valore;
    struct processo *L;
} semaforo;
```

A ogni semaforo sono associati un valore intero e una lista di processi, che contiene i processi che attendono a un semaforo; l'operazione `signal` preleva un processo da tale lista e lo attiva.

L'operazione `wait` del semaforo si può definire come segue:

```
void wait(semaforo S) {
    S.valore--;
    if (S.valore < 0) {
        aggiungi questo processo a S.L;
        block();
    }
}
```

l'operazione `signal` del semaforo si può definire come segue:

```
void signal(semaforo S) {
    S.valore++;
    if (S.valore <= 0) {
        togli un processo P da S.L;
        wakeup(P);
    }
}
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'operazione `block` sospende il processo che la invoca; l'operazione `wakeup(P)` pone in stato di pronto per l'esecuzione un processo P bloccato. Queste due operazioni sono fornite dal sistema operativo come chiamate del sistema di base.

Occorre notare che, mentre la definizione classica di semaforo ad attesa attiva è tale che il valore del semaforo non è mai negativo, questa definizione può condurre a valori negativi. Se il valore del semaforo è negativo, la sua dimensione è data dal numero dei processi che attendono a quel semaforo. Ciò avviene a causa dell'inversione dell'ordine del decremento e della verifica nel codice dell'operazione `wait`. La lista dei processi che attendono a un semaforo si può facilmente realizzare inserendo un campo puntatore in ciascun descrittore di processo (PCB). Ogni semaforo contiene un valore intero e un puntatore a una lista di PCB. Per aggiungere e togliere processi dalla lista assicurando un'attesa limitata si può usare una coda FIFO, della quale il semaforo contiene i puntatori al primo e all'ultimo elemento. In generale si può usare *qualsiasi* criterio d'accodamento; il corretto uso dei semafori non dipende dal particolare criterio impiegato.

Poiché occorre garantire che due processi non possano eseguire contemporaneamente operazioni `wait` e `signal` sullo stesso semaforo, l'esecuzione dei semafori deve essere di tipo atomico. Si tratta di un problema di sezione critica che si può risolvere in due modi. In un ambiente con unità d'elaborazione singola, vale a dire con una CPU, è sufficiente inibire le interruzioni mentre si eseguono le operazioni `wait` e `signal`, sicché le istruzioni di processi diversi non si possono intercalare e, poiché lo scheduler non può riacquisire il controllo, si possono eseguire soltanto le istruzioni del processo corrente.

In un ambiente con più unità d'elaborazione, invece, l'inibizione delle interruzioni non funziona: le istruzioni di processi diversi, in esecuzione su unità d'elaborazione diverse, possono essere intercalate in modo arbitrario. Se la macchina non dispone di istruzioni speciali, si può usare una qualsiasi delle soluzioni del problema della sezione critica (descritte nel Paragrafo 7.2), trattando come sezioni critiche le parti di codice che costituiscono le operazioni `wait` e `signal`.

È importante rilevare che questa definizione delle operazioni `wait` e `signal` non consente di eliminare completamente l'attesa attiva, ma piuttosto di rimuoverla dalle sezioni d'ingresso dei programmi d'applicazione. Inoltre, l'attesa attiva si limita alle sezioni critiche delle operazioni `wait` e `signal`, che sono abbastanza brevi; se sono convenientemente codificate, non sono più lunghe di 10 istruzioni. Quindi, la sezione critica non è quasi mai occupata e l'attesa attiva si presenta raramente e per breve tempo. Una situazione completamente diversa si verifica con i programmi d'applicazione le cui sezioni critiche possono essere lunghe minuti o anche ore, oppure occupate spesso. In questi casi l'attesa attiva è assai inefficiente.

7.4.3 Stallo e attesa indefinita

La realizzazione di un semaforo con coda d'attesa può condurre a situazioni in cui ciascun processo di un insieme di processi attende indefinitamente un evento — l'esecuzione di un'operazione `signal` — che può essere causato solo da uno dei processi dello stesso insieme. Quando si verifica una situazione di questo tipo si dice che i processi sono in **stallo** (*deadlock*).

Per illustrare questo fenomeno si consideri un insieme di due processi, P_0 e P_1, ciascuno dei quali ha accesso a due semafori, `S` e `Q`, impostati al valore 1:

```
   P0                P1
wait(S);          wait(Q);
wait(Q);          wait(S);
   .                 .
   .                 .
   .                 .
signal(S);        signal(Q);
signal(Q);        signal(S);
```

Si supponga che P_0 esegua `wait(S)` e quindi P_1 esegua `wait(Q)`; eseguita `wait(Q)`, P_0 deve attendere che P_1 esegua `signal(Q)`; analogamente, quando P_1 esegue `wait(S)`, deve attendere che P_0 esegua `signal(S)`. Poiché queste operazioni `signal` non si possono eseguire, P_0 e P_1 sono in stallo.

Un insieme di processi è in stallo se ciascun processo dell'insieme attende un evento che può essere causato solo da un altro processo dell'insieme. In questo contesto si considerano principalmente gli *eventi di acquisizione e rilascio di risorse*, tuttavia anche altri tipi di eventi possono produrre situazioni di stallo (si veda il Capitolo 8, che descrive anche i meccanismi che servono ad affrontare questo tipo di problema).

Un altro problema connesso alle situazioni di stallo è quello dell'**attesa indefinita** (nota anche col termine *starvation*, letteralmente, *inedia*). Con tale termine si definisce una situazione d'attesa indefinita nella coda di un semaforo, che si può ad esempio presentare se i processi si aggiungono e si rimuovono dalla lista associata a un semaforo secondo un criterio LIFO.

7.4.4 Semafori binari

Il costrutto semaforo descritto nei paragrafi precedenti è comunemente noto come **semaforo contatore**, poiché il suo valore intero può variare in un dominio logicamente non limitato. Un **semaforo binario** è un semaforo il cui valore intero può essere soltanto 0 o 1. Secondo l'architettura sottostante, un semaforo binario può essere più semplice da realizzare di un semaforo contatore. In questo paragrafo si dimostra come sia possibile realizzare un semaforo contatore adoperando i semafori binari.

Sia `S` un semaforo contatore, la sua realizzazione in termini di semafori binari richiede le seguenti strutture di dati:

```
semaforo_binario S1, S2;
int C;
```

Inizialmente `S1 = 1` e `S2 = 0`; l'intero `C` s'imposta al valore iniziale del semaforo contatore `S`.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'operazione wait, per il semaforo contatore S, si può realizzare come segue:

```
wait(S1);
C--;
if (C < 0) {
    signal(S1);
    wait(S2);
}
signal(S1);
```

L'operazione signal, per il semaforo contatore S, si può realizzare come segue:

```
wait(S1);
C++;
if (C <= 0)
    signal(S2);
else
    signal(S1);
```

7.5 Problemi tipici di sincronizzazione

In questo paragrafo s'illustrano diversi problemi di sincronizzazione come esempi di una vasta classe di problemi connessi al controllo della concorrenza. Questi problemi sono utili per verificare quasi tutte le nuove proposte di schemi di sincronizzazione. Nelle soluzioni s'impiegano i semafori.

7.5.1 Produttori e consumatori con memoria limitata

Il problema dei *produttori e consumatori con memoria limitata*, trattato anche nel Paragrafo 7.1, si usa generalmente per illustrare la potenza delle primitive di sincronizzazione. In questa sede si presenta uno schema generale di soluzione, senza far riferimento a nessuna realizzazione particolare. Si supponga di disporre di una certa quantità di memoria rappresentata da un vettore con *n* posizioni, ciascuna capace di contenere un elemento. Il semaforo mutex garantisce la mutua esclusione degli accessi al vettore ed è inizializzato al valore 1. I semafori vuote e piene conteggiano rispettivamente il numero di posizioni vuote e il numero di posizioni piene nel vettore. Il semaforo vuote si inizializza al valore n; il semaforo piene si inizializza al valore 0.

La Figura 7.12 riporta la struttura generale del processo produttore, la Figura 7.13 riporta la struttura generale del processo consumatore. È interessante notare la simmetria esistente tra il produttore e il consumatore. Il codice si può interpretare nel senso di produzione, da parte del produttore, di posizioni piene per il consumatore; oppure di produzione, da parte del consumatore, di posizioni vuote per il produttore.


```
do {
    ...
    produce un elemento in appena_prodotto
    ...
    wait(vuote);
    wait(mutex);
    ...
    inserisci in vettore l'elemento in appena_prodotto
    ...
    signal(mutex);
    signal(piene);
} while (1);
```

Figura 7.12 Struttura generale del processo produttore.

```
do {
    wait(piene);
    wait(mutex);
    ...
    rimuovi un elemento da vettore e mettilo in da_consumare
    ...
    signal(mutex);
    signal(vuote);
    ...
    consuma l'elemento contenuto in da_consumare
    ...
} while (1);
```

Figura 7.13 Struttura generale del processo consumatore.

7.5.2 Problema dei lettori e degli scrittori

Si consideri un insieme di dati, ad esempio un file, che si deve condividere tra numerosi processi concorrenti. Alcuni processi possono richiedere solo la lettura del contenuto dell'oggetto condiviso, mentre altri possono richiedere un aggiornamento, vale a dire una lettura e una scrittura, dello stesso oggetto. Questi due processi sono distinti, e si indicano chiamando lettori quelli interessati alla sola lettura e scrittori gli altri. Naturalmente, se due lettori accedono nello stesso momento all'insieme di dati condiviso, non si ha alcun effetto negativo; viceversa, se uno scrittore e un altro processo (lettore o scrittore) accedono contemporaneamente allo stesso insieme di dati, si possono produrre incoerenze.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per impedire l'insorgere di difficoltà di questo tipo è necessario che gli scrittori abbiano un accesso esclusivo all'insieme di dati condiviso. Questo problema di sincronizzazione si chiama **problema dei lettori e degli scrittori**. Da quando tale problema fu enunciato, è stato usato per verificare quasi tutte le nuove primitive di sincronizzazione. Il problema dei lettori e degli scrittori ha diverse varianti, che implicano tutte l'esistenza di priorità; la più semplice, cui si fa riferimento come al *primo* problema dei lettori e degli scrittori, richiede che nessun lettore attenda, a meno che uno scrittore abbia già ottenuto il permesso di usare l'insieme di dati condiviso. In altre parole, nessun lettore deve attendere che altri lettori terminino l'operazione solo perché uno scrittore attende l'accesso ai dati. Il *secondo* problema dei lettori e degli scrittori si fonda sul presupposto che uno scrittore, una volta pronto, esegua il proprio compito di scrittura al più presto. In altre parole, se uno scrittore attende per l'accesso all'insieme di dati, nessun nuovo lettore deve iniziare la lettura.

STARVATION

La soluzione del primo problema e la soluzione del secondo possono condurre a uno stato d'attesa indefinita: degli scrittori, nel primo caso; dei lettori, nel secondo. Per questo motivo sono state proposte altre varianti. In questo paragrafo si presenta una soluzione del primo problema dei lettori e degli scrittori; indicazioni attinenti a soluzioni immuni all'attesa indefinita si trovano nelle Note bibliografiche.

La soluzione del primo problema dei lettori e degli scrittori prevede dunque la condivisione da parte dei processi lettori delle seguenti strutture di dati:

```
semaforo mutex, scrittura;
int numlettori;
```

I semafori `mutex` e `scrittura` sono inizializzati a 1; `numlettori` è inizializzato a 0. Il semaforo `scrittura` è comune a entrambi i tipi di processi (lettura e scrittura). Il semaforo `mutex` si usa per assicurare la mutua esclusione al momento dell'aggiornamento di `numlettori`. La variabile `numlettori` contiene il numero dei processi che stanno attualmente leggendo l'insieme di dati. Il semaforo `scrittura` funziona come semaforo di mutua esclusione per gli scrittori e serve anche al primo o all'ultimo lettore che entra o esce dalla sezione critica. Non serve, invece, ai lettori che entrano o escono mentre altri lettori si trovano nelle rispettive sezioni critiche.

La Figura 7.14 illustra la struttura generale di un processo scrittore; la Figura 7.15 illustra la struttura generale di un processo lettore. Occorre notare che se uno scrittore si

semaforo che nel caso degli scrittori funziona come semaforo x la mutua esclusione

```
wait(scrittura);
    ...
    esegui l'operazione di scrittura
    ...
signal(scrittura);
```

Figura 7.14 Struttura generale di un processo scrittore.

semaforo x la mutua esclusione x i lettori

```
wait(mutex);
numlettori++;
if (numlettori == 1)
    wait(scrittura);
signal(mutex);
    ...
    esegui l'operazione di lettura
    ...
wait(mutex);
numlettori--;
if (numlettori == 0)
    signal(scrittura);
signal(mutex);
```

se semaforo scrittura chiamato solo dal primo lettore che entra e dall'ultimo che esce dalla sezione critica.

Figura 7.15 Struttura generale di un processo lettore.

trova nella sezione critica e n lettori attendono di entrarvi, si accoda un lettore a `scrittura` e $n - 1$ lettori a `mutex`. Inoltre, se uno scrittore esegue `signal(scrittura)` si può riprendere l'esecuzione o dei lettori in attesa oppure di un singolo scrittore in attesa. La scelta è fatta dallo scheduler.

7.5.3 Problema dei cinque filosofi

Si considerino cinque filosofi che passano la vita pensando e mangiando. I filosofi condividono un tavolo rotondo circondato da cinque sedie, una per ciascun filosofo. Al centro del tavolo si trova una zuppiera colma di riso, e la tavola è apparecchiata con cinque bacchette (Figura 7.16). Quando un filosofo pensa, non interagisce con i colleghi; quando gli viene fame, tenta di prendere le bacchette più vicine: quelle che si trovano tra lui e i commensali alla sua destra e alla sua sinistra. Un filosofo può prendere una bacchetta alla volta e non può prendere una bacchetta che si trova già nelle mani di un suo vicino. Quando un filosofo affamato tiene in mano due bacchette contemporaneamente, mangia senza lasciare le bacchette. Terminato il pasto, le posa e riprende a pensare.

Il problema dei cinque filosofi è considerato un classico problema di sincronizzazione, non certo per la sua importanza pratica, e neanche per antipatia verso i filosofi da parte degli informatici, ma perché rappresenta una vasta classe di problemi di controllo della concorrenza, in particolare i problemi caratterizzati dalla necessità di assegnare varie risorse a diversi processi evitando situazioni di stallo e d'attesa indefinita.

Una semplice soluzione consiste nel rappresentare ogni bacchetta con un semaforo: un filosofo tenta di afferrare ciascuna bacchetta eseguendo un'operazione `wait` su quel semaforo e la posa eseguendo operazioni `signal` sui semafori appropriati. Quindi, i dati condivisi sono

```
semaforo bacchetta[5];
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

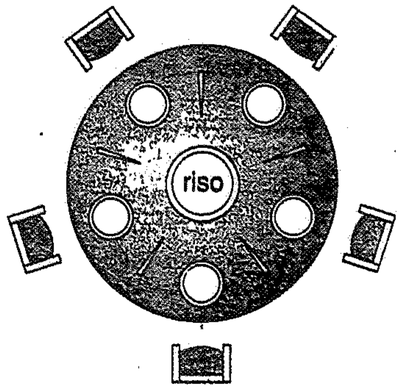


Figura 7.16 Situazione dei cinque filosofi.

```
do {
    wait(bacchetta[i]);
    wait(bacchetta[(i + 1) % 5]);
        ...
        mangia
        ...
    signal(bacchetta[i]);
    signal(bacchetta[(i + 1) % 5]);
        ...
        pensa
        ...
} while (1);
```

Figura 7.17 Struttura del filosofo *i*.

dove tutti gli elementi `bacchetta` sono inizializzati a `1`. La struttura del filosofo `i` è illustrata nella Figura 7.17.

Questa soluzione garantisce che due vicini non mangino contemporaneamente, ma è insufficiente poiché non esclude la possibilità che si abbia una situazione di stallo. Si supponga che tutti e cinque i filosofi abbiano fame contemporaneamente e che ciascuno tenti di afferrare la bacchetta di sinistra; tutti gli elementi di `bacchetta` diventano uguali a zero, perciò ogni filosofo che tenta di afferrare la bacchetta di destra entra in stallo.

Nel Paragrafo 7.7 si descrive una soluzione del problema dei cinque filosofi che impedisce il verificarsi di situazioni di stallo. Di seguito sono elencate diverse possibili soluzioni per tali situazioni di stallo:

- solo quattro filosofi possono stare contemporaneamente a tavola;
- un filosofo può prendere le sue bacchette solo se sono entrambe disponibili (occorre notare che quest'operazione si deve eseguire in una sezione critica);
- si adotta una soluzione asimmetrica: un filosofo dispari prende prima la bacchetta di sinistra e poi quella di destra, invece un filosofo pari prende prima la bacchetta di destra e poi quella di sinistra.

Inoltre, una soluzione soddisfacente per il problema dei cinque filosofi deve escludere la possibilità di situazioni d'attesa indefinita, in altre parole che uno dei filosofi muoia di fame (da qui il termine *starvation*) — una soluzione immune alle situazioni di stallo non esclude necessariamente la possibilità di situazioni d'attesa indefinita.

7.6 Regioni critiche

Sebbene i semafori siano un meccanismo utile ed efficace per la sincronizzazione dei processi, se si usano in modo scorretto possono produrre errori di sincronizzazione. Tali errori sono difficilmente individuabili, poiché si presentano solo con particolari sequenze d'esecuzione.

Un esempio riguardante questo tipo d'errori s'è visto nel Paragrafo 7.1 nell'uso dei contatori nella soluzione del problema dei produttori e consumatori: il problema si presentava raramente e persino in tali occasioni il valore del contatore sembrava ragionevole (diverso solo di 1 dal valore esatto), ciononostante la soluzione non era accettabile, e ha condotto all'introduzione dei semafori.

Sfortunatamente gli errori di sincronizzazione si possono verificare anche se s'impiegano i semafori. Per illustrare come ciò avvenga si può riscrivere la soluzione del problema della sezione critica usando i semafori: tutti i processi condividono una variabile semaforo `mutex`, inizializzata a `1`; ogni processo, prima di entrare nella sezione critica, esegue `wait(mutex)`, e all'uscita esegue `signal(mutex)`. Se non si rispetta questa sequenza, i due processi si possono trovare contemporaneamente nelle rispettive sezioni critiche.

Occorre esaminare le difficoltà che possono sorgere in questi casi, osservando che dipendono da una codifica scorretta dei *singoli* processi. Questa situazione può essere causata da un semplice errore di programmazione o da programmatori che non collaborano abbastanza.

- Si supponga che un processo scambi l'ordine d'esecuzione delle operazioni `wait` e `signal` sul semaforo `mutex`; avverrebbe l'esecuzione seguente:

http://www.adultpdf.com
Created by TIFF Combine - (trial version, http://www.tiffcombine.com) This mark, please register this software.


```
signal(mutex);
    ...
    sezione critica
    ...
wait(mutex);
```

che potrebbe causare l'esecuzione contemporanea di più processi nelle rispettive sezioni critiche, violando così il requisito di mutua esclusione. Un simile errore si può individuare solo se più processi sono contemporaneamente attivi nelle rispettive sezioni critiche. Occorre notare che questa situazione potrebbe non essere riproducibile.

- Si supponga che un processo sostituisca `signal(mutex)` con `wait(mutex)`, avverrebbe l'esecuzione seguente:

```
wait(mutex);
    ...
    sezione critica
    ...
wait(mutex);
```

e si avrebbe uno stallo.

- Si supponga che un processo ometta `wait(mutex)` oppure `signal(mutex)` oppure entrambe. In questo caso o si avrebbe una violazione della mutua esclusione oppure uno stallo.

Questi esempi evidenziano come un errato uso dei semafori possa condurre al verificarsi di errori. Problemi analoghi si possono presentare anche negli altri schemi di sincronizzazione, trattati nel Paragrafo 7.5.

Per evitare l'insieme degli errori appena presentati, i ricercatori hanno introdotto diversi costrutti di linguaggio. In questo paragrafo si descrive un fondamentale costrutto di sincronizzazione di alto livello, la **regione critica**, nota anche come **regione critica condizionale**. Un altro costrutto di sincronizzazione fondamentale, il **monitor**, è trattato nel Paragrafo 7.7. Nella presentazione di questi costrutti si presuppone che un processo sia composto da alcuni dati locali e da un programma sequenziale che può operare su di essi. Ai dati locali può accedere solo il programma sequenziale, che è incapsulato all'interno del processo stesso. Ciò significa che i processi non possono accedere direttamente ai dati locali di altri processi, pur potendo condividere i dati globali.

Il costrutto di sincronizzazione ad alto livello, detto regione critica, richiede che una variabile v di tipo T, che deve essere condivisa tra molti processi, sia dichiarata come segue:

```
v: shared T;
```

Dichiarazione di variabile condivisa

Alla variabile v si può accedere solo dall'interno di un'istruzione `region` della forma

```
region v when (B) do S;
```


significa che, mentre si esegue l'istruzione S, nessun altro processo può accedere alla variabile v. L'espressione B è un'espressione booleana che controlla l'accesso alla regione critica.

Quando un processo vuole accedere alla variabile condivisa v nella regione critica, si valuta l'espressione booleana B; se risulta vera, si esegue l'istruzione S; altrimenti il processo rilascia la mutua esclusione ed è ritardato fino a che B diventa vera e nessun altro processo si trova nella regione associata a v. Quindi, se si eseguono in modo concorrente le due istruzioni

```
region v when (true) do S1;
region v when (true) do S2;
```

in processi sequenziali distinti, il risultato corrisponde all'esecuzione sequenziale di S1 seguita da S2, oppure di S2 seguita da S1.

Il costrutto della regione critica consente di evitare gli errori di programmazione connessi all'uso di semafori nella soluzione del problema della sezione critica. Occorre notare che il costrutto non elimina necessariamente tutti gli errori di sincronizzazione, ma ne riduce il numero. Se si verifica un errore nella logica del programma, può essere difficile riprodurre una particolare sequenza di eventi.

xché dovrebbero sì pure all'uso del costrutto della regione critica

Il costrutto della regione critica si può usare efficacemente per risolvere vari tipi di problemi di sincronizzazione; ad esempio il problema dei produttori e consumatori con memoria limitata. Il vettore di memoria e i suoi puntatori sono incapsulati nella seguente struttura di dati:

```
struct vettore {
    elemento gruppo[n];
    int contatore, inserisci, preleva;
};
```

Il processo produttore inserisce un nuovo elemento appena_prodotto nel vettore condiviso eseguendo il seguente codice:

```
region vettore when (contatore < n) {
    gruppo[inserisci] = appena_prodotto;
    inserisci = (inserisci + 1) % n;
    contatore++;
};
```

Il processo consumatore rimuove un elemento dal vettore condiviso e lo inserisce in da_consumare eseguendo il seguente codice:

```
region vettore when (contatore > 0) {
    da_consumare = gruppo[preleva];
    preleva = (preleva + 1) % n;
    contatore--;
};
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

A questo punto è possibile spiegare in che modo un compilatore può tradurre la regione critica. A ogni variabile condivisa si associano le seguenti variabili:

```
semaforo mutex, primo_ritardo, secondo_ritardo;
int primo_contatore, secondo_contatore;
```

Il semaforo `mutex` si inizializza a `1`, i semafori `primo_ritardo` e `secondo_ritardo` si inizializzano a `0`; le variabili intere `primo_contatore` e `secondo_contatore` si inizializzano a `0`.

L'accesso mutuamente esclusivo alla sezione critica è garantito da `mutex`. Se un processo non può entrare nella sezione critica perché la condizione booleana `B` è falsa, il processo attende inizialmente al semaforo `primo_ritardo`. Prima di poter rivalutare la propria condizione booleana `B`, un processo che attende al semaforo `primo_ritardo` viene spostato al semaforo `secondo_ritardo`. Con `primo_contatore` e `secondo_contatore` si conteggiano rispettivamente il numero dei processi che attendono a `primo_ritardo` e a `secondo_ritardo`.

Un processo che lascia la sua sezione critica può aver modificato il valore di una condizione booleana `B` che impediva a un altro processo di entrare nella propria sezione critica. Di conseguenza, quando un processo esce dalla sezione critica occorre scandire le code dei processi che attendono a `primo_ritardo` e `secondo_ritardo` (in quest'ordine), permettendo a ogni processo di verificare la propria condizione booleana. Se per un certo processo la sua condizione booleana vale `true`, tale processo può entrare nella propria sezione critica; altrimenti deve continuare ad attendere ai semafori `primo_ritardo` e `secondo_ritardo`, come si è descritto precedentemente. Quindi per una variabile condivisa `x`, l'istruzione

```
region x when (B) do S;
```

si può realizzare com'è illustrato nella Figura 7.18. Occorre notare che, con questa soluzione, ogni volta che un processo abbandona la sezione critica si deve rivalutare l'espressione `B` di ogni processo in attesa. Se più processi attendono che le loro rispettive espressioni booleane assumano il valore `true`, il carico richiesto dalla rivalutazione può essere una causa d'inefficienza. Per ridurre tale carico si possono usare diversi metodi di ottimizzazione (si vedano le Note bibliografiche).

7.7 Monitor

Un altro costrutto di sincronizzazione di alto livello è il tipo monitor. Un monitor è caratterizzato da un insieme di operatori definiti dal programmatore. La rappresentazione di un tipo monitor è formata di dichiarazioni di variabili i cui valori definiscono lo stato di un'istanza del tipo, e di corpi delle procedure o funzioni che realizzano le operazioni sul tipo stesso. La Figura 7.19 illustra la sintassi di un monitor.

La rappresentazione di un tipo monitor non può essere usata direttamente dai diversi processi. Quindi, una procedura definita all'interno di un monitor può accedere so-


```
wait(mutex);
while (!B) {
    primo_contatore++;
    if (secondo_contatore > 0)
      signal(secondo_ritardo);
    else
      signal(mutex);
    wait(primo_ritardo);
    primo_contatore--;
    secondo_contatore++;
    if (primo_contatore > 0)
      signal(primo_ritardo);
    else
      signal(secondo_ritardo);
    wait(secondo_ritardo);
    secondo_contatore--;
}
S;
if (primo_contatore > 0)
    signal(primo_ritardo);
else if (secondo_contatore > 0)
    signal(secondo_ritardo);
else
    signal(mutex);
```

Figura 7.18 Realizzazione del costrutto di regione critica.

lo ai suoi parametri formali e alle variabili dichiarate come locali all'interno dello stesso monitor. Analogamente, alle variabili locali di un monitor possono accedere soltanto le procedure locali.

Il costrutto monitor assicura che all'interno di un monitor può essere attivo un solo processo alla volta, sicché non si deve codificare esplicitamente il vincolo di mutua esclusione (Figura 7.20). Tale definizione di monitor non è abbastanza potente per esprimere alcuni schemi di sincronizzazione, sono perciò necessari ulteriori meccanismi che, in questo caso, sono forniti dal costrutto `condition`. Un programmatore che deve scrivere un proprio schema di sincronizzazione può definire una o più variabili condizionali:

```
condition x, y;
```

Le uniche operazioni eseguibili su una variabile `condition` sono `wait` e `signal`. L'operazione

```
x.wait();
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
monitor nome_monitor
{
    dichiarazioni di variabili condivise

    procedure body P1 (...) {
        ...
    }
    procedure body P2 (...) {
        ...
    }
        .
        .
        .
    procedure body Pn (...) {
        ...
    }
    {
        codice d'inizializzazione
    }
}
```

Figura 7.19 Sintassi di un monitor.

implica che il processo che la invoca rimanga sospeso fino a che un altro processo non invoca l'operazione

```
x.signal();
```

che risveglia esattamente un processo sospeso. Se non esistono processi sospesi l'operazione `signal` non ha alcun effetto, vale a dire che lo stato di `x` resta immutato, come se l'operazione non fosse stata eseguita; la situazione è descritta nella Figura 7.21.

Tutto ciò contrasta con l'operazione `signal` associata ai semafori, poiché questa influisce sempre sullo stato del semaforo.

Si supponga, ad esempio, che quando un processo *P* invoca l'operazione `x.signal`, esista un processo sospeso *Q* associato alla variabile `x` di tipo `condition`. Chiaramente, se al processo sospeso *Q* si permette di riprendere l'esecuzione, il processo segnalante *P* è costretto ad attendere, altrimenti *P* e *Q* sarebbero contemporaneamente attivi all'interno del monitor. Occorre in ogni modo notare che, concettualmente, entrambi i processi possono continuare l'esecuzione. Sussistono quindi due possibilità:

1. *P* attende che *Q* lasci il monitor o attenda a un'altra variabile `condition`;
2. *Q* attende che *P* lasci il monitor o attenda a un'altra variabile `condition`.

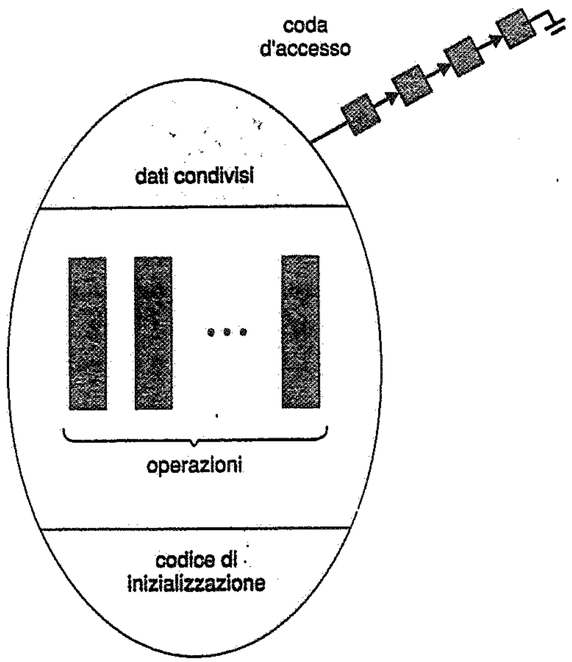


Figura 7.20 Schema di un monitor.

Siccome *P* era già in esecuzione nel monitor, sembrerebbe più ragionevole la seconda soluzione; tuttavia, consentendo al processo *P* di continuare, la condizione logica attesa da *Q* può non valere più nell'istante in cui si riprende *Q*.

La prima soluzione fu sostenuta da C. A. R. Hoare, soprattutto perché l'argomento portato in precedenza a suo favore si traduce direttamente in regole di dimostrazione più semplici e più eleganti.

Nel linguaggio Concurrent C si è adottato un compromesso tra le due soluzioni; quando il processo *P* esegue l'operazione `signal`, si riprende immediatamente il processo *Q*. Questo modello è meno potente di quello di Hoare, poiché durante una singola chiamata di procedura di monitor, un processo non può effettuare una `signal` più di una volta.

Prob. dei 5 filosofi senza stallo

Nel seguito s'illustrano questi concetti presentando una soluzione priva di situazioni di stallo per il problema dei cinque filosofi. Occorre ricordare che un filosofo può prendere le proprie bacchette solo se sono entrambe disponibili. Per codificare questa soluzione si devono distinguere i tre diversi stati in cui può trovarsi un filosofo. A tale scopo si introduce la seguente struttura di dati:

```
enum {pensa, affamato, mangia} stato[5];
```

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

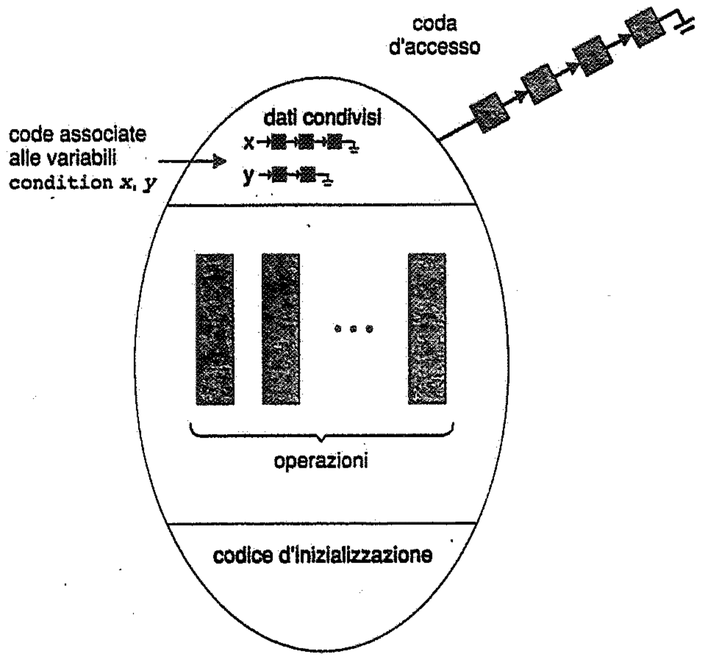


Figura 7.21 Monitor con variabili condition.

Il filosofo i può impostare la variabile `stato[i] = mangia` solo se i suoi vicini non stanno mangiando: `((stato[(i + 4) % 5] != mangia) && (stato[(i + 1) % 5] != mangia))`.

Inoltre, occorre impiegare la seguente struttura di dati:

```
condition auto[5];
```

dove il filosofo i può ritardare se stesso quando ha fame ma non riesce a ottenere le bacchette di cui ha bisogno. A questo punto si può descrivere la soluzione. La distribuzione delle bacchette è controllata dal monitor fc (Figura 7.22). Ciascun filosofo, prima di cominciare a mangiare, deve invocare l'operazione prende; ciò può determinare la sospensione del processo filosofo. Completata con successo l'operazione, il filosofo può mangiare; in seguito, il filosofo invoca l'operazione posa e comincia a pensare. Il filosofo i deve quindi chiamare le operazioni prende e posa nella seguente sequenza:

```
fc.prende(i);
...
mangia
...
fc.posa(i);
```



```
monitor fc
{
    enum {pensa, affamato, mangia} stato[5];
    condition auto[5];

    void prende(int i) {
       stato[i] = affamato;
       verifica(i);
       if (stato[i] != mangia)
          auto[i].wait();
    }

    void posa(int i) {
       stato[i] = pensa;
       verifica((i + 4) % 5);
       verifica((i + 1) % 5);
    }

    void verifica(int i) {
       if ((stato[(i + 4) % 5] =! mangia) &&
          (stato[i] == affamato) &&
          (stato[(i + 1) % 5] != mangia)) {
             stato[i] = mangia;
             auto[i].signal();
       }
    }

    void inizializzazione() {
       for (int i = 0; i < 5; i++)
          stato[i] = pensa;
    }
}
```

Può determinare la sospensione del processo filosofo se ha fame ma non ha le bacchette.

Figura 7.22 Una soluzione con monitor al problema dei cinque filosofi.

È facile dimostrare che questa soluzione assicura che due vicini non mangino contemporaneamente e che non si verifichino situazioni di stallo. Occorre però notare che un filosofo può attendere indefinitamente. La soluzione di questo problema è lasciata come esercizio per il lettore. A questo punto si considera la possibilità di realizzare il meccanismo del monitor usando i semafori. A ogni monitor si associa un semaforo `mutex`, inizializzato a 1; un processo deve eseguire `wait(mutex)` prima di entrare nel monitor, e `signal(mutex)` dopo aver lasciato il monitor.

Poiché un processo che esegue una `signal` deve attendere fino a che il processo risvegliato si mette in attesa o lascia il monitor, si introduce un altro semaforo, `prossimo`, inizializzato a 0, al quale i processi che eseguono una `signal` possono autosospendersi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Cheeco ... Ti TANCHI
02/1/03

Per contare i processi sospesi al semaforo `prossimo`, si usa una variabile intera `prossimo_contatore`. Quindi, ogni procedura esterna di monitor `F` si sostituisce col seguente codice:

```
wait(mutex);
    ...
    corpo di F
    ...
if (prossimo_contatore > 0)
    signal(prossimo);
else
    signal(mutex);
```

In questo modo si assicura la mutua esclusione all'interno del monitor.

A questo punto si può descrivere la realizzazione delle variabili `condition`. Per ogni variabile `x` di tipo `condition` si introducono un semaforo `x_sem` e una variabile intera `x_contatore`, entrambi inizializzati a `0`. L'operazione `x.wait` si può realizzare come segue:

```
x_contatore++;
if (prossimo_contatore > 0)
    signal(prossimo);
else
    signal(mutex);
wait(x_sem);
x_contatore--;
```

L'operazione `x.signal` si può realizzare come segue:

```
if (x_contatore > 0) {
    prossimo_contatore++;
    signal(x_sem);
    wait(prossimo);
    prossimo_contatore--;
}
```

Questa soluzione è applicabile alle definizioni di monitor date da C. A. R. Hoare e da P. Brinch-Hansen. In alcuni casi, tuttavia, questo livello di generalità della codifica non è necessario, e si possono apportare notevoli miglioramenti all'efficienza. La soluzione a questo problema è lasciata al lettore nell'Esercizio 7.13.

A questo punto si discute il problema dell'ordine di ripresa dei processi all'interno di un monitor. Se più processi sono sospesi alla condizione `x`, e se qualche processo esegue l'operazione `x.signal`, è necessario stabilire quale tra i processi sospesi si debba riattivare per primo. Una semplice soluzione consiste nell'usare un ordinamento FCFS, secondo cui il processo che attende da più tempo viene ripreso per primo. Tuttavia, in

molti casi uno schema di scheduling di questo tipo non risulta adeguato; in questi casi si può usare un costrutto di attesa condizionale della forma

```
x.wait(c);
```

dove con `c` si indica un'espressione intera che si valuta al momento dell'esecuzione dell'operazione `wait`. Il valore di `c`, si chiama **numero di priorità**, e si memorizza col nome del processo sospeso. Quando si esegue `x.signal`, si riprende il processo cui è associato il numero di priorità più basso.

Per comprendere questo nuovo meccanismo, si consideri il monitor illustrato nella Figura 7.23; tale monitor ha il compito di assegnare una particolare risorsa a processi in competizione. Quando richiede l'assegnazione di una delle sue risorse, ogni processo specifica il tempo massimo per il quale prevede di usare la risorsa. Il monitor assegna la risorsa al processo con la richiesta di assegnazione più breve.

Per accedere alla risorsa in questione il processo deve rispettare la sequenza

```
R.acquisizione(t);
    ...
    accesso alla risorsa
    ...
R.rilascio();
```

dove `R` è un'istanza di tipo `assegnazione_risorse`.

```
monitor assegnazione_risorse
{
    boolean occupato;
    condition x;

    void acquisizione(int tempo) {
      if (occupato)
        x.wait(tempo);
      occupato = true;
    }

    void rilascio() {
      occupato = false;
      x.signal();
    }

    void inizializzazione() {
      occupato = false;
    }
}
```

Figura 7.23 Un monitor per l'assegnazione di una singola risorsa.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Sfortunatamente il concetto di monitor non può garantire che la precedente sequenza d'accesso sia rispettata. In particolare, può accadere quel che segue:

- un processo può accedere alla risorsa senza prima ottenere il permesso d'accesso;
- una volta che ne ha ottenuto l'accesso, un processo può non rilasciare più la risorsa;
- un processo può tentare di rilasciare una risorsa che non ha mai richiesto;
- un processo può richiedere due volte la stessa risorsa, senza rilasciarla prima della seconda richiesta.

Le stesse difficoltà si sono incontrate col costrutto della regione critica e sono, in realtà, simili alle difficoltà che condussero allo sviluppo dei costrutti di regione critica e di monitor. In precedenza ci si è preoccupati del corretto uso dei semafori, ora ci si deve preoccupare del corretto uso delle operazioni ad alto livello definite dal programmatore, senza poter avere, a questo livello, l'assistenza del compilatore. Una possibile soluzione del problema precedente prevede l'inclusione delle operazioni d'accesso alle risorse all'interno del monitor `assegnazione_risorse`. Comunque, adottando questa soluzione, per lo scheduling delle risorse si userebbe l'algoritmo di scheduling del monitor anziché quello desiderato.

Per garantire che i processi rispettino le sequenze appropriate, è necessario controllare tutti i programmi che usano il monitor `assegnazione_risorse` e la risorsa da esso gestita. Per stabilire la correttezza del sistema è necessario verificare le seguenti condizioni: i processi utenti devono sempre impiegare il monitor secondo una sequenza corretta; è necessario assicurare che un processo non cooperante non cerchi di aggirare la mutua esclusione offerta dal monitor e tenti di accedere direttamente alla risorsa condivisa senza usare i protocolli d'accesso. Soltanto se si assicurano queste due condizioni, si può garantire l'assenza di errori di sincronizzazione e che l'algoritmo di scheduling sia rispettato. Questo controllo è possibile per sistemi statici di piccole dimensioni, mentre non è ragionevolmente applicabile a sistemi di grandi dimensioni o a sistemi dinamici. Questo problema di controllo dell'accesso si può risolvere solo introducendo ulteriori meccanismi, descritti nel Capitolo 18.

7.8 Sincronizzazione nei sistemi operativi

In questo paragrafo si descrivono i meccanismi di sincronizzazione disponibili nei sistemi operativi Solaris e Windows 2000.

7.8.1 Sincronizzazione nel Solaris 2

Il sistema operativo Solaris 2 è progettato per gestire le elaborazioni in tempo reale, per essere multithread e per gestire architetture dotate di più unità d'elaborazione. Per il controllo dell'accesso alle sezioni critiche, prevede l'uso di semafori di mutua esclusione (*mutex*) adattivi, variabili condizionali, semafori, bloccaggio di lettura e scrittura, e i cosiddetti tornelli (*turnstile*).

Un **mutex adattivo** (*adaptive mutex*) protegge l'accesso a ogni elemento critico di dati; in un sistema con più unità d'elaborazione si attiva come un semaforo ordinario ed è realizzato come un semaforo ad attesa attiva. Se i dati sono bloccati e quindi già in uso, nel mutex adattivo si possono verificare due situazioni: se i dati sono posseduti da un thread che è correntemente in esecuzione in un'altra unità d'elaborazione, il thread che ha fatto la nuova richiesta d'accesso entra in uno stato d'attesa attiva (*spinlock*) mentre aspetta la rimozione del blocco dell'accesso ai dati, poiché è probabile che il thread in possesso dei dati termini la propria elaborazione in breve tempo; viceversa, se quest'ultimo non si trova nello stato d'esecuzione, il thread richiedente si sospende nello stato d'attesa e aspetta d'essere riattivato quando i dati saranno sbloccati. In questo modo si evita il ciclo d'attesa attiva, poiché probabilmente i dati non saranno rilasciati in un tempo ragionevolmente breve. Si ha questa situazione, ad esempio, quando il thread che ha bloccato l'accesso ai dati è a sua volta nello stato d'attesa. Poiché un sistema dotato di una singola CPU può eseguire un solo thread alla volta, il thread che ha bloccato l'accesso ai dati non può essere nello stato d'esecuzione nell'istante in cui un altro thread esegue la verifica della presenza del blocco. Quindi, in un sistema con una CPU, un thread che incontra un blocco sospende la propria esecuzione anziché persistere nel ciclo d'attesa attiva. Il metodo del mutex adattivo si usa per proteggere soltanto i dati cui si accede da segmenti di codice molto corti; in pratica si usa solo se un blocco permane per meno di qualche centinaio di istruzioni. Se il segmento di codice fosse più lungo, l'uso dei cicli d'attesa sarebbe inefficiente. Per segmenti di codice più lunghi il sistema ricorre all'impiego di semafori o variabili condizionali. Se l'oggetto richiesto è bloccato da un altro thread, il thread richiedente invoca una `wait` e si sospende. Quando il thread che possiede il dato ne rilascia il controllo, invia un segnale al successivo thread presente nella coda d'attesa per quel dato. L'ulteriore costo richiesto dalla sospensione e dalla successiva riattivazione del thread, compresi i relativi cambi di contesto, è sicuramente minore del costo dovuto alle parecchie centinaia di istruzioni che si sprecherebbero nel ciclo d'attesa del semaforo ad attesa attiva.

Il metodo dell'accesso bloccante per lettura e scrittura si usa per proteggere i dati cui si accede spesso, e di solito per la sola lettura. In questa situazione l'accesso bloccante per lettura e scrittura è più efficiente dell'uso dei semafori, poiché più thread possono leggere i dati in modo concorrente, mentre un semaforo avrebbe imposto la serializzazione di questi accessi. Poiché la sua realizzazione introduce un costo aggiuntivo, anche l'accesso bloccante per lettura e scrittura si applica solamente alle sezioni di codice lunghe.

Per ordinare la lista dei thread che attendono di ottenere un mutex adattivo o un accesso bloccante per lettura e scrittura il sistema operativo Solaris 2 usa i tornelli. Un **tornello** (*turnstile*) è una struttura a coda che contiene i thread che attendono il rilascio di un oggetto bloccato. Ad esempio, se un thread detiene l'accesso esclusivo a un oggetto sincronizzato, tutti gli altri thread che cercano di acquisire l'oggetto si bloccano ed entrano nel tornello relativo a quel blocco. Quando si rilascia l'oggetto, il nucleo seleziona un thread dal tornello per concedergli l'accesso bloccante all'oggetto. Ogni oggetto sincronizzato con almeno un thread che attende per l'accesso richiede un tornello separato. Tuttavia, anziché associare un tornello a ciascun oggetto sincronizzato, il sistema operativo assegna un tornello a ogni thread del livello del nucleo. Il tornello del primo thread

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

che si blocca a un oggetto sincronizzato diventa il tornello per l'oggetto stesso, i thread successivi si aggiungono allo stesso tornello. Quando il thread iniziale infine rilascia il blocco, acquisisce un nuovo tornello da una lista di tornelli liberi mantenuta dal nucleo. Per prevenire un'inversione delle priorità, i tornelli sono organizzati secondo un **protocollo di ereditarietà delle priorità** (Paragrafo 6.5). Ciò significa che se un thread detiene un accesso bloccante a un oggetto al quale attende un processo a priorità maggiore, il thread a priorità minore eredita temporaneamente la priorità del thread a priorità maggiore. Al rilascio del blocco, il thread riassume la priorità originaria.

Si noti che i meccanismi di bloccaggio usati dal nucleo sono disponibili anche per i thread del livello d'utente, sicché gli stessi tipi di bloccaggio sono disponibili sia all'interno sia all'esterno del nucleo. Una differenza cruciale nella loro realizzazione è il protocollo di ereditarietà delle priorità: le procedure di bloccaggio del nucleo adottano i metodi di ereditarietà delle priorità del nucleo usati dallo scheduler (Paragrafo 6.5); i meccanismi di bloccaggio per i thread del livello d'utente non offrono questa funzione.

Poiché l'accesso bloccante agli oggetti si usa frequentemente, e spesso si usa per funzioni cruciali del nucleo, la sua ottimizzazione può condurre a notevoli incrementi delle prestazioni. Per ottimizzare le prestazioni del Solaris 2 si affinano costantemente i metodi di bloccaggio.

7.8.2 Sincronizzazione nel Windows 2000

Il sistema operativo Windows 2000 ha un nucleo multithread che offre anche la gestione di applicazioni per le elaborazioni in tempo reale e di architetture con più unità d'elaborazione. Quando il nucleo di tale sistema operativo accede a una risorsa globale in un sistema con singola CPU, disabilita temporaneamente le interruzioni aventi procedure di gestione che potrebbero accedere alla stessa risorsa globale. In un sistema con più unità d'elaborazione, si protegge l'accesso alle risorse globali con i semafori ad attesa attiva (*spinlock*). Proprio come nel Solaris 2, il nucleo usa i semafori ad attesa attiva solo per proteggere segmenti di codice brevi. Inoltre, per ragioni di efficienza, il nucleo impedisce che un thread sia sottoposto a prelazione mentre detiene un semaforo ad attesa attiva. Per la sincronizzazione fuori dal nucleo, il sistema operativo offre gli **oggetti dispatcher**, che permettono ai thread di sincronizzarsi servendosi di diversi meccanismi, inclusi mutex, semafori ed eventi. I dati condivisi si possono proteggere richiedendo che un thread entri in possesso di un mutex prima di potervi accedere, e rilasci il mutex al completamento dell'elaborazione di quei dati. Gli eventi sono un meccanismo di sincronizzazione che si può usare in modo simile alle variabili condizionali; cioè, possono notificare il verificarsi di una determinata condizione a un thread che l'attendeva.

Gli oggetti dispatcher possono essere nello stato *signaled* o nello stato *nonsignaled*. Uno stato *signaled* indica che l'oggetto è disponibile e che un thread che tentasse di accedere all'oggetto non sarebbe bloccato; uno stato *nonsignaled* indica che l'oggetto non è disponibile e che qualsiasi thread che tentasse di accedervi sarebbe bloccato. C'è una relazione tra lo stato di un *oggetto dispatcher* e lo stato di un thread. Se un thread si blocca a un *oggetto dispatcher* nello stato *nonsignaled*, il suo stato cambia da pronto per l'esecuzione ad attesa, e il thread viene messo nella coda d'attesa per quell'oggetto. Quando lo stato dell'*oggetto dispatcher* diventa *signaled*, il nucleo verifica che non ci sia alcun thread che attende l'oggetto, altrimenti, ne fa passare uno, o più d'uno, dallo stato di attesa allo stato di pronto per l'esecuzione, dal quale possono riprendere l'esecuzione. Il numero dei thread che il nucleo seleziona dalla coda d'attesa dipende dal tipo di *oggetto dispatcher* al quale attendono. Il nucleo selezionerebbe un solo thread dalla coda d'attesa nel caso di un mutex, poiché un oggetto mutex può essere posseduto da un solo thread. Nel caso di un oggetto evento, il nucleo selezionerebbe tutti i thread che attendono l'evento stesso.

Consideriamo un'attesa a un mutex come esempio per illustrare gli *oggetti dispatcher* e gli stati dei thread. Se cercasse di acquisire un *oggetto dispatcher* di tipo mutex che è nello stato *nonsignaled*, un thread sarebbe sospeso e messo in una coda d'attesa per l'oggetto mutex. Se il mutex passasse allo stato *signaled* (il risultato del rilascio del blocco al mutex da parte di un altro thread), il thread che attende al mutex

1. passerebbe dallo stato d'attesa allo stato di pronto per l'esecuzione,
2. acquisirebbe il mutex.

7.9 Transazioni atomiche

La mutua esclusione nelle sezioni critiche assicura che esse siano eseguite in modo atomico. In altri termini, se due sezioni critiche sono eseguite in modo concorrente, il risultato che si ottiene coincide esattamente con quello che si otterrebbe dall'esecuzione sequenziale delle due sezioni critiche in un qualsiasi ordine. Pur essendo utile in molti ambiti, questa proprietà non è più sufficiente nei molti casi in cui si vuole avere la certezza che una sezione critica costituisca una singola unità logica di lavoro, caratterizzata dal fatto di essere eseguita nella sua totalità o non essere eseguita per niente. Un tipico esempio è costituito dal trasferimento di fondi; un'operazione che comporta l'addebito della valuta da un conto e il contemporaneo accredito su un altro. Chiaramente, affinché i dati mantengano la loro coerenza, o si portano a termine con successo entrambe le operazioni oppure non devono avvenire né l'addebito né l'accredito.

La discussione contenuta nel resto del paragrafo è strettamente correlata alla gestione delle basi di dati. Alle **basi di dati** sono correlati i problemi di archiviazione, recupero e coerenza dei dati. Recentemente si è riscontrato un notevole aumento d'interesse circa l'uso di tecniche proprie delle basi di dati all'interno dei sistemi operativi. I sistemi operativi si possono considerare sistemi per la manipolazione di dati; in questa veste, possono sicuramente trarre beneficio da tecniche e modelli ottenuti nella ricerca sulle basi di dati. Ad esempio, la flessibilità e la potenza di molti metodi per la gestione dei file usati nei sistemi operativi potrebbero migliorare se si sostituissero con metodi più formali propri delle basi di dati. Nei Paragrafi dal 7.9.2 al 7.9.4 si descrivono alcuni di questi metodi e le possibilità di applicazione ai sistemi operativi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

7.9.1 Modello di sistema

Un insieme di istruzioni (operazioni) che esegue una singola funzione logica prende il nome di **transazione**. Uno dei principali motivi per cui si fa ricorso alle transazioni è conservare l'atomicità malgrado la possibilità che si verifichino situazioni anomale all'interno del sistema. Nel Paragrafo 7.9 sono descritti i diversi meccanismi usati per garantire l'atomicità di una transazione; innanzi tutto si prende in considerazione un ambiente nel quale sia possibile eseguire una sola transazione alla volta, per poi passare alla discussione di casi nei quali si possono avere più transazioni simultaneamente attive. Una transazione è un'unità di programma che accede a elementi contenuti in file residenti nella memoria secondaria ed eventualmente li aggiorna. Per gli scopi della presente trattazione è sufficiente considerare una transazione come una sequenza di operazioni `read` e `write`, terminate da un'operazione `commit` o da un'operazione `abort`. L'operazione `commit` indica che la transazione è terminata con successo, mentre l'operazione `abort` significa che la transazione è fallita, ha dovuto arrestare la normale esecuzione a causa di qualche errore logico. L'effetto di una transazione che ha portato a termine con successo la propria esecuzione non si può annullare con un'operazione `abort`.

Può anche accadere che una transazione cessi la propria esecuzione a causa di un guasto del sistema. In entrambi i casi, poiché una transazione fallita può aver già alterato i dati ai quali ha avuto accesso, lo stato di questi potrebbe non coincidere con lo stato nel quale si sarebbero trovati se la transazione fosse stata eseguita in modo atomico. Al fine di assicurare la proprietà di atomicità, la terminazione anomala di una transazione non deve produrre alcun effetto sullo stato dei dati che questa ha già modificato. Quindi, è necessario ripristinare lo stato dei dati adoperati dalla transazione fallita, riportandolo a quello che li caratterizzava appena prima dell'inizio della transazione (*roll back*). Il rispetto di questa proprietà deve essere garantito dal sistema.

Per stabilire il modo in cui un sistema deve garantire l'atomicità delle proprie transazioni, è necessario identificare le proprietà dei dispositivi che si usano per memorizzare i dati ai quali esse accedono. I diversi tipi di dispositivi di memorizzazione si possono caratterizzare secondo la capacità, velocità e attitudine al recupero dai guasti.

- **Memorie volatili.** Le informazioni registrate nelle memorie volatili, ad esempio la memoria centrale o le cache, di solito non sopravvivono ai crolli del sistema. L'accesso a questo tipo di dispositivi è molto rapido, sia grazie alla velocità intrinseca degli accessi alla memoria sia grazie alla possibilità di accedere in modo diretto ai dati in esse contenuti.
- **Memorie non volatili.** Le informazioni registrate in memorie non volatili, ad esempio dischi e nastri magnetici, di solito sopravvivono ai crolli del sistema. I dischi sono più affidabili della memoria centrale, ma meno affidabili dei nastri magnetici. Sia i dischi sia i nastri sono soggetti a guasti che possono causare anche la perdita dei dati in essi registrati. Poiché dischi e nastri sono dispositivi elettromeccanici che richiedono movimenti fisici per accedere ai dati, attualmente i tempi d'accesso alle memorie non volatili superano di diversi ordini di grandezza quelli alle memorie volatili.

- **Memorie stabili.** Le informazioni contenute nelle memorie stabili per definizione non si perdono *mai* (data l'impossibilità teorica di garantire questa proprietà, è però indispensabile considerare quest'affermazione con una certa cautela). Per realizzare un'approssimazione di questi dispositivi è necessario replicare le informazioni in più memorie non volatili (di solito dischi) con tipi di guasto indipendenti, e aggiornare i dati in maniera controllata (si veda in proposito il Paragrafo 14.7).

La presente discussione si limita a descrivere come sia possibile assicurare l'atomicità di una transazione in un ambiente in cui eventuali guasti comporterebbero la perdita delle informazioni contenute in memorie volatili.

7.9.2 Ripristino basato sulla registrazione delle modifiche

Un modo per assicurare l'atomicità è registrare in memorie stabili le informazioni che descrivono tutte le modifiche che la transazione ha apportato ai dati ai quali ha avuto accesso. In tal senso, il metodo più largamente usato è quello della **registrazione con scrittura anticipata** (*write-ahead logging*); il sistema mantiene, nella memoria stabile, una struttura di dati chiamata **giornale delle modifiche** (*log*), in cui ciascun elemento descrive una singola operazione `write` eseguita dalla transazione ed è composto dei seguenti campi:

- **Nome della transazione.** Il nome, unico, della transazione che ha richiesto l'operazione `write`.
- **Nome del dato modificato.** Il nome, unico, del dato scritto dall'operazione `write`.
- **Valore precedente.** Il valore posseduto dall'elemento prima dell'operazione `write`.
- **Nuovo valore.** Il valore che l'elemento avrà una volta terminata l'operazione `write`.

Esistono altri elementi speciali del giornale delle modifiche, che si usano per registrare gli eventi significativi che si possono verificare durante l'elaborazione della transazione, ad esempio l'avvio e il successo o il fallimento della transazione.

Prima dell'avvio di una transazione T_i si registra l'elemento <T_i `start`> nel giornale delle modifiche; durante l'esecuzione, ciascuna operazione `write` di T_i è *preceduta* dalla scrittura dell'apposito nuovo elemento nel giornale delle modifiche. Il successo di T_i è sancito dalla registrazione dell'elemento <T_i `commit`> nel giornale delle modifiche.

Poiché le informazioni contenute nel giornale delle modifiche servono per la ricostruzione dello stato delle strutture di dati alle quali le diverse transazioni hanno avuto accesso, non è ammissibile che l'effettivo aggiornamento di un componente di queste strutture avvenga prima che il corrispondente elemento del giornale delle modifiche sia stato registrato nell'apposito dispositivo di memoria stabile. Quindi, il requisito fondamentale affinché questo metodo abbia successo è che l'elemento relativo a un componente `x` sia registrato nella memoria stabile prima dell'esecuzione dell'operazione `write(x)`.

Naturalmente, questo metodo determina una penalizzazione delle prestazioni, poiché ogni `write` logica richiede in realtà l'esecuzione di due `write` fisiche. Inoltre,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

aumenta la quantità di memoria secondaria usata poiché, oltre allo spazio necessario a contenere i dati, si deve prevedere lo spazio necessario al giornale delle modifiche. In situazioni in cui i dati sono molto importanti e si deve disporre di una rapida funzione di ripristino, tale onere merita di essere sostenuto.

Mediante l'uso dei giornali delle modifiche il sistema può gestire qualsiasi malfunzionamento, purché non sia una perdita delle informazioni contenute nella memoria non volatile. L'algoritmo di ripristino impiega le due seguenti procedure:

- `undo(`T_i`)`, per ripristinare il precedente valore di tutti i dati modificati dalla transazione T_i;
- `redo(`T_i`)`, per assegnare il nuovo valore a tutti i dati modificati dalla transazione T_i.

L'insieme dei dati aggiornati da T_i, con i valori vecchi e nuovi a essi associati, è reperibile nel giornale delle modifiche.

Per garantirne un corretto comportamento anche se si dovesse verificare un guasto durante il procedimento di ripristino, le operazioni undo (annullamento) e redo (ripetizione) devono essere idempotenti (in altri termini, le esecuzioni multiple di un'operazione devono produrre i medesimi risultati di una singola esecuzione).

Se una transazione T_i termina in modo anormale (fallisce), per ripristinare lo stato dei dati da essa modificati è sufficiente eseguire l'operazione `undo(`T_i`)`. Se si verifica un guasto nel sistema, il ripristino di uno stato corretto comporta la consultazione del giornale delle modifiche per determinare quali transazioni si devono annullare e quali si devono ripetere. Tale classificazione delle transazioni avviene secondo i seguenti parametri:

- la transazione T_i deve essere annullata se il giornale delle modifiche contiene l'elemento `<`T_i` start>` ma non l'elemento `<`T_i` commit>`;
- la transazione T_i deve essere ripetuta se il giornale delle modifiche contiene sia l'elemento `<`T_i` start>` sia l'elemento `<`T_i` commit>`.

7.9.3 Punti di verifica

Quando si verifica un guasto nel sistema è necessario consultare il giornale delle modifiche per determinare quali transazioni si devono annullare e quali si devono ripetere. Questo metodo richiede la scansione dell'intero giornale delle modifiche e presenta principalmente due inconvenienti:

1. La ricerca può richiedere un tempo piuttosto lungo.
2. La maggior parte delle transazioni, che secondo questo algoritmo deve essere ripetuta, ha già aggiornato con successo dati che, secondo il giornale delle modifiche, si dovrebbero ancora modificare. Benché la ripetizione delle modifiche non causi alcun danno (per l'idempotenza dell'operazione redo), il processo di ripristino richiederà senza dubbio più tempo.

Per ridurre questo genere di sprechi si introduce il concetto di **punto di verifica** (*checkpoint*). Durante l'esecuzione il sistema esegue la registrazione con scrittura anticipata e registrazioni periodiche, che costituiscono ciascun punto di verifica, definite dall'esecuzione della seguente sequenza di azioni:

1. registrazione in una memoria stabile di tutti gli elementi del giornale delle modifiche correntemente residenti in memorie volatili (di solito la memoria centrale);
2. registrazione in una memoria stabile di tutti i dati modificati correntemente residenti in memorie volatili;
3. registrazione dell'elemento `<checkpoint>` nel giornale delle modifiche residente nella memoria stabile.

La presenza dell'elemento `<checkpoint>` consente al sistema di rendere più efficiente la propria procedura di ripristino. Si consideri una transazione T_i terminata con successo (*committed*) prima dell'ultimo punto di verifica; l'elemento `<`T_i` commit>` precede nel giornale delle modifiche l'elemento `<checkpoint>`; quindi, qualsiasi modifica apportata da T_i è stata sicuramente registrata nella memoria stabile prima del punto di verifica o come parte del punto di verifica stesso, perciò durante un eventuale ripristino sarebbe inutile eseguire l'operazione redo sulla transazione T_i.

Quest'osservazione consente di migliorare il precedente algoritmo di ripristino: in seguito a un malfunzionamento, la procedura di ripristino esamina il giornale delle modifiche per determinare la più recente transazione T_i che ha iniziato la propria esecuzione prima del più recente punto di verifica; scandisce a ritroso il giornale delle modifiche fino al primo elemento `<checkpoint>` e quindi fino al primo elemento `<`T_i` start>`.

Una volta identificata la transazione T_i, le operazioni redo e undo si devono applicare solamente a T_i e a tutte le transazioni T_j cominciate dopo T_i, ignorando il resto del giornale delle modifiche. Detto T l'insieme di queste transazioni, le operazioni necessarie al completamento del ripristino sono le seguenti:

- esecuzione dell'operazione `redo(`T_k`)` per tutte le transazioni T_k appartenenti a T tali che il giornale delle modifiche contiene l'elemento `<`T_k` commit>`;
- esecuzione dell'operazione `undo(`T_k`)` per tutte le transazioni T_k appartenenti a T tali che il giornale delle modifiche non contiene l'elemento `<`T_k` commit>`.

7.9.4 Transazioni atomiche concorrenti

Poiché le transazioni sono atomiche, il risultato dell'esecuzione concorrente di più transazioni deve essere equivalente a quello che si otterrebbe eseguendo le transazioni in una sequenza arbitraria. Questa caratteristica di **serializzabilità** si può rispettare semplicemente eseguendo ciascuna transazione all'interno di una sezione critica. In altre parole, tutte le transazioni condividono un semaforo `mutex` inizializzato a `1`; la prima operazione che una transazione esegue è `wait(mutex)`, mentre l'ultima operazione che si esegue dopo il successo o il fallimento della transazione è `signal(mutex)`.

http://www.adultpdf.com
Created by TIFF To PDF Advanced Edition. To remove this mark, please register this software.

Questo schema assicura l'atomicità di tutte le transazioni che si eseguono in modo concorrente ma è troppo restrittivo; in molte situazioni si può consentire la sovrapposizione dell'esecuzione di più transazioni, pur rispettando la proprietà di serializzabilità; nel seguito si descrivono alcuni algoritmi di **controllo della concorrenza** che assicurano la serializzabilità.

7.9.4.1 Serializzabilità

Si consideri un sistema contenente due oggetti A e B, entrambi letti e modificati da due transazioni T_0 e T_1, e si supponga che le due transazioni siano eseguite in modo atomico, nell'ordine T_0, T_1. Questa **sequenza d'esecuzione** (*schedule*) è rappresentata nella Figura 7.24. Nella sequenza d'esecuzione 1 le istruzioni rispettano un ordinamento cronologico dall'alto verso il basso, con le istruzioni di T_0 disposte nella colonna di sinistra e quelle di T_1 nella colonna di destra.

Una sequenza d'esecuzione nella quale ciascuna transazione è eseguita in modo atomico si chiama **sequenza d'esecuzione seriale**, è composta di una sequenza di istruzioni appartenenti a transazioni diverse ed è caratterizzata dal fatto che tutte le istruzioni appartenenti a una singola transazione sono raggruppate. Quindi, dato un insieme di n transazioni, esistono $n!$ differenti sequenze d'esecuzione seriali valide. Ogni sequenza d'esecuzione seriale è valida, poiché equivale all'esecuzione atomica in un ordine qualsiasi delle transazioni in essa contenute.

Se si consente a due transazioni di sovrapporre le proprie esecuzioni, la sequenza d'esecuzione risultante non è più seriale. Il che non implica necessariamente che l'esecuzione risultante sia scorretta (cioè non equivalente a una sequenza d'esecuzione seriale). Per dimostrare quest'affermazione è necessario definire la nozione di **operazioni conflittuali**. Si consideri una sequenza d'esecuzione S nella quale compaiono in successione due operazioni O_i e O_j appartenenti rispettivamente alle transazioni T_i e T_j. Se tentano di accedere agli stessi dati e se almeno una tra le due è una `write`, le operazioni O_i e O_j sono *conflittuali*. Al fine di illustrare meglio il concetto di operazioni conflittuali, si consideri

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Figura 7.24 Sequenza d'esecuzione 1: sequenza d'esecuzione seriale in cui T_0 è seguita da T_1.

la sequenza d'esecuzione 2, non seriale, nella Figura 7.25. L'operazione `write`(A) di T_0 è in conflitto con `read`(A) di T_1; viceversa, l'operazione `write`(A) di T_1 non è in conflitto con `read`(B) di T_0, poiché le due operazioni accedono a elementi diversi.

Si supponga che O_i e O_j siano due operazioni concatenate all'interno di una sequenza d'esecuzione S; se O_i e O_j appartengono a transazioni diverse e non conflittuali si può invertirne l'ordine d'esecuzione, producendo una nuova sequenza d'esecuzione S'. Ci si aspetta che S ed S' siano equivalenti, poiché tutte le operazioni rispettano il medesimo ordine in entrambe le sequenze d'esecuzione, a eccezione delle sole O_i e O_j il cui ordine non è rilevante ai fini del risultato finale.

Per illustrare meglio il concetto di inversione dell'ordine delle operazioni si consideri la sequenza d'esecuzione 2 della Figura 7.25. Poiché l'operazione `write`(A) di T_1 non è in conflitto con l'operazione `read`(B) di T_0, dallo scambio della loro posizione si ottiene una sequenza d'esecuzione equivalente: indipendentemente dallo stato iniziale, entrambe le sequenze d'esecuzione producono il medesimo risultato. Proseguendo nello scambio delle operazioni non conflittuali si ottiene ciò che segue:

- scambio dell'operazione `read`(B) di T_0 con l'operazione `read`(A) di T_1;
- scambio dell'operazione `write`(B) di T_0 con l'operazione `write`(A) di T_1;
- scambio dell'operazione `write`(B) di T_0 con l'operazione `read`(A) di T_1.

Il risultato finale di questo procedimento è la sequenza d'esecuzione 1 della Figura 7.24, la quale è una sequenza d'esecuzione seriale. Questo ragionamento ha dimostrato come la sequenza d'esecuzione 2 sia equivalente a una sequenza d'esecuzione seriale e implica che, indipendentemente dallo stato iniziale del sistema, la sequenza d'esecuzione 2 produce il medesimo stato finale determinato da una qualsiasi sequenza d'esecuzione seriale equivalente.

Se una sequenza d'esecuzione S si può trasformare in una sequenza d'esecuzione seriale S' con una serie di scambi tra operazioni non conflittuali, si dice che S è in **conflitto serializzabile**. Quindi, la sequenza d'esecuzione 2 è in conflitto serializzabile, poiché si può trasformare nella sequenza d'esecuzione seriale 1.

T_0	T_1
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Figura 7.25 Sequenza d'esecuzione 2: sequenza d'esecuzione concorrente serializzabile.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

7.9.4.2 Protocolli d'accesso bloccante

Uno dei metodi che si usano per garantire la serializzabilità consiste nell'associare un segnale di blocco a ciascun dato, e richiedere che ogni transazione rispetti il **protocollo d'accesso bloccante**, che governa l'acquisizione e il rilascio dei diritti d'accesso con blocco dei dati. A un dato si può accedere in modo bloccante in diversi modi, due dei quali sono i seguenti:

- **Condiviso.** Se una transazione T_i accede a un elemento Q bloccandolo in modo condiviso, T_i può leggere ma non può scrivere nell'elemento (tale forma di blocco si denota con S).
- **Esclusivo.** Se una transazione T_i accede a un elemento Q bloccandolo in modo esclusivo, T_i può leggere e scrivere nell'elemento (tale forma di blocco si denota con X).

Ogni transazione deve richiedere il blocco di un elemento Q nel modo adeguato al tipo di operazione che deve eseguire su di esso.

Per accedere a un elemento Q, una transazione T_i deve innanzi tutto bloccare Q in modo adeguato. Se Q non è attualmente bloccato da un'altra transazione, l'acquisizione dell'elemento da parte di T_i è garantita; se Q è bloccato da un'altra transazione, T_i dovrà attenderne il rilascio. In particolare, si supponga che T_i richieda l'accesso bloccante esclusivo a Q, in questo caso T_i deve attendere che la transazione che ne è attualmente in possesso rilasci il blocco di Q. Viceversa, se l'accesso bloccante richiesto è di tipo condiviso, T_i deve aspettare il rilascio della risorsa solo se Q è attualmente soggetto a un blocco esclusivo; altrimenti ottiene anch'essa il diritto di accedere a Q. Si noti la somiglianza tra questo schema e l'algoritmo dei lettori e degli scrittori trattato nel Paragrafo 7.5.2.

Una transazione può sbloccare un elemento precedentemente bloccato, anche se deve in ogni caso mantenere il blocco per tutto il periodo in cui accede all'elemento. Inoltre, non è sempre desiderabile che una transazione sblocchi un elemento immediatamente dopo il suo ultimo accesso, poiché in questo modo la serializzabilità della transazione potrebbe non essere garantita.

Un protocollo che assicura la serializzabilità è il cosiddetto **protocollo d'accesso bloccante a due fasi**, che esige che ogni transazione richieda l'esecuzione delle operazioni di blocco e di sblocco in due fasi distinte:

- **Fase di crescita.** Una transazione può ottenere nuovi accessi bloccanti ai dati ma non può rilasciarne alcuno in suo possesso.
- **Fase di riduzione.** Una transazione può rilasciare accessi bloccanti ai dati di cui è in possesso ma non può ottenerne di nuovi.

Inizialmente, una transazione si trova nella fase di crescita e acquisisce gli accessi bloccanti ai dati di cui ha bisogno; nel rilasciare un accesso bloccante, la transazione entra nella fase di riduzione e non può richiedere nuovi accessi bloccanti.

Il protocollo d'accesso bloccante a due fasi garantisce la serializzabilità dei conflitti (Esercizio 7.23), ma non elimina la possibilità di situazioni di stallo. Inoltre può accadere che, dato un insieme di transazioni, esistano sequenze d'esecuzione in conflitto serializzabile che tuttavia non si possono ottenere attraverso il protocollo d'accesso bloccante a due fasi. A ogni modo, per migliorare le prestazioni di questo protocollo è indispensabile disporre di ulteriori informazioni relative alle transazioni oppure imporre una qualche struttura o un ordinamento dell'insieme dei dati.

7.9.4.3 Protocolli basati sulla marcatura temporale

Nei precedenti protocolli d'accesso bloccante l'ordinamento tra ogni coppia di transazioni conflittuali è determinato nella fase d'esecuzione dal primo accesso bloccante ai dati richiesto da entrambe e che comporta modi incompatibili. Un altro metodo per determinare l'ordine di serializzabilità consiste nello scegliere in anticipo un ordinamento delle transazioni. Il metodo più comunemente adottato consiste nell'usare uno schema con **ordinamento a marche temporali**.

A ciascuna transazione T_i nel sistema si associa una marca temporale (*timestamp*) unica individuata da $\mathrm{TS}(T_i)$; il sistema assegna la marca temporale prima che la transazione T_i inizi la propria esecuzione. Se una transazione T_i riceve una marca temporale $\mathrm{TS}(T_i)$ e si presenta una nuova transazione T_j, allora $\mathrm{TS}(T_i) < \mathrm{TS}(T_j)$. Esistono due semplici metodi per realizzare questo schema:

- Adoperare come marca temporale il valore dell'orologio di sistema; in altre parole, la marca temporale di una transazione equivale al valore dell'orologio nell'istante in cui essa si presenta nel sistema. Questo metodo non funziona per transazioni che avvengano in sistemi separati o con unità d'elaborazione che non condividono lo stesso orologio.
- Adoperare come marca temporale un contatore logico; in altri termini, la marca temporale di una transazione equivale al valore del contatore nell'istante in cui essa si presenta nel sistema. Il contatore s'incrementa dopo l'assegnazione di una nuova marca temporale.

Le marche temporali delle transazioni determinano l'ordine di serializzabilità. Quindi, se $\mathrm{TS}(T_i) < \mathrm{TS}(T_j)$, il sistema deve garantire che la sequenza d'esecuzione generata sia equivalente alla sequenza d'esecuzione seriale nella quale la transazione T_i appare prima della transazione T_j.

Per realizzare questo schema si associano due valori di marche temporali a ogni elemento Q:

- **R-timestamp**(Q), che denota la maggiore tra le marche temporali di tutte le transazioni che hanno completato con successo un'operazione `read(`Q`)`.
- **W-timestamp**(Q), che denota la maggiore tra le marche temporali di tutte le transazioni che hanno completato con successo un'operazione `write(`Q`)`.

Queste marche temporali si aggiornano dopo l'esecuzione di ogni nuova istruzione `read(`Q`)` o `write(`Q`)`.

http://www.adultpdf.com
Created by TIFF Combine - (no stamps are applied by registered version) To remove this mark, please register this software.

Il protocollo a ordinamento di marche temporali garantisce che l'esecuzione di tutte le operazioni `read` e `write` conflittuali rispetti l'ordinamento stabilito dalle marche temporali e funziona nel modo seguente:

- Si supponga che la transazione T_i sottoponga una `read`(Q).
  - Se TS(T_i) < W-timestamp(Q), questo stato implica che T_i deve leggere un valore di Q che è già stato sovrascritto; quindi, si rifiuta l'operazione `read` e si annulla T_i ripristinando lo stato iniziale (*rollback*).
  - Se TS(T_i) ≥ W-timestamp(Q), si esegue l'operazione `read` e a R-timestamp(Q) si assegna il massimo tra R-timestamp(Q) e TS(T_i).
- Si supponga che la transazione T_i sottoponga una `write`(Q).
  - Se TS(T_i) < R-timestamp(Q), questo stato implica che il valore di Q che T_i sta producendo era necessario in precedenza e che T_i aveva supposto che tale valore non sarebbe mai stato prodotto. Quindi si rifiuta l'operazione `write` e si annulla T_i ripristinando lo stato iniziale.
  - Se TS(T_i) < W-timestamp(Q), questo stato implica che T_i sta cercando di scrivere un valore di Q obsoleto. Quindi si rifiuta l'operazione `write` e si annulla T_i ripristinando lo stato iniziale.
  - Altrimenti, si esegue l'operazione `write`.

Ogni transazione T_i per cui lo schema di controllo della concorrenza ripristina lo stato iniziale come risultato della sottomissione di un'operazione `read` o `write` riceve una nuova marca temporale e viene riavviata.

Per illustrare questo protocollo si consideri la sequenza d'esecuzione 3 di due transazioni T_2 e T_3 proposta nella Figura 7.26. Si assume che ciascuna transazione riceva la propria marca temporale immediatamente prima dell'esecuzione della sua prima istruzione. Quindi, nella sequenza d'esecuzione 3, TS(T_2) < TS(T_3), e la presente sequenza d'esecuzione è consentita dal protocollo basato sulla marcatura temporale.

T_2	T_3
`read`(B)	
	`read`(B)
	`write`(B)
`read`(A)	
	`read`(A)
	`write`(A)

Figura 7.26 Sequenza d'esecuzione 3: sequenza d'esecuzione possibile col protocollo basato sulla marcatura temporale.

Questa sequenza d'esecuzione può essere generata anche dal protocollo d'accesso bloccante a due fasi; in ogni caso, certe sequenze d'esecuzione sono possibili col protocollo d'accesso bloccante a due fasi ma non col protocollo basato sulla marcatura temporale, e viceversa (Esercizio 7.24).

Il protocollo a ordinamento di marche temporali garantisce la serializzabilità del conflitto. Questa proprietà deriva dal fatto che le operazioni conflittuali si elaborano secondo l'ordine stabilito dalle marche temporali. Il protocollo assicura inoltre l'assenza di situazioni di stallo, poiché nessuna transazione rimane in attesa.

7.10 Sommario

Dato un gruppo di processi sequenziali cooperanti che condividono dati, è necessario garantirne la mutua esclusione. Si tratta di assicurare che una sezione critica di codice possa essere usata da un solo processo o thread alla volta. Esistono diversi algoritmi per risolvere il problema della sezione critica basati soltanto sul presupposto che le istruzioni di base del linguaggio di macchina (come `load`, `store` e `test`) siano eseguite in modo atomico.

In queste soluzioni lo svantaggio principale consiste nella necessità di ricorrere all'attesa attiva. I semafori consentono di superare questa difficoltà; si possono usare per risolvere diversi problemi di sincronizzazione e si possono realizzare in modo efficiente, soprattutto se è disponibile un'architettura che permette le operazioni atomiche.

Sono stati presentati diversi problemi di sincronizzazione — come il problema dei produttori e consumatori con memoria limitata, il problema dei lettori e degli scrittori e il problema dei cinque filosofi — che costituiscono esempi rappresentativi di una vasta classe di problemi di controllo della concorrenza. Questi problemi sono stati usati per verificare quasi tutti gli schemi di sincronizzazione proposti.

Il sistema operativo deve fornire mezzi di protezione contro gli errori di sincronizzazione. A tal fine sono stati proposti parecchi costrutti di linguaggio. Per risolvere in modo sicuro ed efficiente i problemi di mutua esclusione e di sincronizzazione si possono usare le regioni critiche. I monitor offrono un meccanismo di sincronizzazione per la condivisione di tipi di dati astratti. Una variabile condizionale consente a una procedura di monitor di sospendere la propria esecuzione finché non riceve un segnale.

Il Solaris 2 è un esempio di sistema operativo moderno che dispone di vari sistemi di bloccaggio per l'elaborazione multitask, multithread (inclusi i thread per elaborazioni in tempo reale) e la multielaborazione. Usa, per motivi di efficienza, mutex adattivi per la protezione dei dati da brevi segmenti di codice. Quando sezioni di codice più lunghe devono accedere ai dati si fa uso di variabili condizionali e dell'accesso bloccante per lettura e scrittura. Il Solaris impiega i tornelli (*turnstile*) per ordinare la lista dei thread che attendono di acquisire un mutex adattivo o un accesso bloccante per lettura e scrittura.

Il sistema operativo Windows 2000 gestisce i processi d'elaborazione in tempo reale e la multielaborazione. Quando il nucleo tenta di accedere a risorse globali in sistemi con

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

una sola unità d'elaborazione, si disabilitano le interruzioni le cui procedure di servizio possono accedere alle risorse globali. Nei sistemi con più unità d'elaborazione, le risorse condivise si proteggono con semafori ad attesa attiva (*spinlock*). All'esterno del nucleo, la sincronizzazione si ottiene impiegando gli oggetti dispatcher. Un oggetto di questo tipo si può usare come un semaforo di mutua esclusione, un semaforo o un evento. Un evento è un tipo di oggetto che funziona in modo simile a una variabile condizionale.

Una transazione è un'unità di programma che si deve eseguire in modo atomico, cioè le operazioni a essa associate si devono eseguire nella loro totalità o non si devono eseguire per niente. Per assicurare la proprietà di atomicità anche nel caso di malfunzionamenti, si può usare la registrazione con scrittura anticipata. Si registrano tutti gli aggiornamenti nel giornale delle modifiche, che si mantiene in una memoria stabile. Se si verifica un crollo del sistema, si usano le informazioni conservate nel giornale delle modifiche per ripristinare lo stato dei dati aggiornati; tale ripristino si esegue usando le operazioni undo e redo. Per ridurre il carico dovuto alla ricerca nel giornale delle modifiche dopo un malfunzionamento del sistema si può usare un metodo basato su punti di verifica.

Quando si sovrappongono le esecuzioni di diverse transazioni, l'esecuzione risultante può non essere equivalente a un'esecuzione ottenuta eseguendo in modo atomico tali transazioni. Per assicurare una corretta esecuzione si deve usare uno schema di controllo della concorrenza che garantisca la serializzabilità. Esistono diversi schemi di controllo della concorrenza che assicurano la serializzabilità differendo un'operazione o arrestando la transazione che ha richiesto l'operazione. I più diffusi sono i protocolli d'accesso bloccante e gli schemi d'ordinamento a marche temporali.

7.11 Esercizi

7.1 Spiegate il significato del termine *attesa attiva* ed elencate gli altri tipi di attesa esistenti in un sistema operativo. Discutete la possibilità di evitare l'attesa attiva.

7.2 Spiegate perché i semafori ad attesa attiva, pur non essendo adatti ai sistemi con una CPU, sono adatti ai sistemi con più unità d'elaborazione.

7.3 Dimostrate che per l'algoritmo del fornaio (Paragrafo 7.2.2) vale la seguente proprietà: se P_i si trova nella propria sezione critica e P_k (dove $k \neq i$) ha già scelto il proprio `numero[k] != 0`, allora `(numero[i], i) < (numero[k], k)`.

7.4 L'algoritmo seguente, concepito da Dekker, è la prima soluzione programmata conosciuta del problema della sezione critica per due processi. I due processi, P_0 e P_1, condividono le seguenti variabili:

```
boolean pronto[2]; /* inizialmente falsa* /
int turno;
```



```
do {
    pronto[i] = true;
    while (pronto[j]) {
        if (turno == j) {
            pronto[i] = false;
            while (turno == j);
            pronto[i] = true;
        }
    }

        sezione critica

    turno = j;
    pronto[i] = false;

        sezione non critica

} while (1);
```

Figura 7.27 Struttura del processo P_i nell'algoritmo di Dekker.

La struttura del processo P_i (`i == 0` oppure `1`), dove P_j (`j == 1` oppure `0`) è l'altro processo, è mostrata nella Figura 7.27. Dimostrate che l'algoritmo soddisfa tutti e tre i requisiti per il problema della sezione critica.

7.5 La prima soluzione programmata corretta del problema della sezione critica per n processi con un limite di $n - 1$ turni d'attesa è stata proposta da Eisenberg e McGuire.

I processi condividono le seguenti variabili:

```
enum statoP {inattivo, rich_in, in_sc};
statoP pronto[n];
int turno;
```

Ciascun elemento di `pronto` è inizialmente `inattivo`; il valore iniziale di `turno` è irrilevante (compreso tra 0 e $n - 1$). La struttura del processo P_i è illustrata nella Figura 7.28. Dimostrate che tale algoritmo soddisfa tutti e tre i requisiti del problema della sezione critica.

7.6 Nel Paragrafo 7.3 si è menzionato che una frequente disabilitazione delle interruzioni in certi casi può influenzare un orologio di sistema. Spiegate perché ciò può accadere e come si può minimizzare tale influenza.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
do {

    while (1) {
      stato[i] = rich_in;
      j = turno;
      while (j != i) {
        if (stato[j] != inattivo)
          j = turno;
        else
          j = (j + 1) % n;
      }
      stato[i] = in_sc;
      j = 0;
      while ((j < n) && (j == i) || stato[i] != in_sc))
        j++;
      if ((j >= n) && (turno == i || stato[turno] == inattivo)) break;
    }
    turno = i;

        sezione critica

    j = (turno + 1) % n;
    while (stato[j] == inattivo)
      j = (j + 1) % n;
    turno = j;
    stato[i] = inattivo;

        sezione non critica

} while (1);
```

Figura 7.28 Struttura del processo P_i nell'algoritmo di Eisenberg e McGuire.

7.7 Dimostrate che, se le operazioni `wait` e `signal` non sono eseguite in modo atomico, si può violare la mutua esclusione.

7.8 Considerate il **problema del barbiere dormiente**. Un negozio di barbiere ha una sala d'attesa con n sedie e la stanza del barbiere con la sedia da lavoro. Se non ci sono clienti da servire, il barbiere dorme; se un cliente entra nel negozio e trova tutte le sedie occupate, lascia il negozio; se il barbiere è occupato, ma ci sono sedie disponibili, il cliente si siede; se il barbiere dorme, il cliente lo sveglia. Scrivete un programma che coordini barbiere e clienti.

7.9 Considerate il **problema dei fumatori di sigarette**. Esso è dato da un sistema con tre processi *fumatori* e un processo *tabaccaio*. Ogni fumatore prepara in continuazione una sigaretta e la fuma, ma per preparare una sigaretta il fumatore deve avere tre ingredienti: tabacco, carta e fiammiferi. Uno dei processi fumatori ha la carta, un altro il tabacco e il terzo i fiammiferi. Il tabaccaio ha una fornitura illimitata dei tre ingredienti e posa due ingredienti sul tavolo. Il fumatore che dispone dell'ingrediente mancante può preparare la sigaretta e fumarla, segnalando al tabaccaio quando ha terminato. Il tabaccaio, allora, mette altri due ingredienti sul tavolo e il ciclo si ripete. Scrivete un programma che sincronizzi tabaccaio e fumatori.

7.10 Dimostrate che monitor, regioni critiche e semafori sono equivalenti, e che, pertanto, consentono di risolvere gli stessi problemi di sincronizzazione.

7.11 Progettate un monitor con memoria limitata, nel quale il vettore di memoria sia parte del monitor stesso.

7.12 All'interno di un monitor, la mutua esclusione fa sì che il monitor con memoria limitata dell'Esercizio 7.11 sia adatto soprattutto a vettori di memoria piccoli.

a) Spiegate perché questa affermazione è *vera*.
b) Progettate un nuovo schema idoneo a vettori di memoria grandi.

7.13 Supponete che l'istruzione `signal` possa apparire solo come ultima istruzione di una procedura di monitor. Spiegate come si può semplificare la realizzazione descritta nel Paragrafo 7.7.

7.14 Considerate un sistema composto dai processi $P_1, P_2, \ldots, P_n$, ciascuno dei quali è dotato di un numero di priorità unico. Scrivete un monitor che assegni a questi processi tre stampanti identiche, usando i numeri di priorità per stabilire l'ordine di assegnazione.

7.15 Un file deve essere condiviso tra diversi processi, ciascuno dei quali ha un numero unico. Al file possono accedere contemporaneamente più processi, sottoposti al seguente vincolo: la somma di tutti i numeri unici associati ai processi che hanno accesso concorrente al file deve essere minore di n. Scrivete un monitor per coordinare l'accesso al file.

7.16 Supponete di sostituire le operazioni di monitor `wait` e `signal` con un costrutto singolo `await(B)`, dove `B` è un'espressione booleana generale che fa sì che il processo che la esegue attenda fino a che `B` diventa vera.

a) Scrivete un monitor usando questo schema per il problema dei lettori e degli scrittori.
b) Spiegate perché, in generale, questo costrutto non si può realizzare in modo efficiente.
c) Elencate le restrizioni che si devono porre all'istruzione `await` per ottenerne una realizzazione efficiente. (Suggerimento: riducete la generalità di `B`; si veda [Kessels 1977].)

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

7.17 Scrivete un monitor per realizzare una 'sveglia' che permetta a un programma chiamante di differire la propria esecuzione per un numero specificato di unità di tempo (*battiti*). Presupponete l'esistenza di un orologio reale, che invoca a intervalli regolari una procedura `battito` del monitor.

7.18 Spiegate perché il sistema operativo Solaris 2 impiega diversi meccanismi di bloccaggio, e in quali casi fa uso di semafori ad attesa attiva, semafori, mutex adattivi, variabili condizionali e accesso bloccante per lettura e scrittura. Dite in quali casi impiega ciascun meccanismo e qual è lo scopo dei tornelli.

7.19 Spiegate perché i sistemi operativi Solaris 2 e Windows 2000 impiegano i semafori ad attesa attiva come meccanismo di sincronizzazione nei soli sistemi con più unità d'elaborazione e non nei sistemi con una CPU.

7.20 Spiegate le differenze, in termini di costo, fra i tre tipi di memorizzazione: volatile, non volatile e stabile.

7.21 Discutete gli scopi del meccanismo dei punti di verifica, la frequenza con cui si devono inserire e come tale frequenza influenzi i seguenti fattori:

- le prestazioni del sistema in assenza di malfunzionamenti;
- il tempo richiesto per il ripristino del sistema dopo un suo crollo;
- il tempo richiesto per il ripristino dopo un crollo di un disco.

7.22 Spiegate il concetto di atomicità di una transazione.

7.23 Illustrate come il protocollo d'accesso bloccante a due fasi assicuri la serializzabilità del conflitto.

7.24 Dimostrate l'esistenza di sequenze d'esecuzione possibili nel caso del protocollo d'accesso bloccante a due fasi, ma impossibili nel caso del protocollo basato sulla marcatura temporale, e viceversa.

7.12 Note bibliografiche

Gli algoritmi di mutua esclusione 1 e 2 per due processi sono stati trattati per la prima volta in [Dijkstra 1965a]. L'algoritmo di Dekker (Esercizio 7.4), la prima soluzione programmata del problema della mutua esclusione per due processi, è stato sviluppato dal matematico olandese T. Dekker. Questo algoritmo è trattato anche in [Dijkstra 1965a]. Una soluzione più semplice del problema della mutua esclusione per due processi è presentata in [Peterson 1981] (Algoritmo 3).

La prima soluzione del problema della mutua esclusione per n processi è stata presentata in [Dijkstra 1965b]. Questa soluzione però non pone un limite superiore al tempo d'attesa di un processo che chieda di essere autorizzato a entrare nella sezione critica. [Knuth 1966] ha presentato il primo algoritmo con un limite pari a 2^n turni. Un perfezionamento dell'algoritmo di Knuth che riduce l'attesa a n^2 turni è proposto in [deBruijn 1967]; [Eisenberg e McGuire 1972] (Esercizio 7.5) propone una soluzione che riduce l'attesa al limite inferiore di $n - 1$ turni. L'algoritmo del fornaio è stato sviluppato da [Lamport 1974]; anche in questo algoritmo il tempo è di $n - 1$ turni, ma l'algoritmo è più semplice da programmare e da capire. [Burns 1978] ha sviluppato un algoritmo, basato sull'architettura di macchina, che soddisfa il requisito d'attesa limitata.

[Lamport 1986] e [Lamport 1991] trattano il problema generale della mutua esclusione. Una raccolta di algoritmi per la mutua esclusione è stata proposta da [Raynal 1986].

[Dijkstra 1965a] ha proposto il concetto di semaforo; [Patil 1971] ha cercato di stabilire se i semafori possano risolvere tutti i possibili problemi di sincronizzazione; [Parnas 1975] ha affrontato alcuni punti deboli contenuti nelle argomentazioni di Patil; [Kosaraju 1973] ha proseguito il lavoro di Patil concependo un problema che non si può risolvere con le operazioni `wait` e `signal`. [Lipton 1974] ha evidenziato i limiti di diverse primitive di sincronizzazione.

I problemi classici del coordinamento dei processi, descritti in questo capitolo, sono paradigmi di una vasta classe di problemi inerenti il controllo della concorrenza. Il problema dei produttori e consumatori con memoria limitata, il problema dei cinque filosofi e il problema del barbiere dormiente (Esercizio 7.8) sono stati suggeriti da [Dijkstra 1965a] e [Dijkstra 1971]. Il problema dei fumatori di sigarette (Esercizio 7.9) è stato sviluppato da [Patil 1971]; il problema dei lettori e degli scrittori è stato suggerito da [Courtois et al. 1971]. L'argomento della lettura e scrittura concorrenti è trattato in [Lamport 1977]; il problema della sincronizzazione di processi indipendenti è discusso in [Lamport 1976].

Il concetto di regione critica è stato suggerito da [Hoare 1972] e da [Brinch Hansen 1972]. Il concetto di monitor è stato sviluppato da [Brinch Hansen 1973]. Una descrizione completa del concetto di monitor si trova in [Hoare 1974]. [Kessels 1977] ha proposto un'estensione del concetto di monitor che permette la segnalazione automatica. [Ben-Ari 1990] offre una trattazione generale della programmazione concorrente.

Dettagli sul meccanismo di bloccaggio del sistema operativo Solaris 2 sono presentati in [Mauro e McDougall 2001]. Si noti che i meccanismi di bloccaggio usati dal nucleo sono disponibili anche per i thread al livello d'utente, in tal modo gli stessi tipi di bloccaggio sono disponibili sia all'interno sia all'esterno del nucleo. In [Solomon e Russinovich 2000] si trova una descrizione dettagliata dei meccanismi di sincronizzazione impiegati nel sistema operativo Windows 2000.

Lo schema di registrazione con scrittura anticipata è stato introdotto per la prima volta nel System R [Gray et al. 1981]. Il concetto di serializzabilità è stato formulato da [Eswaran et al. 1976], in relazione al proprio lavoro sul controllo della concorrenza per il System R. Il protocollo d'accesso bloccante a due fasi è stato introdotto da [Eswaran et al. 1976]. Lo schema di controllo della concorrenza basato sulla marcatura temporale è stato fornito da [Reed 1983]. [Bernstein e Goodman 1980] presenta un'esposizione dei vari algoritmi per il controllo della concorrenza basati sulla marcatura temporale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 8

Stallo dei processi

In un ambiente con multiprogrammazione più processi possono competere per ottenere un numero finito di risorse; se una risorsa non è correntemente disponibile, il processo richiedente passa allo stato d'attesa. Se le risorse richieste sono trattenute da altri processi, a loro volta nello stato d'attesa, il processo potrebbe non cambiare più il suo stato. Situazioni di questo tipo sono chiamate di **stallo** (*deadlock*). Questo argomento è trattato brevemente anche nel Capitolo 7, nello studio dei semafori.

Un efficace esempio di situazione di stallo si può ricavare da una legge dello stato del Kansas approvata all'inizio del XX secolo, che in una sua parte recita: "Quando due treni convergono a un incrocio, ambedue devono arrestarsi, e nessuno dei due può ripartire prima che l'altro si sia allontanato".

In questo capitolo si descrivono i metodi che un sistema operativo può usare per prevenire o affrontare le situazioni di stallo. Occorre in ogni caso notare che la maggior parte dei sistemi operativi attuali non offre strumenti di prevenzione delle situazioni di stallo. Funzioni adatte a questo scopo saranno probabilmente aggiunte presto. Date le tendenze correnti, il problema dello stallo può diventare soltanto più importante: aumento del numero dei processi e delle risorse all'interno di un sistema, programmi multithread e preferenza attribuita ai sistemi con archivi di file e basi di dati anziché ai sistemi a lotti.

8.1 Modello del sistema

Un sistema è composto da un numero finito di risorse da distribuire tra più processi in competizione. Le risorse sono suddivise in tipi differenti, ciascuno formato da un certo numero di istanze identiche. Cicli di CPU, spazio di memoria, file e dispositivi di I/O (come stampanti e unità a nastri), sono tutti esempi di tipi di risorsa. Se un sistema ha due unità d'elaborazione, tale tipo di risorsa ha due istanze. Analogamente, il tipo di risorsa *stampante* può avere cinque istanze.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Se un processo richiede un'istanza relativa a un tipo di risorsa, l'assegnazione di *qualsiasi* istanza di quel tipo può soddisfare la richiesta. Se ciò non si verifica significa che le istanze non sono identiche e le classi di risorse non sono state definite correttamente. Un sistema può, ad esempio, avere due stampanti; se a nessuno interessa sapere quale sia la stampante in funzione, le due stampanti si possono definire come appartenenti alla stessa classe di risorse; se, però, una stampante si trova al nono piano e l'altra al piano terra, allora le due stampanti si possono considerare non equivalenti, e per definire ciascuna delle due può essere necessario ricorrere a classi di risorse distinte.

Prima di adoperare una risorsa, un sistema deve richiederla e, dopo averla usata, deve rilasciarla. Un processo può richiedere tutte le risorse necessarie per eseguire il compito assegnatogli, anche se ovviamente il numero delle risorse richieste non può superare il numero totale delle risorse disponibili nel sistema: un processo non può richiedere tre stampanti se il sistema ne ha solo due.

Nelle ordinarie condizioni di funzionamento un processo può servirsi di una risorsa soltanto se rispetta la seguente sequenza:

1. **Richiesta.** Se la richiesta non si può soddisfare immediatamente — ad esempio, perché la risorsa è attualmente in possesso di un altro processo —, il processo richiedente deve attendere fino a che non può acquisire tale risorsa.
2. **Uso.** Il processo può operare sulla risorsa (se, ad esempio, la risorsa è una stampante, il processo può effettuare una stampa).
3. **Rilascio.** Il processo rilascia la risorsa.

La richiesta e il rilascio di risorse avvengono tramite chiamate del sistema; ne sono esempi le chiamate del sistema `request_device` e `release_device`; `open_file` e `close_file`; e `allocate_memory` e `free_memory`. La richiesta e il rilascio di altre risorse si possono eseguire per mezzo delle operazioni `wait` e `signal` su semafori. Quindi, ogni volta che si usa una risorsa, il sistema operativo controlla che il processo utente ne abbia fatto richiesta e che questa gli sia stata assegnata. Una tabella di sistema registra lo stato di ogni risorsa e, se questa è assegnata, indica il processo relativo. Se un processo richiede una risorsa già assegnata a un altro processo, il processo richiedente può essere accodato agli altri processi che attendono tale risorsa.

Un gruppo di processi entra in stallo quando tutti i processi del gruppo attendono un evento che può essere causato solo da un altro processo che si trova nello stato di attesa. Gli eventi maggiormente discussi in questo capitolo sono l'acquisizione e il rilascio di risorse. Le risorse possono essere sia fisiche, ad esempio, stampanti, unità a nastri, spazio di memoria e cicli di CPU, sia logiche, ad esempio, file, semafori e monitor. Tuttavia anche altri eventi possono condurre a situazioni di stallo, ad esempio, le funzioni di IPC trattate nel Capitolo 4.

Per esaminare una situazione di stallo si consideri un sistema con tre unità a nastri. Si supponga che tre processi possiedano ciascuno una di queste unità a nastri. Se ogni processo richiedesse un'altra unità a nastri, i tre processi entrerebbero in stallo: ciascuno attenderebbe il rilascio dell'unità a nastri, ma quest'evento potrebbe essere causato solo da uno degli altri processi che a loro volta attendono l'ulteriore unità a nastri. Quest'esempio descrive uno stallo causato da processi che competono per acquisire lo stesso tipo di risorsa.

Le situazioni di stallo possono implicare anche diversi tipi di risorsa. Si consideri, ad esempio, un sistema con una stampante e un'unità a nastri, e si supponga che il processo P_i sia in possesso dell'unità a nastri e il processo P_j della stampante. Se P_i richiedesse la stampante e P_j l'unità a nastri, si avrebbe uno stallo.

Un programmatore che sviluppa applicazioni multithread deve prestare una particolare attenzione a questo problema: i programmi multithread sono ottimi candidati alle situazioni di stallo poiché più thread possono competere per le risorse condivise.

8.2 Caratterizzazione delle situazioni di stallo

In una situazione di stallo, i processi non terminano mai l'esecuzione, e si bloccano le risorse del sistema impedendo l'esecuzione di altri processi. Prima di trattarne le soluzioni, si descrivono le caratteristiche del problema dello stallo.

8.2.1 Condizioni necessarie

Si può avere una situazione di stallo *solo se* si verificano contemporaneamente le seguenti quattro condizioni:

1. **Mutua esclusione.** Almeno una risorsa deve essere non condivisibile, vale a dire che può essere usata da un solo processo alla volta. Se un altro processo richiede tale risorsa, si deve ritardare il processo richiedente fino al rilascio della risorsa.
2. **Possesso e attesa.** Un processo in possesso di almeno una risorsa attende di acquisire risorse già in possesso di altri processi.
3. **Impossibilità di prelazione.** Non esiste un diritto di prelazione sulle risorse, vale a dire che una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, dopo aver terminato il proprio compito.
4. **Attesa circolare.** Deve esistere un insieme $\{P_0, P_1, ..., P_n\}$ di processi, tale che P_0 attende una risorsa posseduta da P_1, P_1 attende una risorsa posseduta da P_2, ..., P_{n-1} attende una risorsa posseduta da P_n e P_n attende una risorsa posseduta da P_0.

Occorre sottolineare che tutte e quattro le condizioni devono essere vere, altrimenti non si può avere alcuno stallo. La condizione dell'attesa circolare implica la condizione di possesso e attesa, quindi le quattro condizioni non sono completamente indipendenti; tuttavia è utile considerare separatamente ciascuna condizione (Paragrafo 8.4).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.2.2 Grafo di assegnazione delle risorse

Le situazioni di stallo si possono descrivere con maggior precisione avvalendosi di una rappresentazione detta **grafo di assegnazione delle risorse**. Si tratta di un insieme di vertici V e un insieme di archi E, con l'insieme di vertici V composto da due sottoinsiemi: $P = \{P_1, P_2, ..., P_n\}$, che rappresenta tutti i processi del sistema, e $R = \{R_1, R_2, ..., R_m\}$, che rappresenta tutti i tipi di risorsa del sistema.

Un arco diretto dal processo P_i al tipo di risorsa R_j si indica $P_i \rightarrow R_j$, e significa che il processo P_i ha richiesto un'istanza del tipo di risorsa R_j, e attualmente attende tale risorsa. Un arco diretto dal tipo di risorsa R_j al processo P_i si indica $R_j \rightarrow P_i$, e significa che un'istanza del tipo di risorsa R_j è assegnata al processo P_i. Un arco orientato $P_i \rightarrow R_j$ si chiama **arco di richiesta**, un arco orientato $R_j \rightarrow P_i$ si chiama **arco di assegnazione**.

Graficamente ogni processo P_i si rappresenta con un cerchio e ogni tipo di risorsa R_j si rappresenta con un rettangolo. Giacché il tipo di risorsa R_j può avere più di un'istanza, ciascuna di queste istanze si rappresenta con un puntino all'interno del rettangolo. Occorre notare che un arco di richiesta è diretto soltanto verso il rettangolo R_j, mentre un arco di assegnazione deve designare anche uno dei puntini del rettangolo.

Quando il processo P_i richiede un'istanza del tipo di risorsa R_j, si inserisce un arco di richiesta nel grafo di assegnazione delle risorse. Se questa richiesta può essere esaudita, si trasforma *immediatamente* l'arco di richiesta in un arco di assegnazione, che al rilascio della risorsa viene cancellato.

Nel grafo di assegnazione delle risorse della Figura 8.1 è illustrata la seguente situazione:

- insiemi P, R ed E,
  - $P = \{P_1, P_2, P_3\}$,
  - $R = \{R_1, R_2, R_3, R_4\}$,
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$;

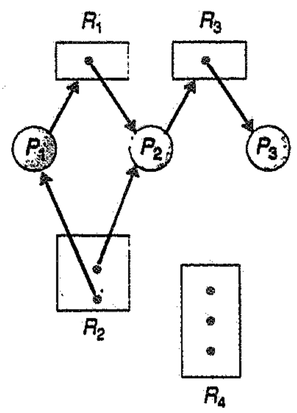


Figura 8.1 Grafo di assegnazione delle risorse.

- istanze delle risorse,
  - un'istanza del tipo di risorsa R_1,
  - due istanze del tipo di risorsa R_2,
  - un'istanza del tipo di risorsa R_3,
  - tre istanze del tipo di risorsa R_4;
- stati dei processi,
  - il processo P_1 possiede un'istanza del tipo di risorsa R_2 e attende un'istanza del tipo di risorsa R_1,
  - il processo P_2 possiede un'istanza dei tipi di risorsa R_1 ed R_2 e attende un'istanza del tipo di risorsa R_3,
  - il processo P_3 possiede un'istanza del tipo di risorsa R_3.

Data la definizione di grafo di assegnazione delle risorse, è facile mostrare che, se il grafo non contiene cicli, nessun processo del sistema subisce uno stallo; se il grafo contiene un ciclo, può sopraggiungere uno stallo.

Se ciascun tipo di risorsa ha esattamente un'istanza, allora l'esistenza di un ciclo implica la presenza di uno stallo; se il ciclo riguarda solo un insieme di tipi di risorsa, ciascuno dei quali ha solo un'istanza, si è verificato uno stallo. Ogni processo che si trovi nel ciclo è in stallo. In questo caso l'esistenza di un ciclo nel grafo è una condizione necessaria e sufficiente per l'esistenza di uno stallo.

Se ogni tipo di risorsa ha più istanze, un ciclo non implica necessariamente uno stallo. In questo caso l'esistenza di un ciclo nel grafo è una condizione necessaria ma non sufficiente per l'esistenza di uno stallo.

Per spiegare questo concetto conviene ritornare al grafo di assegnazione delle risorse della Figura 8.1. Si suppongà che il processo P_3 richieda un'istanza del tipo di risorsa R_2. Poiché attualmente non è disponibile alcuna istanza di risorsa, si aggiunge un arco di richiesta $P_3 \rightarrow R_2$ al grafo, com'è illustrato nella Figura 8.2. A questo punto nel sistema ci sono due cicli minimi:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

I processi P_1, P_2 e P_3 sono in stallo: il processo P_2 attende la risorsa R_3, posseduta dal processo P_3; il processo P_3, invece, attende che il processo P_1 o P_2 rilasci la risorsa R_2; inoltre il processo P_1 attende che il processo P_2 rilasci la risorsa R_1.

Si consideri ora il grafo di assegnazione delle risorse della Figura 8.3. Anche in questo esempio c'è un ciclo:

$$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$

In questo caso, però, non si ha alcuno stallo: il processo P_4 può rilasciare la propria istanza del tipo di risorsa R_2, che si può assegnare al processo P_3, rompendo il ciclo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

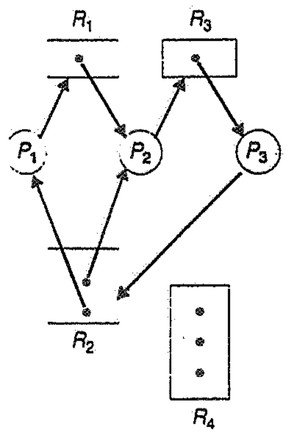


Figura 8.2 Grafo di assegnazione delle risorse con uno stallo.

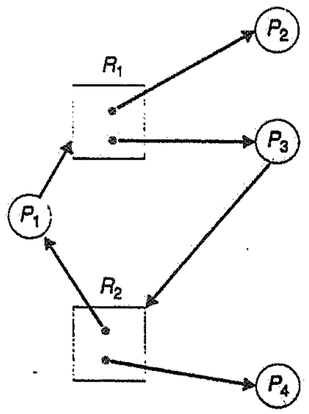


Figura 8.3 Grafo di assegnazione delle risorse con un ciclo, ma senza stallo.

Per concludere, l'assenza di cicli nel grafo di assegnazione delle risorse implica l'*assenza* di situazioni di stallo nel sistema. Viceversa, la presenza di un ciclo non è sufficiente a implicare la presenza di uno stallo nel sistema. Questa osservazione è importante ai fini della gestione del problema delle situazioni di stallo.

8.3 Metodi per la gestione delle situazioni di stallo

Essenzialmente, il problema delle situazioni di stallo si può affrontare impiegando tre metodi:

- si può usare un protocollo per prevenire o evitare le situazioni di stallo, assicurando che il sistema non entri *mai* in stallo;
- si può permettere al sistema di entrare in stallo, individuarlo, e quindi eseguire il ripristino;
- si può ignorare del tutto il problema, 'fingendo' che le situazioni di stallo non possano mai verificarsi nel sistema; è la soluzione usata dalla maggior parte dei sistemi operativi, compreso UNIX.

Nel seguito sono spiegati brevemente tutti questi metodi. Gli algoritmi relativi sono presentati in modo dettagliato nei Paragrafi dall'8.4 all'8.7.

Per assicurare che non si possa mai avere uno stallo, il sistema può servirsi di metodi di prevenzione o di metodi per evitare tale situazione. **Prevenire le situazioni di stallo** significa far uso di metodi atti ad assicurare che non si verifichi almeno una delle condizioni necessarie (Paragrafo 8.2.1). Questi metodi, discussi nel Paragrafo 8.4, prevengono le situazioni di stallo prescrivendo il modo in cui si devono fare le richieste delle risorse. Per **evitare le situazioni di stallo** occorre che il sistema operativo abbia in anticipo informazioni aggiuntive riguardanti le risorse che un processo richiederà e userà durante le sue attività. Con queste informazioni aggiuntive si può decidere se una richiesta di risorse da parte di un processo si può soddisfare o si debba invece sospendere. In tale processo di decisione il sistema tiene conto delle risorse correntemente disponibili, delle risorse correntemente assegnate a ciascun processo, e delle future richieste e futuri rilasci di ciascun processo. Questi metodi sono discussi nel Paragrafo 8.5.

Se un sistema non impiega né un algoritmo per prevenire né un algoritmo per evitare le situazioni di stallo, tali situazioni possono verificarsi. In un ambiente di questo tipo il sistema può servirsi di un algoritmo che esamini lo stato del sistema al fine di stabilire se si è verificato uno stallo; e in tal caso ricorrere a un secondo algoritmo per il ripristino del sistema. Tali argomenti sono discussi nei Paragrafi 8.6 e 8.7.

Se un sistema non garantisce che le situazioni di stallo non possano mai verificarsi e non fornisce alcun meccanismo per l'individuazione delle situazioni di stallo e il ripristino del sistema, possono verificarsi situazioni di stallo senza che ci sia la possibilità di capire cos'è successo. In questo caso la presenza di situazioni di stallo non rilevate può causare un degrado delle prestazioni del sistema; infatti la presenza di risorse assegnate a processi che non si possono eseguire determina lo stallo di un numero crescente di processi che richiedono tali risorse, fino al totale blocco del sistema che dovrà essere riavviato manualmente.

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

Sebbene questo modo di affrontare il problema delle situazioni di stallo non sembri fattibile, è in ogni caso usato in alcuni sistemi operativi perché è più economico dei metodi che altrimenti si dovrebbero adoperare per prevenire le situazioni di stallo, evitarle, o anche individuarle per il successivo ripristino. In molti sistemi, infatti, le situazioni di stallo accadono abbastanza raramente (magari una volta l'anno). In alcune circostanze il sistema è 'congelato' ma non è in stallo. Si consideri, ad esempio, la situazione determinata da un processo d'elaborazione in tempo reale che si esegue con la priorità più elevata (o qualsiasi processo in esecuzione in un sistema con scheduler senza diritto di prelazione) e che non restituisce il controllo al sistema operativo. Il sistema deve così disporre di meccanismi non automatici di ripristino per le situazioni che non sono in modo specifico di stallo, che può impiegare anche per le situazioni di stallo.

8.4 Prevenire le situazioni di stallo

Com'è evidenziato nel Paragrafo 8.2.1, affinché si abbia uno stallo si devono verificare quattro condizioni necessarie; perciò si può *prevenire* il verificarsi di uno stallo assicurando che almeno una di queste condizioni non possa capitare. Questo metodo è descritto nei particolari, con una trattazione di ciascuna delle quattro condizioni necessarie.

8.4.1 Mutua esclusione

Deve valere la condizione di mutua esclusione per le risorse non condivisibili: una stampante non può essere condivisa da più processi. Le risorse condivisibili, invece, non richiedono l'accesso mutuamente esclusivo, perciò non possono essere coinvolte in uno stallo. I file aperti per la sola lettura sono un buon esempio di risorsa condivisibile; è ovviamente consentito l'accesso contemporaneo da parte di più processi. Un processo non deve mai attendere una risorsa condivisibile. Ma poiché alcune risorse sono intrinsecamente non condivisibili, non si possono prevenire in generale le situazioni di stallo negando la condizione di mutua esclusione.

8.4.2 Possesso e attesa

Per assicurare che la condizione di possesso e attesa non si presenti mai nel sistema, occorre garantire che un processo che richiede una risorsa non ne possegga altre. Si può usare un protocollo che ponga la condizione che ogni processo, prima di iniziare la propria esecuzione, richieda tutte le risorse che gli servono e che esse gli siano assegnate. Questa condizione si può realizzare imponendo che le chiamate del sistema che richiedono risorse per un processo precedano tutte le altre chiamate del sistema.

Un protocollo alternativo è quello che permette a un processo di richiedere risorse solo se non ne possiede: un processo può richiedere risorse e adoperarle, ma prima di richiederne altre deve rilasciare tutte quelle che possiede.

Per spiegare la differenza tra questi due protocolli si può considerare un processo che copia i dati da un'unità a nastri a un file in un disco, ordina il contenuto del file e quindi stampa i risultati. Nel caso del primo protocollo, il processo deve richiedere fin dall'inizio l'unità a nastri, il file nel disco e una stampante. La stampante rimane in suo possesso per tutta la durata dell'esecuzione, anche se si usa solo alla fine.

Il secondo protocollo prevede che il processo richieda inizialmente solo l'unità a nastri e il file nel disco. Il processo copia i dati dall'unità a nastri al disco, quindi rilascia le due risorse. A questo punto richiede il file nel disco e la stampante, e dopo aver copiato il file nella stampante rilascia le due risorse e termina.

Questi protocolli presentano due svantaggi principali. Innanzi tutto, l'utilizzo delle risorse può risultare poco efficiente, poiché molte risorse possono essere assegnate, ma non utilizzate, per un lungo periodo di tempo. Nel caso del secondo metodo dell'esempio precedente si possono rilasciare l'unità a nastri e il file nel disco, per poi richiedere il file e la stampante, solo se si ha la certezza che i dati nel file restino immutati tra il rilascio e la seconda richiesta. Se non si può garantire quest'ultima condizione, è necessario richiedere tutte le risorse all'inizio per entrambi i protocolli.

Il secondo svantaggio è dovuto al fatto che si possono verificare situazioni di attesa indefinita. Un processo che richieda più risorse molto usate può trovarsi nella condizione di attenderne indefinitamente la disponibilità, poiché almeno una delle risorse di cui necessita è sempre assegnata a qualche altro processo.

8.4.3 Impossibilità di prelazione

La terza condizione necessaria prevede che non sia possibile avere la prelazione su risorse già assegnate. Per assicurare che questa condizione non persista, si può impiegare il seguente protocollo. Se un processo che possiede una o più risorse ne richiede un'altra che non gli si può assegnare immediatamente (cioè il processo deve attendere), allora si esercita la prelazione su tutte le risorse attualmente in suo possesso. Si ha cioè il rilascio implicito di queste risorse, che si aggiungono alla lista delle risorse che il processo sta attendendo; il processo viene nuovamente avviato solo quando può ottenere sia le vecchie risorse sia quella che sta richiedendo.

In alternativa, quando un processo richiede alcune risorse, si verifica la disponibilità di queste ultime: se sono disponibili vengono assegnate, se non sono disponibili si verifica se sono assegnate a un processo che attende altre risorse. In tal caso si sottraggono le risorse desiderate a quest'ultimo processo e si assegnano al processo richiedente. Se le risorse non sono disponibili né sono possedute da un processo in attesa, il processo richiedente deve attendere. Durante l'attesa si può avere la prelazione su alcune sue risorse; ciò può accadere solo se un altro processo le richiede. Un processo si può avviare nuovamente solo quando riceve le risorse che sta richiedendo e recupera tutte le risorse a esso sottratte durante l'attesa.

Questo protocollo è adatto a risorse il cui stato si può salvare e recuperare facilmente in un secondo tempo, come i registri della CPU e lo spazio di memoria, mentre non si può in generale applicare a risorse come le stampanti e le unità a nastri.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.4.4 Attesa circolare

La quarta e ultima condizione necessaria per una situazione di stallo è l'attesa circolare. Un modo per assicurare che tale condizione d'attesa non si verifichi consiste nell'imporre un ordinamento totale all'insieme di tutti i tipi di risorse e un ordine crescente di numerazione per le risorse richieste da ciascun processo.

Si supponga che $R = \{R_1, R_2, ..., R_m\}$ sia l'insieme dei tipi di risorse. A ogni tipo di risorsa si assegna un numero intero unico che permetta di confrontare due risorse e stabilirne la relazione di precedenza nell'ordinamento stabilito. Formalmente, si definisce una funzione iniettiva, $f: R \rightarrow N$, dove N è l'insieme dei numeri naturali. Se, ad esempio, l'insieme dei tipi di risorsa R contiene unità a nastri, unità a dischi e stampanti, allora la funzione f si può definire come segue:

$$f(\textit{unità a nastri}) = 1;$$
$$f(\textit{unità a dischi}) = 5;$$
$$f(\textit{stampante}) = 12.$$

Per prevenire il verificarsi di situazioni di stallo si può considerare il seguente protocollo: ogni processo può richiedere risorse solo seguendo un ordine crescente di numerazione. Ciò significa che un processo può richiedere inizialmente qualsiasi numero di istanze di un tipo di risorsa, ad esempio R_i, dopo di che il processo può richiedere istanze del tipo di risorsa R_j se e solo se $f(R_j) > f(R_i)$. Se sono necessarie più istanze dello stesso tipo di risorsa si deve presentare una singola richiesta per tutte le istanze. Ad esempio, con la funzione definita precedentemente, un processo che deve impiegare contemporaneamente l'unità a nastri e la stampante deve prima richiedere l'unità a nastri e poi la stampante.

In alternativa, si può stabilire che un processo, prima di richiedere un'istanza del tipo di risorsa R_j, rilasci qualsiasi risorsa R_i tale che $f(R_i) \geq f(R_j)$.

Se si usa uno di questi due protocolli, la condizione di attesa circolare non può sussistere. Ciò si può dimostrare supponendo, per assurdo, che esista un'attesa circolare. Si supponga che l'insieme di processi coinvolti nell'attesa circolare sia $\{P_0, P_1, ..., P_n\}$, dove P_i attende una risorsa R_i, posseduta dal processo P_{i+1}. (Sugli indici si usa l'aritmetica modulare, quindi P_n attende una risorsa R_n posseduta da P_0.) Poiché il processo P_{i+1} possiede la risorsa R_i mentre richiede la risorsa R_{i-1}, è necessario che sia verificata la condizione $f(R_i) < f(R_{i+1})$ per tutti gli i, ma ciò implica che $f(R_0) < f(R_1) < ... < f(R_n) < f(R_0)$. Per la proprietà transitiva, risulta che $f(R_0) < f(R_0)$, il che è impossibile; quindi, non può esservi attesa circolare.

Occorre notare che la funzione f si deve definire secondo l'ordine d'uso normale delle risorse del sistema. Ad esempio, poiché generalmente l'unità a nastri si usa prima della stampante, è sensato definire $f(\textit{unità a nastri}) < f(\textit{stampante})$.

8.5 Evitare le situazioni di stallo

Gli algoritmi di prevenzione delle situazioni di stallo trattati nel Paragrafo 8.4 si basano sul controllo dei modi di richiesta, in modo da assicurare che non si possa verificare almeno una delle condizioni necessarie perché si abbia uno stallo. Questo metodo può però causare effetti collaterali negativi, come uno scarso utilizzo dei dispositivi e una ridotta produttività del sistema.

Un metodo alternativo per evitare le situazioni di stallo consiste nel richiedere ulteriori informazioni sui modi di richiesta delle risorse. In un sistema con un'unità a nastri e una stampante, ad esempio, il processo P può dichiarare che intende richiedere prima l'unità a nastri, poi la stampante, e che rilascerà entrambe solo in seguito. Il processo Q, invece, richiederà prima la stampante e poi l'unità a nastri. Una volta acquisita la sequenza completa delle richieste e dei rilasci di ogni processo, si può stabilire per ogni richiesta se il processo debba attendere. In seguito a ogni richiesta, per stabilire se la richiesta possa essere soddisfatta o debba attendere per evitare un possibile stallo, il sistema deve esaminare le risorse attualmente disponibili, le risorse attualmente assegnate a ogni processo e le richieste e i rilasci futuri per ciascun processo.

Gli algoritmi differiscono tra loro per la quantità e il tipo di informazioni richieste. Il modello più semplice e più utile richiede che ciascun processo dichiari il *numero massimo* delle risorse di ciascun tipo di cui necessita. Data un'informazione a priori per ogni processo sul massimo numero di risorse richiedibili per ciascun tipo, si può costruire un algoritmo capace di assicurare che il sistema non entri in stallo. Questo algoritmo definisce un metodo per **evitare lo stallo**, ed esamina dinamicamente lo stato di assegnazione delle risorse per garantire che non possa esistere una condizione di attesa circolare. Lo *stato* di assegnazione delle risorse è definito dal numero di risorse disponibili e assegnate e dalle richieste massime dei processi.

8.5.1 Stato sicuro

Uno stato si dice *sicuro* se il sistema è in grado di assegnare risorse a ciascun processo (fino al suo massimo) in un certo ordine e impedire il verificarsi di uno stallo. Più formalmente, un sistema si trova in stato sicuro solo se esiste una **sequenza sicura**. Una sequenza di processi $<P_1, P_2, ..., P_n>$ è una sequenza sicura per lo stato di assegnazione attuale se, per ogni P_i, le richieste che P_i può ancora fare si possono soddisfare impiegando le risorse attualmente disponibili più le risorse possedute da tutti i P_j con $j < i$. In questa situazione, se le risorse necessarie al processo P_i non sono disponibili immediatamente, allora P_i può attendere che tutti i P_j abbiano finito, e a quel punto P_i può ottenere tutte le risorse di cui ha bisogno, completare il compito assegnato, restituire le risorse assegnate e terminare. Quando P_i termina, P_{i+1} può ottenere le risorse richieste, e così via. Se non esiste una sequenza di questo tipo, lo stato del sistema si dice *non sicuro*.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Uno stato non sicuro non è uno stato di stallo. Viceversa, uno stato di stallo è uno stato non sicuro; comunque non tutti gli stati non sicuri sono stati di stallo (Figura 8.4). Uno stato non sicuro *può* condurre a uno stallo. Finché lo stato rimane sicuro, il sistema operativo può evitare il verificarsi di stati non sicuri e di stallo. In uno stato non sicuro il sistema operativo non può impedire ai processi di chiedere risorse in modo da causare uno stallo: ciò che accade negli stati non sicuri dipende dal comportamento dei processi.

Per illustrare meglio quel che si è detto sopra, si consideri un sistema con 12 unità a nastri magnetici e 3 processi: P_0, P_1 e P_2. Il processo P_0 può richiedere 10 unità a nastri, il processo P_1 può richiederne 4 e il processo P_2 può richiedere fino a 9 unità a nastri. Supponendo che all'istante t_0 il processo P_0 possieda 5 unità a nastri, e che i processi P_1 e P_2 ne possiedano 2 ciascuno, restano libere 3 unità a nastri.

	Richieste massime	*Unità possedute*
P_0	10	5
P_1	4	2
P_2	9	2

All'istante t_0, il sistema si trova in uno stato sicuro. La sequenza $<P_1, P_0, P_2>$ soddisfa la condizione di sicurezza, poiché al processo P_1 si possono assegnare immediatamente tutte le unità a nastri richieste, che saranno poi restituite (a questo punto sono disponibili 5 unità a nastri), quindi il processo P_0 può avere tutte le unità a nastri richieste e restituirle (il sistema ha 10 unità a nastri disponibili) e infine il processo P_2 potrebbe avere tutte le sue unità a nastri e restituirle (sono disponibili tutte e 12 le unità a nastri).

Un sistema può passare da uno stato sicuro a uno stato non sicuro. Si supponga che all'istante t_1 il processo P_2 richieda un'ulteriore unità a nastri e che questa gli sia assegnata: il sistema non si trova più nello stato sicuro. A questo punto, si possono assegnare tutte le unità a nastri richieste soltanto al processo P_1. Al momento della restituzione, il sistema avrà solo 4 unità a nastri disponibili. Poiché al processo P_0 sono assegnate 5 unità a nastri, ma il numero massimo è 10, il processo può richiederne altre 5, ma poiché queste non sono disponibili il processo P_0 deve attendere. Analogamente, il processo P_2 può richiedere altre 6 unità a nastri ed è costretto ad attendere; il risultato è una situazione di stallo.

L'errore è stato commesso nel soddisfare la richiesta di un'ulteriore unità a nastri fatta dal processo P_2. Se P_2 avesse atteso il termine di uno degli altri processi e il conseguente rilascio delle sue risorse, si sarebbe potuta evitare la situazione di stallo.

Dato il concetto di stato sicuro, si possono definire algoritmi che permettano di evitare le situazioni di stallo. L'idea è semplice: è sufficiente assicurare che il sistema rimanga sempre in uno stato sicuro. Il sistema si trova inizialmente in uno stato sicuro; ogni volta che un processo richiede una risorsa, il sistema deve stabilire se la risorsa è attualmente disponibile oppure se il processo debba attendere. Si soddisfa la richiesta solo se l'assegnazione lascia il sistema in uno stato sicuro.

In questo modo, se un processo richiede una risorsa attualmente disponibile può essere comunque costretto ad attendere. Quindi, l'utilizzo delle risorse può essere inferiore rispetto a quello che si avrebbe in assenza di un algoritmo per evitare le situazioni di stallo.

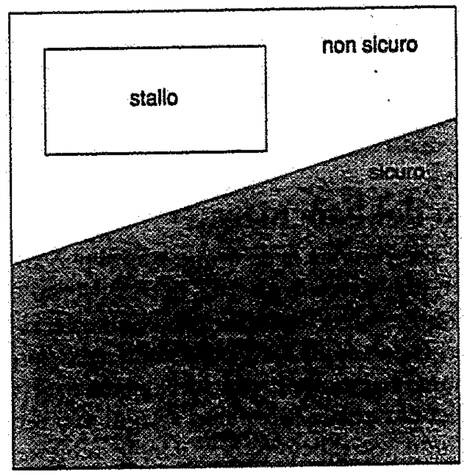


Figura 8.4 Spazi degli stati sicuri, non sicuri e di stallo.

8.5.2 Algoritmo con grafo di assegnazione delle risorse

Quando il sistema per l'assegnazione delle risorse è tale che ogni tipo di risorsa ha una sola istanza, per evitare le situazioni di stallo si può far uso di una variante del grafo di assegnazione delle risorse definito nel Paragrafo 8.2.2.

Oltre agli archi di richiesta e di assegnazione, si introduce un nuovo tipo di arco, l'arco di reclamo. Un **arco di reclamo** $P_i \rightarrow R_j$ indica che il processo P_i può richiedere la risorsa R_j in un qualsiasi momento futuro. Quest'arco ha la stessa direzione dell'arco di richiesta, ma si rappresenta con una linea tratteggiata. Quando il processo P_i richiede la risorsa R_j, l'arco di reclamo $P_i \rightarrow R_j$ diventa un arco di richiesta. Analogamente, quando P_i rilascia la risorsa R_j, l'arco di assegnazione $R_j \rightarrow P_i$ diventa un arco di reclamo $P_i \rightarrow R_j$. Occorre sottolineare che le risorse devono essere reclamate a priori nel sistema. Ciò significa che prima che il processo P_i cominci l'esecuzione, tutti i suoi archi di reclamo devono essere già inseriti nel grafo di assegnazione delle risorse. Questa condizione si può indebolire permettendo l'aggiunta di un arco di reclamo $P_i \rightarrow R_j$ al grafo solo se tutti gli archi associati al processo P_i sono archi di reclamo.

Si supponga che il processo P_i richieda la risorsa R_j. La richiesta si può soddisfare solo se la conversione dell'arco di richiesta $P_i \rightarrow R_j$ nell'arco di assegnazione $R_j \rightarrow P_i$ non causa la formazione di un ciclo nel grafo di assegnazione delle risorse. Occorre ricordare che la sicurezza si controlla con un algoritmo di rilevamento dei cicli, e che un algoritmo per il rilevamento di un ciclo in questo grafo richiede un numero di operazioni dell'ordine di n^2, dove con n si indica il numero dei processi del sistema.

Se non esiste alcun ciclo, l'assegnazione della risorsa lascia il sistema in uno stato sicuro. Se invece si trova un ciclo, l'assegnazione conduce il sistema in uno stato non sicuro, e il processo P_i deve attendere che si soddisfino le sue richieste.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per illustrare questo algoritmo si consideri il grafo di assegnazione delle risorse della Figura 8.5. Si supponga che P_2 richieda R_2. Sebbene sia attualmente libera, R_2 non può essere assegnata a P_2, poiché, come è evidenziato nella Figura 8.6, quest'operazione creerebbe un ciclo nel grafo e un ciclo indica che il sistema è in uno stato non sicuro. Se, a questo punto, P_1 richiedesse R_2, si avrebbe uno stallo.

8.5.3 Algoritmo del banchiere

L'algoritmo con grafo di assegnazione delle risorse non si può applicare ai sistemi di assegnazione delle risorse con più istanze di ciascun tipo di risorsa. L'algoritmo per evitare le situazioni di stallo qui descritto, pur essendo meno efficiente dello schema con grafo di assegnazione delle risorse, si può applicare a tali sistemi, ed è noto col nome di **algoritmo del banchiere**. Questo nome è stato scelto perché l'algoritmo si potrebbe impiegare in un sistema bancario per assicurare che la banca non assegni mai tutto il denaro disponibile, poiché, se ciò avvenisse, non potrebbe più soddisfare le richieste di tutti i suoi clienti.

Quando si presenta, un nuovo processo deve dichiarare il numero massimo delle istanze di ciascun tipo di risorsa di cui necessita. Questo numero non può superare il numero totale delle risorse del sistema. Quando un utente richiede un gruppo di risorse, si deve stabilire se l'assegnazione di queste risorse lasci il sistema in uno stato sicuro. Se si rispetta tale condizione, si assegnano le risorse, altrimenti il processo deve attendere che qualche altro processo rilasci un numero sufficiente di risorse.

La realizzazione dell'algoritmo del banchiere richiede la gestione di alcune strutture di dati che codificano lo stato di assegnazione delle risorse del sistema. Sia n il numero di processi del sistema e m il numero dei tipi di risorsa. Sono necessarie le seguenti strutture di dati:

- **Disponibili.** Un vettore di lunghezza m indica il numero delle istanze disponibili per ciascun tipo di risorsa; *Disponibili*$[j] = k$, significa che sono disponibili k istanze del tipo di risorsa R_j.
- **Massimo.** Una matrice $n \times m$ definisce la richiesta massima di ciascun processo; *Massimo*$[i, j] = k$ significa che il processo P_i può richiedere al più k istanze del tipo di risorsa R_j.
- **Assegnate.** Una matrice $n \times m$ definisce il numero delle istanze di ciascun tipo di risorsa attualmente assegnate a ogni processo; *Assegnate*$[i, j] = k$ significa che al processo P_i sono correntemente assegnate k istanze del tipo di risorsa R_j.
- **Necessità.** Una matrice $n \times m$ indica la necessità residua di risorse relativa a ogni processo; *Necessità*$[i, j] = k$ significa che il processo P_i, per completare il suo compito, può avere bisogno di altre k istanze del tipo di risorsa R_j. Si osservi che *Necessità*$[i, j]$ = *Massimo*$[i, j]$ – *Assegnate*$[i, j]$.

Col trascorrere del tempo, queste strutture di dati variano sia nelle dimensioni sia nei valori.

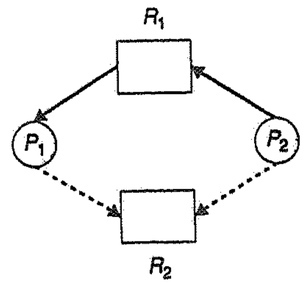


Figura 8.5 Grafo di assegnazione delle risorse per evitare le situazioni di stallo.

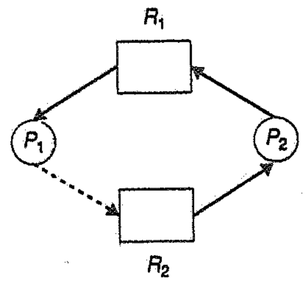


Figura 8.6 Uno stato non sicuro in un grafo di assegnazione delle risorse.

Per semplificare la presentazione dell'algoritmo del banchiere, si usano le seguenti notazioni: supponendo che X e Y siano vettori di lunghezza n, si può affermare che $X \leq Y$ se e solo se $X[i] \leq Y[i]$ per ogni $i = 1, 2, \ldots, n$. Ad esempio, se $X = (1, 7, 3, 2)$ e $Y = (0, 3, 2, 1)$, allora $Y \leq X$; $Y < X$ se $Y \leq X$ e $Y \neq X$.

Tutte le righe delle matrici *Assegnate* e *Necessità* si possono considerare vettori e si possono chiamare rispettivamente *Assegnate*$_i$ e *Necessità*$_i$. Il vettore *Assegnate*$_i$ specifica le risorse correntemente assegnate al processo P_i, mentre il vettore *Necessità*$_i$ specifica le risorse che il processo P_i può ancora richiedere per completare il suo compito.

8.5.3.1 Algoritmo di verifica della sicurezza

L'algoritmo utilizzato per scoprire se il sistema è o non è in uno stato sicuro si può descrivere come segue:
siano *Lavoro* e *Fine* vettori di lunghezza rispettivamente m e n,

1. inizializza *Lavoro* := *Disponibili* e *Fine*$[i]$:= *falso*, per $i = 1, 2, \ldots, n$;

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

2. cerca un indice i tale che valgano contemporaneamente le seguenti relazioni:
   a) $Fine[i] = falso$;
   b) $Necessità_i \leq Lavoro$;

   se tale i non esiste, esegue il passo 4;
3. $Lavoro := Lavoro + Assegnate_i$;

   $Fine[i] := vero$;

   torna al passo 2;
4. se $Fine[i] = vero$ per ogni i, allora il sistema è in uno stato sicuro.

Per decidere se uno stato è sicuro tale algoritmo può richiedere un numero di operazioni dell'ordine di $m \times n^2$.

8.5.3.2 Algoritmo di richiesta delle risorse

Sia $Richieste_i$ il vettore delle richieste per il processo P_i. Se $Richieste_i[j] = k$, allora il processo P_i richiede k istanze del tipo di risorsa R_j. Se il processo P_i fa una richiesta di risorse, si svolgono le seguenti azioni:

1. se $Richieste_i \leq Necessità_i$, esegue il passo 2, altrimenti riporta una condizione d'errore, poiché il processo ha superato il numero massimo di richieste;
2. se $Richieste_i \leq Disponibili$, esegue il passo 3, altrimenti P_i deve attendere poiché le risorse non sono disponibili;
3. il sistema simula l'assegnazione al processo P_i delle risorse richieste modificando come segue lo stato di assegnazione delle risorse:

   $Disponibili := Disponibili - Richieste_i$;

   $Assegnate_i := Assegnate_i + Richieste_i$;

   $Necessità_i := Necessità_i - Richieste_i$.

   Se lo stato di assegnazione delle risorse risultante è sicuro, la transazione è completata e al processo P_i si assegnano le risorse richieste. Tuttavia, se il nuovo stato è non sicuro, P_i deve attendere $Richieste_i$ e si ripristina il vecchio stato di assegnazione delle risorse.

8.5.3.3 Un esempio

Si consideri un sistema con cinque processi, da P_0 a P_4, e tre tipi di risorse: A, B, e C. Il tipo di risorse A ha 10 istanze, il tipo B ha 5 istanze e il tipo C ha 7 istanze. Si supponga che all'istante T_0 si sia verificata la seguente 'istantanea':

	Assegnate	*Massimo*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Il contenuto della matrice *Necessità* è definito come *Massimo* – *Assegnate*:

	Necessità
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Il sistema si trova attualmente in uno stato sicuro; infatti, la sequenza $<P_1, P_3, P_4, P_2, P_0>$ soddisfa i criteri di sicurezza. Si supponga ora che il processo P_1 richieda un'altra istanza del tipo di risorsa A e due istanze del tipo C, quindi $Richieste_1 = (1, 0, 2)$. Per stabilire se questa richiesta si possa esaudire immediatamente si verifica la condizione $Richieste_1 \leq Disponibili$ (vale a dire $(1, 0, 2) \leq (3, 3, 2)$), che risulta vera, quindi, supponendo che questa richiesta sia stata soddisfatta, si ottiene il seguente nuovo stato:

	Assegnate	*Necessità*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Occorre stabilire se questo nuovo stato del sistema sia sicuro; a tale scopo si esegue l'algoritmo di verifica della sicurezza da cui risulta che la sequenza $<P_1, P_3, P_4, P_0, P_2>$ rispetta il requisito di sicurezza. Quindi si può soddisfare immediatamente la richiesta del processo P_1.

Tuttavia, dovrebbe essere chiaro che, quando il sistema si trova in questo stato, una richiesta di (3, 3, 0) da parte di P_4 non si può soddisfare fino a che non siano disponibili le risorse. Una richiesta di (0, 2, 0) da parte di P_0 non si potrebbe soddisfare neanche se le risorse fossero disponibili, poiché lo stato risultante sarebbe non sicuro.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.6 Rilevamento delle situazioni di stallo

Se un sistema non si avvale di un algoritmo per prevenire o per evitare le situazioni di stallo, è possibile che una situazione di stallo si verifichi effettivamente. In tal caso il sistema deve fornire i seguenti algoritmi:

- un algoritmo che esamini lo stato del sistema per stabilire se si è verificato uno stallo;
- un algoritmo che ripristini il sistema dalla condizione di stallo.

Nella discussione che segue sono trattati i suddetti argomenti che riguardano sia sistemi con una sola istanza di ciascun tipo di risorsa, sia sistemi con più istanze di ciascun tipo di risorsa. Tuttavia, a questo punto, occorre notare che uno schema di rilevamento e ripristino richiede un carico che include sia i costi operativi per la memorizzazione delle informazioni necessarie e per l'esecuzione dell'algoritmo di rilevamento, sia i potenziali costi connessi al ripristino da una situazione di stallo.

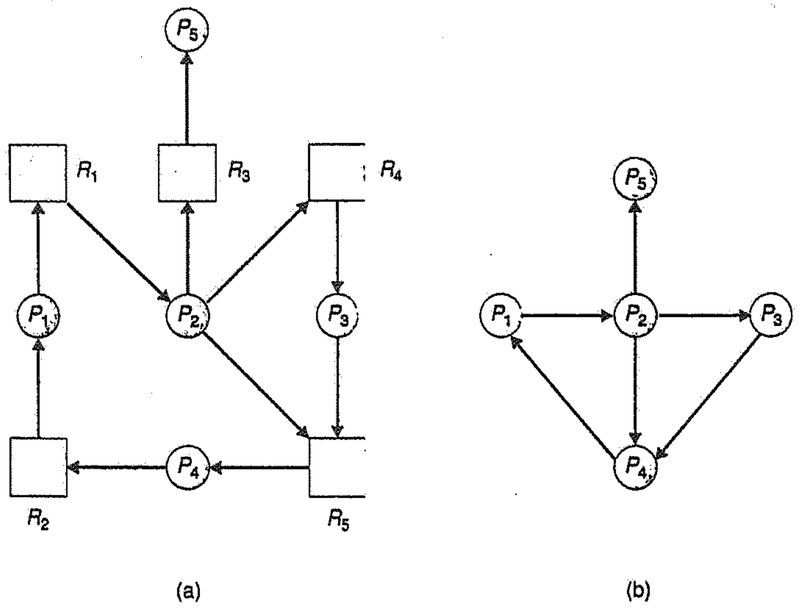


Figura 8.7 (a) Grafo di assegnazione delle risorse; (b) grafo d'attesa corrispondente.

8.6.1 Istanza singola di ciascun tipo di risorsa

Se tutte le risorse hanno una sola istanza si può definire un algoritmo di rilevamento di situazioni di stallo che fa uso di una variante del grafo di assegnazione delle risorse, detta **grafo d'attesa**, ottenuta dal grafo di assegnazione delle risorse togliendo i nodi dei tipi di risorse e componendo gli archi tra i processi.

Più precisamente, un arco da P_i a P_j del grafo d'attesa implica che il processo P_i attende che il processo P_j rilasci una risorsa di cui P_i ha bisogno. Un arco $P_i \rightarrow P_j$ esiste nel grafo d'attesa se e solo se il corrispondente grafo di assegnazione delle risorse contiene due archi $P_i \rightarrow R_q$ e $R_q \rightarrow P_j$ per qualche risorsa R_q. Nella Figura 8.7 sono illustrati un grafo di assegnazione delle risorse e il corrispondente grafo d'attesa.

Come in precedenza, nel sistema esiste uno stallo se e solo se il grafo d'attesa contiene un ciclo. Per individuare le situazioni di stallo, il sistema deve *conservare* il grafo d'attesa e *invocare periodicamente un algoritmo* che cerchi un ciclo all'interno del grafo.

L'algoritmo per il rilevamento di un ciclo all'interno di un grafo richiede un numero di operazioni dell'ordine di n^2, dove con n si indica il numero dei vertici del grafo.

8.6.2 Più istanze di ciascun tipo di risorsa

Lo schema con grafo d'attesa non si può applicare ai sistemi di assegnazione delle risorse con più istanze di ciascun tipo di risorsa. Il seguente algoritmo di rilevamento di situazioni di stallo è, invece, applicabile a tali sistemi. Esso si serve di strutture di dati variabili nel tempo, simili a quelle adoperate nell'algoritmo del banchiere (Paragrafo 8.5.3):

- *Disponibili*. Vettore di lunghezza m che indica il numero delle istanze disponibili per ciascun tipo di risorsa.
- *Assegnate*. Matrice $n \times m$ che definisce il numero delle istanze di ciascun tipo di risorse correntemente assegnate a ciascun processo.
- *Richieste*. Matrice $n \times m$ che indica la richiesta attuale di ciascun processo. Se *Richieste*$[i, j] = k$, significa che il processo P_i sta richiedendo altre k istanze del tipo di risorsa R_j.

La relazione $\leq$ tra due vettori si definisce come nel Paragrafo 8.5.3. Per semplificare la notazione, le righe delle matrici *Assegnate* e *Richieste* si trattano come vettori e, nel seguito, sono indicate rispettivamente come $Assegnate_i$ e $Richieste_i$. L'algoritmo di rilevamento descritto indaga su ogni possibile sequenza di assegnazione per i processi che devono ancora essere completati. Questo algoritmo si può confrontare con l'algoritmo del banchiere del Paragrafo 8.5.3. Siano *Lavoro* e *Fine* vettori di lunghezza rispettivamente m e n,

1. inizializza *Lavoro* := *Disponibili*, per $i = 1, 2, \ldots, n$, se $Assegnate_i \neq 0$, allora *Fine*$[i]$:= *falso*, altrimenti *Fine*$[i]$:= *vero*;
2. cerca un indice i tale che valgano contemporaneamente le seguenti relazioni:
   a) *Fine*$[i]$ = *falso*;
   b) $Richieste_i \leq Lavoro$;

   se tale i non esiste, esegue il passo 4;

http://www.adultpdf.com
Created by TIFF To PDF trial version; to remove this mark, please register this software.

3. *Lavoro* := *Lavoro* + *Assegnate*$_i$;

   Fine[*i*] = *vero*;

   torna al passo 2;

4. se *Fine*[*i*] = *falso* per qualche *i*, $1 \le i \le n$, allora il sistema è in stallo, inoltre, se *Fine*[*i*] = *falso*, il processo P_i è in stallo.

Tale algoritmo richiede un numero di operazioni dell'ordine di $m \times n^2$ per controllare se il sistema è in stallo.

Può meravigliare che le risorse del processo P_i siano richieste (passo 3) non appena risulta valida la condizione *Richieste*$_i$ ≤ *Lavoro* (passo 2.b). Tale condizione garantisce che P_i *non* è correntemente coinvolto in uno stallo, quindi, assumendo un atteggiamento ottimistico, si suppone che P_i non intenda richiedere altre risorse per completare il proprio compito, e che restituisca presto tutte le risorse. Se non si rispetta l'ipotesi fatta, si può verificare uno stallo, che sarà rilevato quando si richiamerà nuovamente l'algoritmo di rilevamento.

Per illustrare questo algoritmo, si consideri un sistema con cinque processi, da P_0 a P_4, e tre tipi di risorse: *A*, *B*, e *C*. Il tipo di risorsa *A* ha 7 istanze, il tipo *B* ha 2 istanze e il tipo *C* ha 6 istanze. Si supponga di avere, all'istante T_0, il seguente stato di assegnazione delle risorse:

	Assegnate	*Richieste*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Il sistema non è in stallo. Infatti eseguendo l'algoritmo per la sequenza <P_0, P_2, P_3, P_1, P_4>, risulta *Fine*[*i*] = *vero* per ogni *i*. Si supponga ora che il processo P_2 richieda un'altra istanza di tipo *C*. La matrice *Richieste* viene modificata come segue:

	Richieste
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

Ora il sistema è in stallo. Anche se si possono reclamare le risorse possedute dal processo P_0, il numero delle risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi, quindi si verifica uno stallo composto dei processi P_1, P_2, P_3 e P_4.

8.6.3 Uso dell'algoritmo di rilevamento

Per sapere quando è necessario ricorrere all'algoritmo di rilevamento si devono considerare i seguenti fattori:

1. *frequenza* (presunta) con la quale si verifica uno stallo;
2. *numero* dei processi che sarebbero influenzati da tale stallo.

Se le situazioni di stallo sono frequenti, è necessario ricorrere spesso all'algoritmo per il loro rilevamento. Le risorse assegnate a processi in stallo rimangono inattive fino all'eliminazione dello stallo. Inoltre, il numero dei processi coinvolti nel ciclo di stallo può aumentare.

Le situazioni di stallo si verificano solo quando qualche processo fa una richiesta che non si può soddisfare immediatamente; può essere una richiesta che chiude una catena di processi in attesa. Il caso estremo è quello nel quale l'algoritmo di rilevamento si usa ogni volta che non si può soddisfare immediatamente una richiesta di assegnazione. In questo caso non si identifica soltanto il gruppo di processi in stallo, ma anche il processo che ha 'causato' lo stallo, anche se, in verità, ciascuno dei processi in stallo è un elemento del ciclo all'interno del grafo di assegnazione delle risorse, quindi tutti i processi sono, congiuntamente, responsabili dello stallo. Se esistono tipi di risorsa diversi, una singola richiesta può causare più cicli nel grafo delle risorse, ciascuno dei quali è completato dalla richiesta più recente ed è causato da un processo identificabile.

Naturalmente, l'uso dell'algoritmo di rilevamento per ogni richiesta, aumenta notevolmente il carico nei termini di tempo di calcolo. Un'alternativa meno dispendiosa è quella in cui l'algoritmo di rilevamento s'invoca a intervalli meno frequenti, ad esempio una volta ogni ora, oppure ogni volta che l'utilizzo della CPU scende sotto il 40 per cento, poiché uno stallo può rendere inefficienti le prestazioni del sistema e quindi causare una drastica riduzione dell'utilizzo della CPU. Non è conveniente richiedere l'algoritmo di rilevamento in momenti arbitrari, poiché nel grafo delle risorse possono coesistere molti cicli e, normalmente, non si può dire quale fra i tanti processi in stallo abbia 'causato' lo stallo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.7 Ripristino da situazioni di stallo

Una situazione di stallo si può affrontare in diversi modi. Una soluzione consiste nell'informare l'operatore della presenza dello stallo, in modo che possa gestirlo manualmente. L'altra soluzione lascia al sistema il ripristino automatico dalla situazione di stallo. Uno stallo si può eliminare in due modi: il primo prevede semplicemente la terminazione di uno o più processi per interrompere l'attesa circolare; il secondo esercita la prelazione su alcune risorse in possesso di uno o più processi in stallo.

8.7.1 Terminazione di processi

Per eliminare le situazioni di stallo attraverso la terminazione di processi si possono adoperare due metodi; in entrambi il sistema recupera immediatamente tutte le risorse assegnate ai processi terminati.

- **Terminazione di tutti i processi in stallo.** Chiaramente questo metodo interrompe il ciclo di stallo, ma l'operazione è molto onerosa; questi processi possono aver già fatto molti calcoli i cui risultati si annullano e probabilmente dovranno essere ricalcolati.
- **Terminazione di un processo alla volta fino all'eliminazione del ciclo di stallo.** Questo metodo causa un notevole carico, poiché, dopo aver terminato ogni processo, si deve impiegare un algoritmo di rilevamento per stabilire se esistono ancora processi in stallo.

Procurare la terminazione di un processo può essere un'operazione tutt'altro che semplice: se il processo si trova nel mezzo dell'aggiornamento di un file, la terminazione lascia il file in uno stato scorretto; analogamente, se il processo si trova nel mezzo di una stampa di dati, prima di stampare un lavoro successivo, il sistema deve reimpostare la stampante riportandola a uno stato corretto.

Se si adopera il metodo di terminazione parziale, dato un insieme di processi in stallo occorre determinare quale processo, o quali processi, costringere alla terminazione nel tentativo di sciogliere la situazione di stallo. Analogamente ai problemi di scheduling della CPU, si tratta di scegliere un criterio. Le considerazioni sono essenzialmente economiche: si dovrebbero arrestare i processi la cui terminazione causa il minimo costo. Sfortunatamente, il termine *minimo costo* non è preciso. La scelta dei processi è influenzata da diversi fattori, tra cui i seguenti:

1. la priorità dei processi;
2. il tempo trascorso dalla computazione e il tempo ancora necessario per completare i compiti assegnati ai processi;
3. la quantità e il tipo di risorse impiegate dai processi (ad esempio, se si può avere facilmente la prelazione sulle risorse);

4. la quantità di ulteriori risorse di cui i processi hanno ancora bisogno per completare i propri compiti;
5. il numero di processi che si devono terminare;
6. il tipo di processi: interattivi o a lotti.

8.7.2 Prelazione su risorse

Per eliminare uno stallo si può esercitare la prelazione sulle risorse: le risorse si sottraggono in successione ad alcuni processi e si assegnano ad altri processi finché si ottiene l'interruzione del ciclo di stallo.

Se per gestire le situazioni di stallo s'impiega la prelazione, si devono considerare i seguenti problemi:

1. **Selezione di una vittima.** Occorre stabilire quali risorse e quali processi si devono sottoporre a prelazione. Come per la terminazione dei processi, è necessario stabilire l'ordine di prelazione allo scopo di minimizzare i costi. I fattori di costo possono includere parametri come il numero delle risorse possedute da un processo in stallo, e la quantità di tempo già spesa durante l'esecuzione da un processo in stallo.
2. **Ristabilimento di un precedente stato sicuro.** Occorre stabilire cosa fare con un processo cui è stata sottratta una risorsa. Poiché è stato privato di una risorsa necessaria, la sua esecuzione non può continuare normalmente, quindi deve essere ricondotto a un precedente stato sicuro dal quale essere riavviato.

   Poiché, in generale, è difficile stabilire quale stato sia sicuro, la soluzione più semplice consiste nel terminare il processo e quindi riavviarlo. Certamente, è più efficace ristabilire un precedente stato sicuro del processo che sia sufficiente allo scioglimento della situazione di stallo, ma questo metodo richiede che il sistema mantenga più informazioni sullo stato di tutti i processi in esecuzione.
3. **Attesa indefinita.** È necessario assicurare che non si verifichino situazioni d'attesa indefinita, occorre cioè garantire che le risorse non siano sottratte sempre allo stesso processo. In un sistema in cui la scelta della vittima avviene soprattutto secondo fattori di costo, può accadere che si scelga sempre lo stesso processo; in questo caso il processo non riesce mai a completare il suo compito; si tratta di una situazione d'attesa indefinita che si deve affrontare in qualsiasi sistema concreto. Chiaramente è necessario assicurare che un processo possa essere prescelto come vittima solo un numero finito (e ridotto) di volte; la soluzione più diffusa prevede l'inclusione del numero di terminazioni, per successivi riavvii, tra i fattori di costo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.8 Sommario

Uno stallo si verifica quando in un insieme di processi ciascun processo dell'insieme attende un evento che può essere causato solo da un altro processo dell'insieme. In linea di principio, i metodi di gestione delle situazioni di stallo sono tre:

- impiegare un protocollo che prevenga o eviti le situazioni di stallo, assicurando che il sistema non entri mai in stallo;
- permettere al sistema di entrare in stallo e quindi effettuarne il ripristino;
- ignorare del tutto il problema fingendo che nel sistema non si verifichino mai situazioni di stallo. Tale 'soluzione' è adottata dalla maggior parte dei sistemi operativi, compreso UNIX.

Una situazione di stallo può presentarsi solo se all'interno del sistema si verificano contemporaneamente quattro condizioni necessarie: mutua esclusione, possesso e attesa, impossibilità di prelazione e attesa circolare. Per prevenire il verificarsi di situazioni di stallo è necessario assicurare che almeno una delle suddette condizioni necessarie non sia soddisfatta.

Un altro metodo per evitare le situazioni di stallo, meno rigido dell'algoritmo di prevenzione, consiste nel disporre di informazioni a priori su come ciascun processo intende impiegare le risorse. L'algoritmo del banchiere, ad esempio, richiede la conoscenza del numero massimo di ogni classe di risorse che può essere richiesto da ciascun processo. Servendosi di queste informazioni si può definire un algoritmo per evitare le situazioni di stallo.

Se un sistema non usa alcun protocollo per assicurare che non avvengano situazioni di stallo, è necessario un metodo di rilevamento e ripristino. Per stabilire se si sia verificato uno stallo, è necessario ricorrere a un algoritmo di rilevamento; nel caso in cui si rilevi uno stallo, il sistema deve attuare il ripristino terminando alcuni processi coinvolti nello stallo, oppure sottraendo risorse a qualcuno di essi.

In un sistema che sceglie le vittime, soprattutto secondo fattori di costo, possono presentarsi situazioni d'attesa indefinita. La conseguenza è che il processo selezionato non completi mai il proprio compito.

8.9 Esercizi

8.1 Elencate tre esempi di situazioni di stallo che non siano connesse a un ambito informatico.

8.2 Verificate la possibilità di uno stallo che coinvolga un solo processo. Motivate la risposta.

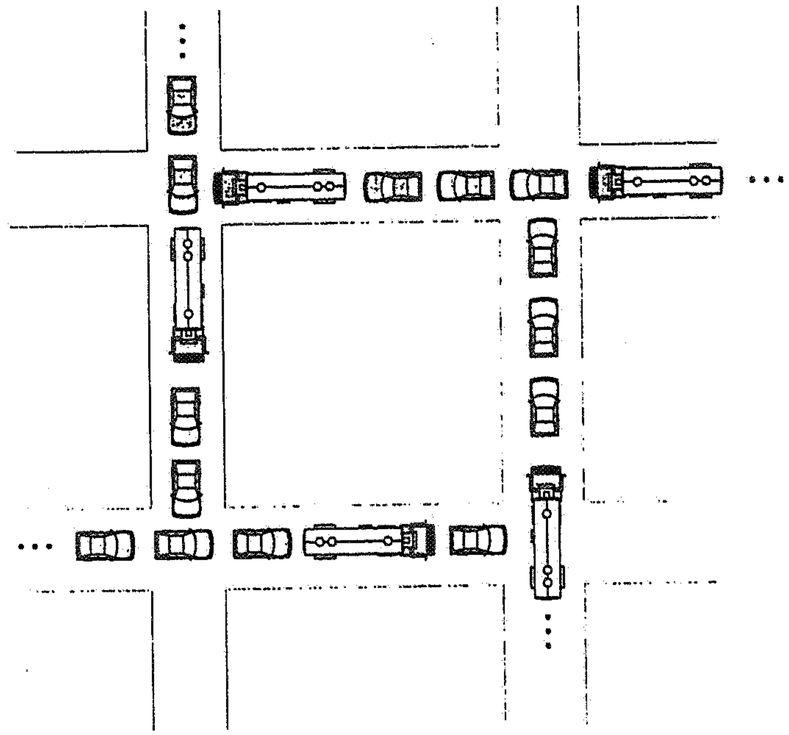


Figura 8.8 Stallo di traffico automobilistico per l'Esercizio 8.4.

8.3 È stato affermato che un'idonea gestione asincrona dell'elaborazione e dell'I/O (*spooling*) eliminerebbe le situazioni di stallo. Certamente eviterebbe la contesa di lettori di schede, stampanti, ecc. Tale tecnica si può impiegare anche per i nastri (si parla di *staging* dei nastri), e lascerebbe come risorse il tempo di CPU, la memoria e i dischi. Dite se si può avere uno stallo che coinvolga queste risorse. Nel caso affermativo dite come potrebbe avvenire tale stallo, nel caso contrario spiegate perché. Dite quale schema, ammesso che esista, sembra essere il migliore per eliminare queste situazioni di stallo, oppure quale condizione viene violata se non esiste alcuno schema.

8.4 Considerate lo stallo di traffico automobilistico illustrato nella Figura 8.8,

a) dimostrate che le quattro condizioni necessarie per lo stallo valgono anche in questo esempio;

b) fissate una semplice regola che eviti le situazioni di stallo in questo sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

8.5 Supponendo che un sistema sia in uno stato non sicuro, dimostrate che i processi possono completare le loro esecuzioni senza entrare in stallo.

8.6 In un sistema reale, né le risorse disponibili, né le richieste di risorse da parte dei processi sono coerenti in lunghi periodi (mesi). Le risorse si guastano o sono sostituite, nuovi processi vanno e vengono, si acquistano e si aggiungono nuove risorse al sistema. Se le situazioni di stallo si controllano con l'algoritmo del banchiere, dite quali tra le seguenti modifiche si possono apportate con sicurezza, senza introdurre la possibilità di situazioni di stallo, e in quali circostanze:

a) aumento di *Disponibili* (aggiunta di nuove risorse);
b) riduzione di *Disponibili* (risorsa rimossa definitivamente dal sistema);
c) aumento di *Massimo* per un processo (il processo necessita di più risorse di quante siano permesse);
d) riduzione di *Massimo* per un processo (il processo decide che non ha bisogno di tante risorse);
e) aumento del numero di processi;
f) riduzione del numero di processi.

8.7 Dimostrate che l'algoritmo di verifica della sicurezza presentato nel Paragrafo 8.5.3 richiede un numero di operazioni dell'ordine di $m \times n^2$.

8.8 Considerate un sistema composto da quattro risorse dello stesso tipo condivise da tre processi, ciascuno dei quali necessita di non più di due risorse. Dimostrate che non si possono verificare situazioni di stallo.

8.9 Considerate un sistema composto da m risorse dello stesso tipo, condivise da n processi. Le risorse possono essere richieste e rilasciate dai processi solo una alla volta. Dimostrate che non si possono verificare situazioni di stallo se si rispettano le seguenti condizioni:

a) la richiesta massima di ciascun processo è compresa tra 1 e m risorse;
b) la somma di tutte le richieste massime è minore di $m + n$.

8.10 Considerate un sistema di calcolo che esegue 5000 lavori d'elaborazione al mese e che non dispone di meccanismi per prevenire o evitare le situazioni di stallo. In media si verificano situazioni di stallo due volte al mese, e l'operatore deve terminare ed eseguire nuovamente circa 10 lavori per ogni stallo. Ogni lavoro costa circa 2 euro (in tempo di CPU) e al momento della terminazione forzata è, in media, eseguito per metà.

Un programmatore di sistemi ha stimato che un algoritmo per evitare le situazioni di stallo, come quello del banchiere, si può installare nel sistema con un aumento medio del tempo d'esecuzione per ogni lavoro d'elaborazione del 10 per cento circa. Poiché il calcolatore presenta attualmente un tempo morto del 30 per cento circa, si potrebbero ancora eseguire tutti i 5000 lavori al mese, anche se il tempo di completamento aumenterebbe mediamente del 20 per cento.

a) Individuate gli argomenti a favore dell'installazione dell'algoritmo per evitare le situazioni di stallo.
b) Individuate gli argomenti contrari all'installazione dell'algoritmo per evitare le situazioni di stallo.

8.11 L'algoritmo del banchiere per un singolo tipo di risorsa si può ottenere dall'algoritmo del banchiere generale, semplicemente riducendo di 1 la dimensione delle diverse matrici. Dimostrate con un esempio che lo schema del banchiere con più tipi di risorsa non si può realizzare applicando separatamente lo schema del singolo tipo di risorsa a ciascun tipo di risorsa.

8.12 Dite se un sistema può rilevare se qualcuno dei suoi processi è in uno stato d'attesa indefinita. Nel caso affermativo spiegate com'è possibile che ciò avvenga, altrimenti spiegate in che modo il sistema può gestire il problema dell'attesa indefinita.

8.13 Considerate la seguente 'istantanea' di un sistema:

	Assegnate	*Massimo*	*Disponibili*
	A B C D	*A B C D*	*A B C D*
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

impiegando l'algoritmo del banchiere rispondete alle seguenti domande:

a) dite qual è il contenuto della matrice *Necessità*;
b) dite se il sistema è in uno stato sicuro;
c) dite se a fronte di una richiesta per (0, 4, 2, 0), fatta dal processo P_1, la richiesta può essere soddisfatta immediatamente.

8.14 Considerate il seguente criterio di assegnazione di risorse. Le richieste e i rilasci di risorse sono ammessi in qualsiasi momento. Se una richiesta di risorse non si può soddisfare immediatamente poiché le risorse non sono disponibili, si controllano tutti i processi bloccati nell'attesa di risorse. Se questi processi posseggono le risorse desiderate, si sottraggono tali risorse per assegnarle al processo richiedente. Il vettore delle risorse attese dal processo viene incrementato per comprendere le risorse sottratte.

Si consideri, ad esempio, un sistema con tre tipi di risorsa e il vettore *Disponibili* inizializzato a (4, 2, 2). Se il processo P_0 richiede (2, 2, 1), le ottiene. Se P_1 richiede (1, 0, 1) le ottiene; quindi, se P_0 richiede (0, 0, 1), viene bloccato (risorsa non disponibile). A questo punto, se P_2 chiede (2, 0, 0), ottiene la risorsa dispo-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nibile (1, 0, 0) e quella che era assegnata a P_0 (poiché P_0 è bloccato). Il vettore *Assegnate* di P_0 decresce a (1, 2, 1) e il suo vettore *Necessità* cresce a (1, 0, 1).

a) Stabilite se si può verificare uno stallo. Nel caso affermativo fornite un esempio, altrimenti dite quale condizione necessaria non può verificarsi.
b) Stabilite se si può verificare un blocco indefinito.

8.15 Supponete di aver codificato l'algoritmo di sicurezza per evitare le situazioni di stallo, ma che ora sia richiesta la realizzazione dell'algoritmo di rilevamento di situazioni di stallo. Dite se è sufficiente utilizzare semplicemente il codice dell'algoritmo di verifica della sicurezza e ridefinire $Massimo_i = Attesa_i + Assegnate_i$, dove $Attesa_i$ è un vettore che specifica le risorse che sta attendendo il processo i, mentre $Assegnate_i$ è com'è definito nel Paragrafo 8.5. Spiegate la risposta.

8.10 Note bibliografiche

[Dijkstra 1965a] è stato uno dei primi e più influenti studiosi della materia delle situazioni di stallo. [Holt 1972] è stato il primo a formalizzare la nozione di stallo con un modello di teoria dei grafi simile a quello presentato in questo capitolo. L'attesa indefinita è trattata in [Holt 1972]. [Hyman 1985] presenta l'esempio di situazione di stallo tratto dalla legislazione del Kansas.

I diversi algoritmi di prevenzione sono stati suggeriti da [Havender 1968], che ha escogitato lo schema di ordinamento delle risorse per il sistema IBM OS/360.

L'algoritmo del banchiere, per evitare le situazioni di stallo, è stato sviluppato per un singolo tipo di risorsa da [Dijkstra 1965a], ed è stato esteso ai tipi di risorsa multipli da [Habermann 1969]. Gli Esercizi 8.8 e 8.9 sono tratti da [Holt 1971].

L'algoritmo di rilevamento di situazioni di stallo per istanze multiple di un tipo di risorsa, descritto nel Paragrafo 8.6.2, è presentato in [Coffman et al. 1971].

[Bach 1987] descrive quanti tra gli algoritmi del nucleo dello UNIX tradizionale gestiscono le situazioni di stallo.

Parte terza

Gestione della memoria

Gestione della memoria
Memoria virtuale
Interfaccia del file system
Realizzazione del file system

Lo scopo principale di un sistema di calcolo è eseguire programmi; durante l'esecuzione, i programmi e i dati di cui questi necessitano devono trovarsi almeno parzialmente nella memoria centrale.

Per migliorare l'utilizzo della CPU e la velocità con cui questa risponde ai suoi utenti, il calcolatore deve tenere nella memoria parecchi processi. Esistono molti metodi di gestione della memoria e l'efficacia di ciascun algoritmo dipende dalla situazione. La scelta di un metodo di gestione della memoria, per un sistema specifico, dipende da molti fattori, in particolar modo dall'*architettura* del sistema; ogni algoritmo richiede infatti specifiche caratteristiche dell'architettura del sistema.

Poiché la memoria centrale è in genere troppo piccola per contenere in modo permanente tutti i dati e tutti i programmi, il calcolatore deve disporre di una memoria secondaria a sostegno della memoria centrale. I calcolatori moderni impiegano i dischi come mezzo primario di registrazione delle informazioni, cioè programmi e dati. Il file system fornisce i meccanismi sia per l'accesso ai dati e ai programmi residenti nei dischi sia per la loro registrazione. Un file è una raccolta d'informazioni tra loro correlate definite dal suo creatore. Il sistema operativo gestisce la corrispondenza tra i file e i dispositivi che li contengono fisicamente; i file sono normalmente organizzati in directory che ne facilitano l'uso.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

3 metodi
- Ribaltamento
- Paginazione
- Segmentazione

Capitolo 9

Gestione della memoria

=> Scopo: avere in memoria il maggior numero di processi

Un sistema monoprogrammato
Nella sua memoria è divisa in 2 parti:
- Una parte x il S.O.
- Una parte x il prog che è in esecuzione in quel momento

Deve essere suddivisa x consentire la presenza di più processi possibili

Tale suddivisione è fatta dinamicamente dal S.O. ed è detta GESTIONE DELLA MEMORIA.

- Affinché la memoria sia assegnata efficientemente deve contenere molti processi.

Nel Capitolo 6 si spiega come sia possibile condividere la CPU tra un insieme di processi. Uno dei risultati dello scheduling della CPU consiste nella possibilità di migliorare sia l'utilizzo della CPU sia la rapidità con cui il calcolatore risponde ai propri utenti; per ottenere questo aumento delle prestazioni occorre tenere nella memoria parecchi processi: la memoria deve, cioè, essere *condivisa*.

In questo capitolo si discutono i diversi metodi di gestione della memoria. Gli algoritmi di gestione della memoria variano dal metodo iniziale sulla nuda macchina ai metodi di paginazione e segmentazione. Ogni metodo presenta vantaggi e svantaggi. La scelta di un metodo specifico di gestione della memoria dipende da molti fattori, in particolar modo dall'architettura del sistema, infatti molti algoritmi richiedono specifiche caratteristiche dell'architettura; progetti recenti integrano in modo molto stretto l'architettura e il sistema operativo.

9.1 Introduzione

Com'è spiegato nel Capitolo 1, la memoria è fondamentale nelle operazioni di un moderno sistema di calcolo; consiste in un ampio vettore di parole o byte, ciascuno con il proprio indirizzo. La CPU preleva le istruzioni dalla memoria sulla base del contenuto del contatore di programma; tali istruzioni possono determinare ulteriori letture (load) e scritture (store) in specifici indirizzi della memoria.

Un tipico ciclo d'esecuzione di un'istruzione, ad esempio, prevede che l'istruzione sia prelevata dalla memoria; decodificata (e ciò può comportare il prelievo di operandi dalla memoria) ed eseguita sugli eventuali operandi; i risultati si possono salvare nella memoria. La memoria 'vede' soltanto un flusso d'indirizzi di memoria, e non 'sa' come sono generati (contatore di programma, indicizzazione, riferimenti indiretti, indirizzamenti immediati e così via), oppure a cosa servano (istruzioni o dati). Di conseguenza, è possibile ignorare *come* un programma genera un indirizzo di memoria, e prestare attenzione solo alla sequenza degli indirizzi di memoria generati dal programma in esecuzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software

gli indirizzi che poniamo al Gezione sono quelli della memoria principale.

- Un processo deve poterci muovere avanti ed indietro.

9.1.1 Associazione degli indirizzi

In genere un programma risiede in un disco in forma di un file binario eseguibile. Il programma per essere eseguito deve essere caricato nella memoria e inserito all'interno di un processo. Secondo il tipo di gestione della memoria adoperato, durante la sua esecuzione, il processo si può trasferire dalla memoria al disco e viceversa. L'insieme dei processi presenti nei dischi e che attendono d'essere trasferiti nella memoria per essere eseguiti forma la **coda d'ingresso.**

La procedura normale consiste nello scegliere uno dei processi appartenenti alla coda d'ingresso e caricarlo nella memoria. Il processo durante l'esecuzione può accedere alle istruzioni e ai dati nella memoria. Quando il processo termina, si dichiara disponibile il suo spazio di memoria.

La maggior parte dei sistemi consente ai processi utenti di risiedere in qualsiasi parte della memoria fisica, quindi, anche se lo spazio d'indirizzi del calcolatore comincia all'indirizzo 00000, il primo indirizzo del processo utente non deve necessariamente essere 00000. Quest'assetto influisce sugli indirizzi che un programma utente può usare. Nella maggior parte dei casi un programma utente, prima di essere eseguito, deve passare attraverso vari stadi — alcuni possono essere facoltativi — (Figura 9.1) in cui gli indirizzi possono essere rappresentati in modi diversi. Generalmente gli indirizzi del programma sorgente sono simbolici (ad esempio, contatore). Un compilatore di solito associa (*bind*) questi indirizzi simbolici a indirizzi rilocabili (ad esempio, '14 byte dall'inizio di questo modulo'). L'editor dei collegamenti (*linkage editor*), o il caricatore (*loader*), fa corrispondere a sua volta questi indirizzi rilocabili a indirizzi assoluti (ad esempio, 74014). Ogni associazione rappresenta una corrispondenza da uno spazio d'indirizzi a un altro.

Generalmente, l'associazione di istruzioni e dati a indirizzi di memoria si può compiere in qualsiasi fase del seguente percorso:

In questa fase non sappiamo dove risiederà il codice di memoria

- **Compilazione.** Se nella fase di compilazione si sa dove il processo risiederà nella memoria, si può generare **codice assoluto.** Se, ad esempio, è noto a priori che un processo utente comincia alla locazione *r*, anche il codice generato dal compilatore comincia da quella locazione. Se, in un momento successivo, la locazione iniziale cambiasse, sarebbe necessario ricompilare il codice. I programmi per l'MS-DOS nel formato identificato dall'estensione .com sono codice assoluto generato nella fase di compilazione.

All'atto di caricamento vado a specificare i veri indirizzi

- **Caricamento.** Se nella fase di compilazione non è possibile sapere in che punto della memoria risiederà il processo, il compilatore deve generare **codice rilocabile.** In questo caso si ritarda l'associazione finale degli indirizzi alla fase del caricamento. Se l'indirizzo iniziale cambia, è sufficiente ricaricare il codice d'utente per incorporare il valore modificato.

- **Esecuzione.** Se durante l'esecuzione il processo può essere spostato da un segmento di memoria a un altro, si deve ritardare l'associazione degli indirizzi fino alla fase d'esecuzione. Per realizzare questo schema sono necessarie specifiche caratteristiche dell'architettura; questo argomento è trattato nel Paragrafo 9.1.2. La maggior parte dei sistemi operativi d'uso generale impiega questo metodo.

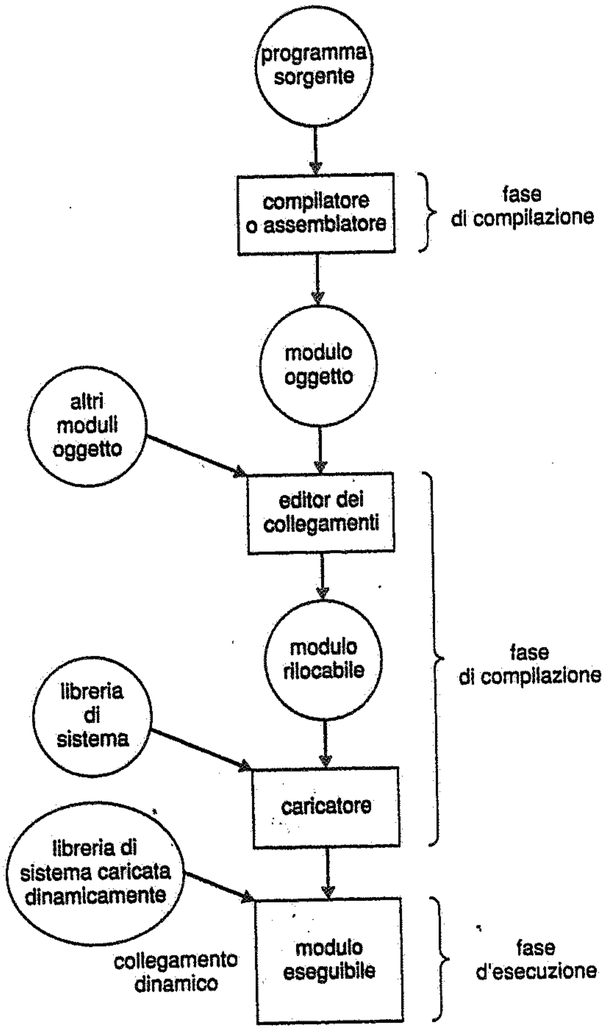


Figura 9.1 Fasi di elaborazione per un programma utente.

Una gran parte di questo capitolo è dedicata alla spiegazione di come i vari tipi di associazione degli indirizzi si possano realizzare efficacemente in un calcolatore e, inoltre, alla discussione delle caratteristiche dell'architettura appropriate alla realizzazione di queste funzioni.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.1.2 Spazi di indirizzi logici e fisici

Un indirizzo generato dalla CPU di solito si indica come **indirizzo logico**, mentre un indirizzo visto dall'unità di memoria, cioè caricato nel **registro dell'indirizzo di memoria** (*memory address register* — MAR) di solito si indica come **indirizzo fisico**.

I metodi di associazione degli indirizzi nelle fasi di compilazione e di caricamento producono indirizzi logici e fisici identici. Con i metodi di associazione nella fase d'esecuzione, invece, gli indirizzi logici non coincidono con gli indirizzi fisici. In questo caso ci si riferisce, di solito, agli indirizzi logici col termine **indirizzi virtuali**; in questo testo si usano tali termini in modo intercambiabile. L'insieme di tutti gli indirizzi logici generati da un programma è lo **spazio degli indirizzi logici**; l'insieme degli indirizzi fisici corrispondenti a tali indirizzi logici è lo **spazio degli indirizzi fisici**. Quindi, con lo schema di associazione degli indirizzi nella fase d'esecuzione, lo spazio degli indirizzi logici differisce dallo spazio degli indirizzi fisici.

L'associazione nella fase d'esecuzione degli indirizzi virtuali agli indirizzi fisici è svolta da un dispositivo detto **unità di gestione della memoria** (*memory-management unit* — MMU). Come si discute nei Paragrafi dal 9.3 al 9.6, si può scegliere tra diversi metodi di realizzazione di tale associazione; di seguito s'illustra un semplice schema di associazione degli indirizzi che impiega una MMU; si tratta di una generalizzazione dello schema con registro di base descritto nel Paragrafo 2.5.3.

Com'è illustrato nella Figura 9.2, questo metodo richiede un'architettura leggermente diversa da quella descritta nel Paragrafo 2.5.3. Il registro di base è ora denominato **registro di rilocazione**: quando un processo utente genera un indirizzo, prima dell'invio all'unità di memoria, si *somma* a tale indirizzo il valore contenuto nel registro di rilocazione. Ad esempio, se il registro di rilocazione contiene il valore 14.000, un tentativo da parte dell'utente di accedere alla locazione 0 è dinamicamente rilocato alla locazione 14.000; un accesso alla locazione 346 corrisponde alla locazione 14.346. Quando il sistema operativo MS-DOS, eseguito sulla famiglia di CPU Intel 80*x*86, carica ed esegue processi impiega quattro registri di rilocazione.

Il programma utente non considera mai gli indirizzi fisici *reali*. Il programma crea un puntatore alla locazione 346, lo memorizza, lo modifica, lo confronta con altri indirizzi, tutto ciò semplicemente come un numero. Solo quando assume il ruolo di un indirizzo di memoria (magari in una `load` o una `store` indiretta), si riloca il numero sulla base del contenuto del registro di rilocazione. Il programma utente tratta indirizzi *logici*, l'architettura del sistema converte gli indirizzi logici in indirizzi *fisici*. Il collegamento nella fase d'esecuzione è trattato nel Paragrafo 9.1.1. La locazione finale di un riferimento a un indirizzo di memoria non è determinata finché non si compie effettivamente il riferimento.

In questo caso esistono due diversi tipi di indirizzi: gli indirizzi logici (nell'intervallo da 0 a *max*) e gli indirizzi fisici (nell'intervallo da $r + 0$ a $r + max$ per un valore di base r). L'utente genera solo indirizzi logici e 'pensa' che il processo sia eseguito nelle posizioni da 0 a *max*. Il programma utente fornisce indirizzi logici che, prima d'essere usati, si devono far corrispondere a indirizzi fisici.

Il concetto di *spazio d'indirizzi logici* associato a uno *spazio d'indirizzi fisici* separato è fondamentale per una corretta gestione della memoria.

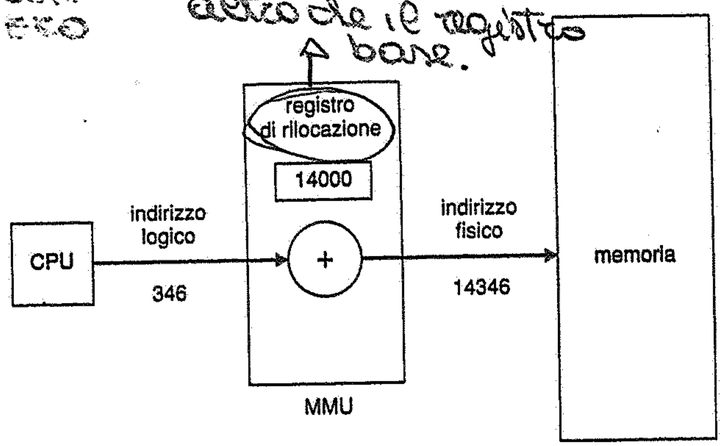


Figura 9.2 Rilocazione dinamica tramite un registro di rilocazione.

9.1.3 Caricamento dinamico

Per migliorare l'utilizzo della memoria si può ricorrere al **caricamento dinamico** (*dynamic loading*), mediante il quale si carica una procedura solo quando viene richiamata; tutte le procedure si tengono nella memoria secondaria in un formato di caricamento rilocabile. Si carica il programma principale nella memoria e quando, durante l'esecuzione, una procedura deve chiamare un'altra procedura, si controlla innanzi tutto che sia stata caricata, altrimenti si chiama il caricatore di collegamento rilocabile per caricare nella memoria la procedura richiesta e aggiornare le tabelle degli indirizzi del programma in modo che registrino questo cambiamento. A questo punto il controllo passa alla procedura appena caricata.

Il vantaggio dato dal caricamento dinamico consiste nel fatto che una procedura che non si adopera non viene caricata. Questo metodo è utile soprattutto quando servono grandi quantità di codice per gestire casi non frequenti, ad esempio le procedure di gestione degli errori. In questi casi, anche se un programma può avere dimensioni totali elevate, la parte effettivamente usata, e quindi effettivamente caricata, può essere molto più piccola.

Il caricamento dinamico non richiede un intervento particolare del sistema operativo. Spetta agli utenti progettare i programmi in modo da trarre vantaggio da un metodo di questo tipo. Il sistema operativo può comunque aiutare il programmatore fornendo librerie di procedure che realizzano il caricamento dinamico.

9.1.4 Collegamento dinamico e librerie condivise

La Figura 9.1 mostra anche le **librerie collegate dinamicamente**. Alcuni sistemi operativi consentono solo il **collegamento statico**, in cui le librerie di sistema del linguaggio sono trattate come qualsiasi altro modulo oggetto e sono combinate dal caricatore nell'immagine binaria del programma. Il concetto di collegamento dinamico è analogo a quello di ca-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ricamento dinamico. Invece di differire il caricamento di una procedura fino al momento dell'esecuzione, si differisce il collegamento. Questa caratteristica si usa soprattutto con le librerie di sistema, ad esempio le librerie di procedure del linguaggio. Senza questo strumento tutti i programmi di un sistema dovrebbero disporre all'interno dell'immagine eseguibile di una copia della libreria di linguaggio (o almeno delle procedure cui il programma fa riferimento). Tutto ciò richiede spazio nei dischi e nella memoria centrale. Con il collegamento dinamico, invece, per ogni riferimento a una procedura di libreria s'inserisce all'interno dell'immagine eseguibile una piccola porzione di codice di riferimento (*stub*), che indica come localizzare la giusta procedura di libreria residente nella memoria o come caricare la libreria se la procedura non è già presente.

Durante l'esecuzione, il codice di riferimento controlla se la procedura richiesta è già nella memoria, altrimenti provvede a caricarla; in ogni caso tale codice sostituisce se stesso con l'indirizzo della procedura, che viene poi eseguita. In questo modo quando si raggiunge nuovamente quel segmento del codice, si esegue direttamente la procedura di libreria, senza costi aggiuntivi per il collegamento dinamico. Con questo metodo tutti i processi che usano una libreria del linguaggio si limitano a eseguire la stessa copia del codice della libreria.

Questa caratteristica si può estendere anche agli aggiornamenti delle librerie, ad esempio, la correzione di errori. Una libreria si può sostituire con una sua nuova versione, e tutti i programmi che fanno riferimento a quella libreria usano automaticamente la nuova versione. Senza il collegamento dinamico tutti i programmi di questo tipo devono subire una nuova fase di collegamento per accedere alla nuova libreria. Affinché i programmi non eseguano accidentalmente nuove versioni di librerie incompatibili, sia nel programma sia nella libreria si inserisce un'informazione relativa alla versione. È possibile caricare nella memoria più di una versione della stessa libreria, ciascun programma si serve dell'informazione sulla versione per decidere quale copia debba usare. Se le modifiche sono di piccola entità, il numero di versione resta invariato; se l'entità delle modifiche diviene rilevante, si aumenta anche il numero di versione. Perciò, solo i programmi compilati con la nuova versione della libreria subiscono gli effetti delle modifiche incompatibili incorporate nella libreria stessa. I programmi collegati prima dell'installazione della nuova libreria continuano ad avvalersi della vecchia libreria. Questo sistema è noto anche con il nome di **librerie condivise**.

A differenza del caricamento dinamico, il collegamento dinamico richiede generalmente l'assistenza del sistema operativo. Se i processi presenti nella memoria sono protetti l'uno dall'altro (Paragrafo 9.3.1), il sistema operativo è l'unica entità che può controllare se la procedura richiesta da un processo è nello spazio di memoria di un altro processo, o che può consentire l'accesso di più processi agli stessi indirizzi di memoria. Questo concetto è sviluppato nel contesto della paginazione, nel Paragrafo 9.4.5.

9.1.5 Sovrapposizione di sezioni

Per permettere a un processo di essere più grande della memoria che gli si assegna, si può usare una tecnica chiamata **sovrapposizione di sezioni** (*overlay*). Il concetto della sovrapposizione di sezioni si fonda sulla possibilità di mantenere nella memoria soltanto le

istruzioni e i dati che si usano con maggior frequenza. Quando sono necessarie altre istruzioni, queste si caricano nello spazio precedentemente occupato dalle istruzioni che non sono più in uso.

Un esempio è quello dell'assemblatore a due passi. Durante il passo 1, l'assemblatore costruisce una tabella dei simboli; durante il passo 2, genera il codice nel linguaggio di macchina. Un siffatto assemblatore si può suddividere in quattro parti: codice del passo 1; codice del passo 2; tabella dei simboli; procedure ausiliarie, comuni al passo 1 e al passo 2. Si supponga che le dimensioni di queste parti siano le seguenti:

Passo 1	70 KB
Passo 2	80 KB
Tabella dei simboli	20 KB
Procedure comuni	30 KB

Per caricare tutto il programma nella memoria sono necessari 200 KB. Se sono disponibili solo 150 KB, il processo non può essere eseguito. Tuttavia, per eseguire il processo non è necessario che il passo 1 e il passo 2 si trovino contemporaneamente nella memoria. È possibile definire due sezioni: la sezione *A* è costituita dalla tabella dei simboli, dalle procedure comuni e dal passo 1; la sezione *B* è costituita dalla tabella dei simboli, dalle procedure comuni e dal passo 2.

Si aggiunge un gestore della sovrapposizione delle sezioni (10 KB), e s'inizia il processo con la sezione *A* nella memoria. Terminato il passo 1, il controllo passa al gestore della sovrapposizione delle sezioni che legge la sezione *B* nella memoria, sovrapponendola alla sezione *A*, quindi trasferisce il controllo al passo 2. La sezione *A* necessita di soli 120 KB, mentre la sezione *B* ne richiede 130, com'è evidenziato nella Figura 9.3. A questo punto l'assemblatore si può eseguire nei 150 KB di memoria disponibili. Esso sarà inoltre caricato più rapidamente poiché prima dell'avvio dell'esecuzione si devono trasferire meno dati. Tuttavia, la sua esecuzione è rallentata dall'operazione di I/O necessaria per leggere il codice della sezione *B* da sovrapporre al codice della sezione *A*.

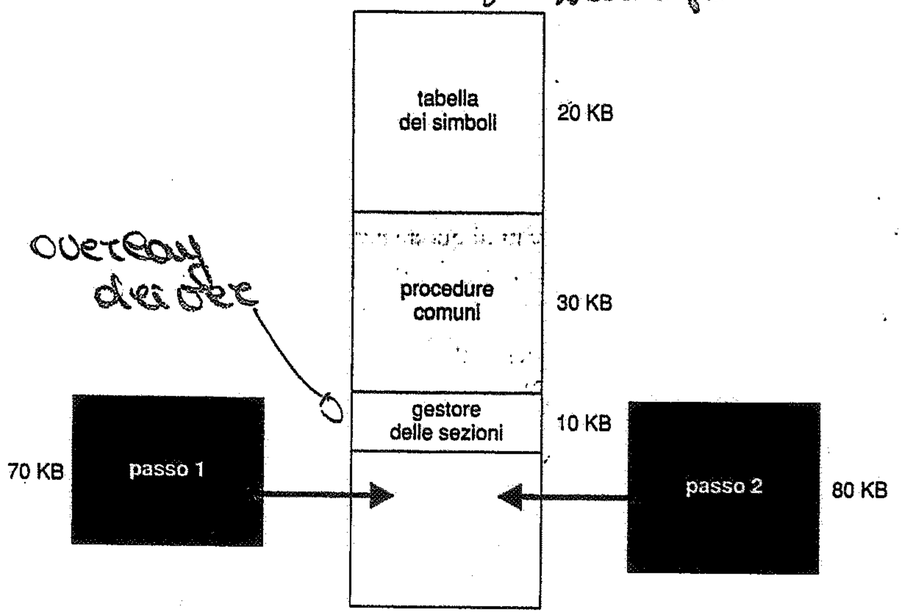
Il codice per la sezione *A* e il codice per la sezione *B* risiedono nella memoria secondaria come immagini di memoria in formato assoluto, e sono letti dal gestore della sovrapposizione delle sezioni secondo le necessità. Per gestire la sovrapposizione delle sezioni sono necessari speciali algoritmi di rilocazione e collegamento.

Questa tecnica non richiede alcun intervento speciale del sistema operativo, ma può essere realizzata direttamente dall'utente per mezzo di semplici strutture di file, copiandone il contenuto nella memoria e quindi trasferendo il controllo a quest'ultima per eseguire le istruzioni appena lette. Il sistema operativo si limita ad annotare la presenza di operazioni di I/O supplementari.

Il programmatore, quindi, deve progettare e programmare adeguatamente la struttura delle sezioni. Questo compito può essere un'impresa considerevole, poiché richiede una conoscenza minuziosa della struttura del programma, del suo codice e delle sue strutture di dati. Poiché, per definizione, il programma è di grandi dimensioni — programmi piccoli non richiedono la sovrapposizione di sezioni — può essere difficile comprenderne sufficientemente la struttura.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

corrispondenza tra le varie funzioni.



overlay dei dati

Figura 9.3 Sovrapposizione di sezioni per un assemblatore a due passi.

Per questi motivi, l'uso della sovrapposizione di sezioni è attualmente limitato ai microcalcolatori e ad altri sistemi dotati di quantità di memoria limitate, e la cui architettura non offre le caratteristiche necessarie alla realizzazione di tecniche più progredite. Alcuni compilatori per microcalcolatori gestiscono la sovrapposizione di sezioni per semplificare il compito dei programmatori. Per l'esecuzione di lunghi programmi in quantità limitate di memoria fisica, sono comunque certamente preferibili le tecniche automatiche di gestione.

9.2 Avvicendamento dei processi

Per essere eseguito, un processo deve trovarsi nella memoria centrale, ma si può trasferire temporaneamente nella memoria ausiliaria (*backing store*) da cui si riporta nella memoria centrale al momento di riprenderne l'esecuzione. Si consideri, ad esempio, un ambiente di multiprogrammazione con un algoritmo circolare (*round-robin*) per lo scheduling della CPU. Trascorso un quanto di tempo, il gestore di memoria scarica dalla memoria il processo appena terminato e carica un altro processo nello spazio di memoria appena liberato; questo procedimento è illustrato nella Figura 9.4 e si chiama avvi-

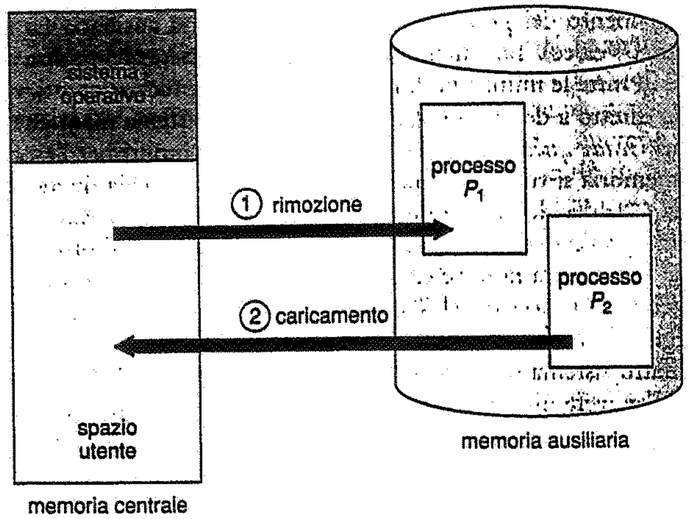


Figura 9.4 Avvicendamento di due processi con un disco come memoria ausiliaria.

si chiama così se utilizziamo un algoritmo RR

cendamento dei processi nella memoria — o, più brevemente, **avvicendamento** o **scambio** (*swapping*). Nel frattempo lo scheduler della CPU assegna un quanto di tempo a un altro processo presente nella memoria. Quando esaurisce il suo quanto di tempo, ciascun processo viene scambiato con un altro processo. In teoria il gestore della memoria può avvicendare i processi in modo sufficientemente rapido da far sì che alcuni processi siano presenti nella memoria, pronti per essere eseguiti, quando lo scheduler della CPU vuole riassegnare la CPU stessa. Anche il quanto di tempo deve essere sufficientemente lungo da permettere che un processo, prima d'essere sostituito, esegua quantità ragionevole di calcolo.

Una variante di questo criterio d'avvicendamento dei processi s'impiega per gli algoritmi di scheduling basati sulle priorità. Se si presenta un processo con priorità maggiore, il gestore della memoria può scaricare dalla memoria centrale il processo con priorità inferiore per fare spazio all'esecuzione del processo con priorità maggiore. Quando il processo con priorità maggiore termina, si può ricaricare nella memoria quello con priorità minore e continuare la sua esecuzione (questo tipo d'avvicendamento è talvolta chiamato *roll out, roll in*) se utilizziamo un algoritmo a priorità.

Normalmente, un processo che è stato scaricato dalla memoria si deve ricaricare nello spazio di memoria occupato in precedenza. Questa limitazione è dovuta al metodo di associazione degli indirizzi. Se l'associazione degli indirizzi logici agli indirizzi fisici della memoria si effettua nella fase di assemblaggio o di caricamento, il processo non può essere ricaricato in posizioni diverse. Se l'associazione degli indirizzi logici agli indirizzi fisici della memoria si compie nella fase d'esecuzione, un processo può essere riversato in uno spazio di memoria diverso, poiché gli indirizzi fisici si calcolano nella fase d'esecuzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'avvicendamento dei processi richiede una memoria ausiliaria (si tratta generalmente di un disco veloce). Tale memoria ausiliaria deve essere abbastanza ampia da contenere le copie di tutte le immagini di memoria di tutti i processi utenti, e deve permettere un accesso diretto a dette immagini di memoria. Il sistema mantiene una coda dei processi pronti (*ready queue*) formata da tutti i processi pronti per l'esecuzione, le cui immagini di memoria si trovano nella memoria ausiliaria o nella memoria. Quando lo scheduler della CPU decide di eseguire un processo, chiama il dispatcher, il quale controlla se il primo processo della coda si trova nella memoria. Se il processo non si trova nella memoria, e in questa non c'è spazio libero, il dispatcher scarica un processo dalla memoria e vi carica il processo richiesto dallo scheduler della CPU, quindi ricarica normalmente i registri e trasferisce il controllo al processo selezionato.

In un siffatto sistema d'avvicendamento, il tempo di cambio di contesto è abbastanza elevato. Per avere un'idea delle sue dimensioni si pensi a un processo utente di 1 MB e a una memoria ausiliaria costituita di un disco ordinario con velocità di trasferimento di 5 MB al secondo. Il trasferimento effettivo del processo di 1 MB da e nella memoria richiede

$$1 \text{ MB}/5 \text{ MB al secondo} = 1/5 \text{ secondo}$$
$$= 200 \text{ millisecondi}$$

Ipotizzando che non siano necessari movimenti della testina (*seek*) e che la latenza media sia di 8 millisecondi, l'operazione richiede 208 millisecondi. Poiché l'avvicendamento richiede lo scaricamento e il caricamento di un processo, il tempo totale è di circa 416 millisecondi.

Per utilizzare efficacemente la CPU è necessario che il tempo d'esecuzione di ciascun processo sia lungo rispetto al tempo d'avvicendamento. Ad esempio, in un algoritmo circolare di scheduling della CPU il quanto di tempo dovrebbe essere maggiore di 0,416 secondi.

Occorre notare che la maggior parte del tempo d'avvicendamento è data dal tempo di trasferimento. Il tempo di trasferimento totale è direttamente proporzionale alla *quantità* di memoria interessata. In un calcolatore con 128 MB di memoria centrale e un sistema operativo residente di 5 MB, la massima dimensione possibile per un processo utente è di 123 MB. Tuttavia possono essere presenti molti processi utenti con dimensione minore, ad esempio 1 MB. Lo scaricamento di un processo di 1 MB può concludersi in 208 millisecondi, mentre per lo scaricamento di 123 MB sono necessari 24,6 millisecondi. Perciò sarebbe utile sapere esattamente quanta memoria *è* effettivamente usata da un processo utente e non solo quanta questo *potrebbe* teoricamente usarne, poiché in questo caso è necessario scaricare solo quanto è effettivamente utilizzato, riducendo il tempo d'avvicendamento. Affinché questo metodo risulti efficace, l'utente deve tenere informato il sistema su tutte le modifiche apportate ai requisiti di memoria; un processo con requisiti di memoria dinamici deve impiegare chiamate del sistema (`request memory` e `release memory`) per informare il sistema operativo delle modifiche da apportare alla memoria.

L'avvicendamento dei processi è soggetto ad altri vincoli. Per scaricare un processo dalla memoria è necessario essere certi che il processo sia completamente inattivo. Particolare importanza ha qualsiasi I/O pendente: mentre si vuole scaricare un processo per liberare la memoria, tale processo può essere nell'attesa del completamento di un'operazione di I/O. Tuttavia, se un dispositivo di l'I/O accede in modo asincrono alle aree di I/O della memoria (*buffer*) d'utente, il processo non può essere scaricato. Si supponga che l'operazione di I/O sia stata accodata perché il dispositivo era occupato. Se il processo P_2 s'avvicendasse al processo P_1, l'operazione di I/O potrebbe tentare di usare la memoria che attualmente appartiene al processo P_2. Questo problema si può risolvere in due modi: non scaricare dalla memoria un processo con operazioni di I/O pendenti, oppure eseguire operazioni di I/O solo in aree di memoria per l'I/O del sistema operativo. In questo modo i trasferimenti fra tali aree del sistema operativo e la memoria assegnata al processo possono avvenire solo quando il processo è presente nella memoria centrale.

L'ipotesi che l'avvicendamento dei processi richieda pochi o nessun movimento delle testine dei dischi merita ulteriori spiegazioni (tale argomento è discusso nel Capitolo 14, che tratta la struttura della memoria secondaria). Affinché il suo uso sia quanto più veloce è possibile, generalmente si assegna l'area d'avvicendamento in una porzione di disco separata da quella riservata al file system.

Attualmente l'avvicendamento semplice si usa in pochi sistemi; esso richiede infatti un elevato tempo di gestione, e consente un tempo di esecuzione troppo breve per essere considerato una soluzione ragionevole al problema di gestione della memoria. Versioni modificate dell'avvicendamento dei processi si trovano comunque in molti sistemi.

Una sua forma modificata era, ad esempio, usata in molte versioni dello UNIX: normalmente era disabilitato e si avviava solo nel caso in cui molti processi si fossero trovati in esecuzione superando una quantità limite di memoria. L'avvicendamento dei processi sarebbe stato di nuovo sospeso qualora il carico del sistema fosse diminuito.

I primi PC non disponevano di un'architettura adatta o di un sistema operativo che ne traesse vantaggio per realizzare metodi di gestione della memoria più progrediti; si usavano però per eseguire più processi di elevate dimensioni con una versione modificata dell'avvicendamento dei processi. Un esempio importante è dato dal sistema operativo Windows 3.1 della Microsoft che consente l'esecuzione concorrente di più processi presenti nella memoria. Se si carica un nuovo processo, ma lo spazio disponibile nella memoria centrale è insufficiente, si trasferisce nel disco un processo già presente nella memoria centrale. Non si tratta di vero e proprio avvicendamento dei processi poiché è l'utente, e non lo scheduler, che decide il momento in cui scaricare un processo in favore di un altro; inoltre, ciascun processo scaricato resta in tale stato fino a quando sarà selezionato, per l'esecuzione, dall'utente. Il sistema operativo Windows NT, sfrutta le MMU attualmente disponibili anche nei PC. Nel Paragrafo 9.6 si descrive l'architettura per la gestione della memoria della famiglia di CPU Intel 80386 usata in molti PC, inoltre si descrive la gestione della memoria impiegata in tali CPU da un altro moderno sistema operativo per PC: l'IBM OS/2.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.3 Assegnazione contigua della memoria

La memoria centrale deve contenere sia il sistema operativo sia i vari processi utenti; perciò è necessario assegnare le diverse parti della memoria centrale nel modo più efficiente possibile. In questo paragrafo si tratta l'assegnazione contigua della memoria.

La memoria centrale di solito si divide in due partizioni, una per il sistema operativo residente e una per i processi utenti. Il sistema operativo si può collocare sia nella memoria bassa sia nella memoria alta. Il fattore che incide in modo decisivo su tale scelta è generalmente la posizione del vettore delle interruzioni. Poiché questo si trova spesso nella memoria bassa, i programmatori di solito collocano anche il sistema operativo nella memoria bassa.

Di solito si vuole che più processi utenti risiedano contemporaneamente nella memoria centrale. Perciò è necessario considerare come assegnare la memoria disponibile ai processi presenti nella coda d'ingresso che attendono di essere caricati nella memoria. Con l'assegnazione contigua della memoria, ciascun processo è contenuto in una singola sezione contigua della memoria.

9.3.1 Protezione della memoria

Prima di trattare l'assegnazione della memoria, si deve discutere la questione della sua protezione: la protezione del sistema operativo dai processi utenti, e la protezione dei processi utenti dagli altri processi utenti. Tale protezione si può realizzare usando un registro di rilocazione, come si descrive nel Paragrafo 9.1.2, con un registro di limite, come si descrive nel Paragrafo 2.5.3. Il registro di rilocazione contiene il valore dell'indirizzo fisico minore; il registro di limite contiene l'intervallo di indirizzi logici, ad esempio, *rilocazione* = 100.040 e *limite* = 74.600. Con i registri di rilocazione e di limite, ogni indirizzo logico deve essere minore del contenuto del registro di limite; la MMU fa corrispondere *dinamicamente* l'indirizzo fisico all'indirizzo logico sommando a quest'ultimo il valore contenuto nel registro di rilocazione (Figura 9.5).

Quando lo scheduler della CPU seleziona un processo per l'esecuzione, il dispatcher, durante l'esecuzione del cambio di contesto, carica il registro di rilocazione e il registro di limite con i valori corretti. Poiché si confronta ogni indirizzo generato dalla CPU con i valori contenuti in questi registri, si possono proteggere il sistema operativo e i programmi e i dati di altri utenti da tentativi di modifiche da parte del processo in esecuzione.

Lo schema con registro di rilocazione consente al sistema operativo di cambiare dinamicamente le proprie dimensioni. Tale flessibilità è utile in molte situazioni; il sistema operativo, ad esempio, contiene codice e spazio di memoria per i driver dei dispositivi; se uno di questi, o un altro servizio del sistema operativo, non è comunemente usato, è inutile tenerne nella memoria codice e dati, poiché lo spazio occupato si potrebbe usare per altri scopi. Talvolta questo codice si chiama codice **transiente** del sistema operativo, poiché s'inserisce secondo le necessità; l'uso di tale codice cambia le dimensioni del sistema operativo durante l'esecuzione del programma.

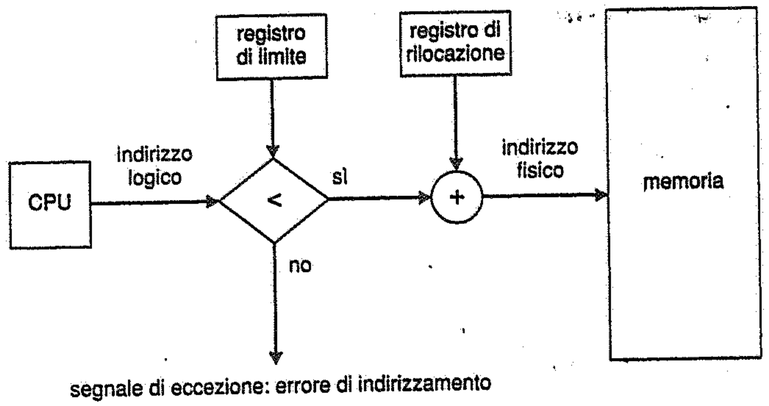


Figura 9.5 Registri di rilocazione e di limite.

9.3.2 Assegnazione della memoria

Uno dei metodi più semplici per l'assegnazione della memoria consiste nel dividere la stessa in **partizioni** di dimensione fissa. Ogni partizione deve contenere esattamente un processo, quindi il grado di multiprogrammazione è limitato dal numero di partizioni. Con questo metodo quando una partizione è libera può essere occupata da un processo presente nella coda d'ingresso; terminato il processo, la partizione diviene nuovamente disponibile per un altro processo. Originariamente questo metodo, detto MFT, si usava nel sistema operativo IBM OS/360, ma attualmente non è più in uso. Il metodo descritto di seguito, detto MVT, è una generalizzazione del metodo con partizioni fisse e si usa soprattutto in ambienti d'elaborazione a lotti. Si noti, comunque, che molte idee a esso relative si possono applicare agli ambienti a partizione del tempo d'elaborazione nei quali si fa uso della segmentazione semplice per la gestione della memoria (Paragrafo 9.5).

Il sistema operativo conserva una tabella nella quale sono indicate le partizioni di memoria disponibili e quelle occupate. Inizialmente tutta la memoria è a disposizione dei processi utenti; si tratta di un grande blocco di memoria disponibile, un **buco**. Quando si carica un processo che necessita di memoria, occorre cercare un buco sufficientemente grande da contenerlo. Se ne esiste uno si assegna solo la parte di memoria necessaria al processo; la parte rimanente si usa per soddisfare eventuali richieste future.

Quando entrano nel sistema, i processi vengono inseriti in una coda d'ingresso. Per determinare a quali processi si debba assegnare la memoria, il sistema operativo tiene conto dei requisiti di memoria di ciascun processo e della quantità di spazio di memoria disponibile. Quando a un processo si assegna dello spazio, il processo stesso viene caricato nella memoria e può quindi competere per il controllo della CPU. Al termine, rilascia la memoria che gli era stata assegnata, e il sistema operativo può impiegarla per un altro processo presente nella coda d'ingresso.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

È sempre disponibile un elenco delle dimensioni dei blocchi liberi e della coda d'ingresso. Il sistema operativo può ordinare la coda d'ingresso secondo un algoritmo di scheduling. La memoria si assegna ai processi della coda finché si possono soddisfare i requisiti di memoria del processo successivo, cioè finché esiste un blocco di memoria (o buco) disponibile che sia sufficientemente grande da accogliere quel processo. Il sistema operativo può quindi attendere che si renda disponibile un blocco sufficientemente grande, oppure può scorrere la coda d'ingresso per verificare se è possibile soddisfare le richieste di memoria più limitate di qualche altro processo.

In generale, è sempre presente un *insieme* di buchi di diverse dimensioni sparsi per la memoria. Quando si presenta un processo che necessita di memoria, il sistema cerca nel gruppo un buco di dimensioni sufficienti per contenerlo. Se è troppo grande, il buco viene diviso in due parti: si assegna una parte al processo in arrivo e si riporta l'altra nell'insieme dei buchi. Quando termina, un processo rilascia il blocco di memoria, che si reinserisce nell'insieme dei buchi; se si trova accanto ad altri buchi, si uniscono tutti i buchi adiacenti per formarne uno più grande. A questo punto il sistema deve controllare se vi siano processi nell'attesa di spazio di memoria, e se la memoria appena liberata e ricombinata possa soddisfare le richieste di qualcuno fra tali processi.

Questa procedura è una particolare istanza del più generale problema di **assegnazione dinamica della memoria**, che consiste nel soddisfare una richiesta di dimensione *n* data una lista di buchi liberi. Le soluzioni sono numerose. Si esamina il gruppo di buchi per stabilire quale sia quello migliore per l'assegnazione. I criteri più usati per scegliere un buco libero tra quelli disponibili nell'insieme sono i seguenti:

- Si assegna il **primo** buco abbastanza grande. La ricerca può cominciare sia dall'inizio dell'insieme di buchi sia dal punto in cui era terminata la ricerca precedente. Si può fermare la ricerca non appena s'individua un buco libero di dimensioni sufficientemente grandi. Questo criterio è noto come *first-fit*.
- Si assegna il **più piccolo** buco in grado di contenere il processo. Si deve compiere la ricerca in tutta la lista, sempre che questa non sia ordinata per dimensione. Questo criterio produce le parti di buco inutilizzate più piccole, ed è noto come *best-fit*.
- Si assegna il **buco più grande**. Anche in questo caso si deve esaminare tutta la lista, sempre che questa non sia ordinata per dimensione. Questo criterio produce le parti di buco inutilizzate più grandi, che possono essere più utili delle parti più piccole ottenute col criterio precedente, ed è noto come *worst-fit*.

Con l'uso di simulazioni si è dimostrato che i criteri di scelta del primo buco abbastanza grande e del buco più piccolo dànno risultati migliori del criterio di scelta del buco più grande, poiché riducono il tempo e l'utilizzo della memoria; d'altra parte nessuno dei due è chiaramente migliore dell'altro per quel che riguarda l'utilizzo della memoria, ma, in genere, la scelta del primo buco richiede meno tempo.

Questi algoritmi soffrono di **frammentazione esterna**: quando si caricano e si rimuovono i processi dalla memoria, si frammenta lo spazio libero della memoria in tante piccole parti. Si ha la frammentazione esterna se lo spazio di memoria totale è sufficiente per soddisfare una richiesta, ma non è contiguo; la memoria è frammentata in tanti

piccoli buchi. Questo problema di frammentazione può essere molto grave; nel caso peggiore può esservi un blocco di memoria libera, sprecata, tra ogni coppia di processi. Se tutta questa memoria si trovasse in un unico blocco libero di grandi dimensioni, si potrebbero eseguire molti più processi.

L'impiego di un determinato criterio di scelta può influire sulla quantità di frammentazione: in alcuni sistemi dà migliori risultati la scelta del primo buco abbastanza grande, in altri dà migliori risultati la scelta del più piccolo tra i buchi abbastanza grandi; inoltre è necessario sapere qual è l'estremità assegnata di un blocco libero (se la parte inutilizzata è quella in alto o quella in basso). A prescindere dal tipo di algoritmo usato, la frammentazione esterna è un problema.

La sua gravità dipende dalla quantità totale di memoria e dalla dimensione media dei processi. L'analisi statistica dell'algoritmo che segue il criterio di scelta del primo buco abbastanza grande, ad esempio, rivela che, pur con una certa ottimizzazione, per *n* blocchi assegnati, si perdono altri 0,5*n* blocchi a causa della frammentazione, ciò significa che potrebbe essere inutilizzabile un terzo della memoria. Questa caratteristica è nota con il nome di **regola del 50 per cento**.

9.3.3 Frammentazione

La frammentazione può essere sia interna sia esterna. Si consideri il metodo d'assegnazione con più partizioni con un buco di 18.464 byte. Supponendo che il processo successivo richieda 18.462 byte, assegnando esattamente il blocco richiesto rimane un buco di 2 byte. Il carico necessario per tenere traccia di questo buco è sostanzialmente più grande del buco stesso. Il metodo generale prevede di suddividere la memoria fisica in blocchi di dimensione fissata, che costituiscono le unità d'assegnazione. Con questo metodo la memoria assegnata può essere leggermente maggiore della memoria richiesta. La **frammentazione interna** consiste nella differenza tra questi due numeri; la memoria è interna a una partizione, ma non è in uso.

Una soluzione al problema della frammentazione esterna è data dalla **compattazione**. Lo scopo è quello di riordinare il contenuto della memoria per riunire la memoria libera in un unico grosso blocco. La compattazione non è sempre possibile: non si può realizzare se la rilocazione è statica ed è fatta nella fase di assemblaggio o di caricamento; è possibile solo se la rilocazione è dinamica e si compie nella fase d'esecuzione. Se gli indirizzi sono rilocati dinamicamente, la rilocazione richiede solo lo spostamento del programma e dei dati, e quindi la modifica del registro di rilocazione in modo che rifletta il nuovo indirizzo di base. Se è possibile eseguire la compattazione, è necessario determinarne il costo. Il più semplice algoritmo di compattazione consiste nello spostare tutti i processi verso un'estremità della memoria, mentre tutti i buchi vengono spostati nell'altra direzione formando un grosso buco di memoria. Questo metodo può essere assai oneroso.

Un'altra possibile soluzione del problema della frammentazione esterna è data dal consentire la non contiguità dello spazio degli indirizzi logici di un processo, permettendo così di assegnare la memoria fisica ai processi dovunque essa sia disponibile. Due tecniche complementari conseguono questo risultato: la paginazione (Paragrafo 9.4) e la segmentazione (Paragrafo 9.5). Queste tecniche si possono anche combinare (Paragrafo 9.6).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.4 Paginazione

= è il metodo di gestione della memoria che permette che lo spazio degli indirizzi fisici di un processo non sia contiguo.

La paginazione è un metodo di gestione della memoria che permette che lo spazio degli indirizzi fisici di un processo non sia contiguo. Elimina il gravoso problema della sistemazione di blocchi di memoria di diverse dimensioni nella memoria ausiliaria, un problema che riguarda la maggior parte dei metodi di gestione della memoria analizzati. Quando alcuni frammenti di codice o dati residenti nella memoria centrale devono essere scaricati, si deve trovare lo spazio necessario nella memoria ausiliaria. I problemi di frammentazione relativi alla memoria centrale valgono anche per la memoria ausiliaria, con la differenza che in questo caso l'accesso è molto più lento, quindi è impossibile eseguire la compattazione. Grazie ai vantaggi offerti rispetto ai metodi precedenti, la paginazione nelle sue varie forme è comunemente usata in molti sistemi operativi.

Tradizionalmente, l'architettura del sistema offre specifiche caratteristiche per la gestione della paginazione. Recenti progetti (soprattutto le CPU a 64 bit) prevedono che il sistema di paginazione sia realizzato integrando strettamente l'architettura e il sistema operativo

9.4.1 Metodo di base

(frame)

Si suddivide la memoria fisica in blocchi di memoria di dimensioni fisse (noti anche come pagine fisiche o *frame*), e la memoria logica in blocchi di eguale dimensione detti pagine. Quando si deve eseguire un processo, si caricano le sue pagine nei blocchi di memoria disponibili, prendendole dalla memoria ausiliaria, che è divisa in blocchi di dimensione fissa, uguale a quella dei blocchi di memoria.

L'architettura d'ausilio alla paginazione è illustrata nella Figura 9.6; ogni indirizzo generato dalla CPU è diviso in due parti: un numero di pagina (p), e uno scostamento di pagina (d). Il numero di pagina serve come indice per la tabella delle pagine, che contiene l'indirizzo di base nella memoria fisica di ogni pagina. Questo indirizzo di base si appaia allo scostamento di pagina per definire l'indirizzo della memoria fisica, che s'invia all'unità di memoria. La Figura 9.7 illustra il modello di paginazione della memoria.

La dimensione di una pagina, così come quella di un blocco di memoria, è definita dall'architettura del calcolatore ed è, tipicamente, una potenza di 2 compresa tra 512 byte e 16 MB. La scelta di una potenza di 2 come dimensione della pagina facilita notevolmente la traduzione di un indirizzo logico nei corrispondenti numero di pagina e scostamento di pagina. Se la dimensione dello spazio degli indirizzi logici è 2^m e la dimensione di una pagina è di 2^n unità di indirizzamento (byte o parole), allora gli $m - n$ bit più significativi di un indirizzo logico indicano il numero di pagina, e gli n bit meno significativi indicano lo scostamento di pagina. L'indirizzo logico ha quindi la forma seguente:

numero di pagina	scostamento di pagina
p	d
$m - n$	n

• Si ha solo frammentazione interna solo all'ultimo frame.

frame → corrispondenza

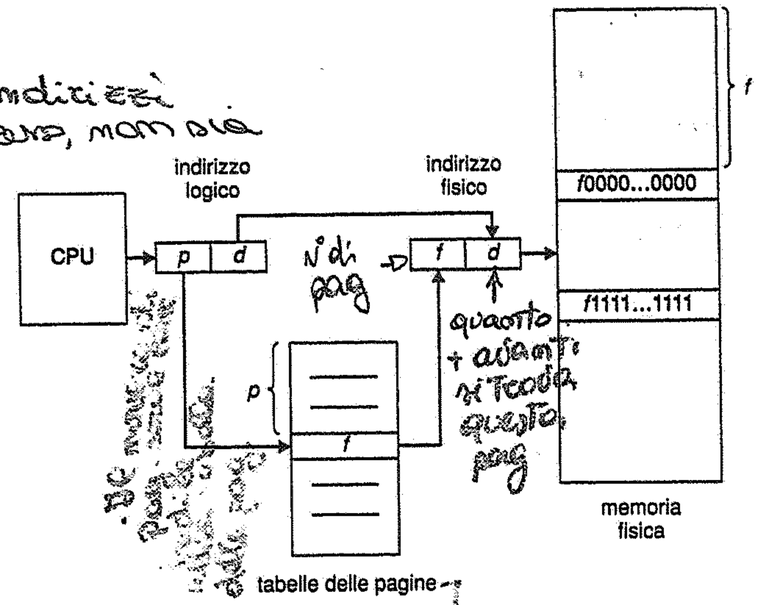


Figura 9.6 Architettura di paginazione.

contiene l'indirizzo di base nella memoria fisica di ogni pag.

dove p è un indice della tabella delle pagine e d è lo scostamento all'interno della pagina indicata da p.

Come esempio concreto, anche se minimo, si consideri la memoria illustrata nella Figura 9.8; con pagine di 4 byte e una memoria fisica di 32 byte (8 pagine), si mostra come si faccia corrispondere la memoria vista dall'utente alla memoria fisica. L'indirizzo logico 0 è la pagina 0 con scostamento 0. Secondo la tabella delle pagine, la pagina 0 si trova nel blocco di memoria 5. Quindi all'indirizzo logico 0 corrisponde l'indirizzo fisico 20 (= (5 × 4) + 0). All'indirizzo logico 3 (pagina 0, scostamento 3) corrisponde l'indirizzo fisico 23 (= (5 × 4) + 3). Per quel che riguarda l'indirizzo logico 4 (pagina 1, scostamento 0), secondo la tabella delle pagine, alla pagina 1 corrisponde il blocco di memoria 6, quindi, all'indirizzo logico 4 corrisponde l'indirizzo fisico 24 (= (6 × 4) + 0). All'indirizzo logico 13 corrisponde l'indirizzo fisico 9.

Il lettore può aver notato che la paginazione non è altro che una forma di rilocazione dinamica: a ogni indirizzo logico l'architettura di paginazione fa corrispondere un indirizzo fisico. L'uso della tabella delle pagine è simile all'uso di una tabella di registri di base (o di rilocazione), uno per ciascun blocco di memoria.

Con la paginazione si può evitare la frammentazione esterna: *qualsiasi* blocco di memoria libero si può assegnare a un processo che ne abbia bisogno; tuttavia si può avere la frammentazione interna. I blocchi di memoria si assegnano come unità. Poiché in generale lo spazio di memoria richiesto da un processo non è un multiplo delle dimen-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

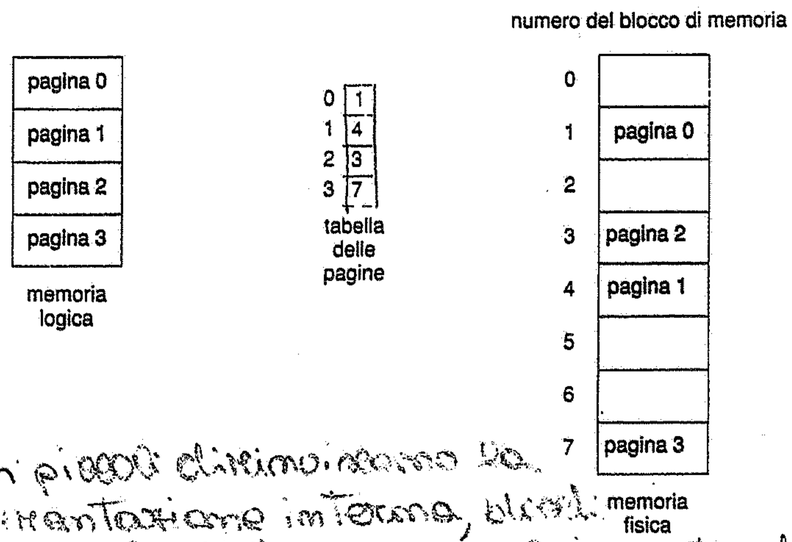


Figura 9.7 Modello di paginazione di memoria logica e memoria fisica.

sioni delle pagine, l'*ultimo* blocco di memoria assegnato può non essere completamente pieno. Se, ad esempio, le pagine sono di 2048 byte, un processo di 72.766 byte necessita di 35 pagine più 1086 byte. Si assegnano 36 blocchi di memoria, quindi si ha una frammentazione interna di 2048 – 1086 = 962 byte. Il caso peggiore si ha con un processo che necessita di n pagine più un byte: si assegnano $n + 1$ blocchi di memoria, quindi si ha una frammentazione interna di quasi un intero blocco di memoria.

Se la dimensione del processo è indipendente dalla dimensione della pagina, ci si deve aspettare una frammentazione interna media di mezza pagina per processo. Questa considerazione suggerisce che conviene usare pagine di piccole dimensioni; tuttavia, a ogni elemento della tabella delle pagine è associato un carico che si può ridurre aumentando le dimensioni delle pagine. Inoltre, con un maggior numero di dati da trasferire, l'I/O su disco è più efficiente (Capitolo 14). Generalmente la dimensione delle pagine cresce col passare del tempo, come i processi, gli insiemi di dati e la memoria centrale; attualmente la dimensione tipica delle pagine è compresa tra 4 KB e 8 KB; in alcuni sistemi può essere anche maggiore. Alcune CPU e alcuni nuclei di sistemi operativi gestiscono anche pagine di diverse dimensioni; il sistema Solaris ad esempio usa pagine di 4 KB o 8 KB, secondo il tipo dei dati memorizzati nelle pagine. Sono in fase di studio e progettazione sistemi di paginazione che consentono la variazione dinamica della dimensione delle pagine.

Ciascun elemento della tabella delle pagine di solito è lungo 4 byte, ma anche questa dimensione può variare; un elemento di 32 bit può puntare a uno dei 2^{32} blocchi di memoria; quindi se un blocco di memoria è di 4 KB, un sistema con elementi di 4 byte può accedere a 2^{36} byte (o 64 GB) di memoria fisica.

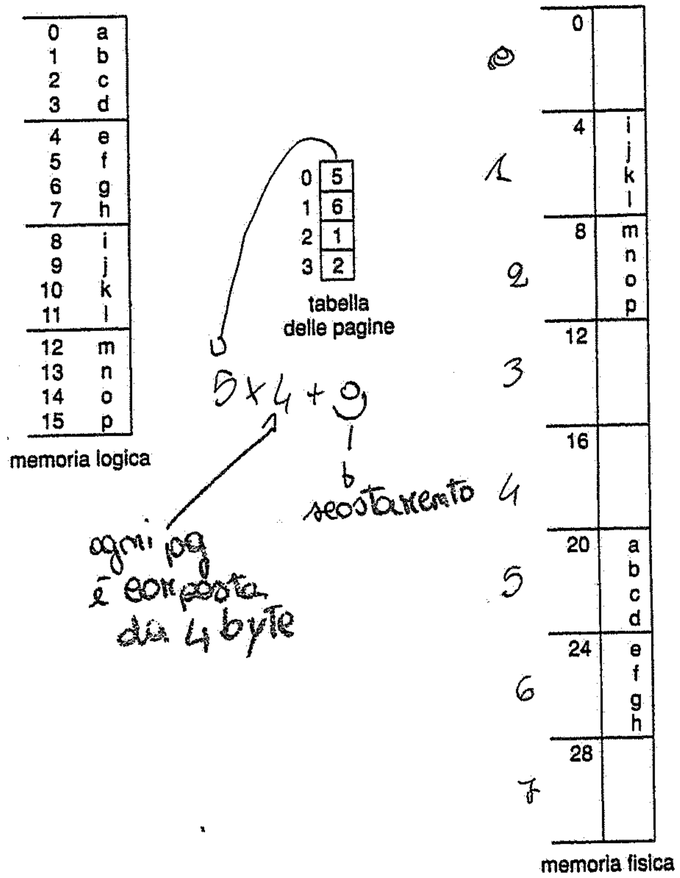


Figura 9.8 Esempio di paginazione per una memoria di 32 byte con pagine di 4 byte.

Quando si deve eseguire un processo, si esamina la sua dimensione espressa in pagine. Poiché ogni pagina del processo necessita di un blocco di memoria, se il processo richiede n pagine, devono essere disponibili almeno n blocchi di memoria, che, se ci sono, si assegnano al processo stesso. Si carica la prima pagina del processo in uno dei blocchi di memoria assegnati e s'inserisce il numero del blocco di memoria nella tabella delle pagine relativa al processo in questione. La pagina successiva si carica in un altro blocco di memoria e, anche in questo caso, s'inserisce il numero del blocco di memoria nella tabella delle pagine, e così via (Figura 9.9).

Un aspetto importante della paginazione è la netta distinzione tra la memoria vista dall'utente e l'effettiva memoria fisica: il programma utente 'vede' la memoria come un unico spazio contiguo, contenente solo il programma stesso; in realtà, il programma utente è sparso in una memoria fisica che contiene anche altri programmi. La differenza

http://www.adobepdf.com
Created by TIFF To PDF trial version, please register this software.

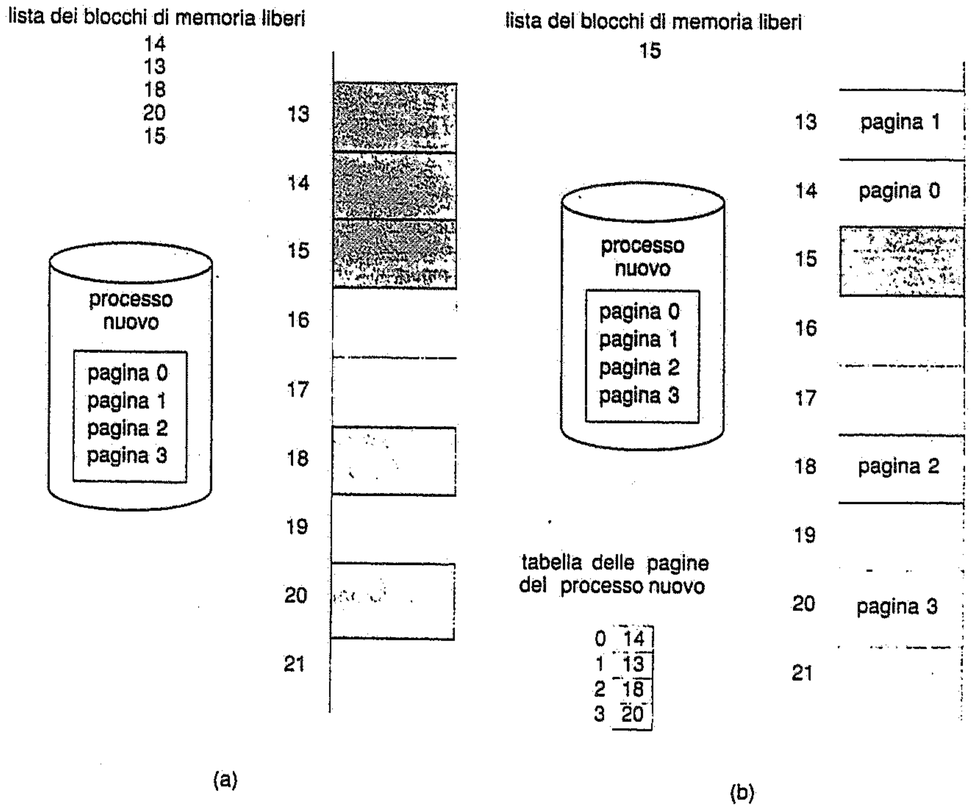


Figura 9.9 Blocchi di memoria liberi; a) prima e b) dopo l'assegnazione.

tra la memoria vista dall'utente e la memoria fisica è colmata dall'architettura di traduzione degli indirizzi, che fa corrispondere gli indirizzi fisici agli indirizzi logici generati dai processi utenti. Queste operazioni non sono visibili agli utenti e sono controllate dal sistema operativo. Si noti che un processo utente, per definizione, non può accedere alle zone della memoria che non gli appartengono. Non ha modo di accedere alla memoria oltre quel che è previsto dalla sua tabella delle pagine; tale tabella riguarda soltanto le pagine che appartengono al processo.

Poiché il sistema operativo gestisce la memoria fisica, deve essere informato dei relativi particolari di assegnazione: quali blocchi di memoria sono assegnati, quali sono disponibili, il numero totale dei blocchi di memoria, e così via. In genere queste informazioni sono contenute in una struttura di dati chiamata **tabella dei blocchi di memoria**, che contiene un elemento per ogni blocco di memoria, indicante se il blocco di memoria è libero oppure assegnato e, se è assegnato, a quale pagina di quale processo o di quali processi.

Inoltre, il sistema operativo deve sapere che i processi utenti operano nello spazio d'utente, e tutti gli indirizzi logici si devono far corrispondere a indirizzi fisici. Se un utente usa una chiamata del sistema con un indirizzo di memoria come parametro, ad esempio per eseguire un'operazione di I/O all'indirizzo specificato, si deve tradurre tale indirizzo nell'indirizzo fisico corretto. Il sistema operativo conserva una copia della tabella delle pagine per ciascun processo, così come conserva una copia dei valori contenuti nel contatore di programma e nei registri. Questa copia si usa per tradurre gli indirizzi logici in indirizzi fisici ogni volta che il sistema operativo deve associare esplicitamente un indirizzo fisico a un indirizzo logico. La stessa copia è usata anche dal dispatcher della CPU per impostare l'architettura di paginazione quando a un processo sta per essere assegnata la CPU. La paginazione fa quindi aumentare la durata dei cambi di contesto.

9.4.2 Architettura di paginazione

Ogni sistema operativo segue metodi propri per memorizzare le tabelle delle pagine. La maggior parte dei sistemi impiega una tabella delle pagine per ciascun processo. Il descrittore di processo contiene, insieme col valore di altri registri, come il registro delle istruzioni, un puntatore alla tabella delle pagine. Per avviare un processo, il dispatcher ricarica i registri d'utente e imposta i corretti valori della tabella delle pagine fisiche, usando la tabella delle pagine presente nella memoria e relativa al processo.

L'architettura d'ausilio alla tabella delle pagine si può realizzare in modi diversi. Nel caso più semplice, si usa uno specifico insieme di **registri**. Per garantire un'efficiente traduzione degli indirizzi di paginazione, questi registri devono essere costruiti in modo da operare a una velocità molto elevata. Tale efficienza è determinante, poiché ogni accesso alla memoria passa attraverso il sistema di paginazione. Il dispatcher della CPU ricarica questi registri proprio come ricarica gli altri registri, ma le istruzioni di caricamento e modifica dei registri della tabella delle pagine sono privilegiate, quindi soltanto il sistema operativo può modificare la mappa della memoria. Il DEC PDP-11 è un esempio di tale tipo di architettura. Un indirizzo è costituito di 16 bit e la dimensione di una pagina è di 8 KB, la tabella delle pagine consiste quindi di otto elementi che sono mantenuti in registri veloci.

L'uso di registri per la tabella delle pagine è efficiente se la tabella stessa è ragionevolmente piccola, dell'ordine, ad esempio, di 256 elementi. La maggior parte dei calcolatori contemporanei usa comunque tabelle molto grandi, ad esempio di un milione di elementi, quindi non si possono impiegare i registri veloci per realizzare la tabella delle pagine; quest'ultima si mantiene nella memoria principale e un **registro di base della tabella delle pagine** (*page-table base register* — PTBR) punta alla tabella stessa. Il cambio delle tabelle delle pagine richiede soltanto di modificare questo registro, riducendo considerevolmente il tempo dei cambi di contesto.

Questo metodo presenta un problema connesso al tempo necessario per accedere a una locazione della memoria d'utente. Per accedere alla locazione *i*, occorre fare riferimento alla tabella delle pagine usando il valore contenuto nel PTBR aumentato del numero di pagina relativo a *i*, perciò si deve accedere alla memoria. Si ottiene il numero del blocco di memoria che, associato allo scostamento di pagina, produce l'indirizzo cercato;

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

1) x la tabella
2) x le istruzioni/dati (in questo caso x il byte)

a questo punto è possibile accedere alla posizione di memoria desiderata. Con questo metodo, per accedere a un byte occorrono *due* accessi alla memoria (uno per l'elemento della tabella delle pagine e uno per il byte stesso), quindi l'accesso alla memoria è rallentato di un fattore 2. Nella maggior parte dei casi un tale ritardo è intollerabile; sarebbe più conveniente ricorrere all'avvicendamento dei processi!

TLB = registri ad acc. veloce per ridurre il problema del 2° accesso alla memoria

La soluzione tipica a questo problema consiste nell'impiego di una speciale, piccola cache di ricerca veloce, detta TLB (*translation look-aside buffer*). Il TLB è una memoria associativa ad alta velocità in cui ogni suo elemento consiste di due parti: una chiave (o indicatore) e un valore. Quando si presenta un elemento, la memoria associativa lo confronta contemporaneamente con tutte le chiavi; se trova una corrispondenza, riporta il valore correlato. La ricerca è molto rapida ma le memorie associative sono molto costose. Il numero degli elementi in un TLB è piccolo, spesso è compreso tra 64 e 1024.

Il TLB si usa insieme con le tabelle delle pagine nel modo seguente: il TLB contiene una piccola parte degli elementi della tabella delle pagine; quando la CPU genera un indirizzo logico, si presenta il suo numero di pagina al TLB, se tale numero è presente, il corrispondente numero di blocco di memoria è immediatamente disponibile e si usa per accedere alla memoria. Tutta l'operazione può richiedere un tempo inferiore al 10 per cento in più del tempo che sarebbe richiesto per un riferimento alla memoria senza paginazione.

Se il numero di pagina non è presente nel TLB, situazione nota come **insuccesso del TLB** (*TLB miss*), si deve consultare la tabella delle pagine nella memoria. Il numero del blocco di memoria così ottenuto si può eventualmente usare per accedere alla memoria (Figura 9.10). Inoltre, inserendo i numeri della pagina e del blocco di memoria nel TLB, al riferimento successivo la ricerca sarà molto più rapida. Se il TLB è già pieno d'elementi, il sistema operativo deve sceglierne uno per sostituirlo. I criteri di sostituzione variano dalla scelta dell'elemento usato meno recentemente (LRU) alla scelta casuale. Inoltre alcuni TLB consentono che certi elementi siano vincolati (*wired down*), cioè non si possano rimuovere dal TLB; tipicamente si vincolano gli elementi per il codice del nucleo.

Alcuni TLB memorizzano gli **identificatori dello spazio d'indirizzi** (*address-space identifier* — ASID) in ciascun elemento del TLB. Un ASID identifica in modo univoco ciascun processo e si usa per fornire al processo corrispondente la protezione del suo spazio d'indirizzi. Quando tenta di trovare i valori corrispondenti ai numeri delle pagine virtuali, il TLB assicura che l'ASID per il processo attualmente in esecuzione corrisponda all'ASID associato alla pagina virtuale. La mancata corrispondenza dell'ASID si tratta come un insuccesso del TLB. Inoltre per fornire la protezione dello spazio d'indirizzi, l'ASID consente che il TLB contenga nello stesso istante elementi di diversi processi. Se il TLB non permette l'uso di ASID distinti, ogni volta che si seleziona una nuova tabella delle pagine, ad esempio a ogni cambio di contesto, si deve **cancellare** il TLB in modo da assicurare che il successivo processo in esecuzione non faccia uso di errate informazioni di traduzione. Potrebbero altrimenti esserci vecchi elementi del TLB che contengono indirizzi virtuali validi ma con indirizzi fisici corrispondenti sbagliati o non validi, lasciati in sospeso dal precedente processo.

x identificare ogni processo in memoria

La percentuale di volte che un numero di pagina si trova nel TLB si chiama **tasso di successi** (*hit ratio*). Un tasso di successi dell'80 per cento significa che il numero di pagina desiderato si trova nel TLB nell'80 per cento dei casi. Se la ricerca nel TLB richiede 20

Ci accorgo se quella pag è presente o è stata sostituita a caricare

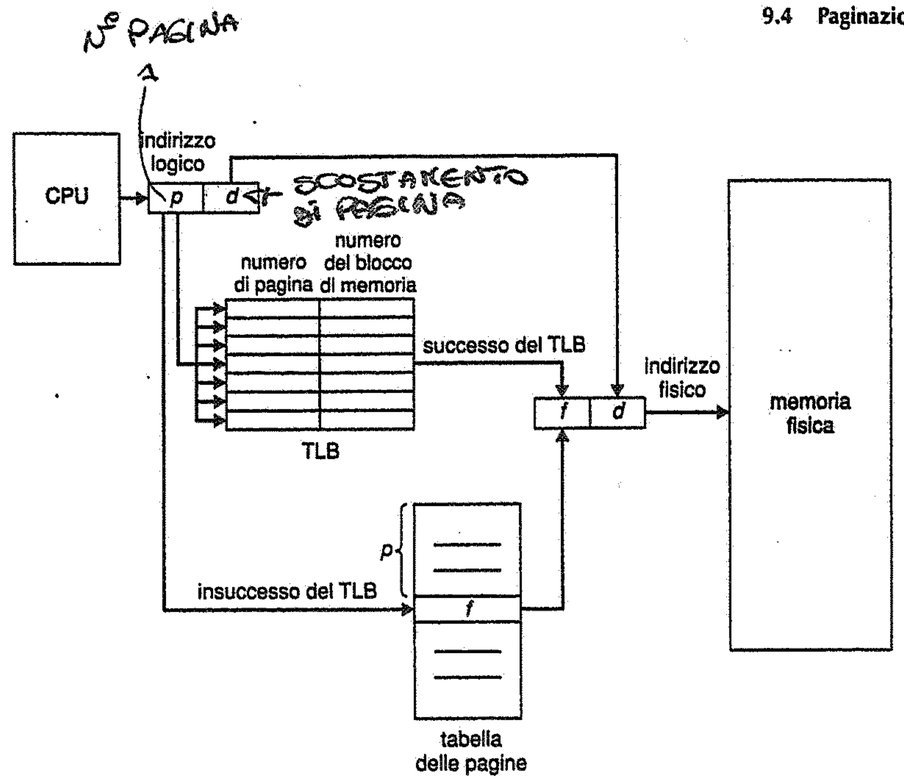


Figura 9.10 Architettura di paginazione con TLB.

nanosecondi e sono necessari 100 nanosecondi per accedere alla memoria, allora, supponendo che il numero di pagina si trovi nel TLB, un accesso alla memoria richiede 120 nanosecondi. Se, invece, il numero non è contenuto nel TLB (20 nanosecondi), occorre accedere alla memoria per arrivare alla tabella delle pagine e al numero di blocco di memoria (100 nanosecondi), quindi accedere al byte desiderato nella memoria (100 nanosecondi); in totale sono necessari 220 nanosecondi. Per calcolare il **tempo effettivo d'accesso alla memoria** occorre tenere conto della probabilità dei due casi:

$$\textit{tempo effettivo d'accesso} = 0{,}80 \times 120 + 0{,}20 \times 220 = 140 \text{ nanosecondi}$$

In questo esempio si verifica un rallentamento del 40 per cento nel tempo d'accesso alla memoria (da 100 a 140 nanosecondi).

Per un tasso di successi del 98 per cento si ottiene il seguente risultato:

$$\textit{tempo effettivo d'accesso} = 0{,}98 \times 120 + 0{,}02 \times 220 = 122 \text{ nanosecondi}$$

Aumentando il tasso di successi, il rallentamento del tempo d'accesso alla memoria scende al 22 per cento. L'impatto del tasso di successi nei TLB è ulteriormente analizzato nel Capitolo 10.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.4.3 Protezione

In un ambiente paginato, la protezione della memoria è assicurata dai bit di protezione associati a ogni blocco di memoria; normalmente tali bit si trovano nella tabella delle pagine. Un bit può determinare se una pagina si può leggere e scrivere oppure soltanto leggere. Tutti i riferimenti alla memoria passano attraverso la tabella delle pagine per trovare il numero di blocco di memoria giusto; quindi mentre si calcola l'indirizzo fisico, si possono controllare i bit di protezione per verificare che non si scriva in una pagina di sola lettura. Un talè tentativo causerebbe l'invio di un segnale di eccezione al sistema operativo, si ha cioè una violazione della protezione della memoria.

Questo metodo di protezione si può facilmente estendere per fornire un livello di protezione più perfezionato. Si può progettare un'architettura che fornisca la protezione di sola lettura, di sola scrittura o di sola esecuzione. In alternativa, con bit di protezione distinti per ogni tipo d'accesso, si può ottenere una qualsiasi combinazione di tali tipi d'accesso; i tentativi illegali causano l'invio di un segnale di eccezione al sistema operativo.

Di solito si associa un ulteriore bit, detto **bit di validità**, a ciascun elemento della tabella delle pagine. Tale bit impostato a *valido*, indica che la pagina corrispondente è nello spazio d'indirizzi logici del processo, quindi è una pagina valida; impostato a *non valido*, indica che la pagina non è nello spazio d'indirizzi logici del processo. Il bit di validità consente quindi di riconoscere gli indirizzi illegali e di notificarne la presenza attraverso un segnale di eccezione. Il sistema operativo concede o revoca la possibilità d'accesso a una pagina impostando in modo appropriato tale bit. Ad esempio, in un sistema con uno spazio di indirizzi di 14 bit (da 0 a 16.383) si può avere un programma che deve usare soltanto gli indirizzi da 0 a 10.468 Con una dimensione delle pagine di 2 KB si ha la situazione mostrata nella Figura 9.11. Gli indirizzi nelle pagine 0, 1, 2, 3, 4 e 5 sono tradotti normalmente tramite la tabella delle pagine. D'altra parte, ogni tentativo di generare un indirizzo nelle pagine 6 o 7 trova non valido il bit di validità; in questo caso il calcolatore invia un segnale di eccezione al sistema operativo (riferimento di pagina non valido).

Poiché il programma si estende solo fino all'indirizzo 10.468, quindi ogni riferimento oltre tale indirizzo è illegale, i riferimenti alla pagina 5 sono comunque classificati come validi, e ciò rende validi gli accessi sino all'indirizzo 12.287; solo gli indirizzi da 12.288 a 16.383 sono non validi. Tale problema è dovuto alla dimensione delle pagine di 2 KB e riflette la frammentazione interna della paginazione.

Capita raramente che un processo faccia uso di tutto il suo intervallo di indirizzi, infatti molti processi utilizzano solo una piccola frazione dello spazio d'indirizzi di cui dispongono. In questi casi è un inutile spreco creare una tabella di pagine con elementi per ogni pagina dell'intervallo di indirizzi, poiché una gran parte di questa tabella resta inutilizzata e occupa prezioso spazio di memoria. Alcune architetture dispongono di registri, detti **registri di lunghezza della tabella delle pagine** (*page-table length register* — PTLR), per indicare le dimensioni della tabella. Questo valore si controlla rispetto a ogni indirizzo logico per verificare che quest'ultimo si trovi nell'intervallo valido per il processo, un errore causa l'invio di un segnale di eccezione al sistema operativo.

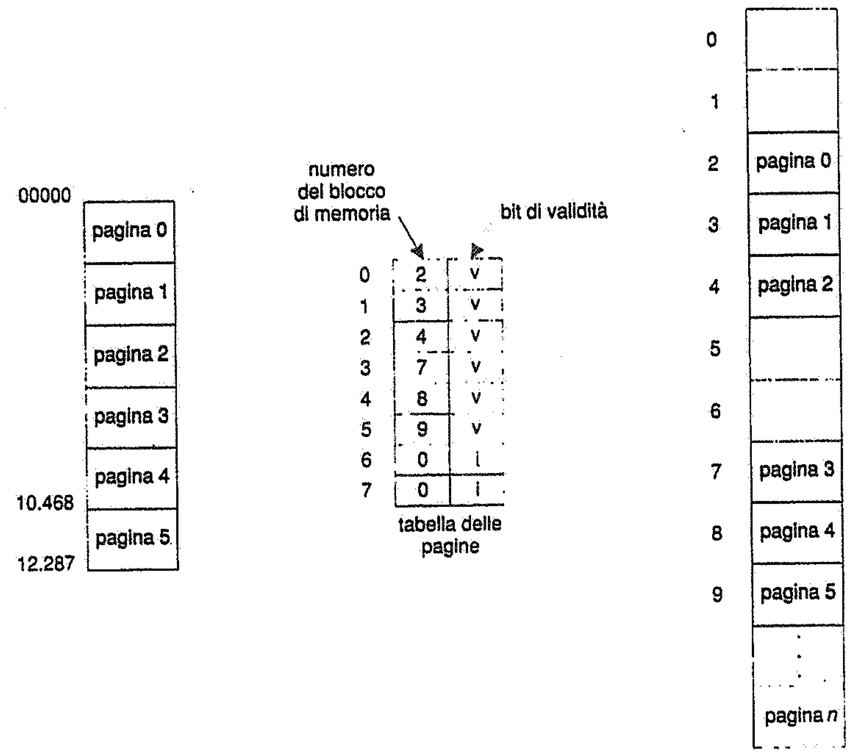


Figura 9.11 Bit di validità in una tabella delle pagine.

9.4.4 Struttura della tabella delle pagine

Nei Paragrafi dal 9.4.4.1 al 9.4.4.3 si descrivono alcune tra le tecniche più comuni per strutturare la tabella delle pagine.

9.4.4.1 Paginazione gerarchica

La maggior parte dei moderni calcolatori dispone di uno spazio d'indirizzi logici molto grande (da 2^{32} a 2^{64} elementi). In un ambiente di questo tipo la stessa tabella delle pagine finirebbe per diventare eccessivamente grande. Si consideri, ad esempio, un sistema con uno spazio d'indirizzi logici a 32 bit. Se la dimensione di ciascuna pagina è di 4 KB (2^{12}), la tabella delle pagine potrebbe arrivare a contenere fino a 1 milione di elementi ($2^{32}/2^{12}$). Se ogni elemento consiste di 4 byte, ciascun processo potrebbe richiedere fino a 4 MB di spazio d'indirizzi fisico solo per la tabella delle pagine. Chiaramente, sarebbe meglio evitare di collocare la tabella delle pagine in modo contiguo nella memoria centrale. Una semplice soluzione a questo problema consiste nel dividere la tabella delle pagine in parti più piccole; questo risultato si può ottenere in molti modi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

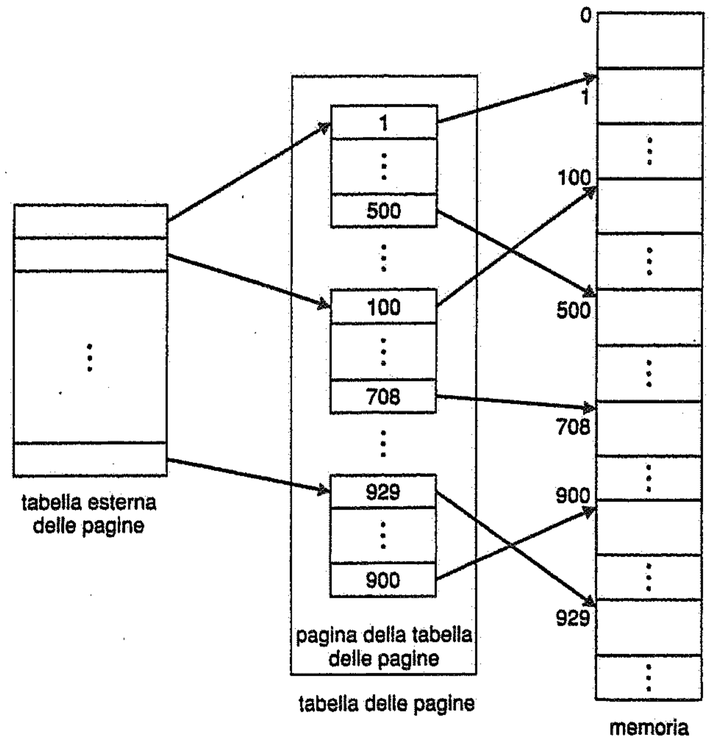


Figura 9.12 Schema di una tabella delle pagine a due livelli.

Un metodo consiste nell'adottare un algoritmo di paginazione a due livelli, nel quale si pagina la stessa tabella delle pagine (Figura 9.12). Si consideri il precedente esempio di macchina a 32 bit con dimensione delle pagine di 4 KB. Si suddivide ciascun indirizzo logico in un numero di pagina di 20 bit e in uno scostamento di pagina di 12 bit. Paginando la tabella delle pagine, anche il numero di pagina è a sua volta diviso in un numero di pagina di 10 bit e uno scostamento di pagina di 10 bit. Quindi, l'indirizzo logico è:

numero di pagina		scostamento di pagina
p_1	p_2	d
10	10	12

dove p_1 è un indice della tabella delle pagine di primo livello, o tabella esterna delle pagine, e p_2 è lo scostamento all'interno della pagina indicata dalla tabella esterna delle pagine. Il metodo di traduzione degli indirizzi seguito da questa architettura è mostrato

• Per accedere al dato devo fare 3 accessi di cui 2 alle tabelle ed 1 alla memoria e quindi l'overhead è di 1/3 del tempo totale.

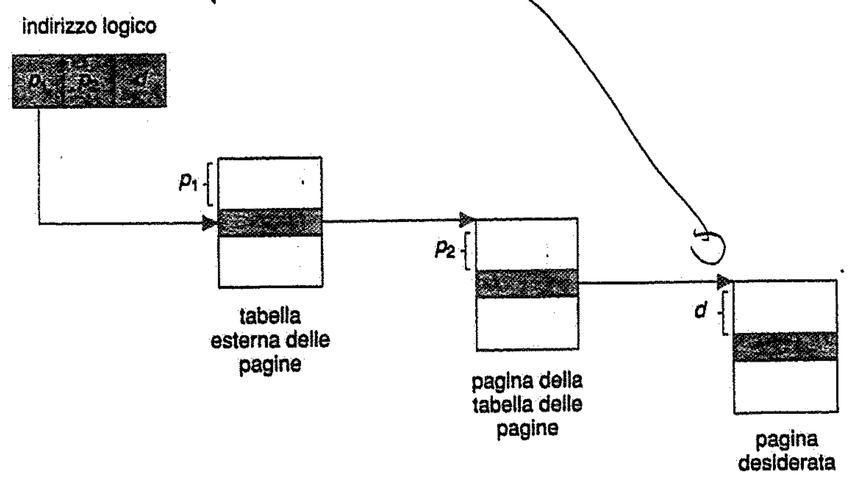


Figura 9.13 Traduzione degli indirizzi per un'architettura a 32 bit con paginazione a due livelli.

nella Figura 9.13. Poiché la traduzione degli indirizzi si svolge dalla tabella esterna delle pagine verso l'interno, questo metodo (usato ad esempio nel Pentium II) è anche noto come **tabella delle pagine ad associazione diretta** (*forward-mapped page table*).

Anche l'architettura VAX ha una variante della paginazione a due livelli. Il VAX è una macchina a 32 bit con pagine di 512 byte. Lo spazio d'indirizzi logici di un processo è diviso in quattro sezioni uguali, ciascuna di 2^{30} byte; ogni sezione rappresenta una parte differente dello spazio d'indirizzi logici di un processo. I 2 bit più significativi dell'indirizzo logico identificano la sezione appropriata, i successivi 21 bit rappresentano il numero logico di pagina all'interno di tale sezione e gli ultimi 9 bit lo scostamento nella pagina richiesta. Dividendo in questo modo la tabella delle pagine, il sistema operativo può lasciare inutilizzate le diverse parti fino al momento in cui un processo ne fa richiesta. Nell'architettura VAX un indirizzo ha quindi la forma seguente:

sezione	pagina	scostamento
s	p	d
2	21	9

dove s denota il numero della sezione, p è un indice per la tabella delle pagine e d è lo scostamento all'interno della pagina.

La dimensione di una tabella delle pagine a un livello per un processo in un sistema VAX che usa una sezione è ancora 2^{21} bit × 4 byte per elemento = 8 MB. Per ridurre ulteriormente l'uso della memoria centrale, il VAX pagina le tabelle delle pagine dei processi utenti.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Lo schema di paginazione a due livelli non è più adatto nel caso di sistemi con uno spazio di indirizzi logici a 64 bit. Per illustrare questo aspetto, si supponga che la dimensione delle pagine di questo sistema sia di 4 KB (2^{12}). In questo caso, la tabella delle pagine conterrà fino a 2^{52} elementi. Adottando uno schema di paginazione a due livelli, le tabelle interne delle pagine possono occupare convenientemente una pagina, o contenere 2^{10} elementi di 4 byte. Gli indirizzi si presentano come segue:

pagina della tabella esterna	pagina della tabella interna	scostamento
p_1	p_2	d
42	10	12

La tabella esterna delle pagine consiste di 2^{42} elementi, o 2^{44} byte. La soluzione ovvia per evitare una tabella tanto grande consiste nel dividere la tabella in parti più piccole. Questo metodo si adotta anche in alcune CPU a 32 bit allo scopo di fornire una maggiore flessibilità ed efficienza.

La tabella delle pagine si può dividere in vari modi. Si può paginare la tabella esterna delle pagine, ottenendo uno schema di paginazione a tre livelli. Si supponga che la tabella esterna delle pagine sia costituita di pagine di dimensione ordinaria (2^{10} elementi, o 2^{12} byte); uno spazio d'indirizzi a 64 bit è ancora enorme:

pagina della tabella esterna di secondo livello	pagina della tabella esterna	pagina della tabella interna	scostamento
p_1	p_2	p_3	d
32	10	10	12

La tabella esterna delle pagine è ancora di 2^{34} byte.

Il passo successivo sarebbe uno schema di paginazione a quattro livelli, nel quale si pagina anche la tabella esterna di secondo livello delle pagine. L'architettura SPARC (con indirizzamento a 32 bit) offre uno schema di paginazione a tre livelli, mentre la CPU a 32 bit Motorola 68030 offre uno schema di paginazione a quattro livelli.

Comunque, per le architetture a 64 bit le tabelle delle pagine gerarchiche sono generalmente considerate inappropriate. L'UltraSPARC ad esempio richiederebbe sette livelli di paginazione; un numero di accessi alla memoria proibitivo per tradurre ciascun indirizzo logico a ogni insuccesso del TLB.

9.4.4.2 Tabella delle pagine di tipo hash

Un metodo di gestione molto comune degli spazi d'indirizzi relativi ad architetture oltre i 32 bit consiste nell'impiego di una **tabella delle pagine di tipo hash**, in cui l'argomento della funzione di hash è il numero della pagina virtuale. Per la gestione delle collisioni, ogni elemento della tabella hash contiene una lista concatenata di elementi che la funzione hash fa corrispondere alla stessa locazione. Ciascun elemento è composto da tre campi: (a) il numero della pagina virtuale; (b) l'indirizzo del blocco della memoria fisica (pagina fisica) che corrisponde alla pagina virtuale; (c) un puntatore al successivo elemento della lista.

L'algoritmo opera come segue: si applica la funzione hash al numero della pagina virtuale contenuto nell'indirizzo virtuale, identificando un elemento della tabella. Si confronta il numero di pagina virtuale con il campo (a) del primo elemento della lista concatenata corrispondente. Se i valori coincidono, si usa l'indirizzo del relativo blocco di memoria (campo (b)) per generare l'indirizzo fisico desiderato. Altrimenti, l'algoritmo esamina allo stesso modo gli elementi successivi della lista concatenata (Figura 9.14). Le tabelle delle pagine di tipo hash sono particolarmente utili per gli spazi d'indirizzi sparsi, in cui i riferimenti alla memoria non sono contigui ma distribuiti per tutto lo spazio d'indirizzi.

Poiché per spazi d'indirizzi relativi ad architetture a 64 bit ciascun numero di pagina virtuale (64 bit) più il puntatore al successivo elemento della lista (64 bit) richiedono 128 bit per 64 bit di informazione utile, è stata proposta una variante di questo schema. Si tratta della **tabella delle pagine a gruppi** (*clustered page table*), è simile alla tabella hash ma ciascun elemento della tabella delle pagine contiene i riferimenti alle pagine fisiche

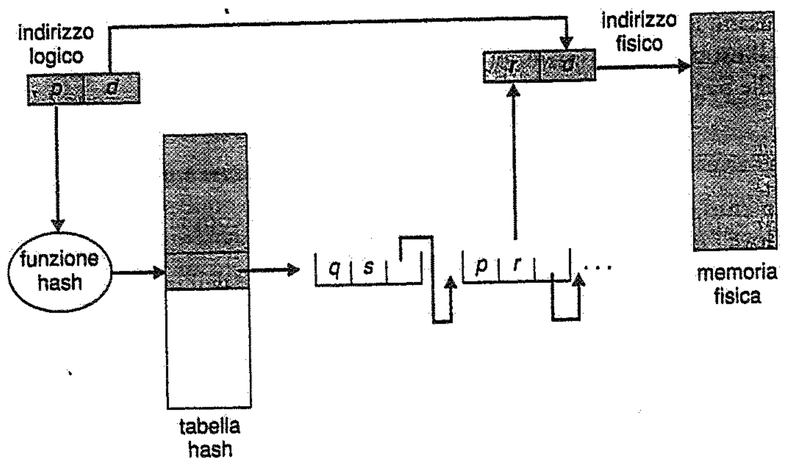


Figura 9.14 Tabella delle pagine di tipo hash.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

corrispondenti a un gruppo di pagine virtuali contigue (ad esempio 16). In questo modo si riduce lo spazio di memoria richiesto. Inoltre, poiché lo spazio degli indirizzi di molti programmi non è arbitrariamente sparso su pagine isolate ma distribuito per 'raffiche' di riferimenti, le tabelle delle pagine a gruppi sfruttano bene questa caratteristica.

9.4.4.3 Tabella delle pagine invertita → limito la memoria necessaria x memorizzare le pag

Generalmente, si associa una tabella delle pagine a ogni processo e tale tabella contiene un elemento per ogni pagina virtuale che il processo sta utilizzando, oppure un elemento per ogni indirizzo virtuale a prescindere dalla validità di quest'ultimo. Questa è una rappresentazione naturale della tabella, poiché i processi fanno riferimento alle pagine tramite gli indirizzi virtuali delle pagine stesse, che il sistema operativo deve poi tradurre in indirizzi di memoria fisica. Poiché la tabella è ordinata per indirizzi virtuali, il sistema operativo può calcolare in che punto della tabella si trova l'elemento dell'indirizzo fisico associato, e usare direttamente tale valore. Uno degli inconvenienti insiti in questo metodo è costituito dalla dimensione di ciascuna tabella delle pagine, che può contenere milioni di elementi e occupare grandi quantità di memoria fisica, necessaria proprio per sapere com'è impiegata la rimanente memoria fisica.

Per risolvere questo problema si può fare uso della **tabella delle pagine invertita**. Una tabella delle pagine invertita ha un elemento per ogni pagina reale (blocco di memoria). Ciascun elemento è quindi costituito dell'indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria, con informazioni sul processo che possiede tale pagina. Quindi, nel sistema esiste una sola tabella delle pagine che ha un solo elemento per ciascuna pagina di memoria fisica. Nella Figura 9.15 sono mostrate le operazioni di una tabella delle pagine invertita; si confronti questa figura con la Figura 9.6, che illustra il modo di operare per una tabella delle pagine ordinaria. Esempi di architetture che usano le tabelle delle pagine invertite sono l'UltraSPARC a 64 bit e la PowerPC.

Per illustrare questo metodo è possibile descrivere una versione semplificata della tabella delle pagine invertita dell'IBM RT. Ciascun indirizzo virtuale è una tripla del tipo seguente:

<id-processo, numero di pagina, scostamento>

Ogni elemento della tabella delle pagine invertita è una coppia *<id-processo, numero di pagina>* dove l'*id-processo* assume il ruolo di identificatore dello spazio d'indirizzi. Quando si fa un riferimento alla memoria, si presenta una parte dell'indirizzo virtuale, formato da *<id-processo, numero di pagina>*, al sottosistema di memoria. Quindi si cerca una corrispondenza nella tabella delle pagine invertita. Se si trova tale corrispondenza, ad esempio sull'elemento *i*, si genera l'indirizzo fisico *<i, scostamento>*. Se invece non si trova alcuna corrispondenza è stato tentato un accesso illegale a un indirizzo.

Sebbene questo schema riduca la quantità di memoria necessaria per memorizzare ogni tabella delle pagine, aumenta però il tempo di ricerca nella tabella quando si fa riferimento a una pagina. Poiché la tabella delle pagine invertita è ordinata per indirizzi fisici, mentre le ricerche si fanno per indirizzi virtuali, per trovare una corrispondenza occorre esaminare tutta la tabella; questa ricerca richiede molto tempo. Per limitare l'entità del problema si può impiegare una tabella hash (come si descrive nel Paragrafo 9.4.4.2), che

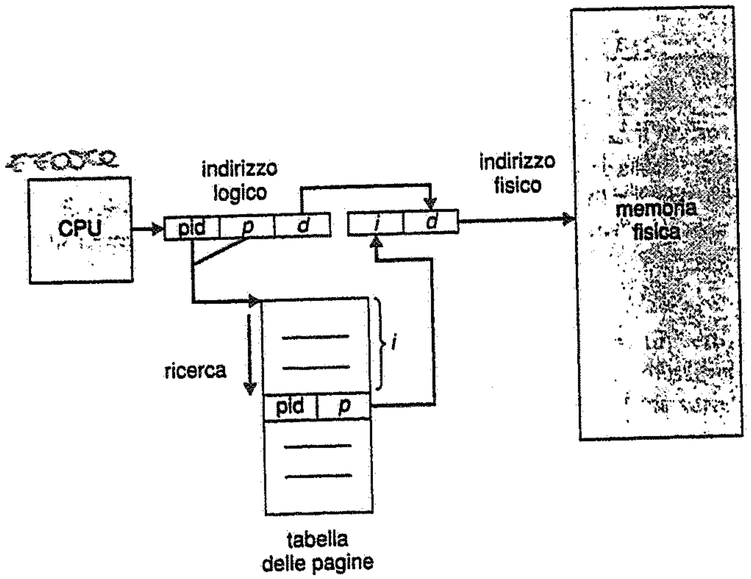


Figura 9.15 Tabella delle pagine invertita.

riduce la ricerca a un solo, o a pochi, elementi della tabella delle pagine. Naturalmente, ogni accesso alla tabella hash aggiunge al procedimento un riferimento alla memoria, quindi un riferimento alla memoria virtuale richiede almeno due letture della memoria reale: una per l'elemento della tabella hash e l'altro per la tabella delle pagine. Per migliorare le prestazioni, la ricerca si effettua prima nel TLB, quindi si consulta la tabella hash.

9.4.5 Pagine condivise

Un altro vantaggio della paginazione consiste nella possibilità di *condividere* codice comune. Questa considerazione è importante soprattutto in un ambiente a partizione del tempo. Si consideri un sistema con 40 utenti, ciascuno dei quali usa un elaboratore di testi. Se tale programma è formato di 150 KB di codice e 50 KB di spazio di dati, per gestire i 40 utenti sono necessari 8000 KB. Se però il codice è rientrante, può essere condiviso; la Figura 9.16 mostra un elaboratore di testi, di tre pagine di 50 KB ciascuna (l'ampia dimensione delle pagine ha lo scopo di rendere più chiara la figura), condiviso da tre processi, ciascuno dei quali dispone della propria pagina di dati.

Il **codice rientrante**, detto anche **codice puro**, è un codice non automodificante: non cambia durante l'esecuzione. Quindi, due o più processi possono eseguire lo stesso codice nello stesso momento. Ciascun processo dispone di una propria copia dei registri e di una memoria dove conserva i dati necessari alla propria esecuzione. I dati per due differenti processi variano, ovviamente, per ciascun processo.

Nella memoria fisica è presente una sola copia dell'elaboratore di testi: la tabella delle pagine di ogni utente fa corrispondere gli stessi blocchi di memoria contenenti l'e-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

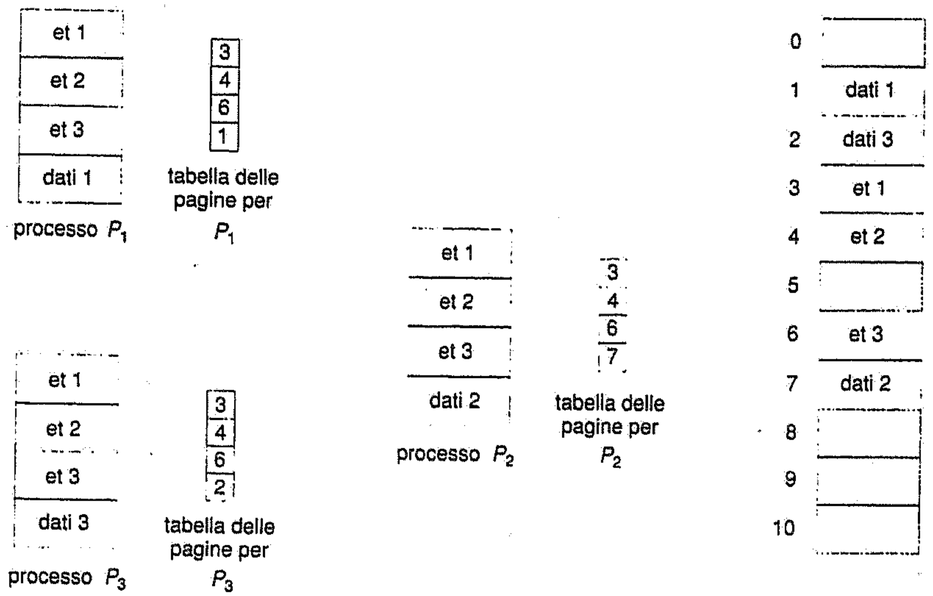


Figura 9.16 Condivisione di codice in un ambiente paginato.

laboratore di testi, mentre le pagine dei dati si fanno corrispondere a blocchi di memoria diversi. Quindi per gestire 40 utenti sono sufficienti una copia dell'elaboratore di testi (150 KB) e 40 copie dei 50 KB di spazio di dati per ciascun utente; per un totale di 2150 KB, invece di 8000 KB; il risparmio è notevole.

Si possono condividere anche altri programmi d'uso frequente: compilatori, interfacce a finestre, librerie a collegamento dinamico, sistemi di basi di dati e così via. Per essere condivisibile, il codice deve essere rientrante. La natura di sola lettura del codice condiviso non si può affidare alla sola correttezza intrinseca del codice stesso, ma deve essere fatta rispettare dal sistema operativo. Tale condivisione della memoria tra processi di un sistema è simile al modo in cui i thread condividono lo spazio d'indirizzi di un task (Capitolo 5). Inoltre con riferimento al Capitolo 4, dove si descrive la memoria condivisa come un metodo di comunicazione tra processi, alcuni sistemi operativi realizzano la memoria condivisa impiegando le pagine condivise.

Nei sistemi che fanno uso delle tabelle delle pagine invertite, la condivisione della memoria è di più difficile realizzazione. Infatti, mentre la memoria condivisa di solito si realizza facendo corrispondere più indirizzi virtuali (uno per ogni processo che condivide la memoria) allo stesso indirizzo fisico, tale metodo non si può usare in questo caso, poiché è presente un solo elemento di pagina virtuale per ogni pagina fisica, perciò una pagina fisica non può corrispondere a più pagine virtuali, e quindi essere condivisa.

Oltre a permettere che più processi condividano le stesse pagine fisiche, l'organizzazione della memoria in pagine offre numerosi altri benefici; alcuni sono trattati nel Capitolo 10.

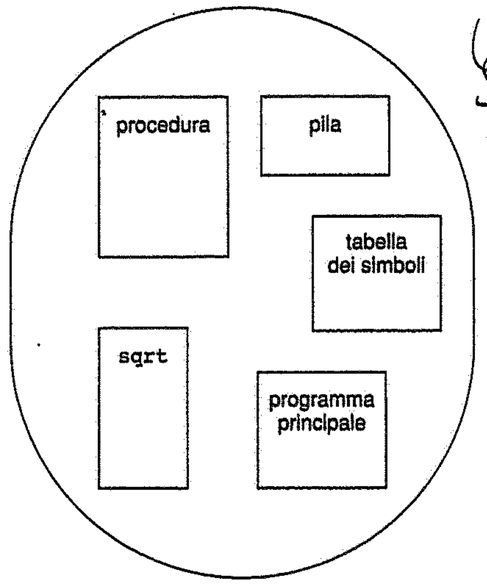
9.5 Segmentazione

Un aspetto importante della gestione della memoria, inevitabile alla presenza della paginazione, è quello della separazione tra la visione della memoria dell'utente e l'effettiva memoria fisica. Lo spazio d'indirizzi 'visto' dall'utente non è quello dell'effettiva memoria fisica, ma si fa corrispondere alla memoria fisica. I metodi che stabiliscono questa corrispondenza consentono di separare la memoria logica dalla memoria fisica.

9.5.1 Metodo di base

Ci si potrebbe chiedere se l'utente può considerare la memoria come un vettore lineare di byte, alcuni dei quali contengono istruzioni e altri dati. Molti risponderebbero di no. Gli utenti la vedono piuttosto come un insieme di segmenti di dimensione variabile non necessariamente ordinati (Figura 9.17).

La tipica struttura di un programma con cui i programmatori hanno familiarità è costituita di una parte principale e di un gruppo di procedure, funzioni o moduli, insieme con diverse strutture di dati come tabelle, matrici, pile, variabili e così via. Ciascuno di questi moduli o elementi di dati si identifica con un nome: "tabella dei simboli", "funzione `sqrt`", "programma principale"; indipendentemente dagli indirizzi che questi ele-



In questo caso l'idea dei blocchi della paginazione non va + bene xché non abbiamo + parti della stessa dimensione.

Figura 9.17 Un programma dal punto di vista dell'utente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

menti occupano nella memoria. Non è necessario preoccuparsi del fatto che la tabella dei simboli sia memorizzata prima o dopo la funzione `sqrt`. Ciascuno di questi segmenti ha una lunghezza variabile, definita intrinsecamente dallo scopo che il segmento stesso ha all'interno del programma. Gli elementi che si trovano all'interno di un segmento sono identificati dal loro scostamento, misurato dall'inizio del segmento: la prima istruzione del programma, il diciassettesimo elemento della tabella dei simboli, la quinta istruzione della funzione `sqrt`, e così via.

La **segmentazione** è uno schema di gestione della memoria che consente di gestire questa visione della memoria da parte dell'utente. Uno spazio d'indirizzi logici è una raccolta di segmenti, ciascuno dei quali ha un nome e una lunghezza. Gli indirizzi specificano sia il nome sia lo scostamento all'interno del segmento, quindi l'utente fornisce ogni indirizzo come una coppia ordinata di valori: un nome di segmento e uno scostamento. Questo schema contrasta con la paginazione, nella quale l'utente fornisce un indirizzo singolo, che l'architettura di paginazione divide in un numero di pagina e uno scostamento, non visibili dal programmatore.

Per semplicità i segmenti sono numerati, e ogni riferimento si compie per mezzo di un numero anziché di un nome; quindi un indirizzo logico è una *coppia*

<numero di segmento, scostamento>

Normalmente il programma utente è stato compilato, e il compilatore struttura automaticamente i segmenti secondo il programma sorgente. Un compilatore per il linguaggio Pascal può creare segmenti distinti per i seguenti elementi di un programma:

1. le variabili globali;
2. la pila per le chiamate di procedure (per memorizzare i parametri e gli indirizzi di ritorno);
3. il codice di ogni procedura e funzione;
4. le variabili locali di ciascuna procedura e funzione.

Un compilatore per il linguaggio FORTRAN crea un segmento separato per ogni blocco comune. Ai vettori si possono assegnare segmenti distinti. Il caricatore preleva questi segmenti e assegna loro i numeri di segmento.

9.5.2 Architettura di segmentazione

Sebbene l'utente possa fare riferimento agli oggetti del programma per mezzo di un indirizzo bidimensionale, la memoria fisica è in ogni caso una sequenza di byte unidimensionale. Per questo motivo occorre tradurre gli indirizzi bidimensionali definiti dall'utente negli indirizzi fisici unidimensionali. Questa operazione si compie tramite una **tabella dei segmenti**; ogni suo elemento è una coppia ordinata: la *base* del segmento e il *limite* del segmento. La base del segmento contiene l'indirizzo fisico iniziale della memoria al quale il segmento risiede, il limite del segmento contiene la lunghezza del segmento.

L'STBR punta alla locazione in memoria della tabella dei segmenti
L'STLR indica il numero di segmenti utilizzati dal prog.

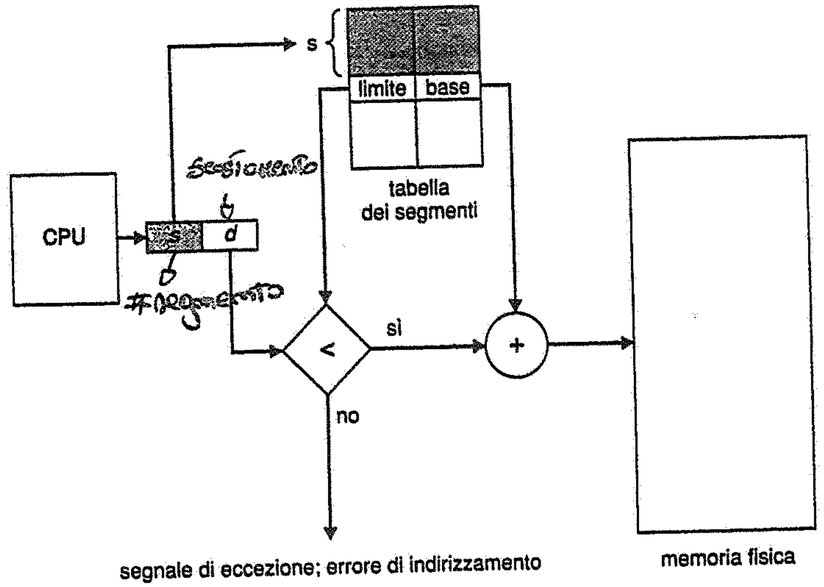


Figura 9.18 Architettura di segmentazione.

L'uso della tabella dei segmenti è illustrato nella Figura 9.18. Un indirizzo logico è formato da due parti: un numero di segmento *s* e uno scostamento in tale segmento *d*. Il numero di segmento si usa come indice per la tabella dei segmenti; lo scostamento *d* dell'indirizzo logico deve essere compreso tra 0 e il limite del segmento, altrimenti s'invia un segnale di eccezione al sistema operativo (tentativo di indirizzamento logico oltre la fine del segmento). Se tale condizione è rispettata, si somma lo scostamento alla base del segmento per produrre l'indirizzo della memoria fisica a cui si trova il byte desiderato. Quindi la tabella dei segmenti è fondamentalmente un vettore di coppie di registri di base e di limite.

Come esempio si può considerare la situazione illustrata nella Figura 9.19. Sono dati cinque segmenti numerati da 0 a 4, memorizzati nella memoria fisica. La tabella dei segmenti ha un elemento distinto per ogni segmento, indicante l'indirizzo iniziale del segmento nella memoria fisica (la base) e la lunghezza di quel segmento (il limite). Ad esempio, il segmento 2 è lungo 400 byte e comincia alla locazione 4300, quindi un riferimento al byte 53 del segmento 2 si fa corrispondere alla locazione 4300 + 53 = 4353. Un riferimento al segmento 3, byte 852, si fa corrispondere alla locazione 3200 (la base del segmento 3) + 852 = 4052. Un riferimento al byte 1222 del segmento 0 causa l'invio di un segnale di eccezione al sistema operativo, poiché questo segmento è lungo 1000 byte.

• Ricollocazione dinamica
• condivisione = si ottengono segmenti condivisi puntando allo stesso indirizzo di segmento.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

• La segmentazione in maniera notevole si permette di condividere segmenti di codice e segmenti di dati -

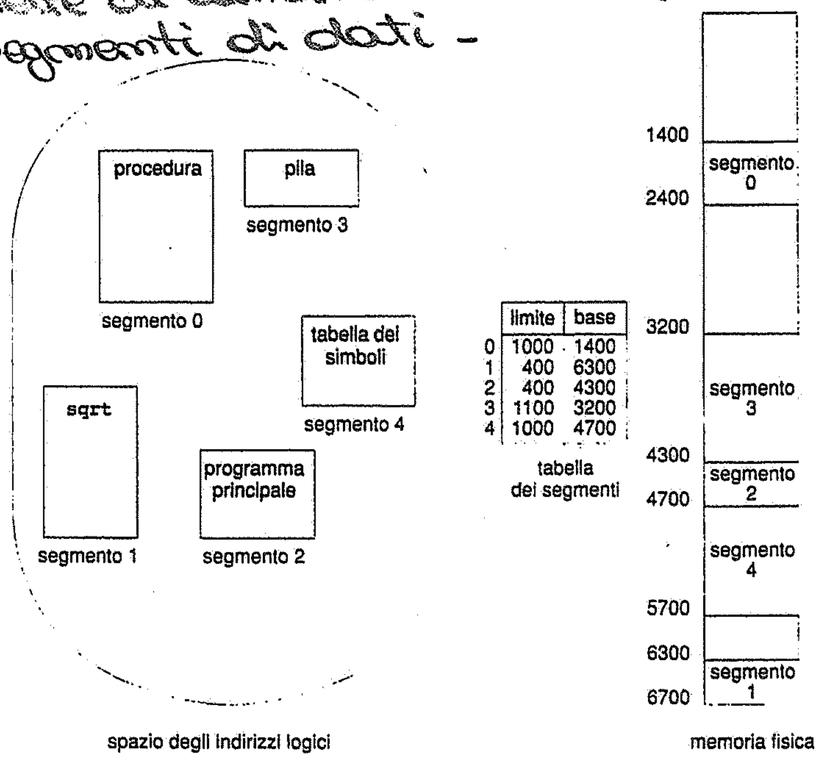


Figura 9.19 Esempio di segmentazione.

9.5.3 Protezione e condivisione

Un vantaggio particolare dato dalla segmentazione è l'associazione della protezione con i segmenti. Poiché i segmenti rappresentano una porzione del programma ben definita dal punto di vista semantico, è probabile che tutti gli elementi del segmento siano usati nello stesso modo; alcuni segmenti contengono istruzioni, altri contengono dati. In un'architettura moderna le istruzioni sono non automodificanti, quindi i segmenti contenenti istruzioni si possono definire come di sola lettura o di sola esecuzione. L'architettura di traduzione degli indirizzi della memoria controlla i bit di protezione associati a ogni elemento della tabella dei segmenti, in modo da impedire accessi illegali alla memoria, ad esempio tentativi di scrittura in un segmento di sola lettura oppure tentativi d'uso di un segmento di sola esecuzione come un insieme di dati. Se si costruisce un segmento che contiene solo un vettore, l'architettura di gestione della memoria controlla automaticamente che gli indici del vettore siano legali e non fuoriescano dai suoi limiti. In questo modo si possono individuare molti errori di programmazione prima che questi possano causare danni gravi.

Un altro vantaggio della segmentazione riguarda la *condivisione* di codice o di dati. Ogni processo ha una tabella di segmenti, associata al proprio PCB, che il dispatcher usa per definire la tabella fisica dei segmenti quando assegna la CPU al processo. I segmenti si condividono se gli elementi delle tabelle dei segmenti di due processi diversi puntano alle stesse locazioni fisiche (Figura 9.20).

Poiché la condivisione avviene al livello dei segmenti, si può condividere qualsiasi informazione definita come un segmento, e siccome si possono condividere parecchi segmenti, si può condividere anche un programma composto da più segmenti.

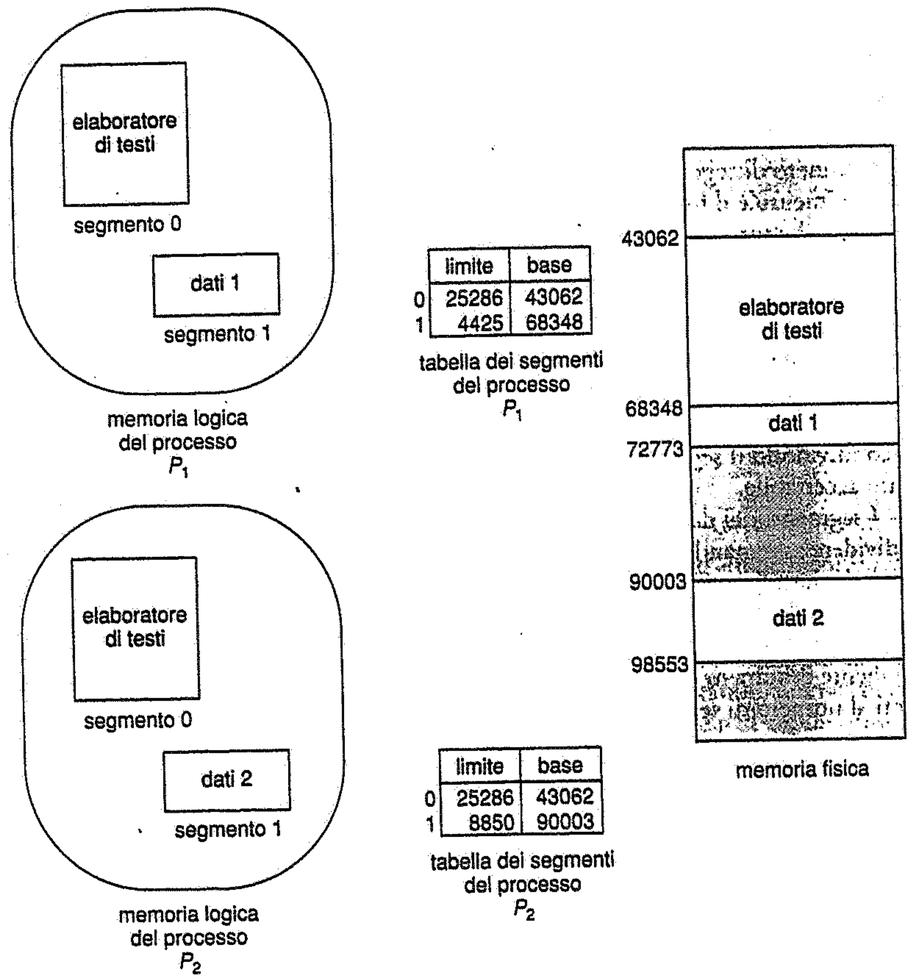


Figura 9.20 Condivisione di segmenti in un sistema con memoria segmentata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Si consideri, ad esempio, l'uso di un elaboratore di testi in un sistema a partizione del tempo. Un intero programma di questo tipo può essere abbastanza grande, e quindi essere composto di molti segmenti, che tutti gli utenti possono condividere, limitando la quantità di memoria fisica necessaria. Anche in questo caso si carica una sola copia dell'elaboratore di testi, anziché *n* copie. Ogni utente, però, necessita di segmenti unici e distinti nei quali memorizzare le variabili locali; naturalmente questi segmenti non si possono condividere.

Si possono anche condividere solo alcune parti di un programma; più utenti ad esempio possono condividere insiemi di procedure e funzioni comuni, ammesso che siano definiti come segmenti condivisibili di sola lettura. Due programmi scritti in FORTRAN, ad esempio, possono usare la stessa funzione `sqrt`, ma è necessaria una sola sua copia fisica.

Sebbene questo tipo di condivisione sembri semplice, è necessario fare alcune sottili considerazioni. Tipicamente i segmenti di codice contengono riferimenti a se stessi; ad esempio, un salto condizionale ha generalmente un indirizzo di trasferimento costituito da un numero di segmento e da uno scostamento. Il numero di segmento nell'indirizzo di trasferimento è il numero del segmento di codice. Per condividere questo segmento è necessario che tutti i processi coinvolti nella condivisione associno al segmento di codice condiviso lo stesso numero di segmento.

Se ad esempio si presenta la necessità di condividere la funzione `sqrt`, e un processo intende denotarla come segmento 4 e un altro come segmento 17, occorre stabilire come la procedura `sqrt` debba fare riferimento a se stessa. Poiché esiste una sola copia fisica di `sqrt`, questa deve fare riferimento a se stessa sempre nello stesso modo, deve avere, in altre parole, un unico numero di segmento. Aumentando il numero degli utenti che condividono il segmento, aumenta anche la difficoltà di trovare un numero di segmento accettabile.

I segmenti dei dati di sola lettura, che non contengono puntatori fisici, si possono condividere assegnando a ciascun segmento un numero diverso, così come i segmenti di codice che non fanno riferimento a se stessi in modo diretto, ma solo in modo indiretto. Ad esempio, le istruzioni condizionali che specificano l'indirizzo di diramazione, come uno scostamento dal valore corrente del contatore di programma o relativo a un registro contenente il numero del segmento corrente, permettono al codice di evitare riferimenti diretti al numero di segmento corrente.

9.5.4 Frammentazione

Lo scheduler a lungo termine deve trovare e assegnare spazio di memoria per tutti i segmenti di un programma utente. Questa situazione è analoga alla paginazione, con la differenza che in questo caso i segmenti hanno lunghezza *variabile*, mentre le pagine hanno tutte le stesse dimensioni. Quindi, come accade anche con lo schema delle partizioni di dimensione variabile, l'assegnazione della memoria diventa un problema di assegnazione dinamica, che generalmente si risolve con gli algoritmi descritti nel Paragrafo 9.3.2.

La segmentazione può causare anche frammentazione esterna se tutti i blocchi di memoria libera sono troppo piccoli perché contengano un segmento. In questo caso è possibile ritardare il processo fino a che non si renda disponibile più memoria (o almeno un buco più grande), oppure finché la compattazione crei un buco più grande. La segmentazione è, per sua natura, un algoritmo di rilocazione dinamico, grazie al quale la memoria si può compattare in qualsiasi momento. Se lo scheduler della CPU deve attendere un processo a causa di un problema di assegnazione di memoria, può (o non può) esaminare la coda d'attesa per la CPU alla ricerca di un processo da eseguire che sia più piccolo e con priorità più bassa.

Per stabilire la serietà di un problema di frammentazione esterna in uno schema di segmentazione, e decidere se questo si possa risolvere per mezzo dello scheduling a lungo termine e con la compattazione, occorre conoscere la dimensione media dei segmenti. Un caso estremo, che riconduce allo schema delle partizioni di dimensione variabile, è l'associazione di un segmento a ogni processo; all'estremo opposto, si ha l'associazione di un proprio segmento a ogni byte, e ciascun byte è rilocato separatamente. Questa soluzione elimina la frammentazione esterna, ma ogni byte necessita di un registro di base per essere rilocato, quindi la quantità di memoria richiesta raddoppia. Naturalmente, il passo logico successivo — piccoli segmenti di dimensione fissa — è la paginazione. In genere, se la dimensione media dei segmenti è piccola, è piccola anche la frammentazione esterna (si consideri per analogia la collocazione di un insieme di valigie nel baule di un'automobile; lo spazio a disposizione si sfrutta meglio disponendo gli oggetti singolarmente anziché contenuti nelle valigie). Poiché i singoli segmenti sono più piccoli dell'intero processo, è più probabile che riescano a entrare nei blocchi di memoria disponibili.

9.6 Segmentazione con paginazione

La paginazione e la segmentazione presentano vantaggi e svantaggi. Infatti, considerando due tipi di CPU tra i più diffusi quali la serie Motorola 68000, basata su uno spazio d'indirizzi piatto, e la famiglia Intel 80*x*86, basata sulla segmentazione si vede come in entrambi i casi s'impiegano modelli di memoria combinati, che presentano sia caratteristiche di paginazione sia di segmentazione. Questi due metodi si combinano per migliorarsi a vicenda. Tale combinazione è illustrata nel migliore dei modi dall'architettura dell'Intel 80386.

Per la gestione della memoria, la CPU Intel 80386 adotta la segmentazione con paginazione, il numero massimo di segmenti per processo è 16 KB, ciascun segmento può avere dimensioni fino a 4 GB e la dimensione delle pagine è di 4 KB. Questa discussione non intende fornire una descrizione completa di come la CPU Intel 80386 gestisce la memoria, ma piuttosto presentarne le idee fondamentali.

Lo spazio degli indirizzi logici di un processo è diviso in due partizioni: la prima contiene segmenti di dimensioni fino a 8 KB riservati al processo; la seconda contiene segmenti di dimensioni fino a 8 KB, condivisi fra tutti i processi. Le informazioni riguardanti la prima partizione sono contenute nella **tabella locale dei descrittori** (*local*

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

descriptor table — LDT), quelle relative alla seconda partizione sono memorizzate nella **tabella globale dei descrittori** (*global descriptor table* — GDT). Ciascun elemento nella LDT e nella GDT è composto da 8 byte e contiene informazioni dettagliate riguardanti uno specifico segmento, tra cui l'indirizzo della base e la lunghezza.

Un indirizzo logico è una coppia <*selettore*, *scostamento*>, il selettore è un numero di 16 bit avente la seguente forma:

s	g	p
13	1	2

dove s indica il numero del segmento, g indica se il segmento si trova nella GDT o nella LDT e p contiene informazioni relative alla protezione. Lo scostamento è un numero di 32 bit che indica la posizione del byte (o parola) all'interno del segmento in questione.

La macchina ha sei registri di segmento che consentono a un processo di fare riferimento contemporaneamente a sei segmenti; inoltre possiede sei registri di microprogramma di 8 byte per i corrispondenti descrittori prelevati dalla LDT o dalla GDT. Questa cache evita all'Intel 80386 di dover prelevare dalla memoria i descrittori per ogni riferimento alla memoria.

Un indirizzo fisico dell'Intel 80386 è lungo 32 bit e si genera come segue: il registro del segmento punta all'elemento appropriato all'interno della LDT o della GDT; le informazioni relative alla base e al limite di tale segmento si usano per generare un indirizzo lineare. Innanzi tutto si usa il valore del limite per controllare la validità dell'indirizzo; se non è valido, si ha un errore di riferimento alla memoria che causa l'invio di un segnale di eccezione al sistema operativo; altrimenti si somma il valore dello scostamento al valore della base, ottenendo un indirizzo lineare di 32 bit, che a sua volta è tradotto in un indirizzo fisico.

Come s'è accennato, ogni segmento è paginato in pagine di 4 KB ciascuna; ciò implica la possibilità di avere tabelle delle pagine contenenti fino a 1 milione di elementi. Poiché ciascun elemento della tabella delle pagine occupa 4 byte, ogni processo potrebbe richiedere 4 MB di spazio fisico per la sola tabella delle pagine. Chiaramente, sarebbe meglio evitare di collocare la tabella delle pagine in un'area contigua della memoria centrale. L'Intel 80386 risolve il problema adottando uno schema di paginazione a due livelli. L'indirizzo lineare è suddiviso in un numero di pagina di 20 bit e uno scostamento di pagina di 12 bit. Essendo paginata anche la tabella delle pagine, il numero di pagina è ulteriormente suddiviso in un puntatore, di 10 bit, alla directory delle pagine e in un puntatore alla tabella delle pagine, anch'esso di 10 bit. L'indirizzo logico ha quindi la forma seguente:

numero di pagina		scostamento di pagina
p_1	p_2	d
10	10	13

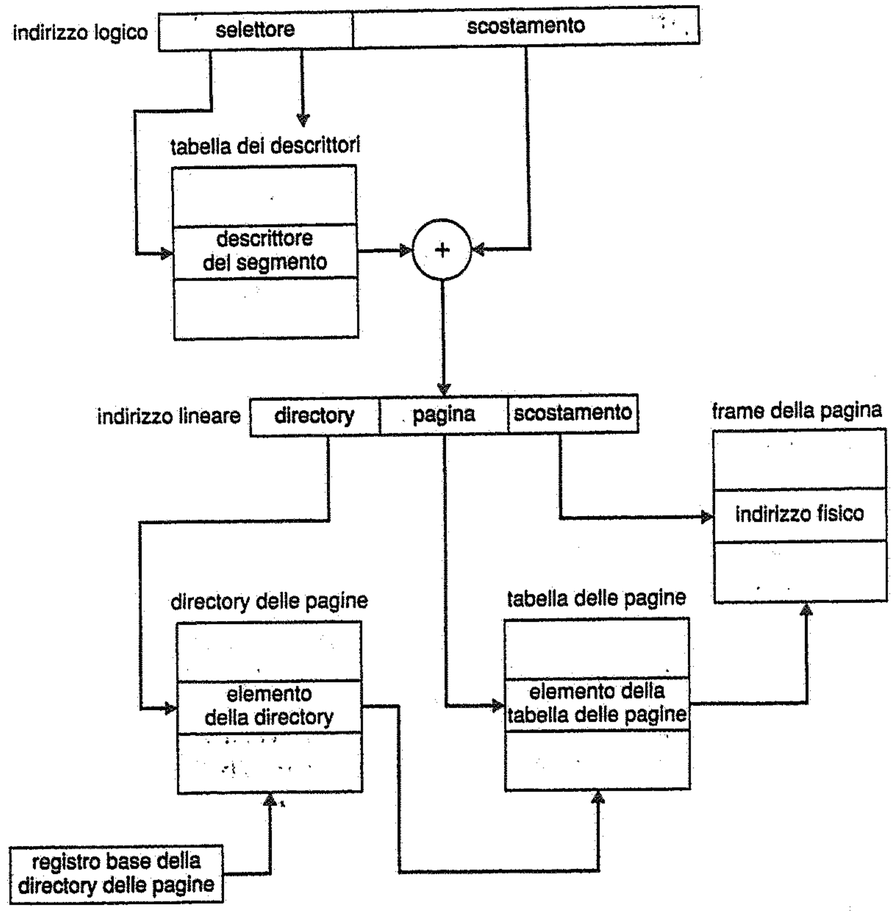


Figura 9.21 Traduzione degli indirizzi nell'Intel 80386.

Lo schema di traduzione degli indirizzi adottato in questa architettura è simile a quello mostrato nella Figura 9.13. La traduzione degli indirizzi dell'Intel 80386 è illustrata con maggiori dettagli nella Figura 9.21. Per migliorare l'efficienza nell'utilizzo della memoria fisica, le tabelle delle pagine dell'Intel 80386 si possono trasferire nella memoria secondaria. In questo caso si aggiunge un bit di validità a ciascun elemento della directory delle pagine, che indica se la tabella cui l'elemento fa riferimento si trova nella memoria centrale o nella memoria secondaria. In quest'ultimo caso il sistema operativo può usare i restanti 31 bit per indicare la posizione della tabella nella memoria secondaria; quindi, la tabella delle pagine si può caricare, a richiesta, nella memoria centrale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.7 Sommario

Gli algoritmi di gestione della memoria per sistemi operativi multiprogrammati variano da un metodo semplice, per sistema con singolo utente, fino alla segmentazione paginata. Il fattore che incide maggiormente sulla scelta del metodo da seguire in un sistema particolare è costituito dall'architettura disponibile. È necessario controllare la validità di ogni indirizzo di memoria generato dalla CPU; inoltre tali indirizzi, se sono corretti, devono essere tradotti in un indirizzo fisico. Tali controlli per essere efficienti devono essere effettuati dall'architettura del sistema, che secondo le sue caratteristiche pone vincoli al sistema operativo.

Gli algoritmi di gestione della memoria analizzati (assegnazione contigua, paginazione, segmentazione e combinazione di paginazione e segmentazione) differiscono per molti aspetti. Per confrontare i diversi metodi di gestione della memoria si possono considerare i seguenti elementi:

- **Architettura.** Un semplice registro di base oppure una coppia di registri di base e di limite è sufficiente per i metodi con partizione singola e con più partizioni, mentre la paginazione e la segmentazione necessitano di tabelle di traduzione per definire la corrispondenza degli indirizzi.
- **Prestazioni.** Aumentando la complessità dell'algoritmo, aumenta anche il tempo necessario per tradurre un indirizzo logico in un indirizzo fisico. Nei sistemi più semplici è sufficiente fare un confronto o sommare un valore all'indirizzo logico; si tratta di operazioni rapide. La paginazione e la segmentazione possono essere altrettanto rapide se per realizzare la tabella s'impiegano registri veloci. Se però la tabella si trova nella memoria, gli accessi alla memoria d'utente possono essere assai più lenti. Un TLB può limitare il calo delle prestazioni a un livello accettabile.
- **Frammentazione.** Un sistema multiprogrammato esegue le elaborazioni generalmente in modo più efficiente se ha un più elevato livello di multiprogrammazione. Per un dato gruppo di processi, il livello di multiprogrammazione si può aumentare solo compattando più processi nella memoria. Per eseguire questo compito occorre ridurre lo spreco di memoria o la frammentazione. Sistemi con unità di assegnazione di dimensione fissa, come lo schema con partizione singola e la paginazione, soffrono di frammentazione interna. Sistemi con unità di assegnazione di dimensione variabile, come lo schema con più partizioni e la segmentazione, soffrono di frammentazione esterna.
- **Rilocazione.** Una soluzione al problema della frammentazione esterna è data dalla compattazione. Che implica lo spostamento di un programma nella memoria, senza che il programma stesso si 'accorga' del cambiamento. Ciò richiede che gli indirizzi logici siano rilocati dinamicamente al momento dell'esecuzione. Se gli indirizzi si rilocano solo al momento del caricamento, non è possibile compattare la memoria.

- **Avvicendamento dei processi.** L'avvicendamento dei processi (*swapping*) si può incorporare in ogni algoritmo. I processi si copiano dalla memoria centrale alla memoria ausiliaria, e successivamente si ricopiano nella memoria centrale a intervalli fissati dal sistema operativo, e generalmente stabiliti dai criteri di scheduling della CPU. Questo schema permette di inserire contemporaneamente nella memoria più processi da eseguire.
- **Condivisione.** Un altro mezzo per aumentare il livello di multiprogrammazione è quello della condivisione del codice e dei dati tra diversi utenti. Poiché deve fornire piccoli pacchetti d'informazioni (pagine o segmenti) che si possano condividere, generalmente la condivisione richiede l'uso della paginazione o della segmentazione. La condivisione permette di eseguire molti processi con una quantità di memoria limitata, ma i programmi e i dati condivisi si devono progettare con estrema cura.
- **Protezione.** Con la paginazione o la segmentazione, diverse sezioni di un programma d'utente si possono dichiarare di sola esecuzione, di sola lettura oppure di lettura e scrittura. Questa limitazione è necessaria per il codice e i dati condivisi, ed è utile, in genere, nei casi in cui sono richiesti semplici controlli nella fase d'esecuzione per l'individuazione degli errori di programmazione.

9.8 Esercizi

9.1 Esponete due differenze tra gli indirizzi logici e gli indirizzi fisici.

9.2 Spiegate la differenza tra la frammentazione interna e la frammentazione esterna.

9.3 Descrivete e commentate i seguenti criteri di assegnazione dei buchi di memoria liberi:

a) primo buco abbastanza grande (*first-fit*);
b) più piccolo tra i buchi abbastanza grandi (*best-fit*);
c) più grande tra i buchi abbastanza grandi (*worst-fit*).

9.4 Date le seguenti partizioni di memoria: 100 KB, 500 KB, 200 KB, 300 KB e 600 KB (nell'ordine), descrivete come sono disposti dagli algoritmi dell'Esercizio 9.3 i processi di 212 KB, 417 KB, 112 KB e 426 KB (nell'ordine). Dite quale di questi algoritmi utilizza la memoria nel modo più efficiente.

9.5 Se un processo viene estratto dalla memoria centrale, perde la capacità di servirsi della CPU (almeno per un istante). Descrivete un'altra situazione in cui un processo perde la capacità di servirsi della CPU, nella quale, però, il processo resta nella memoria centrale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

VANTAGGI → Risparmio di memoria
SVANTAGGI → I dati possono essere modificati

9.6 Considerate un sistema nel quale un programma si può dividere in due parti: codice e dati. La CPU sa se ha bisogno di un'istruzione (prelievo di un'istruzione) o di dati (prelievo o scrittura, nella memoria, di dati). Sono quindi fornite due coppie di registri di base e di limite: una per le istruzioni e una per i dati. La coppia di registri di base e di limite riservata alle istruzioni è automaticamente impostata per la sola lettura, quindi più utenti possono condividere i programmi. Discutete vantaggi e svantaggi di tale schema.

9.7 Spiegate perché le dimensioni delle pagine sono sempre potenze di 2. → Poiché facilita notevolmente la traduzione da indirizzo fisico a logico

9.8 Considerate uno spazio d'indirizzi logici di otto pagine di 1024 parole ciascuna, cui corrisponde una memoria fisica di 32 blocchi di memoria. Calcolate i seguenti valori: $1024 = 2^{10}$ 2^5

a) il numero dei bit presenti nell'indirizzo logico; 10-8=7
b) il numero dei bit presenti nell'indirizzo fisico. 3

9.9 Spiegate perché in un sistema con paginazione un processo non può accedere a locazioni di memoria che non possiede. Dite in che modo il sistema operativo può consentire l'accesso a tale memoria; spiegate perché ciò è possibile, o, nel caso contrario, perché non lo è.

9.10 Considerate un sistema di paginazione con la tabella delle pagine nella memoria centrale.

a) Se un riferimento alla memoria richiede 200 nanosecondi, dite quanto tempo richiede un riferimento a una memoria paginata. 200 ns

b) Se si aggiungono TLB e il 75 per cento di tutti i riferimenti alla tabella delle pagine si trova in questi ultimi, dite qual è il tempo effettivo di riferimento alla memoria. (Supponete che il reperimento di un elemento presente nella tabella delle pagine contenuta nei TLB richieda un tempo nullo.)

$\alpha = 0{,}75$
$\varepsilon = 0$
$\gamma = 200$
$(200+0)\,0{,}75 + (400+0)\,0{,}25 = 200 \cdot 0{,}75 + 400 \cdot 0{,}25$

9.11 Dite cosa accadrebbe se si permettesse a due elementi di una tabella delle pagine di puntare allo stesso blocco di memoria. Spiegate come si può usare questo metodo per ridurre il tempo necessario a copiare o spostare il contenuto di una vasta area della memoria. Dite quale sarebbe l'effetto causato su una pagina dall'aggiornamento dell'altra. → Che si può condividere i dati

Impostando i bit di validità

9.12 Spiegate perché talvolta la segmentazione e la paginazione si combinano in un unico schema. X separare la memoria logica dalla fisica

9.13 Descrivete un meccanismo che consenta a un segmento di appartenere allo spazio d'indirizzi di due processi diversi. Si condivide il segmento tra processo A e B

Poiché la condivisione avviene x mezzo dei segmenti

9.14 Spiegate perché è più facile condividere un modulo di codice rientrante usando la segmentazione anziché la paginazione pura. Uno x blocco del S.O.

9.15 Nei sistemi con segmentazione e collegamento dinamico si possono condividere i segmenti tra i processi senza che i segmenti debbano avere lo stesso numero.

a) Definite un sistema che permetta il collegamento statico e la condivisione di segmenti senza richiedere che questi abbiano gli stessi numeri.

b) Descrivete uno schema di paginazione che permetta la condivisione delle pagine senza richiedere che i numeri delle pagine siano gli stessi.

9.16 Considerate la seguente tabella dei segmenti:

Segmento	Base	Lunghezza
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Segmento

Calcolate gli indirizzi fisici corrispondenti ai seguenti indirizzi logici:

a) <0, 430>
b) <1, 10>
c) <2, 500>
d) <3, 400>
e) <4, 112>

bisogna verificare che questi numeri siano < dalla lunghezza se sono è certo si può calcolare l'indirizzo altrimenti no.

9.17 Considerate lo schema di traduzione degli indirizzi dell'Intel 80386 mostrato nella Figura 9.21.

a) Descrivete tutti i passi eseguiti dall'Intel 80386 nel tradurre un indirizzo logico in un indirizzo fisico.

b) Esponete i vantaggi offerti a un sistema operativo da un'architettura dotata di un così complesso sistema di traduzione degli indirizzi.

c) Dite se ci sono svantaggi in questo sistema di traduzione degli indirizzi; se sì, dite quali sono; altrimenti spiegate perché non è impiegato da tutti i costruttori di CPU.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

9.18 Nell'IBM/370 la memoria è protetta tramite *chiavi*; si tratta di quantità di 4 bit. Si associa una chiave a ogni blocco di memoria di 2 KB (la chiave di memoria) e una chiave alla CPU (la chiave di protezione). Un'operazione di memorizzazione è permessa solo se le due chiavi sono uguali, oppure se una è nulla. Dite quali tra i seguenti metodi di gestione della memoria si potrebbe usare con successo con questo tipo d'architettura:

a) nuda macchina;
b) sistema con utente singolo;
c) multiprogrammazione con un numero fisso di processi;
d) multiprogrammazione con un numero variabile di processi;
e) paginazione;
f) segmentazione.

9.9 Note bibliografiche

L'assegnazione dinamica della memoria è discussa nel Paragrafo 2.5 di [Knuth 1973], che, tramite i risultati di simulazioni, ha trovato che il criterio di scelta del primo buco abbastanza grande (*first-fit*) è in genere più vantaggioso del criterio di scelta del più piccolo tra i buchi abbastanza grandi (*best-fit*). [Knuth 1973] Discute la regola del 50 per cento.

Il concetto di paginazione si può attribuire ai progettisti del sistema Atlas, descritto da [Kilburn et al. 1961] e [Howarth et al. 1961]. Il concetto di segmentazione è stato discusso per la prima volta da [Dennis 1965]. Il primo sistema con segmentazione paginata è stato il GE 645, per il quale era stato originariamente progettato e realizzato il sistema operativo MULTICS [Organick 1972].

Le tabelle delle pagine per spazi di indirizzi a 64 bit (tabelle delle pagine a gruppi) sono trattate in [Talluri et al. 1995]. Le tabelle delle pagine invertite sono trattate in un articolo sulla gestione della memoria dell'IBM RT, [Chang e Mergen 1988].

[Jacob e Mudge 1997] tratta la traduzione degli indirizzi gestita dal sistema operativo, anziché dall'architettura.

Le memorie cache, comprese le memorie associative, sono descritte e analizzate da [Smith 1982], che contiene anche una vasta bibliografia sull'argomento. [Hennessy e Patterson 1996] tratta gli aspetti architetturali di TLB, cache e MMU. [Dougan et al. 1999] descrive una tecnica di gestione dei TLB.

La famiglia di CPU Motorola 68000 è descritta in [Motorola 1989a]. In [Intel 1986] e [Tanenbaum 2001] si trovano informazioni sull'architettura di paginazione dell'Intel 80386. L'architettura dell'Intel 80486 è trattata in [Intel 1989].

[Jacob e Mudge 1998a] descrive la gestione della memoria per diverse architetture: Pentium II, PowerPC, UltraSPARC, ecc.

Capitolo 10

Memoria virtuale

Nel Capitolo 9 si sono esaminate le strategie di gestione della memoria impiegate nei calcolatori. Hanno tutte lo stesso scopo: tenere contemporaneamente nella memoria più processi per permettere la multiprogrammazione; tuttavia esse tendono a richiedere che l'intero processo si trovi nella memoria prima di essere eseguito.

La **memoria virtuale** è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti nella memoria. Il vantaggio principale offerto da questa tecnica è quello di permettere che i programmi siano più grandi della memoria fisica; inoltre la memoria virtuale astrae la memoria centrale in un vettore molto grande e uniforme, separando la memoria logica, com'è vista dall'utente, dalla memoria fisica. Questa tecnica libera i programmatori da quel che riguarda i limiti della memoria. La memoria virtuale permette inoltre ai processi di condividere facilmente file e spazi d'indirizzi, e fornisce un meccanismo efficiente per la creazione dei processi

La memoria virtuale è però difficile da realizzare e, s'è usata scorrettamente, può ridurre di molto le prestazioni del sistema. In questo capitolo si esamina la memoria virtuale nella forma della paginazione su richiesta e se ne valutano la complessità e i costi.

10.1 Introduzione

Gli algoritmi di gestione della memoria delineati nel Capitolo 9 sono necessari perché, per l'attivazione di un processo, le istruzioni da eseguire si devono trovare all'interno della memoria fisica. Il primo metodo per far fronte a tale requisito consiste nel collocare l'intero spazio d'indirizzi logici del processo relativo nella memoria fisica. La sovrapposizione di sezioni (*overlay*) e il caricamento dinamico possono aiutare ad attenuare tale limitazione, ma generalmente richiedono particolari precauzioni e un ulteriore impegno dei programmatori. Questa limitazione sembra necessaria e ragionevole, ma riduce le dimensioni dei programmi a valori strettamente correlati alle dimensioni della memoria fisica.

ciò che accade con la Paginaz. e Segm.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

In molti casi l'intero programma non è infatti necessario; si considerino ad esempio le seguenti situazioni:

- Spesso i programmi dispongono di codice per la gestione di condizioni d'errore insolite. Poiché questi errori sono rari, se non inesistenti, anche i relativi segmenti di codice non si eseguono quasi mai.
- Spesso a matrici, liste e tabelle si assegna più memoria di quanta necessitino effettivamente. Una matrice si può dichiarare di 100 per 100 elementi, anche se raramente contiene più di 10 per 10 elementi. Una tabella dei simboli di un assemblatore può avere spazio per 3000 simboli, anche se un programma medio ha meno di 200 simboli.
- Alcune opzioni e caratteristiche di un programma possono essere usate solo di radò.

Anche nei casi in cui è necessario disporre di tutto il programma è possibile che non serva tutto in una volta, ad esempio nel caso della sovrapposizione di sezioni.

La possibilità di eseguire un programma che si trova solo parzialmente nella memoria può essere vantaggiosa per i seguenti motivi:

- Un programma non è più vincolato alla quantità di memoria fisica disponibile. Gli utenti possono scrivere programmi per uno **spazio degli indirizzi virtuali** molto grande, semplificando così le operazioni di programmazione.
- Poiché ogni utente impiega meno memoria fisica, si possono eseguire molti più programmi contemporaneamente, ottenendo un corrispondente aumento dell'utilizzo e della produttività della CPU senza aumentare il tempo di risposta o di completamento.
- Per caricare (o scaricare) ogni programma utente nella memoria sono necessarie meno operazioni di I/O, perciò ogni programma utente è eseguito più rapidamente.

Quindi la possibilità di eseguire un programma che non si trova completamente nella memoria apporterebbe vantaggi sia al sistema sia all'utente.

La **memoria virtuale** rappresenta la separazione della memoria logica, vista dall'utente, dalla memoria fisica. Questa separazione permette di offrire ai programmatori una memoria virtuale molto ampia anche se la memoria fisica disponibile è più piccola, com'è illustrato nella Figura 10.1. La memoria virtuale facilita la programmazione, poiché il programmatore non deve preoccuparsi della quantità di memoria fisica disponibile o di quale codice si debba inserire nelle sezioni sovrapponibili, ma può concentrarsi sul problema da risolvere; in un sistema dotato di memoria virtuale, la sovrapposizione di sezioni di fatto scompare.

Oltre alla separazione della memoria logica dalla memoria fisica, la memoria virtuale permette la condivisione di file e memoria tra diversi processi tramite la condivisione delle pagine (Paragrafo 9.4.5). La condivisione delle pagine consente anche un miglioramento delle prestazioni durante la creazione dei processi.

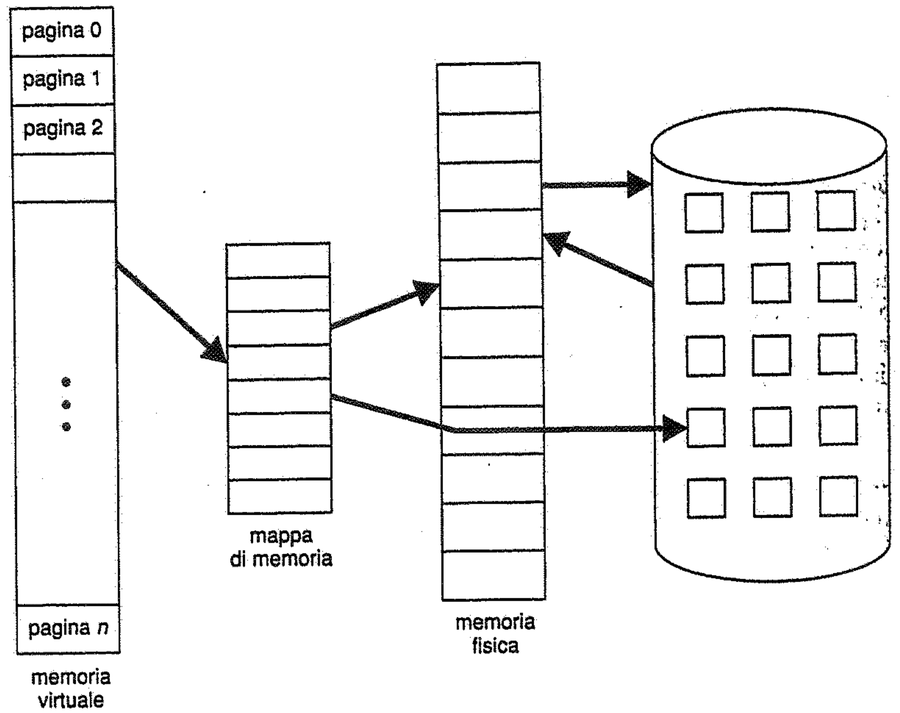


Figura 10.1 Schema di memoria virtuale più grande della memoria fisica.

La memoria virtuale generalmente si realizza nella forma della **paginazione su richiesta**, ma si può realizzare anche in un sistema a segmentazione. Parecchi sistemi offrono uno schema di segmentazione paginata, dove i segmenti sono suddivisi in pagine. Quindi l'utente vede la segmentazione, ma il sistema operativo può realizzare questa visione con la paginazione su richiesta. Per ottenere una memoria virtuale si può usare anche la **segmentazione su richiesta**. I calcolatori Burroughs hanno impiegato la segmentazione su richiesta. Anche il sistema operativo IBM OS/2 si serve della segmentazione su richiesta. Tuttavia, gli algoritmi di sostituzione dei segmenti sono più complicati degli algoritmi di sostituzione delle pagine, poiché i segmenti hanno dimensione variabile. La segmentazione su richiesta non è trattata in questo testo; si possono trovare alcune indicazioni nelle Note Bibliografiche.

http://www.adultpdf.com
Created by TIFF To PDF trial version, get complete version at www.adultpdf.com to remove this watermark, please register this software.

10.2 Paginazione su richiesta

Un sistema di paginazione su richiesta è analogo a un sistema paginato con avvicendamento dei processi nella memoria; si veda la Figura 10.2. I processi risiedono nella memoria secondaria, generalmente costituita di uno o più dischi. Per eseguire un processo occorre caricarlo nella memoria. Tuttavia, anziché caricare nella memoria l'intero processo, si può seguire un criterio d'avvicendamento 'pigro' (*lazy swapping*), non si carica mai nella memoria una pagina che non sia necessaria. Poiché ora si considera un processo come una sequenza di pagine, invece che come un unico ampio spazio d'indirizzi contiguo, l'uso del termine *avvicendamento dei processi* non è appropriato: non si manipolano interi processi ma singole pagine di processi. Nell'ambito della paginazione su richiesta, il modulo del sistema operativo che si occupa della sostituzione delle pagine si chiama **paginatore**.

10.2.1 Concetti fondamentali

Quando un processo sta per essere caricato nella memoria, il paginatore fa una predizione delle pagine che saranno usate, prima che il processo sia nuovamente scaricato dalla memoria. Anziché caricare nella memoria tutto il processo, il paginatore trasferisce nella memoria solo le pagine che ritiene necessarie. In questo modo è possibile evitare il trasferimento nella memoria di pagine che non sono effettivamente usate, riducendo il tempo d'avvicendamento e la quantità di memoria fisica richiesta.

Con tale schema è necessario che l'architettura disponga di un qualche meccanismo che consenta di distinguere le pagine presenti nella memoria da quelle nei dischi. A tal fine si può adoperare lo schema basato sul bit di validità descritto nel Paragrafo 9.4.3. In questo caso, però, il bit impostato come 'valido' significa che la pagina corrispondente è valida ed è presente nella memoria; il bit impostato come 'non valido' indica che la pagina non è valida (cioè non appartiene allo spazio d'indirizzi logici del processo) oppure è valida ma è attualmente nel disco. L'elemento della tabella delle pagine di una pagina caricata nella memoria s'imposta come il solito, mentre l'elemento della tabella delle pagine corrispondente a una pagina che attualmente non è nella memoria è semplicemente contrassegnato come non valido o contiene l'indirizzo che consente di trovare la pagina nei dischi. Tale situazione è illustrata nella Figura 10.3.

Occorre notare che indicare una pagina come non valida non sortisce alcun effetto se il processo non tenta mai di accedervi. Quindi, se la predizione del paginatore è esatta e si caricano tutte e solo le pagine che servono effettivamente, il processo è eseguito proprio come se fossero state caricate tutte le pagine. Durante l'esecuzione, il processo accede alle pagine **residenti nella memoria**, e l'esecuzione procede come di consueto.

Se il processo tenta di accedere a una pagina che non era stata caricata nella memoria, l'accesso a una pagina contrassegnata come non valida causa un'**eccezione di pagina mancante** (*page fault trap*). L'architettura di paginazione, traducendo l'indirizzo attraverso la tabella delle pagine, nota che il bit è non valido e invia un segnale di eccezione al sistema operativo; tale eccezione è dovuta a un 'insuccesso' del sistema operativo nella scelta delle pagine da caricare nella memoria (nel tentativo di limitare i requisiti di memoria

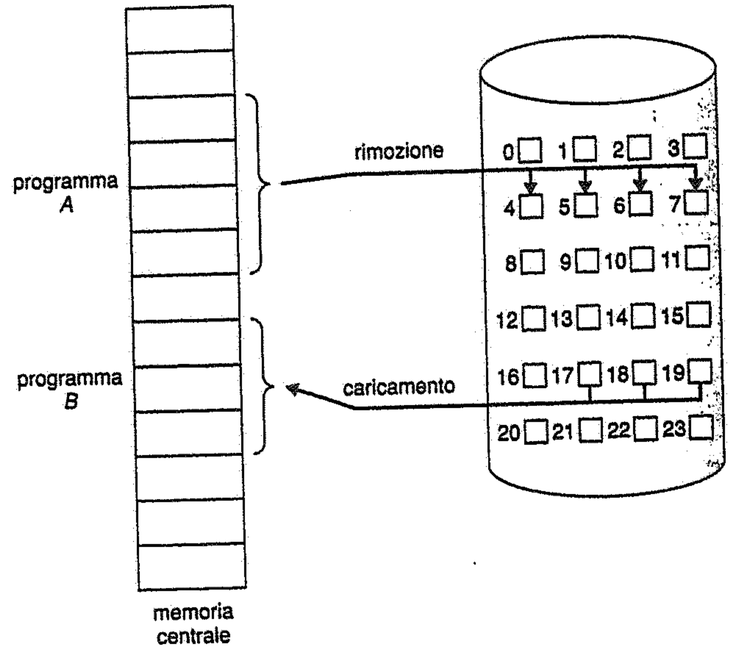


Figura 10.2 Trasferimento di una memoria paginata nello spazio contiguo di un disco.

e il carico dovuto al trasferimento dai dischi), piuttosto che a un errore di indirizzo non valido risultante dal tentativo di accedere a un indirizzo illegale della memoria (come un errato indice di un vettore). A tale 'insuccesso', detto **assenza di pagina** (*page fault*), si deve quindi rimediare. La procedura di gestione dell'eccezione di pagina mancante (Figura 10.4) è chiara:

1. Si controlla una tabella interna per questo processo; in genere tale tabella è conservata insieme con il descrittore di processo, allo scopo di stabilire se il riferimento fosse un accesso alla memoria valido o non valido.
2. Se il riferimento non era valido, si termina il processo. Se era un riferimento valido, ma la pagina non era ancora stata portata nella memoria, se ne effettua l'inserimento.
3. Si individua un blocco di memoria libero, ad esempio prelevandone uno dall'elenco dei blocchi liberi.
4. Si programma un'operazione sui dischi per trasferire la pagina desiderata nel blocco di memoria appena assegnato.
5. Quando la lettura dal disco è completata, si modificano la tabella interna, conservata con il processo, e la tabella delle pagine per indicare che la pagina attualmente si trova nella memoria.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

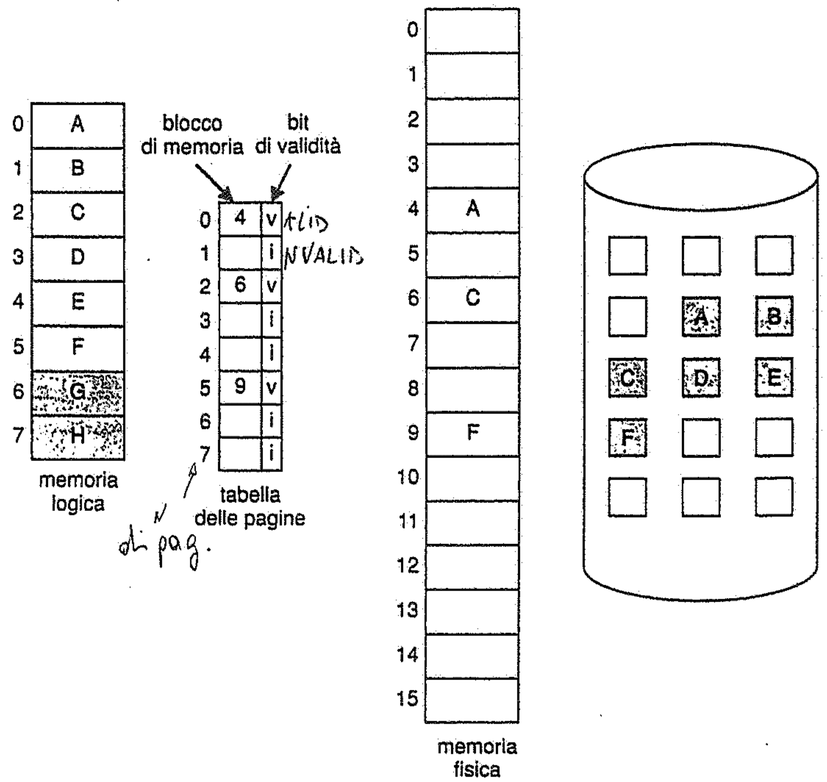


Figura 10.3 Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale.

6. Si riavvia l'istruzione che era stata interrotta dal segnale di eccezione. A questo punto il processo può accedere alla pagina come se questa fosse già presente nella memoria.

È importante capire che, salvando lo stato del processo (registri, codici di condizione, contatore di programma) interrotto col verificarsi dell'eccezione di pagina mancante, si può riavviare il processo *esattamente* nello stesso punto e nello stesso stato, con la sola differenza che ora la pagina desiderata si trova nella memoria ed è accessibile. In questo modo si può eseguire un processo anche se alcune delle sue parti non sono ancora presenti nella memoria. Quando il processo tenta di accedere a locazioni che non si trovano nella memoria, l'architettura del calcolatore emette un segnale d'eccezione per il sistema operativo (eccezione di pagina mancante). Il sistema operativo carica nella memoria la pagina desiderata e riavvia il processo come se la tale pagina fosse già presente nella memoria.

"ARCHITETTURA DEL PAGE FAULT"

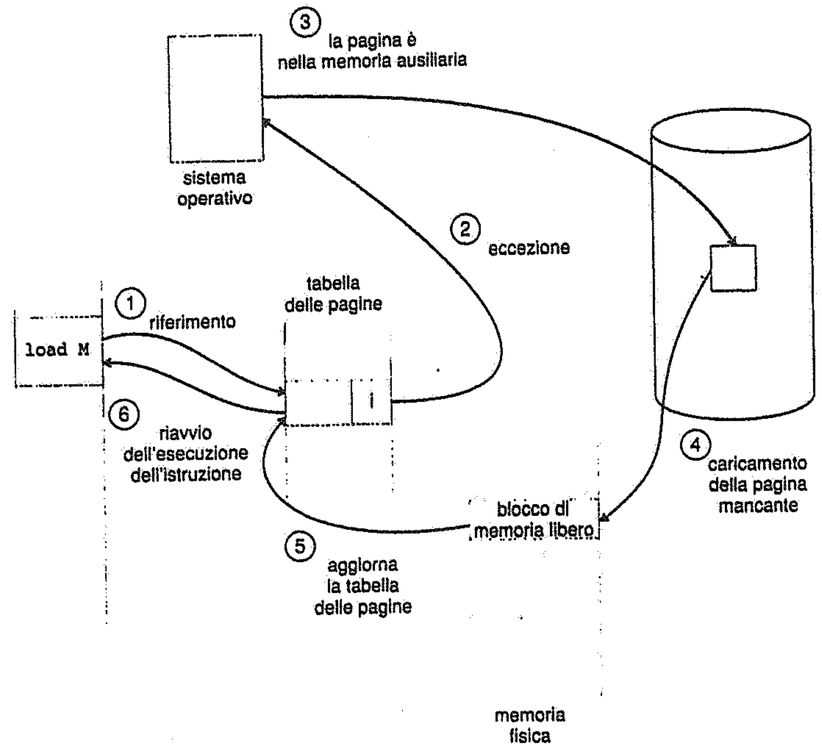


Figura 10.4 Fasi di gestione di un'eccezione di pagina mancante.

È addirittura possibile avviare l'esecuzione di un processo *senza* pagine nella memoria. Quando il sistema operativo carica nel contatore di programma l'indirizzo della prima istruzione del processo, che è in una pagina non residente nella memoria, il processo accusa un'assenza di pagina. Una volta portata la pagina nella memoria, il processo continua l'esecuzione, subendo assenze di pagine fino a che tutte le pagine necessarie non si trovano effettivamente nella memoria; a questo punto si può eseguire il processo senza ulteriori richieste. Lo schema descritto è una **paginazione su richiesta pura**, vale a dire che una pagina non si trasferisce nella memoria fino a che non è richiesta.

In teoria alcuni programmi possono accedere a più pagine di memoria all'esecuzione di ogni istruzione (una pagina per l'istruzione e molte per i dati), causando più possibili eccezioni di pagine mancanti per ogni istruzione. In un caso simile le prestazioni del sistema sarebbero inaccettabili. Un'analisi dei processi in esecuzione mostra che questo comportamento è molto improbabile. I programmi tendono ad avere una **località dei riferimenti**, descritta nel Paragrafo 10.6.1, quindi le prestazioni della paginazione su richiesta rientrano in un campo ragionevole.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I meccanismi d'ausilio alla paginazione su richiesta che l'architettura del calcolatore deve offrire sono quelli richiesti per la paginazione e l'avvicendamento dei processi nella memoria:

- **Tabella delle pagine.** Questa tabella ha la capacità di contrassegnare un elemento come non valido attraverso un bit di validità oppure un valore speciale dei bit di protezione.
- **Memoria ausiliaria.** Questa memoria conserva le pagine che non sono presenti nella memoria centrale. Generalmente la memoria ausiliaria è costituita di un disco ad alta velocità detto dispositivo d'avvicendamento; la sezione del disco usata a questo scopo si chiama **area d'avvicendamento**, o **area di scambio** (*swap space*). L'assegnazione dell'area d'avvicendamento è trattata nel Paragrafo 14.4.

Oltre queste caratteristiche dell'architettura, occorre un notevole lavoro di programmazione ed è necessario imporre ulteriori vincoli architettonici. Uno cruciale riguarda la necessità di riavviare qualsiasi istruzione dopo un'assenza di pagina; nella maggior parte dei casi questo requisito si soddisfa facilmente. Un'assenza di pagina si può verificare per qualsiasi riferimento alla memoria. Se l'assenza di pagina si presenta durante la fase di prelievo di un'istruzione, l'esecuzione si può riavviare prelevando nuovamente tale istruzione. Se si verifica durante il prelievo di un operando, l'istruzione deve essere di nuovo prelevata e decodificata, quindi si può prelevare l'operando.

Come caso limite si consideri un'istruzione in tre indirizzi, come ad esempio la somma (ADD) del contenuto di A al contenuto di B, con risultato posto in C. I passi necessari per eseguire l'istruzione sono i seguenti:

1. prelievo e decodifica dell'istruzione (ADD);
2. prelievo del contenuto di A;
3. prelievo del contenuto di B;
4. addizione del contenuto di A al contenuto di B;
5. memorizzazione della somma in C.

Se l'assenza di pagina avviene al momento della memorizzazione in C, poiché C si trova in una pagina che non è nella memoria, occorre prelevare la pagina desiderata, caricarla nella memoria, correggere la tabella delle pagine e riavviare l'istruzione. Il riavvio dell'istruzione richiede una nuova operazione di prelievo della stessa, con nuova decodifica e nuovo prelievo dei due operandi; infine occorre ripetere l'addizione. In ogni modo il lavoro da ripetere non è molto, meno di un'istruzione completa, e la ripetizione è necessaria solo nel caso si verifichi un'assenza di pagina.

La difficoltà maggiore si presenta quando un'istruzione può modificare parecchie locazioni diverse. Si consideri, ad esempio, l'istruzione MVC (*move character*) del sistema IBM 360/370: quest'istruzione può spostare una sequenza di byte (fino a 256) da una locazione a un'altra con possibilità di sovrapposizione. Se una delle sequenze (quella d'origine o quella di destinazione) esce dal confine di una pagina, e se lo spostamento è stato effettuato solo in parte, si può verificare un'assenza di pagina. Inoltre, se le sequenze d'origine e di destinazione si sovrappongono, è probabile che la sequenza d'origine sia stata modificata, in tal caso non è possibile limitarsi a riavviare l'istruzione.

Il problema si può risolvere in due modi. In una delle due soluzioni il microcodice computa e tenta di accedere alle estremità delle due sequenze di byte. Un'eventuale assenza di pagina si può verificare solo in questa fase, prima che si apporti qualsiasi modifica. A questo punto si può compiere lo spostamento perché tutte le pagine interessate si trovano nella memoria. L'altra soluzione si serve di registri temporanei per conservare i valori delle locazioni sovrascritte. Nel caso di un'assenza di pagina, si riscrivono tutti i vecchi valori nella memoria prima che sia emesso il segnale di eccezione di pagina mancante. Questa operazione riporta la memoria allo stato in cui si trovava prima che l'istruzione fosse avviata, perciò si può ripetere la sua esecuzione.

Un analogo problema architettonico si presenta in calcolatori che impiegano modi d'indirizzamento speciali che comprendono modi d'autodecremento e autoincremento, ad esempio il PDP-11. Questi modi d'indirizzamento usano come puntatore un registro che si decrementa e incrementa automaticamente. L'autodecremento decrementa automaticamente il registro *prima* di usarne il contenuto come indirizzo per l'operando; l'autoincremento, invece, incrementa automaticamente il registro *dopo* averne usato il contenuto come indirizzo per l'operando. Quindi, l'istruzione

```
MOV (R2)+, -(R3)
```

copia il contenuto della locazione puntata dal registro R2 nella locazione puntata dal registro R3. Il registro R2 viene incrementato (di 2 per una parola, perché il PDP-11 è un calcolatore indirizzabile per byte) dopo essere stato usato come puntatore; il registro R3 viene decrementato, di 2, prima di essere usato come puntatore. Ora è possibile esaminare cosa accade nel caso di un'assenza di pagina, se si fa un tentativo di memorizzazione nella locazione puntata dal registro R3. Per riavviare l'istruzione si devono reimpostare i due registri sui valori che avevano prima di iniziare l'esecuzione dell'istruzione. Una soluzione consiste nel creare un nuovo registro di stato per memorizzare il numero di registro e l'entità della modifica per ogni registro modificato durante l'esecuzione di un'istruzione. Questo registro di stato permette al sistema operativo di *annullare* gli effetti di un'istruzione eseguita solo in parte, che ha causato un'eccezione di pagina mancante.

Questi non sono gli unici problemi architettonici che possono derivare dall'aggiunta della paginazione a un'architettura esistente allo scopo di permettere la paginazione su richiesta, ma illustrano alcune difficoltà che si possono incontrare. Il sistema di paginazione si colloca tra la CPU e la memoria di un calcolatore e deve essere completamente trasparente al processo utente. L'opinione comune che la paginazione si possa aggiungere a qualsiasi sistema è vera per gli ambienti senza paginazione su richiesta nei quali un'eccezione di pagina mancante rappresenta un errore fatale, ma è falsa nei casi in cui un'eccezione di pagina mancante implica la necessità di caricare nella memoria un'altra pagina e quindi riavviare il processo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

10.2.2 Prestazioni della paginazione su richiesta

La paginazione su richiesta può avere un effetto rilevante sulle prestazioni di un calcolatore. Il motivo si può comprendere calcolando il **tempo d'accesso effettivo** per una memoria con paginazione su richiesta. Attualmente, nella maggior parte dei calcolatori il tempo d'accesso alla memoria, che si denota *ma*, varia da 10 a 200 nanosecondi. Fino a che non si verificano assenze di pagine, il tempo d'accesso effettivo è uguale al tempo d'accesso alla memoria. Se però si verifica un'assenza di pagina, occorre prima leggere dal disco la pagina interessata e quindi accedere alla parola della memoria desiderata.

Supponendo che p sia la probabilità che si verifichi un'assenza di pagina ($0 \le p \le 1$), è probabile che p sia molto vicina allo zero, cioè che ci siano solo poche assenze di pagine. Il **tempo d'accesso effettivo** è dato dalla seguente espressione:

$$\textit{tempo d'accesso effettivo} = (1 - p) \times ma + p \times \textit{tempo di gestione dell'assenza di pagina}$$

Per calcolare il tempo d'accesso effettivo occorre conoscere il tempo necessario alla gestione di un'assenza di pagina. Alla presenza di un'assenza di pagina si esegue la seguente sequenza:

1. segnale d'eccezione al sistema operativo;
2. salvataggio dei registri d'utente e dello stato del processo;
3. determinazione della natura di eccezione di pagina mancante del segnale di eccezione;
4. controllo della correttezza del riferimento alla pagina e determinazione della locazione della pagina nel disco;
5. lettura dal disco e trasferimento in un blocco di memoria libero:
   a) attesa nella coda relativa a questo dispositivo fino a che la richiesta di lettura non è servita,
   b) attesa del tempo di posizionamento e latenza del dispositivo,
   c) inizio del trasferimento della pagina in un blocco di memoria libero;
6. durante l'attesa, assegnazione della CPU a un altro processo utente (scheduling della CPU, facoltativo);
7. segnale d'interruzione emesso dal controllore del disco (I/O completato);
8. salvataggio dei registri e dello stato dell'altro processo utente (se è stato eseguito il passo 6);
9. determinazione della provenienza dal disco dell'interruzione;
10. aggiornamento della tabella delle pagine e di altre tabelle per segnalare che la pagina richiesta è attualmente presente nella memoria;
11. attesa che la CPU sia nuovamente assegnata a questo processo;
12. recupero dei registri d'utente, dello stato del processo e della nuova tabella delle pagine, quindi ripresa dell'istruzione interrotta.

Non sempre sono necessari tutti i passi sopra elencati. Nel passo 6, ad esempio, si ipotizza che la CPU sia assegnata a un altro processo durante un'operazione di I/O. Tale possibilità permette la multiprogrammazione per mantenere occupata la CPU, ma una volta completato il trasferimento di I/O implica un dispendio di tempo per riprendere la procedura di servizio dell'eccezione di pagina mancante.

In ogni caso, il tempo di servizio dell'eccezione di pagina mancante comporta tre operazioni principali:

1. servizio del segnale di eccezione di pagina mancante;
2. lettura della pagina;
3. riavvio del processo.

La prima e la terza operazione si possono realizzare, per mezzo di un'accurata codifica, in parecchie centinaia d'istruzioni. Ciascuna di queste operazioni può richiedere da 1 a 100 microsecondi. D'altra parte, il tempo di cambio di pagina è probabilmente vicino a 24 millisecondi. Un disco ha tipicamente un tempo di latenza di 8 millisecondi, un tempo di posizionamento di 15 millisecondi e un tempo di trasferimento di 1 millisecondo, quindi il tempo totale della paginazione è dell'ordine di 25 millisecondi, compresi i tempi d'esecuzione del codice relativo e delle operazioni dei dispositivi fisici coinvolti. Nel calcolo si è considerato solo il tempo di servizio del dispositivo. Se una coda di processi è nell'attesa del dispositivo (altri processi che hanno accusato un'assenza di pagina) è necessario considerare anche il tempo di accodamento del dispositivo, poiché occorre attendere che il dispositivo di paginazione sia libero per servire la richiesta, quindi il tempo d'avvicendamento aumenta ulteriormente.

Considerando un tempo medio di servizio dell'eccezione di pagina mancante di 25 millisecondi e un tempo d'accesso alla memoria di 100 nanosecondi, il tempo effettivo d'accesso in nanosecondi è il seguente:

$$\begin{aligned}\textit{tempo d'accesso effettivo} &= (1 - p) \times 100 + p \times (25 \text{ millisecondi})\\ &= (1 - p) \times 100 + p \times 25.000.000\\ &= 100 + 24.999.900 \times p\end{aligned}$$

Il tempo d'accesso effettivo è direttamente proporzionale alla frequenza delle assenze di pagine. Se un accesso su 1000 accusa un'assenza di pagina, il tempo d'accesso effettivo è di 25 microsecondi. Impiegando la paginazione su richiesta, il calcolatore è rallentato di un fattore pari a 250. Se si desidera un rallentamento inferiore al 10 per cento, occorre che valgano le seguenti condizioni:

$$\begin{aligned}&110 > 100 + 25.000.000 \times p\\ &10 > 25.000.000 \times p\\ &p < 0{,}0000004\end{aligned}$$

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

Cioè, per mantenere a un livello ragionevole il rallentamento dovuto alla paginazione, si può permettere meno di un'assenza di pagina ogni 2.500.000 accessi alla memoria.

In un sistema con paginazione su richiesta, è importante tenere bassa la frequenza delle assenze di pagine, altrimenti il tempo effettivo d'accesso aumenta, rallentando molto l'esecuzione del processo.

Un altro aspetto della paginazione su richiesta è la gestione e l'uso generale dell'area d'avvicendamento. L'I/O di un disco relativo all'area d'avvicendamento è generalmente più rapido di quello relativo al file system, poiché essa è composta di blocchi assai più grandi e non s'impiegano ricerche di file e metodi d'assegnazione indiretta (Paragrafo 14.4). Perciò il sistema può migliorare l'efficienza della paginazione copiando tutta l'immagine di un file nell'area d'avvicendamento all'avvio del processo e di lì eseguire la paginazione su richiesta. Un'altra possibilità consiste nel richiedere inizialmente le pagine al file system, ma scrivere le pagine nell'area d'avvicendamento al momento della sostituzione. Questo metodo assicura che si leggano sempre dal file system solo le pagine necessarie, ma che tutta la paginazione successiva sia fatta dall'area d'avvicendamento. Quando s'impiegano file binari, alcuni sistemi tentano di limitare tale area: le pagine richieste per questi file si prelevano direttamente dal file system; quando è richiesta una sostituzione di pagine, queste, se è necessario, sono ancora lette dal file system. Seguendo questo criterio, lo stesso file system funziona da memoria ausiliaria (*backing store*). L'area d'avvicendamento si deve in ogni caso usare per le pagine che non sono relative ai file; queste comprendono la **pila** (*stack*) e lo **heap** di un processo. Questa tecnica che sembra essere un buon compromesso si usa in diversi sistemi tra cui Solaris 2 e BSD UNIX.

10.3 Creazione dei processi

Nel Paragrafo 10.2 si spiega come l'esecuzione di un processo si avvii usando la paginazione su richiesta. Per mezzo di questa tecnica, un processo può cominciare rapidamente la propria elaborazione, facendo in modo che il sistema di paginazione su richiesta carichi la pagina che contiene la prima istruzione. La paginazione e la memoria virtuale offrono altri vantaggi per quel che riguarda la creazione dei processi. In questo paragrafo, s'illustrano due tecniche, rese possibili dal meccanismo della memoria virtuale, che migliorano le prestazioni relative alla creazione ed esecuzione dei processi.

10.3.1 Copiatura su scrittura *

Nella lettura dei file dai dischi alla memoria, inclusi i file eseguibili in formato binario, s'impiega la paginazione su richiesta. Tuttavia, quando si crea un processo per mezzo della chiamata del sistema `fork`, si può evitare la paginazione su richiesta usando una tecnica simile alla condivisione delle pagine (trattata nel Paragrafo 9.4.5). Questa tecnica permette una creazione dei processi molto rapida e minimizza il numero di pagine che si devono assegnare al nuovo processo.

* quando il processo figlio [illegible] del processo padre allora viene fatta la copia del processo genitore.

Si ricordi che la chiamata del sistema `fork` crea un processo figlio come duplicato del genitore. Nella sua versione originale la `fork` creava per il figlio una copia dello spazio d'indirizzi del genitore, duplicando le pagine appartenenti al processo genitore. Considerando che molti processi figli eseguono subito dopo la loro creazione la chiamata del sistema `exec`, questa operazione di copiatura risulta inutile. In alternativa, si può impiegare una tecnica nota come **copiatura su scrittura** (*copy-on-write*), il cui funzionamento si fonda sulla condivisone iniziale delle pagine da parte dei processi genitori e dei processi figli. Le pagine condivise si contrassegnano come pagine da copiare su scrittura, a significare che, se un processo (genitore o figlio) scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina. Si consideri ad esempio un processo figlio che cerca di modificare una pagina che contiene parti della pila; il sistema operativo considera questa una pagina da copiare su scrittura e ne crea una copia nello spazio degli indirizzi del processo figlio. Il processo figlio modifica la sua copia della pagina e non la pagina appartenente al processo genitore. È chiaro che, adoperando la tecnica di copiatura su scrittura, si copiano soltanto le pagine modificate da uno dei due processi, mentre tutte le altre possono essere condivise dai processi genitore e figlio. Si noti inoltre che soltanto le pagine modificabili si devono contrassegnare come da copiare su scrittura, mentre quelle che non si possono modificare (ad esempio, le pagine che contengono codice eseguibile) possono essere condivise dai processi genitore e figlio. La tecnica di copiatura su scrittura è piuttosto comune e si usa in diversi sistemi operativi, tra i quali Windows 2000, LINUX e Solaris 2.

Quando è necessaria la duplicazione di una pagina secondo la tecnica di copiatura su scrittura, è importante capire da dove si attingerà la pagina libera necessaria. Molti sistemi operativi forniscono, per queste richieste, un gruppo (*pool*) di pagine libere, che di solito si assegnano quando la pila o il cosiddetto *heap* di un processo devono espandersi, oppure proprio per gestire pagine da copiare su scrittura. L'assegnazione di queste pagine di solito avviene secondo una tecnica nota come **azzeramento su richiesta** (*zero-fill-on-demand*); si riempiono di zeri le pagine prima dell'assegnazione, cancellandone in questo modo tutto il contenuto precedente. Se s'impiega la copiatura su scrittura, la pagina da copiare sarà copiata in una pagina azzerata. Allo stesso modo le pagine assegnate per la pila o per lo heap sono pagine azzerate.

Diverse versioni dello UNIX (compreso il Solaris 2) offrono anche una variante della chiamata del sistema `fork` detta `vfork` (per *virtual memory fork*). La `vfork` offre un'alternativa all'uso della `fork` con copiatura su scrittura. Con la `vfork` si sospende il processo genitore e il processo figlio usa lo spazio d'indirizzi del genitore. Poiché la `vfork` non usa la copiatura su scrittura, se il processo figlio modifica qualche pagina dello spazio d'indirizzi del genitore, le pagine modificate saranno visibili al processo genitore non appena riprenderà il controllo. Quindi, è necessaria molta attenzione nell'uso della `vfork`, per assicurarsi che il processo figlio non modifichi lo spazio d'indirizzi del genitore. La chiamata del sistema `vfork` è adatta al caso in cui il processo figlio esegue una `exec` immediatamente dopo la sua creazione. Poiché non richiede alcuna copiatura delle pagine, la `vfork` è un metodo di creazione dei processi molto efficiente ed è in alcuni casi impiegato per realizzare le interfacce degli interpreti dei comandi nello UNIX.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

10.3.2 Associazione dei file alla memoria

Si consideri un accesso sequenziale per lettura a un file in un disco, per mezzo delle normali chiamate del sistema open, `read` e `write`. Ogni accesso al file richiede una chiamata del sistema e un accesso al disco. In alternativa, si possono usare le tecniche di memoria virtuale, considerando l'I/O relativo a un file come un ordinario accesso alla memoria. Questo metodo si chiama **associazione alla memoria** di un file e consiste nel permettere che una parte dello spazio degli indirizzi virtuali sia associata logicamente a un file. Questo metodo si realizza facendo corrispondere un blocco di disco a una pagina (o più pagine) della memoria. L'accesso iniziale al file avviene per mezzo dell'ordinario meccanismo di paginazione su richiesta, che determina un'assenza di pagina cui segue, però, il caricamento in una pagina fisica di una porzione di file della dimensione di una pagina, leggendola dal file system (in alcuni sistemi c'è la possibilità di leggere porzioni maggiori di un blocco della dimensione di una pagina). Le operazioni successive di lettura e scrittura sul file si gestiscono come normali accessi alla memoria; in questo modo si semplifica l'accesso e l'uso dei file, si permette la manipolazione degli stessi attraverso la memoria e si evita il carico dovuto alle chiamate del sistema `read` e `write`. Si noti che le operazioni di scrittura nei file associati alla memoria non corrispondono necessariamente a scritture immediate nei file presenti nella memoria secondaria. Alcuni sistemi potrebbero optare per l'aggiornamento degli effettivi file durante le periodiche verifiche svolte dal sistema operativo riguardo eventuali modifiche alle pagine di memoria corrispondenti ai file. La chiusura di un file comporta la scrittura nella memoria secondaria di tutti i dati presenti nelle corrispondenti pagine di memoria e la rimozione del processo dalla memoria virtuale.

Alcuni sistemi operativi offrono l'associazione alla memoria soltanto attraverso specifiche chiamate del sistema e gestiscono tutte le altre operazioni di I/O su file attraverso le normali chiamate del sistema. Tuttavia alcuni sistemi decidono di associare un file alla memoria indipendentemente dal fatto che ciò sia richiesto oppure no. Si consideri come esempio il sistema operativo Solaris 2: se per un file si richiede l'associazione alla memoria (usando la chiamata del sistema `mmap`), il sistema operativo mette il file in corrispondenza con lo spazio d'indirizzi del processo; tuttavia, se un file è aperto vi si accede usando le ordinarie chiamate del sistema come open, `read`, e `write`; l'assenza di un'esplicita richiesta in questo senso comporta in ogni caso l'associazione alla memoria del file, anche se in questo caso si fa corrispondere allo spazio d'indirizzi del nucleo. Indipendentemente da come si aprono i file, il sistema operativo gestisce tutte le operazioni di I/O sui file come associate alla memoria, permettendo l'accesso ai file attraverso la memoria.

Si può permettere l'associazione dello stesso file alla memoria virtuale di più processi, allo scopo di condividere dati. Le operazioni di scrittura fatte da uno qualunque di questi processi modificano i dati presenti nella memoria virtuale e sono visibili a tutti gli altri processi coinvolti nella condivisione. Date le nozioni acquisite in questo capitolo, dovrebbe essere chiaro come funziona il meccanismo con il quale si condividono nella memoria le sezioni di file: la mappa della memoria virtuale di ciascun processo coinvolto nella condivisione punta alla stessa pagina di memoria fisica, in particolare alla pagina che contiene una copia del blocco di disco. Questo schema di condivisione della memoria è illustrato nella Figura 10.5. Le chiamate del sistema per l'associazione alla memoria possono anche disporre della funzione di copiatura su scrittura, permettendo ai processi di condividere un file per la sola lettura, ma tenendo una propria copia di ogni dato da essi modificato. Per coordinare opportunamente l'accesso ai dati condivisi, i processi coinvolti possono servirsi di uno dei meccanismi di mutua esclusione descritti nel Capitolo 7.

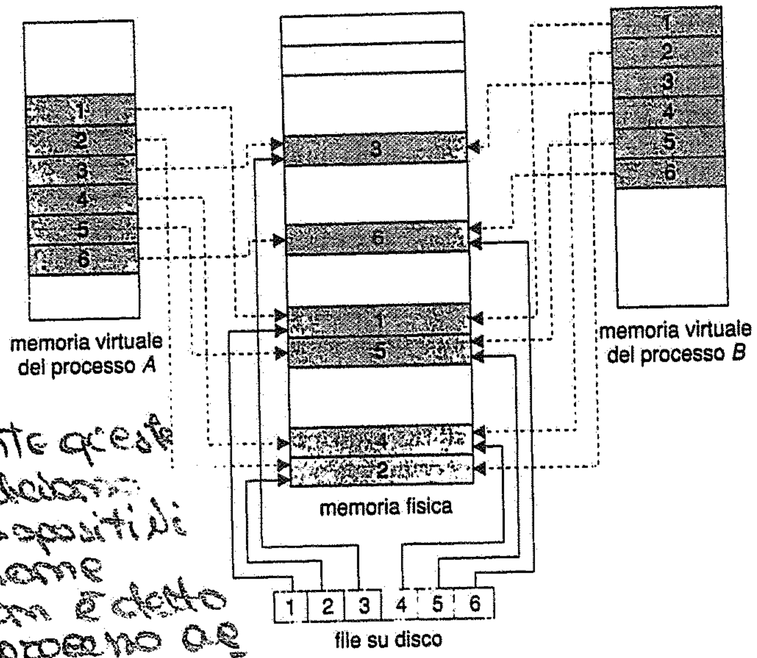


Figura 10.5 Associazione dei file alla memoria.

10.4 Sostituzione delle pagine

Nelle descrizioni fatte finora, la frequenza delle assenze di pagine non è stata un problema grave, giacché ogni pagina poteva essere assente al massimo una volta, e precisamente la prima volta in cui si effettuava un riferimento a essa. Tale rappresentazione non è molto precisa. Se un processo di 10 pagine ne impiega effettivamente solo la metà, la paginazione su richiesta fa risparmiare l'I/O necessario per caricare le cinque pagine che non sono mai usate. Il grado di multiprogrammazione potrebbe essere aumentato eseguendo il doppio dei processi. Quindi, disponendo di 40 blocchi di memoria, si potrebbero eseguire otto processi anziché i quattro che si eseguirebbero se ciascuno di essi richiedesse 10 blocchi di memoria, cinque dei quali non sarebbero mai usati.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Aumentando il grado di multiprogrammazione, si sovrassegna la memoria. Eseguendo sei processi, ciascuno dei quali è formato da 10 pagine, delle quali solo cinque sono effettivamente usate, s'incrementerebbero l'utilizzo e la produttività della CPU e si risparmierebbero 10 blocchi di memoria. Tuttavia è possibile che ciascuno di questi processi, per un insieme particolare di dati, abbia improvvisamente necessità di impiegare tutte le 10 pagine, perciò sarebbero necessari 60 blocchi di memoria mentre ne sono disponibili solo 40. La probabilità di una situazione di questo tipo aumenta con il livello di multiprogrammazione, di modo che la quantità media della memoria adoperata si avvicina alla memoria fisica disponibile. (Nel nostro esempio, perché fermarsi a un livello di multiprogrammazione di 6 avendo la possibilità di raggiungere un livello di 7 o di 8?)

Si consideri inoltre che la memoria del sistema non si usa solo per contenere pagine di programmi: le aree di memoria per l'I/O impegnano una rilevante quantità di memoria. Ciò può aumentare le difficoltà agli algoritmi di assegnazione della memoria. Decidere quanta memoria assegnare all'I/O e quanta alle pagine dei programmi è un problema rilevante. Alcuni sistemi riservano una quota fissata di memoria per l'I/O, altri permettono sia ai processi utenti sia al sottosistema di I/O di competere per tutta la memoria del sistema.

La sovrassegnazione si può illustrare come segue. Durante l'esecuzione di un processo utente si verifica un'assenza di pagina, che causa l'invio di un segnale d'eccezione al sistema operativo, il quale controlla le sue tabelle interne per verificare che si tratti di un'eccezione di pagina mancante e non di un accesso illegale alla memoria. Il sistema operativo determina la locazione del disco in cui la pagina desiderata risiede, ma poi scopre che l'elenco dei blocchi di memoria liberi è *vuoto*: tutta la memoria è in uso; si veda a questo proposito la Figura 10.6.

A questo punto il sistema operativo può scegliere tra diverse possibilità, ad esempio può terminare il processo utente. Tuttavia, la paginazione su richiesta è un tentativo che il sistema operativo fa per migliorare l'utilizzo e la produttività del sistema di calcolo. Gli utenti non devono sapere che i loro processi sono eseguiti su un sistema paginato. La paginazione deve essere logicamente trasparente per l'utente, quindi la terminazione del processo non costituisce la scelta migliore.

Il sistema operativo può scaricare dalla memoria l'intero processo, liberando tutti i suoi blocchi di memoria e riducendo il livello di multiprogrammazione. Questa possibilità, buona in certe situazioni, è considerata nel Paragrafo 10.6. In questa sede si discute una possibilità più interessante: la sostituzione delle pagine.

10.4.1 Schema di base

La sostituzione delle pagine segue il seguente criterio: se nessun blocco di memoria è libero si può liberarne uno attualmente inutilizzato. Un blocco di memoria si può liberare scrivendo il suo contenuto nell'area d'avvicendamento e modificando la tabella delle pagine (e tutte le altre tabelle) per indicare che la pagina non si trova più nella memoria; si veda a questo proposito la Figura 10.7. Il blocco di memoria liberato si può usare per conservare la pagina che ha causato l'eccezione. Si modifica la procedura di servizio dell'eccezione di pagina mancante in modo da includere la sostituzione della pagina:

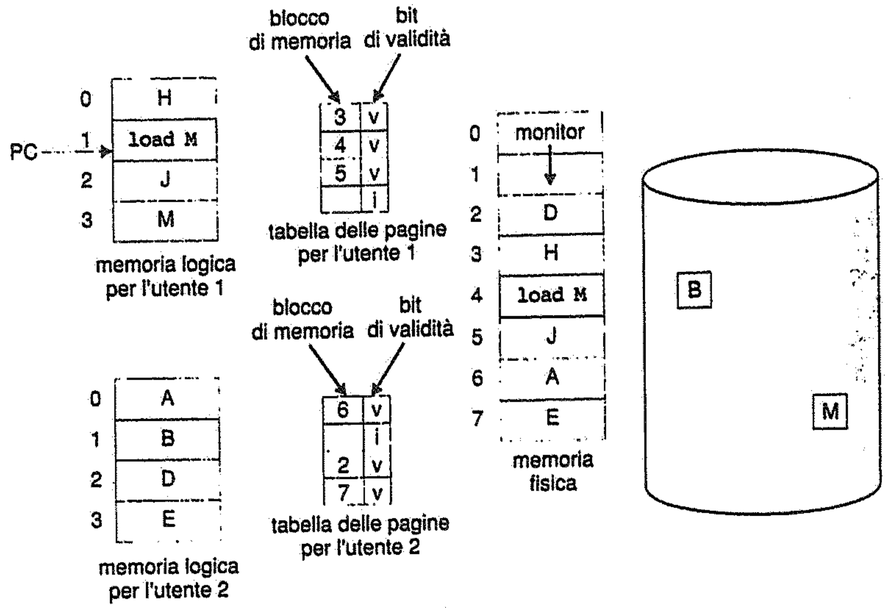


Figura 10.6 Necessità di sostituzione di pagine.

1. s'individua la locazione nel disco della pagina richiesta;
2. si cerca un blocco di memoria libero:
   a) se esiste, lo si usa,
   b) altrimenti si impiega un algoritmo di sostituzione delle pagine per scegliere un blocco di memoria 'vittima',
   c) si scrive la pagina 'vittima' nel disco; si modificano nel modo che ne consegue le tabelle delle pagine e dei blocchi di memoria;
3. si scrive la pagina richiesta nel blocco di memoria appena liberato; si modificano le tabelle delle pagine e dei blocchi di memoria;
4. si riavvia il processo utente.

Occorre notare che se non esiste alcun blocco di memoria libero sono necessari *due* trasferimenti di pagine, uno fuori e uno dentro la memoria. Questa situazione raddoppia il tempo di servizio dell'eccezione di pagina mancante e aumenta di conseguenza anche il tempo effettivo d'accesso.

Questo sovraccarico si può ridurre usando un bit di modifica (*dirty bit*). L'architettura fisica del sistema di calcolo può disporre di un bit di modifica, associato a ogni pagina (o blocco di memoria), che imposta automaticamente ogni volta che nella pagina si

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

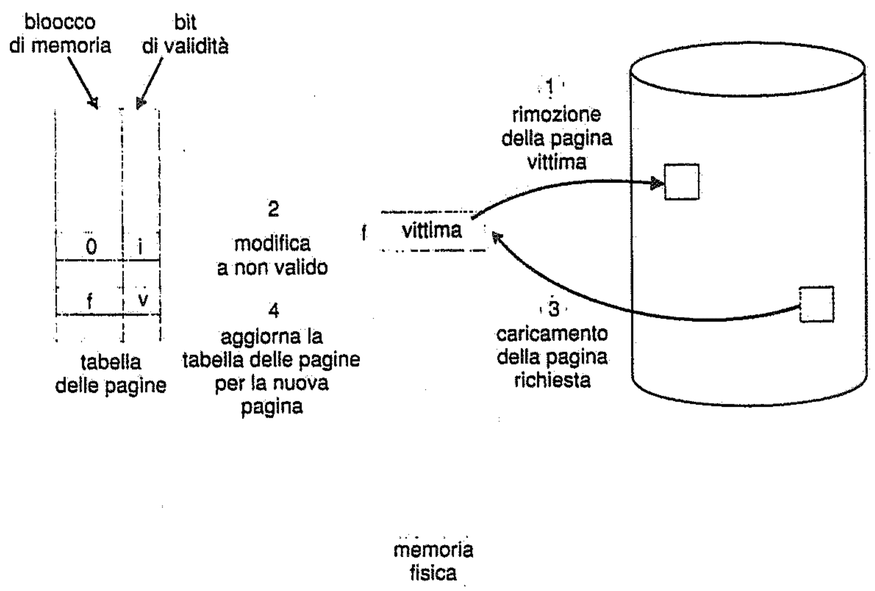


Figura 10.7 Sostituzione di una pagina.

scrive una parola o un byte, indicando che la pagina è stata modificata. Quando si sceglie una pagina da sostituire si esamina il suo bit di modifica; se è attivo, significa che quella pagina è stata modificata rispetto a quando era stata letta dal disco; in questo caso la pagina deve essere scritta nel disco. Se il bit di modifica non è attivo, significa che la pagina *non* è stata modificata da quando è stata caricata nella memoria, quindi se la sua copia nel disco non è stata sovrascritta, ad esempio da qualche altra pagina, non è necessario scrivere nel disco la pagina di memoria: c'è già. Questa tecnica vale anche per le pagine di sola lettura, ad esempio pagine di codice binario. Queste pagine non possono essere modificate, quindi si possono rimuovere in ogni momento. Questo schema può ridurre in modo considerevole il tempo per il servizio dell'eccezione di pagina mancante, poiché dimezza il tempo di I/O, *se* la pagina non è stata modificata.

La sostituzione di una pagina è fondamentale al fine della paginazione su richiesta, perché completa la separazione tra la memoria logica e la memoria fisica. Con questo meccanismo si può mettere a disposizione dei programmatori una memoria virtuale molto ampia con una memoria fisica più piccola. Senza la paginazione su richiesta, gli indirizzi d'utente si fanno corrispondere a indirizzi fisici, perciò i due insiemi di indiriz-

zi possono essere abbastanza diversi. Tutte le pagine di un processo devono però essere nella memoria fisica. Con la paginazione su richiesta la dimensione dello spazio degli indirizzi logici non è più limitata dalla memoria fisica. Ad esempio, un processo utente formato di 20 pagine si può eseguire in 10 blocchi di memoria semplicemente usando la paginazione su richiesta e l'algoritmo di sostituzione per localizzare un blocco di memoria libero ogni qual volta è necessario. Se una pagina che è stata modificata deve essere sostituita, si copia il suo contenuto nel disco. Un successivo riferimento a quella pagina causa un'eccezione di pagina mancante. In quel momento, la pagina viene riportata nella memoria e può sostituire un'altra pagina del processo.

Per realizzare la paginazione su richiesta è necessario risolvere due problemi principali: occorre sviluppare un algoritmo di assegnazione dei blocchi di memoria e un algoritmo di sostituzione delle pagine. Se nella memoria sono presenti più processi, occorre decidere quanti blocchi di memoria si devono assegnare a ciascun processo. Inoltre, quando è richiesta una sostituzione di pagina occorre scegliere i blocchi di memoria da sostituire. La progettazione di algoritmi idonei a risolvere questi problemi è un compito importante, poiché l'I/O nei dischi è piuttosto oneroso. Anche miglioramenti minimi ai metodi di paginazione su richiesta apportano notevoli miglioramenti alle prestazioni del sistema.

Esistono molti algoritmi di sostituzione delle pagine; probabilmente ogni sistema operativo ha il proprio schema di sostituzione. È quindi necessario stabilire un criterio per selezionare un algoritmo di sostituzione particolare; generalmente si sceglie quello con la minima frequenza delle assenze di pagine.

Un algoritmo si valuta effettuandone l'esecuzione su una particolare successione di riferimenti alla memoria e calcolando il numero di assenze di pagine. La successione dei riferimenti alla memoria si chiama, appunto, successione dei riferimenti. Queste successioni si possono generare artificialmente (ad esempio con un generatore di numeri casuali), oppure analizzando un dato sistema e registrando l'indirizzo di ciascun riferimento alla memoria. Quest'ultima opzione permette di ottenere un numero elevato di dati, dell'ordine di un milione di indirizzi al secondo. Per ridurre questa quantità di dati occorre notare due fatti.

Innanzi tutto, di una pagina di date dimensioni, generalmente fissate dall'architettura del sistema, si considera solo il numero della pagina anziché l'intero indirizzo. In secondo luogo, quando si fa riferimento a una pagina *p*, i riferimenti alla stessa pagina *immediatamente* successivi al primo non accusano assenze di pagine: dopo il primo riferimento, la pagina *p* è presente nella memoria. Esaminando un processo si potrebbe ad esempio registrare la seguente successione di indirizzi:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

che, a 100 byte per pagina, si riduce alla seguente successione di riferimenti:

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per stabilire il numero di assenze di pagine relativo a una particolare successione di riferimenti e a un particolare algoritmo di sostituzione delle pagine, occorre conoscere anche il numero di blocchi di memoria disponibili. Naturalmente, aumentando il numero di blocchi di memoria disponibili diminuisce il numero di assenze di pagine. Per la successione dei riferimenti precedentemente esaminata, ad esempio, dati tre o più blocchi di memoria si possono verificare tre sole assenze di pagine: una per il primo riferimento di ogni pagina. D'altra parte, se si dispone di un solo blocco di memoria è necessaria una sostituzione per ogni riferimento, con il risultato di 11 assenze di pagine. In generale è prevista una curva simile a quella della Figura 10.8. Aumentando il numero di blocchi di memoria, il numero di assenze di pagine diminuisce fino al livello minimo. Naturalmente aggiungendo memoria fisica il numero di blocchi di memoria aumenta.

Per illustrare gli algoritmi di sostituzione delle pagine s'impiega la seguente successione di riferimenti:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1,

con tre blocchi di memoria.

10.4.2 Sostituzione delle pagine secondo l'ordine d'arrivo

L'algoritmo di sostituzione delle pagine più semplice è un algoritmo FIFO. Questo algoritmo associa a ogni pagina l'istante di tempo in cui quella pagina è stata portata nella memoria. Se si deve sostituire una pagina, si seleziona quella presente nella memoria da più tempo. Occorre notare che non è strettamente necessario registrare l'istante in cui si carica una pagina nella memoria; infatti tutte le pagine presenti nella memoria si possono strutturare secondo una coda FIFO. In questo caso si sostituisce la pagina che si trova nel primo elemento della coda. Quando si carica una pagina nella memoria, la si inserisce nell'ultimo elemento della coda.

Nella successione di riferimenti adottata, i tre blocchi di memoria sono inizialmente vuoti. I primi tre riferimenti (7, 0, 1) accusano ciascuno un'assenza di pagina con conseguente caricamento delle relative pagine nei blocchi di memoria vuoti. Il riferimento successivo (2) causa la sostituzione della pagina 7, perché essa è stata caricata per prima nella memoria. Siccome 0 è il riferimento successivo e si trova già nella memoria, per questo riferimento non ha luogo alcuna assenza di pagina. Il primo riferimento a 3 causa la sostituzione della pagina 0, che era la prima fra le tre pagine nella memoria (0, 1, e 2) da caricare. A causa di questa sostituzione il riferimento successivo, a 0, accuserà un'assenza di pagina. La pagina 1 è allora sostituita dalla pagina 0. Questo processo prosegue come è illustrato nella Figura 10.9. Sono indicate le pagine presenti nei tre blocchi di memoria ogni volta che si verifica un'assenza di pagina. Complessivamente si hanno 15 assenze di pagine.

L'algoritmo FIFO di sostituzione delle pagine è facile da capire e da programmare; tuttavia la sue prestazioni non sono sempre buone. La pagina sostituita potrebbe essere un modulo di inizializzazione usato molto tempo prima e che non serve più; ma potrebbe anche contenere una variabile molto usata, inizializzata precedentemente, e ancora in uso.

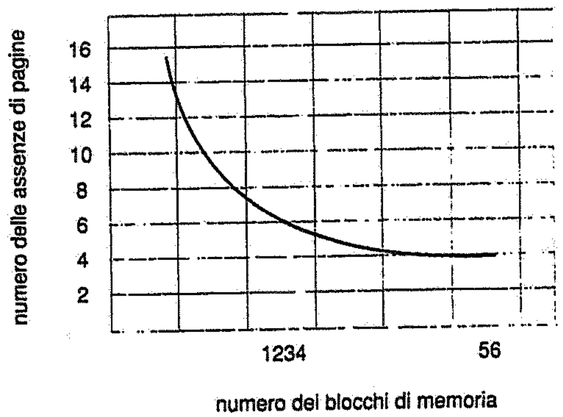


Figura 10.8 Numero delle assenze di pagine rispetto al numero dei blocchi di memoria.

Occorre notare che anche se si sceglie una pagina da sostituire che è in uso attivo, tutto continua a funzionare correttamente. Dopo aver rimosso una pagina attiva per inserirne una nuova, quasi immediatamente si verifica un'eccezione di pagina mancante per riprendere la pagina attiva. Per riportare la pagina attiva nella memoria è necessario sostituire un'altra pagina. Quindi, scegliendo una sostituzione errata, aumenta la frequenza di pagine mancanti che rallentata l'esecuzione del processo, ma non vengono causati errori nell'esecuzione.

Per illustrare i problemi che possono insorgere con l'uso dell'algoritmo di sostituzione delle pagine FIFO, si consideri la seguente successione di riferimenti:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

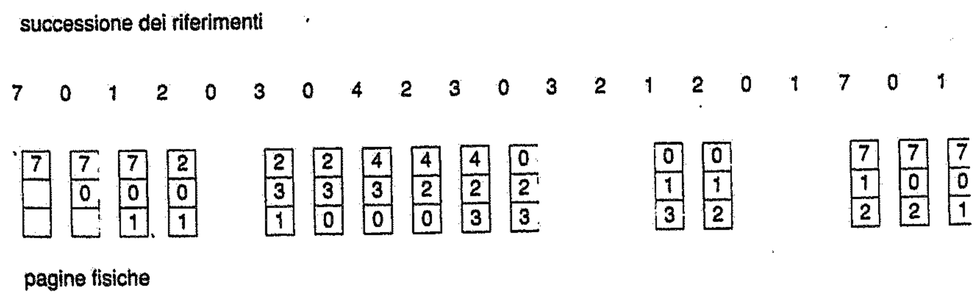


Figura 10.9 Algoritmo di sostituzione delle pagine FIFO.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Aumentando il numero di blocchi da 3 a 4 aumenta il numero di PAGE FAULT

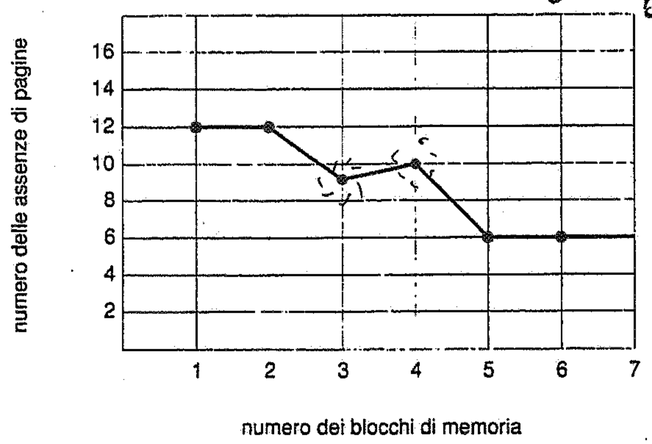


Figura 10.10 Curva delle assenze di pagine per sostituzione FIFO su una successione di riferimenti.

Nella Figura 10.10 è illustrata la curva delle assenze di pagine in funzione del numero di blocchi di memoria disponibili. Occorre notare che il numero delle assenze di pagine (10) per quattro blocchi di memoria è *maggiore* del numero delle assenze di pagine (9) per tre blocchi di memoria. Questo inatteso risultato è noto col nome di **anomalia di Belady**, e riflette il fatto che con alcuni algoritmi di sostituzione delle pagine la frequenza delle assenze di pagine può *aumentare* con l'aumentare del numero di blocchi di memoria assegnati. A prima vista sembra logico supporre che fornendo più memoria a un processo le prestazioni di quest'ultimo migliorino. Si è invece notato che questo presupposto non sempre è vero; l'anomalia di Belady ne è la prova.

10.4.3 Sostituzione delle pagine ottimale

In seguito alla scoperta dell'anomalia di Belady, la ricerca si è diretta verso un **algoritmo ottimale di sostituzione delle pagine**. Tale algoritmo è quello che fra tutti gli algoritmi presenta la minima frequenza di assenze di pagine e non presenta mai l'anomalia di Belady. Questo algoritmo esiste ed è stato chiamato OPT o MIN. È semplicemente:

Si sostituisce la pagina che non si userà per il periodo di tempo più lungo.

L'uso di quest'algoritmo di sostituzione delle pagine assicura la frequenza di assenze di pagine più bassa possibile per un numero fissato di blocchi di memoria.

Ad esempio, nella successione dei riferimenti considerata, l'algoritmo ottimale di sostituzione delle pagine produce nove assenze di pagine, come è mostrato nella Figura 10.11. I primi tre riferimenti causano assenze di pagine che riempiono i tre blocchi di memoria vuoti. Il riferimento alla pagina 2 determina la sostituzione della pagina 7, perché la 7 non è usata fino al riferimento 18, mentre la pagina 0 viene usata al 5 e la pagina 1 al 14. Il riferimento alla pagina 3 causa la sostituzione della pagina 1, poiché la pagina 1 è l'ultima delle tre pagine nella memoria cui si fa nuovamente riferimento. Con sole nove assenze di pagine, la sostituzione ottimale risulta assai migliore di quella ottenuta con un algoritmo FIFO, dove le assenze di pagine erano 15. Ignorando le prime tre assenze di pagine, che si verificano con tutti gli algoritmi, la sostituzione ottimale è due volte migliore rispetto all'algoritmo FIFO; nessun algoritmo di sostituzione può gestire questa successione di riferimenti a tre blocchi di memoria con meno di nove assenze di pagine.

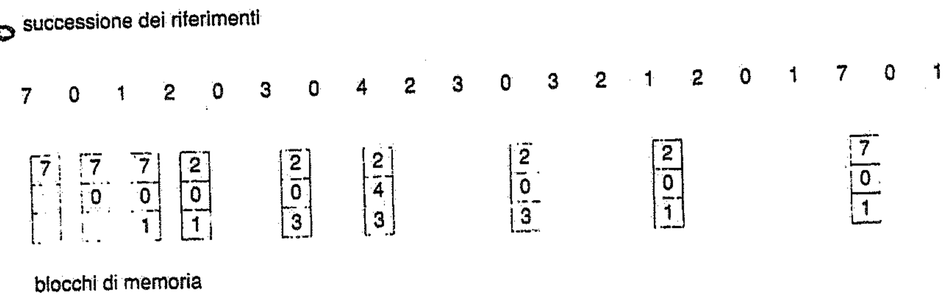


Figura 10.11 Algoritmo ottimale di sostituzione delle pagine.

Sfortunatamente l'algoritmo ottimale di sostituzione delle pagine è difficile da realizzare, perché richiede la conoscenza futura della successione dei riferimenti. Una situazione analoga si è riscontrata con l'algoritmo SJF di scheduling della CPU, nel Paragrafo 6.3.2. Quindi, l'algoritmo ottimale si impiega soprattutto per studi comparativi. Ad esempio, può risultare abbastanza utile sapere che sebbene un algoritmo nuovo non sia ottimale, nel peggiore dei casi le sue prestazioni sono inferiori del 12,3 per cento rispetto a quelle dell'algoritmo ottimale, e mediamente questa percentuale è del 4,7 per cento.

10.4.4 Sostituzione delle pagine usate meno recentemente

Se l'algoritmo ottimale non è realizzabile, è forse possibile realizzarne un'approssimazione. La distinzione fondamentale tra gli algoritmi FIFO e OPT, oltre quella di guardare avanti o indietro nel tempo, consiste nel fatto che l'algoritmo FIFO impiega l'istante in cui una pagina è stata caricata nella memoria, mentre l'algoritmo OPT impiega l'istante in cui una pagina è *usata*. Usando come approssimazione di un futuro vicino un passato recente, si sostituisce la pagina che *non è stata usata* per il periodo più lungo; si veda a questo proposito la Figura 10.12. Il metodo appena descritto è noto come algoritmo LRU — *least recently used*.

La sostituzione LRU associa a ogni pagina l'istante in cui è stata usata per l'ultima volta. Quando occorre sostituire una pagina, l'algoritmo LRU sceglie quella che non è stata usata per il periodo più lungo. Questa strategia costituisce l'algoritmo ottimale di sostituzione delle pagine con ricerca all'indietro nel tempo, anziché in avanti. Infatti,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

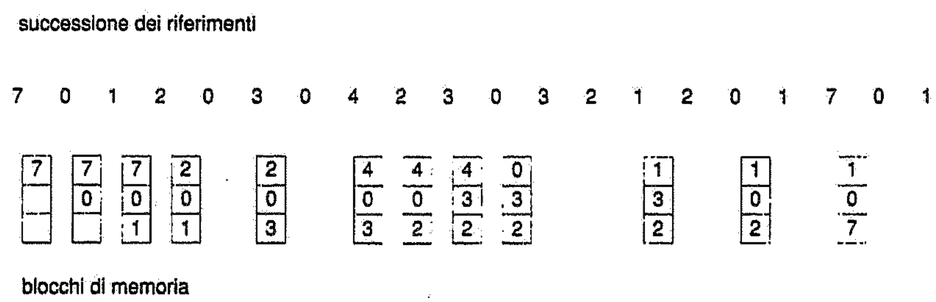


Figura 10.12 Algoritmo di sostituzione delle pagine LRU.

supponendo che S^R sia la successione inversa di una successione di riferimenti S, la frequenza di assenze di pagine per l'algoritmo OPT su S è uguale a quella per l'algoritmo LRU su S^R. Allo stesso modo, la frequenza di assenze di pagine per l'algoritmo LRU su S è uguale a quella per l'algoritmo OPT su S^R.

Il risultato dell'applicazione dell'algoritmo LRU alla successione dei riferimenti dell'esempio è illustrato nella Figura 10.12. L'algoritmo LRU produce 12 assenze di pagine. Occorre notare che le prime cinque assenze di pagine sono le stesse della sostituzione ottimale. Quando si presenta il riferimento alla pagina 4, però, l'algoritmo LRU trova che, fra i tre blocchi di memoria, quello usato meno recentemente è della pagina 2. La pagina usata più recentemente è la pagina 0, usata appena prima della pagina 3. Quindi, l'algoritmo LRU sostituisce la pagina 2 senza sapere che sta per essere usata. Quando si verifica l'assenza della pagina 2, l'algoritmo LRU sostituisce la pagina 3, poiché, fra le tre pagine nella memoria (0, 3, 4), la pagina 3 è quella usata meno recentemente. Nonostante questi problemi, la sostituzione LRU, con 12 assenze di pagine, è in ogni modo migliore della sostituzione FIFO, con 15 assenze di pagine.

Il criterio LRU si usa spesso come algoritmo di sostituzione delle pagine ed è considerato valido. Il problema principale riguarda la realizzazione della sostituzione stessa. Un algoritmo di sostituzione delle pagine LRU può richiedere una notevole assistenza da parte dell'architettura del sistema di calcolo. Il problema consiste nel determinare un ordine per i blocchi di memoria definito secondo il momento dell'ultimo uso. Si possono realizzare le due seguenti soluzioni:

- **Contatori.** Nel caso più semplice, a ogni elemento della tabella delle pagine si associa un campo del momento d'uso, e alla CPU si aggiunge un contatore che si incrementa a ogni riferimento alla memoria. Ogni volta che si fa un riferimento a una pagina, si copia il contenuto del registro contatore nel campo del momento d'uso nella tabella relativa a quella specifica pagina. In questo modo è sempre possibile conoscere il momento in cui è stato fatto l'ultimo riferimento a ogni pagina. Si sostituisce la pagina con il valore associato più piccolo. Questo schema implica una ricerca all'interno della tabella delle pagine per individuare la pagina usata meno recentemente (LRU), e una scrittura nella memoria (nel campo del momento d'uso della tabella delle pagine) per ogni accesso alla memoria. I riferimenti temporali si devono mantenere anche quando, a seguito dello scheduling della CPU, si modificano le tabelle delle pagine. Occorre infine considerare il superamento della capacità del contatore (*overflow*).

- **Pila.** Un altro metodo per la realizzazione della sostituzione delle pagine LRU prevede la presenza di una pila dei numeri delle pagine. Ogni volta che si fa un riferimento a una pagina, la si estrae dalla pila e la si colloca in cima a quest'ultima. In questo modo, in cima alla pila si trova sempre la pagina usata per ultima, mentre in fondo si trova la pagina usata meno recentemente, com'è illustrato dalla Figura 10.13. Poiché alcuni elementi si devono estrarre dal centro della pila, la migliore realizzazione si ottiene usando una lista doppiamente concatenata, con un puntatore all'elemento iniziale e uno a quello finale. Per estrarre una pagina dalla pila e collocarla sulla sua cima, nel caso peggiore è necessario modificare sei puntatori. Ogni aggiornamento è un po' più costoso, ma per una sostituzione non si deve compiere alcuna ricerca; il puntatore dell'elemento di coda punta al fondo della pila, vale a dire la pagina usata meno recentemente. Questo metodo è adatto soprattutto alle realizzazioni programmate (o microprogrammate) della sostituzione delle pagine LRU.

Né la sostituzione ottimale né quella LRU sono soggette all'anomalia di Belady. Esiste una classe di algoritmi di sostituzione delle pagine, chiamati **algoritmi a pila**, che non presenta l'anomalia di Belady. Un algoritmo a pila è un algoritmo per il quale è possibile mostrare che l'insieme delle pagine nella memoria per n blocchi di memoria è sempre un

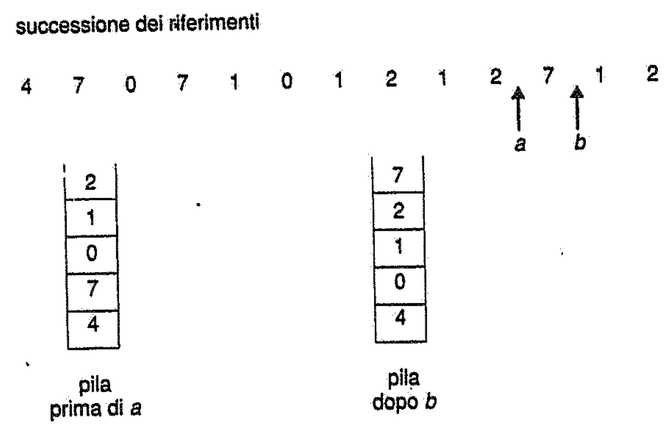


Figura 10.13 Uso di una pila per registrare i più recenti riferimenti alle pagine.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sottoinsieme dell'insieme delle pagine che dovrebbero essere nella memoria per $n + 1$ blocchi di memoria. Per la sostituzione LRU, l'insieme di pagine nella memoria è costituito delle n pagine cui si è fatto riferimento più recentemente. Se il numero di blocchi di memoria è aumentato, queste n pagine continuano a essere quelle cui si è fatto riferimento più recentemente e quindi restano nella memoria.

Oltre i registri TLB standard, senza l'ausilio dell'architettura sarebbe inconcepibile anche la realizzazione della sostituzione LRU. L'aggiornamento dei campi del contatore o della pila si deve effettuare per *ogni* riferimento alla memoria. Se per ogni riferimento si dovesse adoperare un segnale d'interruzione per permettere alle procedure del sistema operativo di modificare tali strutture di dati, tutti i riferimenti alla memoria sarebbero rallentati di un fattore almeno pari a 10, quindi anche tutti i processi utenti sarebbero rallentati di un fattore pari a 10. Pochi sistemi possono permettersi un tale sovraccarico per la gestione della memoria.

10.4.5 Sostituzione delle pagine per approssimazione a LRU

Sono pochi i sistemi di calcolo che dispongono di un'architettura adatta a una vera sostituzione LRU delle pagine. Nei sistemi che non offrono tali caratteristiche specifiche si devono impiegare altri algoritmi di sostituzione delle pagine, ad esempio l'algoritmo FIFO. Molti sistemi possono fornire un aiuto: un **bit di riferimento**. Il bit di riferimento a una pagina è impostato automaticamente dall'architettura del sistema ogni volta che si fa un riferimento a quella pagina, che sia una lettura o una scrittura su qualsiasi byte della pagina. I bit di riferimento sono associati a ciascun elemento della tabella delle pagine.

Inizialmente, il sistema operativo azzera tutti i bit. Quando s'inizia l'esecuzione di un processo utente, l'architettura del sistema imposta a 1 il bit associato a ciascuna pagina cui si fa riferimento. Dopo qualche tempo è possibile stabilire quali pagine sono state usate semplicemente esaminando i bit di riferimento. Non è però possibile conoscere l'*ordine* d'uso. Queste informazioni parziali sull'ordinamento portano ad alcuni algoritmi di sostituzione delle pagine che approssimano la sostituzione LRU.

10.4.5.1 Algoritmo con bit supplementari di riferimento

Ulteriori informazioni sull'ordinamento si possono ottenere registrando i bit di riferimento a intervalli regolari. È possibile conservare in una tabella nella memoria una serie di bit per ogni pagina. A intervalli regolari, ad esempio di 100 millisecondi, un segnale d'interruzione del temporizzatore del sistema trasferisce il controllo al sistema operativo. Questo sposta il bit di riferimento per ciascuna pagina nel bit più significativo della sequenza, traslando gli altri bit a destra di 1 bit e scartando il bit meno significativo. Questi registri a scorrimento, ad esempio di 8 bit, contengono l'ordine d'uso delle pagine relativo agli ultimi otto periodi di tempo. Se il registro a scorrimento contiene la successione di bit 00000000, significa che la pagina associata non è stata usata da otto periodi di tempo; a una pagina usata almeno una volta per ogni periodo corrisponde la successione 11111111 nel registro a scorrimento.

Una pagina cui corrisponde la successione 11000100 nel relativo registro, è stata usata più recentemente di quanto non lo sia stata una cui è associata la successione 01110111. Interpretando queste successioni di bit come interi senza segno, la pagina cui è associato il numero minore è la pagina LRU, e può essere sostituita. In ogni caso l'unicità dei numeri non è garantita. Si possono sostituire (o scaricare dalla memoria all'area d'avvicendamento) tutte le pagine con il valore minore, oppure si può ricorrere a una selezione FIFO.

Il numero dei bit può ovviamente essere variato: si stabilisce secondo l'architettura disponibile per accelerarne al massimo la modifica. Nel caso limite tale numero si riduce a zero, lasciando soltanto il bit di riferimento e definendo un algoritmo noto come **algoritmo di sostituzione delle pagine con seconda chance**.

10.4.5.2 Algoritmo con seconda chance

L'algoritmo di base per la sostituzione con seconda chance è un algoritmo di sostituzione di tipo FIFO. Dopo aver selezionato una pagina si controlla il bit di riferimento, se il suo valore è 0, si sostituisce la pagina; se il bit di riferimento è impostato a 1, si dà una seconda chance alla pagina e la selezione passa alla successiva pagina FIFO. Quando una pagina riceve la seconda chance, si azzera il suo bit di riferimento e si aggiorna il suo istante d'arrivo al momento attuale. In questo modo, una pagina cui si offre una seconda chance non viene mai sostituita fino a che tutte le altre pagine non sono state sostituite, oppure non è stata offerta loro una seconda chance. Inoltre, se una pagina è usata abbastanza spesso, in modo che il suo bit di riferimento sia sempre impostato a 1, non viene mai sostituita.

Un metodo per realizzare l'algoritmo con seconda chance, detto anche a orologio (*clock*), è basato sull'uso di una coda circolare, nella quale un puntatore indica qual è la prima pagina da sostituire. Quando serve un blocco di memoria, si fa avanzare il puntatore fino a che si trova in corrispondenza di una pagina con il bit di riferimento 0, a ogni passo si azzera il bit di riferimento appena esaminato (Figura 10.14). Una volta trovata una pagina 'vittima', la si sostituisce e si inserisce la nuova pagina nella coda circolare nella posizione corrispondente. Si noti che nel caso peggiore, quando tutti i bit sono impostati a 1, il puntatore percorre un ciclo su tutta la coda, dando a ogni pagina una seconda chance. Prima di selezionare la pagina da sostituire, azzera tutti i bit di riferimento. Se tutti i bit sono a 1, la sostituzione con seconda chance si riduce a una sostituzione FIFO.

10.4.5.3 Algoritmo con seconda chance migliorato

L'algoritmo con seconda chance descritto precedentemente si può migliorare considerando i bit di riferimento e di modifica (si vedano i Paragrafi 10.4.1 e 10.4.5) come una coppia ordinata, con cui si possono ottenere le seguenti quattro classi:

1. (0, 0) né recentemente usato né modificato — migliore pagina da sostituire;
2. (0, 1) non usato recentemente, ma modificato — la pagina non così buona poiché prima di essere sostituita deve essere scritta nella memoria secondaria;

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

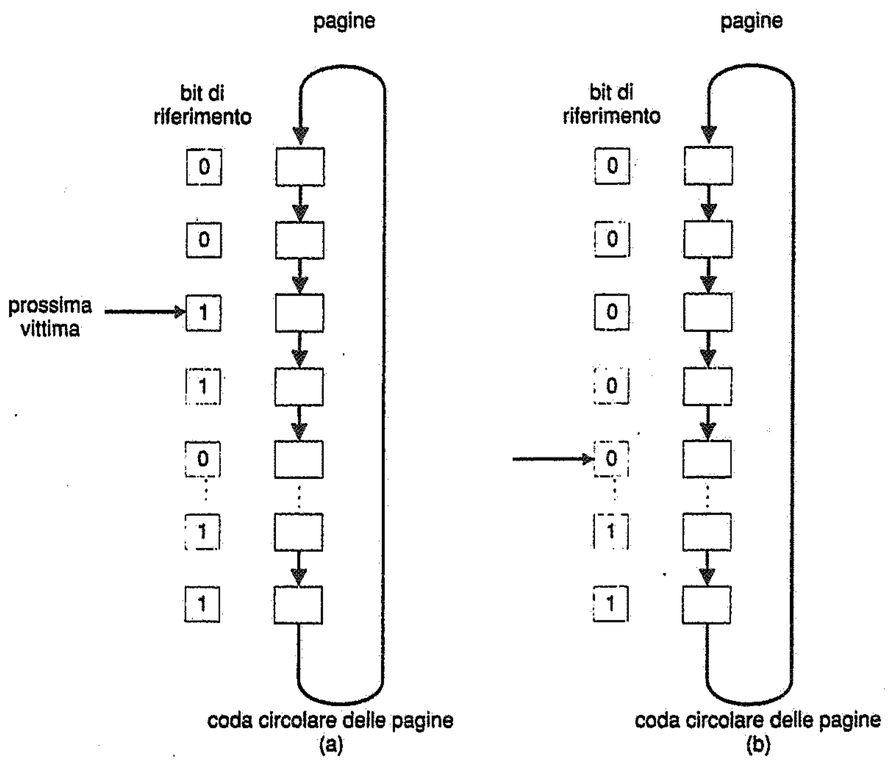


Figura 10.14 Algoritmo di sostituzione delle pagine con seconda chance (orologio).

3. (1, 0) usato recentemente ma non modificato — probabilmente la pagina sarà presto ancora usata;
4. (1, 1) usato recentemente e modificato — probabilmente la pagina sarà presto ancora usata e dovrà essere scritta nella memoria secondaria prima di essere sostituita.

Alla richiesta di una sostituzione di pagina, ogni pagina è in una tra le suddette classi. Si usa lo stesso schema impiegato nell'algoritmo a orologio, ma anziché controllare se la pagina puntata ha il bit di riferimento impostato a 1, si esaminano le classi cui la pagina appartiene e si sostituisce la prima pagina che si trova nella classe minima non vuota. Si noti che la coda circolare deve essere scandita più volte prima di trovare una pagina da sostituire.

Questo algoritmo è usato nello schema di gestione della memoria virtuale dell'Apple Macintosh. La differenza principale tra questo algoritmo e il più semplice algoritmo a orologio è che nel primo si dà la preferenza alle pagine che sono state modificate, al fine di ridurre il numero di I/O richiesti.

10.4.6 Sostituzione delle pagine basata su conteggio

Esistono molti altri algoritmi che si possono usare per la sostituzione delle pagine. Ad esempio, si potrebbe usare un contatore del numero dei riferimenti che sono stati fatti a ciascuna pagina e sviluppare i due seguenti schemi:

- Algoritmo di sostituzione delle pagine meno frequentemente usate (*least frequently used* — LFU), richiede che si sostituisca la pagina con il conteggio più basso. La ragione di questa scelta è che una pagina usata attivamente deve avere un conteggio di riferimento alto. Il punto debole di questo algoritmo è rappresentato dai casi in cui una pagina è usata molto intensamente durante la fase iniziale di un processo, ma poi non viene più usata. Poiché è stata usata intensamente il suo conteggio è alto, quindi rimane nella memoria anche se non è più necessaria. Una soluzione può essere quella di spostare i conteggi a destra di un bit a intervalli regolari, formando un conteggio per l'uso medio con esponente decrescente.
- Algoritmo di sostituzione delle pagine più frequentemente usate (*most frequently used* — MFU), è basato sul fatto che, probabilmente, la pagina con il contatore più basso è stata appena inserita e non è stata ancora usata.

Le sostituzioni MFU e LFU non sono molto comuni, poiché la realizzazione di questi algoritmi è abbastanza onerosa; inoltre, tali algoritmi non approssimano bene la sostituzione OPT.

10.4.7 Algoritmi con memorizzazione transitoria delle pagine

Oltre a uno specifico algoritmo, per la sostituzione delle pagine si usano spesso anche altre procedure; ad esempio, i sistemi hanno generalmente un gruppo di blocchi di memoria liberi. Quando si verifica un'assenza di pagina, si sceglie innanzi tutto un blocco di memoria vittima, ma prima che essa sia scritta nella memoria secondaria, si trasferisce la pagina richiesta in un blocco di memoria libero del gruppo. Questa procedura permette al processo di ricominciare al più presto, senza attendere che la pagina vittima sia scritta nella memoria secondaria. Quando nel seguito si scrive la vittima nella memoria secondaria, si aggiunge il suo blocco di memoria al gruppo dei blocchi liberi.

Quest'idea si può estendere conservando un elenco delle pagine modificate: ogniqualvolta il dispositivo di paginazione è inattivo, si sceglie una pagina modificata, la si scrive nel disco e si reimposta il suo bit di modifica. Questo schema aumenta la probabilità che, al momento della selezione per la sostituzione, la pagina non abbia subito modifiche e non debba essere scritta nella memoria secondaria.

Anche un'altra modifica prevede l'uso di un gruppo di blocchi di memoria liberi, ma, in questo caso, per ricordare quale pagina era contenuta in ciascun blocco di memoria. Poiché quando si scrive il contenuto di un blocco di memoria in un disco tale contenuto non cambia, se è necessaria, la vecchia pagina può essere ancora usata direttamente

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dal gruppo dei blocchi di memoria liberi, prima che quel blocco di memoria sia riusato. In questo caso non è necessario alcun I/O. Se si verifica un'assenza di pagina occorre controllare se la pagina richiesta si trova nel gruppo dei blocchi liberi; se non c'è si deve individuare un blocco di memoria libero e trasferirvi la pagina.

Questa tecnica, insieme con l'algoritmo di sostituzione FIFO, è usata dal sistema VAX/VMS. Quando l'algoritmo FIFO sostituisce per errore una pagina che è ancora in uso, la si ricupera rapidamente dal gruppo dei blocchi di memoria liberi senza ricorrere a operazioni di I/O. Il gruppo dei blocchi di memoria liberi protegge contro l'algoritmo di sostituzione FIFO, relativamente povero, ma semplice. Questo metodo è necessario poiché le prime versioni del VAX non disponevano del bit di riferimento correttamente realizzato.

10.5 Assegnazione dei blocchi di memoria

A questo punto occorre stabilire un criterio per l'assegnazione della memoria libera ai diversi processi. Come esempio, è possibile considerare un caso nel quale 93 blocchi di memoria liberi si debbano assegnare a due processi.

Il caso più semplice di memoria virtuale è il sistema con utente singolo. Si consideri un microcalcolatore con utente singolo che dispone di 128 KB di memoria, con pagine di 1 KB. Complessivamente sono presenti 128 blocchi di memoria. Il sistema operativo può occupare 35 KB, lasciando 93 blocchi di memoria per il processo utente. In condizioni di paginazione su richiesta pura, tutti i 93 blocchi di memoria sono inizialmente posti nell'elenco dei blocchi liberi. Quando comincia l'esecuzione, il processo utente genera una sequenza di eccezioni di pagine mancanti. Le prime 93 pagine assenti ricevono i blocchi di memoria liberi dall'elenco. Una volta esaurito l'elenco, per stabilire quale tra le 93 pagine presenti nella memoria si debba sostituire con la novantaquattresima, si può usare un algoritmo di sostituzione delle pagine. Terminato il processo, si reinseriscono i 93 blocchi di memoria nell'elenco dei blocchi liberi.

Questa strategia è semplice, ma può subire molte variazioni. Si può richiedere che il sistema operativo assegni tutto lo spazio richiesto dalle proprie strutture di dati attingendo all'elenco dei blocchi liberi. Quando questo spazio non è usato dal sistema operativo può essere sfruttato per la paginazione d'utente. Un'altra variante prevede di riservare sempre tre blocchi liberi, in modo che quando si verifica un'assenza di pagina sia disponibile un blocco di memoria libero in cui trasferire la pagina richiesta. Mentre ha luogo il trasferimento della pagina, si può fare una sostituzione, la pagina coinvolta viene poi scritta nel disco mentre il processo utente continua l'esecuzione.

Sono possibili anche altre varianti, ma la strategia di base è chiara: al processo utente si assegna qualsiasi blocco di memoria libero.

Un altro problema nasce quando la paginazione su richiesta è unita alla multiprogrammazione. Infatti, questa colloca nella memoria più processi contemporaneamente.

I due tipi di allocazione:
- Allocazione fissa
- Allocazione con priorità

è definito dall'architettura del sistema, è max dalla quantità di memoria fisica disponibile

10.5.1 Numero minimo di blocchi di memoria

Le strategie di assegnazione dei blocchi di memoria sono soggette a parecchi vincoli. Non si possono assegnare più blocchi di memoria di quanti siano disponibili, sempre che non vi sia condivisione di pagine. Inoltre, è necessario assegnare almeno un numero minimo di blocchi di memoria. Naturalmente, col diminuire del numero dei blocchi di memoria assegnati a ciascun processo aumenta la frequenza delle assenze di pagine, con conseguente rallentamento dell'esecuzione del processo.

Oltre ai fattori negativi riguardanti le prestazioni, connessi con l'assegnazione di pochi blocchi di memoria, esiste un numero minimo per i blocchi di memoria che si devono assegnare. Questo numero minimo è definito dalla struttura della serie delle istruzioni di macchina. Bisogna ricordare che quando si verifica un'assenza di pagina prima che sia stata completata l'esecuzione di un'istruzione, quest'ultima deve essere riavviata. Di conseguenza, i blocchi di memoria disponibili devono essere in numero sufficiente per contenere tutte le pagine cui ogni singola istruzione può fare riferimento.

Si consideri, ad esempio, un calcolatore in cui tutte le istruzioni di riferimento alla memoria hanno solo un indirizzo di memoria; in questo caso occorre almeno un blocco di memoria per l'istruzione e uno per il riferimento alla memoria. Inoltre se è ammesso un indirizzamento indiretto a un livello, ad esempio, un'istruzione load presente nella pagina 16 può fare riferimento a un indirizzo della pagina 0, che costituisce a sua volta un riferimento indiretto alla pagina 23; così la paginazione richiede almeno tre blocchi di memoria per ogni processo. Si consideri che cosa accadrebbe nel caso di un processo che dispone di due soli blocchi di memoria.

Il numero minimo dei blocchi di memoria è definito dalla data architettura del calcolatore. Ad esempio, l'istruzione MOV del PDP-11, per alcuni modi di indirizzamento, è costituita di più di una parola, quindi la stessa istruzione può stare a cavallo tra due pagine. Inoltre, ciascuno dei suoi due operandi può essere un riferimento indiretto, per un totale di sei blocchi di memoria. Il caso peggiore che può verificarsi per l'IBM 370 è probabilmente quello relativo all'istruzione MVC. Poiché l'istruzione è da memoria a memoria, può occupare 6 byte e stare a cavallo tra due pagine. Anche la sequenza di caratteri da spostare e l'area su cui effettuare lo spostamento possono essere a cavallo tra due pagine; questa situazione richiede sei blocchi di memoria. In effetti, la situazione peggiore si presenta quando l'istruzione MVC è l'operando di un'istruzione EXECUTE che sta a cavallo di un limite di pagina; in questo caso occorrono otto blocchi di memoria.

Il caso peggiore si può presentare nelle architetture di calcolatori che permettono riferimenti indiretti a più livelli, ad esempio ogni parola di 16 bit può contenere un indirizzo di 15 bit più un indicatore indiretto di 1 bit. In teoria, una semplice istruzione di caricamento può fare riferimento a un indirizzo indiretto che a sua volta può fare riferimento a un indirizzo indiretto (su un'altra pagina) anch'esso facente riferimento a un indirizzo indiretto su un'altra pagina ancora, e così via, fino a che tutte le pagine della memoria virtuale sono state chiamate in causa. Quindi, nel caso peggiore, tutta la memoria virtuale si deve trovare nella memoria fisica. Per superare questa difficoltà occorre porre

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

un limite al livello dei riferimenti indiretti, ad esempio limitando un'istruzione a un massimo di 16 livelli. Quando si verifica il riferimento indiretto di primo livello, si imposta un contatore al valore 16, per decrementarlo a ciascun livello successivo relativo a questa istruzione. Se il contatore si riduce a 0 si verifica un segnale di eccezione (livello di riferimenti indiretti eccessivo). Tale limitazione riduce a 17 il numero massimo dei riferimenti alla memoria per ogni istruzione, richiedendo un pari numero di blocchi di memoria.

Il numero minimo di blocchi di memoria per ciascun processo è definito dall'architettura, mentre il numero massimo è definito dalla quantità di memoria fisica disponibile. Rimane ancora aperta la questione della scelta dell'assegnazione dei blocchi di memoria.

Allocazione fissa

10.5.2 Algoritmi di assegnazione

Il modo più semplice per suddividere m blocchi di memoria tra n processi è quello per cui a ciascuno si dà una parte uguale, m/n blocchi di memoria. Dati 93 blocchi di memoria e cinque processi, ogni processo riceve 18 blocchi di memoria. I tre blocchi di memoria lasciati liberi si potrebbero usare come gruppo dei blocchi liberi (si veda il Paragrafo 10.4.7). Questo schema è chiamato **assegnazione uniforme**.

Un'alternativa consiste nel riconoscere che diversi processi hanno bisogno di quantità di memoria diverse. Si consideri un sistema con blocchi di memoria di 1 KB. Se un piccolo processo utente di 10 KB e una base di dati interattiva di 127 KB sono gli unici due processi in esecuzione su un sistema con 62 blocchi di memoria liberi, non ha senso assegnare a ciascun processo 31 blocchi di memoria. Al processo utente non ne servono più di 10, quindi gli altri 21 sarebbero semplicemente sprecati.

Per risolvere questo problema è possibile ricorrere all'**assegnazione proporzionale**, secondo cui la memoria disponibile si assegna a ciascun processo secondo la sua dimensione. Si supponga che s_i sia la dimensione della memoria virtuale per il processo p_i. Si definisce la seguente quantità:

• S = somma di tutte le grandezze dei processi

$$\boxed{S = \sum s_i.}$$

• S_i = la grandezza del processo P_i

Quindi, se il numero totale dei blocchi di memoria disponibili è m, al processo p_i si assegnano a_i blocchi di memoria, dove a_i è approssimativamente

$$\boxed{a_i = s_i/S \times m.}$$

Naturalmente è necessario scegliere ciascun a_i in modo che sia un intero maggiore del numero minimo dei blocchi di memoria richiesti dalla struttura della serie di istruzioni di macchina e in modo che la somma di tutti gli a_i non sia maggiore di m.

Usando l'assegnazione proporzionale, per suddividere 62 blocchi di memoria tra due processi, uno di 10 e uno di 127 pagine, si assegnano rispettivamente 4 e 57 blocchi di memoria, infatti

$$10/137 \times 62 \approx 4,$$
$$127/137 \times 62 \approx 57.$$

In questo modo entrambi i processi condividono i blocchi di memoria disponibili secondo le rispettive necessità, e non in modo uniforme.

→ Allocazione a priorità → si utilizza la stessa di allocazione proporzionale, impiegando delle priorità, piuttosto che la dimensione. Se a processo con alta priorità manca memoria • si seleziona x dei suoi blocchi

Sia nell'assegnazione uniforme sia in quella proporzionale, l'assegnazione a ogni processo può variare rispetto al livello di multiprogrammazione. Se si aumenta il livello di multiprogrammazione, ciascun processo perde alcuni blocchi di memoria per fornire la memoria necessaria per il nuovo processo. D'altra parte, se il livello di multiprogrammazione diminuisce, i blocchi di memoria che erano stati assegnati al processo allontanato si possono distribuire tra i processi che restano.

Occorre notare che sia con l'assegnazione uniforme sia con l'assegnazione proporzionale, un processo a priorità elevata è trattato come un processo a bassa priorità anche se, per definizione, si vorrebbe che al processo con elevata priorità fosse assegnata più memoria per accelerarne l'esecuzione, a discapito dei processi a bassa priorità.

Un modo di affrontare tale problema prevede l'uso di uno schema di assegnazione proporzionale in cui il rapporto dei blocchi di memoria non dipende dalle dimensioni relative dei processi, ma dalle priorità degli stessi oppure da una combinazione di dimensioni e priorità.

• si seleziona x da sostituzione a blocco di a processo con alta priorità tocca

10.5.3 Assegnazione globale e assegnazione locale

Un altro importante fattore che riguarda il modo in cui si assegnano i blocchi di memoria ai vari processi è la sostituzione delle pagine. Nei casi in cui vi siano più processi in competizione per i blocchi di memoria, gli algoritmi di sostituzione delle pagine si possono classificare in due categorie generali: **sostituzione globale** e **sostituzione locale**. La sostituzione globale permette che per un processo si scelga un blocco di memoria per la sostituzione dall'insieme di tutti i blocchi di memoria, anche se quel blocco di memoria è correntemente assegnato a qualche altro processo; un processo può dunque sottrarre un blocco di memoria a un altro processo. La sostituzione locale richiede invece che per ogni processo si scelga un blocco di memoria solo dal suo insieme di blocchi di memoria.

Si consideri ad esempio uno schema di assegnazione che, per una sostituzione a favore dei processi ad alta priorità, permetta di sottrarre blocchi di memoria ai processi a bassa priorità. Per un processo si può stabilire una sostituzione che attinga tra i suoi blocchi di memoria oppure tra i blocchi di memoria di qualsiasi processo con priorità minore. Questo metodo permette a un processo ad alta priorità di aumentare il proprio livello di assegnazione di blocchi di memoria a discapito del processo a bassa priorità.

Con la strategia di sostituzione locale, il numero di blocchi di memoria assegnati a un processo non cambia. Con la sostituzione globale, invece, può accadere che per un certo processo si selezionino solo blocchi di memoria assegnati ad altri processi, aumentando così il numero di blocchi di memoria assegnati a quel processo, purché per altri processi non si scelgano per la sostituzione i *suoi* blocchi di memoria.

L'algoritmo di sostituzione globale risente di un problema: un processo non può controllare la propria frequenza di assenze di pagine, infatti l'insieme di pagine che si trova nella memoria per un processo non dipende solo dal comportamento di paginazione di quel processo, ma anche dal comportamento di paginazione di altri processi. Quindi, lo stesso processo può comportarsi in modi piuttosto diversi, ad esempio impiegando

• la sostituzione globale garantisce > throughput ⇒ più utilizzate nei S.O. più diffuse

http://www.chinapdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

0,5 secondi per un'esecuzione e 10,3 secondi per l'esecuzione successiva, a causa di circostanze esterne. Con l'algoritmo di sostituzione locale questo problema non si presenta. Infatti l'insieme di pagine nella memoria per un processo subisce l'effetto del comportamento di paginazione di quel solo processo. Dal canto suo, la sostituzione locale può limitare un processo, non rendendogli disponibili altre pagine di memoria meno usate. Generalmente, la sostituzione globale genera una maggiore produttività del sistema, perciò è il metodo più usato.

10.6 Attività di paginazione degenere

Se il numero dei blocchi di memoria assegnati a un processo con priorità bassa diviene inferiore al numero minimo richiesto dall'architettura del calcolatore, occorre sospendere l'esecuzione del processo, e quindi togliere le pagine restanti, liberando tutti i blocchi di memoria assegnati. Questa operazione introduce un livello intermedio di scheduling.

Infatti, si consideri un qualsiasi processo che non disponga di un numero sufficiente di blocchi di memoria. Anche se tecnicamente si può ridurre al valore minimo il numero dei blocchi di memoria assegnati, esiste un certo (in generale grande) numero di pagine in uso attivo. Se non dispone di questo numero di blocchi di memoria, il processo accusa immediatamente un'assenza di pagina. A questo punto si deve sostituire qualche pagina; ma, poiché tutte le sue pagine sono in uso attivo, si deve sostituire una pagina che sarà immediatamente necessaria, e di conseguenza si verificano subito parecchie assenze di pagine. Il processo continua a subire assenze di pagine, facendo sostituire pagine che saranno immediatamente trattate come assenti e dovranno essere riprese.

Questa intensa quanto degenere attività di paginazione (nota come *thrashing*) si verifica quando si spende più tempo per la paginazione che per l'esecuzione dei processi.

10.6.1 Cause dell'attività di paginazione degenere

La degenerazione dell'attività di paginazione causa parecchi problemi di prestazioni. Si consideri il seguente scenario, basato sul comportamento effettivo dei primi sistemi di paginazione.

Il sistema operativo vigila sull'utilizzo della CPU. Se questo è basso, aumenta il grado di multiprogrammazione introducendo un nuovo processo. Si usa un algoritmo di sostituzione delle pagine globale, che sostituisce le pagine senza tener conto del processo al quale appartengono. Per ora si ipotizza che un processo entri in una nuova fase d'esecuzione e richieda più blocchi di memoria; se ciò si verifica si ha una serie di assenze di pagine, cui segue la sottrazione di nuove pagine ad altri processi. Questi processi hanno però bisogno di quelle pagine e quindi subiscono anch'essi delle assenze di pagine, con conseguente sottrazione di pagine ad altri processi. Per effettuare il caricamento e lo scaricamento delle pagine per questi processi si deve usare il dispositivo di paginazione. Mentre si mettono i processi in coda per il dispositivo di paginazione, la coda dei processi pronti per l'esecuzione si svuota, quindi l'utilizzo della CPU diminuisce.

Lo scheduler della CPU rileva questa riduzione dell'utilizzo della CPU e *aumenta* il grado di multiprogrammazione. Si tenta di avviare il nuovo processo sottraendo pagine ai processi in esecuzione, causando ulteriori assenze di pagine e allungando la coda per il dispositivo di paginazione. L'utilizzo della CPU scende ulteriormente, e lo scheduler della CPU tenta di aumentare ancora il grado di multiprogrammazione. L'attività di paginazione è degenerata in una situazione patologica che fa precipitare la produttività del sistema. La frequenza delle assenze di pagine aumenta in modo impressionante, e di conseguenza aumenta il tempo effettivo d'accesso alla memoria. I processi non svolgono alcun lavoro, poiché si sta spendendo tutto il tempo nell'attività di paginazione.

Questo fenomeno è illustrato nella Figura 10.15, nella quale si riporta l'utilizzo della CPU in funzione del grado di multiprogrammazione. Aumentando il grado di multiprogrammazione aumenta anche l'utilizzo della CPU, anche se più lentamente, fino a raggiungere un massimo. Se a questo punto si aumenta ulteriormente il grado di multiprogrammazione, l'attività di paginazione degenera e fa crollare l'utilizzo della CPU. In questa situazione, per aumentare l'utilizzo della CPU occorre *ridurre* il grado di multiprogrammazione.

Gli effetti di questa situazione si possono limitare usando un **algoritmo di sostituzione locale**, o **algoritmo di sostituzione per priorità**. Con la sostituzione locale, se l'attività di paginazione per un processo degenera, non si possono sottrarre blocchi di memoria a un altro processo e quindi non si può far degenerare la relativa attività di paginazione. Le pagine si sostituiscono tenendo conto del processo di cui fanno parte. Tuttavia, se i processi sono in situazioni di paginazione degeneri, rimangono nella coda d'attesa del dispositivo di paginazione per la maggior parte del tempo. Il tempo di servizio medio di un eccezione di pagina mancante aumenta a causa dell'allungamento medio della coda d'attesa del dispositivo di paginazione. Di conseguenza, il tempo effettivo d'accesso al dispositivo di paginazione aumenta anche per gli altri processi.

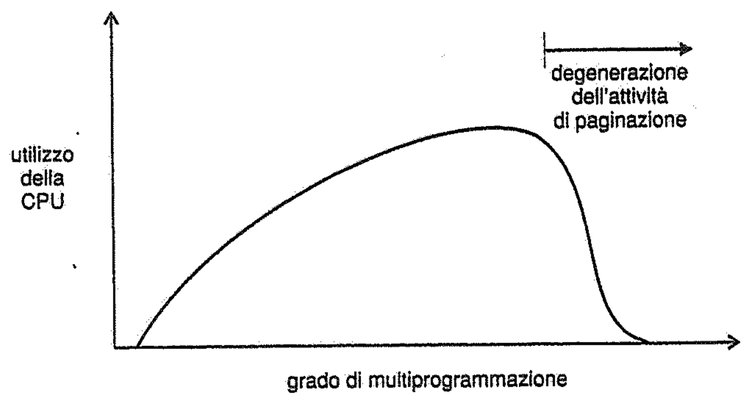


Figura 10.15 Degenerazione dell'attività di paginazione (*thrashing*).

http://www.adultpdf.com
Created by TIFF To PDF trial version, please visit www.tiffpdf.com to remove this watermark, please register this software.

* Perché avviene il thrashing?
dim. località > dim. della memoria (disponibile x il processo)

Per evitare il verificarsi di queste situazioni, occorre fornire a un processo tutti i blocchi di memoria di cui necessita. Per cercare di sapere ~~di sapere~~ quanti blocchi di memoria 'servono' a un processo si impiegano diverse tecniche. Il modello dell'insieme di lavoro, trattato nel Paragrafo 10.6.2, comincia osservando quanti sono i blocchi di memoria che un processo sta effettivamente usando. Questo metodo definisce il **modello di località d'esecuzione del processo.** *

Il modello di località stabilisce che un processo, durante la sua esecuzione, si sposta di località in località. Una località è un insieme di pagine usate attivamente, com'è illustrato nella Figura 10.16. Generalmente un programma è formato di parecchie località diverse, che si possono sovrapporre.

Ad esempio, quando s'invoca una procedura, essa definisce una nuova località. In questa località si fanno riferimenti alla memoria alle istruzioni della procedura, alle sue variabili locali e a un sottoinsieme delle variabili globali.

Quando la procedura termina, il processo lascia questa località, poiché le variabili locali e le istruzioni della procedura non sono più usate attivamente. Quindi, le località sono definite dalla struttura del programma e dalle relative strutture di dati. Il modello di località sostiene che tutti i programmi mostrano questa struttura di base di riferimenti alla memoria. Si noti che il modello di località è il principio non dichiarato sottostante alla discussione fin qui svolta sul caching. Se gli accessi ai vari tipi di dati fossero casuali, anziché strutturati in località, il caching sarebbe inutile.

Si supponga di assegnare a un processo un numero di blocchi di memoria sufficiente per sistemare le sue località attuali. Fino a che tutte queste pagine non si trovano nella memoria, si verificano le assenze delle pagine relative a tali località; quindi non hanno luogo altre assenze di pagine fino a che le località non vengono modificate. Se si assegnano meno blocchi di memoria rispetto alla dimensione della località attuale, l'attività di paginazione relativa al processo degenera, poiché non si possono tenere nella memoria tutte le pagine che il processo sta usando attivamente.

10.6.2 Modello dell'insieme di lavoro

Il **modello dell'insieme di lavoro** (*working-set model*) è basato sull'ipotesi di località. Questo modello usa un parametro, δ, per definire la **finestra dell'insieme di lavoro**. L'idea consiste nell'esaminare i più recenti δ riferimenti alle pagine. L'insieme di pagine nei più recenti δ riferimenti è l'insieme di lavoro; si veda a questo proposito la Figura 10.17. Se una pagina è in uso attivo si trova nell'insieme di lavoro; se non è più usata esce dall'insieme di lavoro δ unità di tempo dopo il suo ultimo riferimento. Quindi, l'insieme di lavoro non è altro che un'approssimazione della località del programma.

Ad esempio, data la successione di riferimenti alla memoria mostrata nella Figura 10.17, se $\delta = 10$ riferimenti alla memoria, l'insieme di lavoro all'istante t_1 è $\{1, 2, 5, 6, 7\}$. All'istante t_2 l'insieme di lavoro è diventato $\{3, 4\}$.

La precisione dell'insieme di lavoro dipende dalla scelta del valore di δ. Se δ è troppo piccolo non include l'intera località, se è troppo grande può sovrapporre più località. Al limite, se δ è infinito l'insieme di lavoro coincide con l'insieme di pagine cui il processo fa riferimento durante la sua esecuzione.

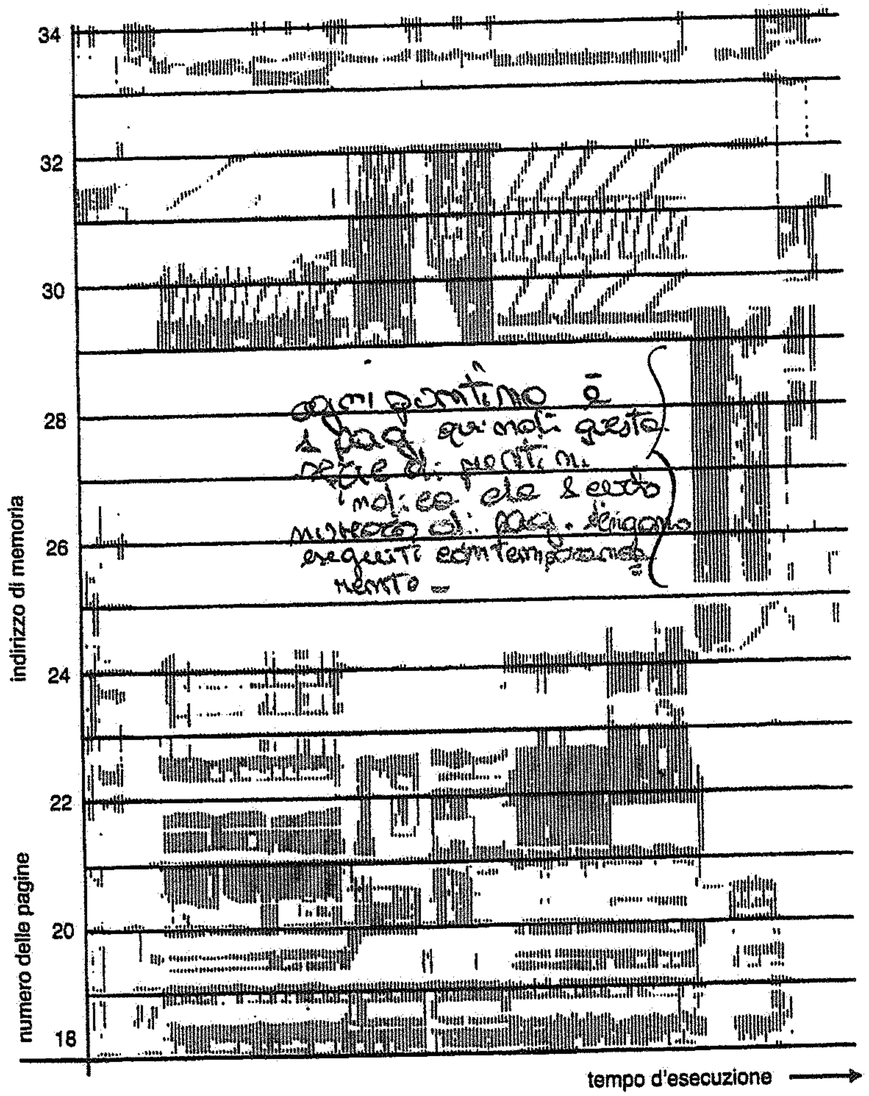


Figura 10.16 Località di riferimenti alla memoria.

La caratteristica più importante dell'insieme di lavoro è la sua dimensione. Calcolando la dimensione dell'insieme di lavoro, WSS_i, per ciascun processo p_i del sistema, si può determinare la richiesta totale di blocchi di memoria:

$$D = \sum WSS_i$$

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

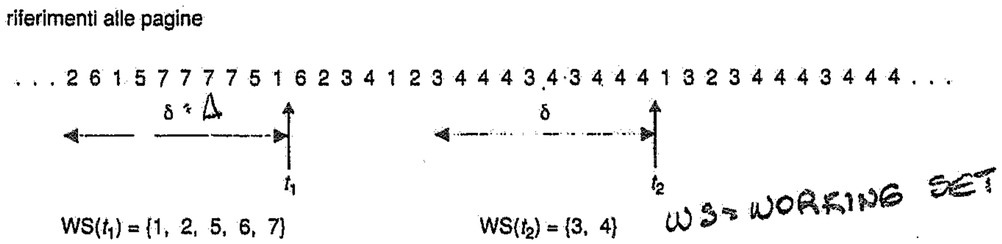


Figura 10.17 Modello dell'insieme di lavoro.

Ogni processo usa attivamente le pagine del proprio insieme di lavoro. Quindi, il processo i necessita di WSS_i blocchi di memoria. Se la richiesta totale è maggiore del numero totale di blocchi liberi ($D > m$), l'attività di paginazione degenera, poiché alcuni processi non dispongono di un numero sufficiente di blocchi di memoria.

Come funziona il WORKING SET

L'uso del modello dell'insieme di lavoro è abbastanza semplice. Il sistema operativo controlla l'insieme di lavoro di ogni processo e gli assegna un numero di blocchi di memoria sufficiente, rispetto alle dimensioni del suo insieme di lavoro. Se i blocchi di memoria ancora liberi sono in numero sufficiente, si può iniziare un altro processo. Se la somma delle dimensioni degli insiemi di lavoro aumenta, superando il numero totale dei blocchi di memoria disponibili, il sistema operativo individua un processo da sospendere. Scrive nella memoria secondaria le pagine di quel processo e assegna i suoi blocchi di memoria ad altri processi. Il processo sospeso può essere ripreso successivamente.

Questa strategia impedisce la degenerazione dell'attività di paginazione, mantenendo il grado di multiprogrammazione più alto possibile, sicché ottimizza l'utilizzo della CPU.

Poiché la finestra dell'insieme di lavoro è una finestra dinamica, la difficoltà insita in questo modello consiste nel tener traccia degli elementi che compongono l'insieme di lavoro stesso. A ogni riferimento alla memoria, a un'estremità appare un riferimento nuovo e il riferimento più vecchio fuoriesce dall'altra estremità. Una pagina si trova nell'insieme di lavoro se esiste un riferimento a essa in qualsiasi punto della finestra dell'insieme di lavoro. Il modello dell'insieme di lavoro si può approssimare con un segnale d'interruzione del temporizzatore a intervalli fissi e con un bit di riferimento.

Si supponga, ad esempio, che δ sia pari a 10.000 riferimenti e che sia possibile ottenere un segnale d'interruzione dal temporizzatore ogni 5000 riferimenti. Quando si verifica uno di tali segnali d'interruzione, i valori dei bit di riferimento di ciascuna pagina vengono copiati e poi azzerati. Così, quando si verifica un'assenza di pagina si può esaminare il bit di riferimento corrente e i 2 bit nella memoria per stabilire se una pagina è stata usata entro gli ultimi 10.000 - 15.000 riferimenti. Se lo è stata, almeno uno di questi bit è attivo. Se non lo è stata, questi bit sono tutti inattivi. Le pagine con almeno un bit attivo si considerano appartenenti all'insieme di lavoro. Occorre notare che questo schema non è del tutto preciso, poiché non è possibile stabilire dove si è verificato un

riferimento entro un intervallo di 5000. L'incertezza si può ridurre aumentando il numero dei bit cronologici e la frequenza dei segnali d'interruzione, ad esempio, 10 bit e un'interruzione ogni 1000 riferimenti. Tuttavia, il costo per servire questi segnali d'interruzione più frequenti aumenta in modo corrispondente.

10.6.3 Frequenza delle assenze di pagine

Il modello dell'insieme di lavoro riscuote un discreto successo, e la sua conoscenza può servire per la prepaginazione (discussa nel Paragrafo 10.8.1), ma appare un modo alquanto goffo per controllare la degenerazione dell'attività di paginazione. La strategia basata sulla **frequenza delle assenze di pagine** (*page fault frequency* — PFF) è più diretta.

Il problema specifico è la prevenzione delle situazioni di paginazione degeneri. La frequenza delle assenze di pagine in tali situazioni è alta, ed è proprio questa che si deve controllare. Se la frequenza delle assenze di pagine è eccessiva, significa che il processo necessita di più blocchi di memoria. Analogamente, se la frequenza delle assenze di pagine è molto bassa, allora il processo potrebbe disporre di troppi blocchi di memoria. Si può fissare un limite inferiore e un limite superiore per la frequenza desiderata delle assenze di pagine, com'è illustrato nella Figura 10.18. Se la frequenza effettiva delle assenze di pagine per un processo supera il limite superiore, occorre assegnare a quel processo un altro blocco di memoria; se la frequenza scende sotto il limite inferiore, si sottrae un blocco di memoria a quel processo. Quindi, per prevenire la degenerazione dell'attività di paginazione, si può misurare e controllare direttamente la frequenza delle assenze di pagine.

Come nel caso dell'insieme di lavoro, può essere necessaria la sospensione di un processo. Se la frequenza delle assenze di pagine aumenta e non ci sono blocchi di memoria disponibili, occorre selezionare un processo e sospenderlo. I blocchi di memoria liberati si distribuiscono ai processi con elevate frequenze di assenze di pagine.

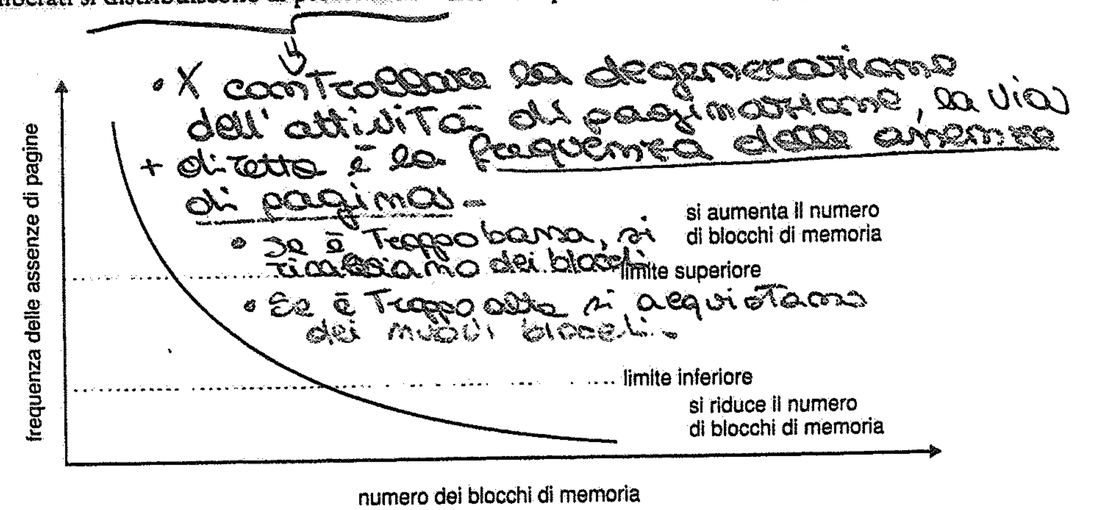


Figura 10.18 Frequenza delle assenze di pagine. = PAGE FAULT = quando a pag. non è trovata in memoria centrale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

10.7 Esempi tra i sistemi operativi

In questo paragrafo si descrive la realizzazione della memoria virtuale nei sistemi operativi Windows NT e Solaris 2.

10.7.1 Windows NT

Il sistema operativo Windows NT realizza la memoria virtuale impiegando la paginazione su richiesta per gruppi di pagine (*demand paging with clustering*). Tale tecnica consiste nel gestire le assenze di pagine caricando nella memoria, non solo la pagina richiesta, ma più pagine a essa adiacenti. Alla sua creazione, un processo riceve i valori del minimo insieme di lavoro e del massimo insieme di lavoro. Il **minimo insieme di lavoro** è il minimo numero di pagine caricate nella memoria di un processo che il sistema garantisce di assegnare, ma se la memoria è sufficiente, il sistema potrebbe assegnare un numero di pagine pari al suo **massimo insieme di lavoro** (in alcuni casi, si potrebbe anche permettere di superare questo valore). Il gestore della memoria virtuale mantiene una lista di pagine fisiche libere, con associato un valore di soglia che indica se è disponibile una quantità sufficiente di memoria libera oppure no. Se si verifica un'assenza di pagina per un processo che è sotto il suo massimo insieme di lavoro, il gestore della memoria virtuale assegna una pagina dalla lista delle pagine libere; se invece un processo è già al suo massimo insieme di lavoro e si verifica un'assenza di pagina, il gestore deve scegliere una pagina da sostituire usando un criterio di sostituzione locale. Nel caso in cui la quantità di memoria libera scenda sotto la soglia, il gestore della memoria virtuale usa un metodo noto come **regolazione automatica dell'insieme di lavoro** (*automatic working-set trimming*) per riportare il valore sopra la soglia. Si tratta sostanzialmente di valutare il numero di pagine assegnate a ciascun processo; se a un processo sono state assegnate più pagine del suo minimo insieme di lavoro, il gestore della memoria virtuale rimuove pagine fino a raggiungere quel valore; a un processo che è al suo minimo insieme di lavoro, può assegnare altre pagine prendendole dalla lista delle pagine fisiche libere, non appena è disponibile una quantità sufficiente di memoria libera.

L'algoritmo impiegato per stabilire quale pagina rimuovere da un insieme di lavoro dipende dal tipo di unità d'elaborazione disponibile: nei sistemi 80*x*86 con una CPU, il sistema operativo Windows NT usa una variante dell'algoritmo a orologio, presentato nel Paragrafo 10.4.5.2; nei sistemi basati su CPU Alpha e nei sistemi con più unità d'elaborazione 80*x*86, l'azzeramento del bit di riferimento può richiedere l'invalidamento dell'elemento corrispondente nel TLB delle altre unità d'elaborazione. Perciò anziché accettare questo onere, nel sistema Windows NT si usa una variante dell'algoritmo FIFO discusso nel Paragrafo 10.4.2.

10.7.2 Solaris 2

Il nucleo del sistema operativo Solaris 2 assegna una pagina a un thread ogni volta che si verifica un'assenza di pagina, prendendola dalla lista delle pagine libere che il nucleo stesso mantiene. È quindi essenziale che il nucleo riesca a mantenere una quantità sufficiente di memoria libera. Un parametro, *lotsfree*, associato alla lista delle pagine libere, rappresenta una soglia per l'inizio del processo di paginazione, e il suo è di solito fissato a 1/64 della dimensione della memoria fisica. Il nucleo verifica, quattro volte al secondo, se la quantità di memoria libera è inferiore a *lotsfree*.

Se il numero di pagine libere scende sotto *lotsfree*, si avvia un processo noto come pageout. Questo processo è simile all'algoritmo con seconda chance descritto nel Paragrafo 10.4.5.2 — noto anche come algoritmo a orologio con due lancette (*two-handed-clock*) —, che funziona come segue: la prima lancetta (*hand*) esamina tutte le pagine nella memoria, azzerandone il bit di riferimento; successivamente, la seconda lancetta esamina il bit di riferimento per le pagine nella memoria restituendo alla lista delle pagine libere tutte quelle pagine il cui bit di riferimento è ancora 0.

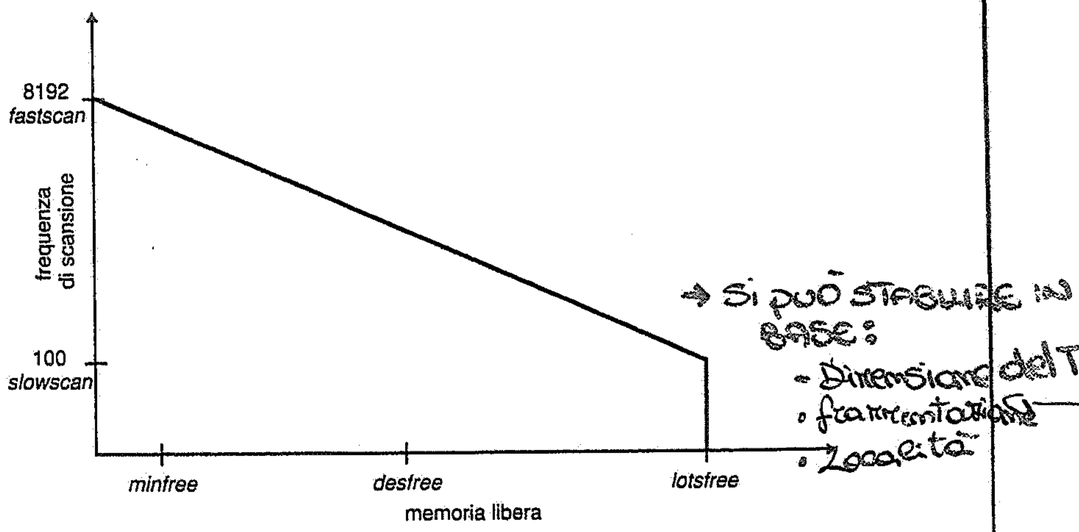
L'algoritmo pageout si serve di diversi parametri per controllare la frequenza di scansione delle pagine (chiamata anche *scanrate*). Questa frequenza è espressa in pagine al secondo ed è compresa tra i valori *slowscan* e *fastscan*. Quando la memoria libera scende sotto *lotsfree*, la scansione delle pagine avviene alla frequenza *slowscan*, e sale fino a *fastscan* secondo la quantità di memoria libera disponibile. Il valore predefinito di *slowscan* è 100, mentre *fastscan* è di solito fissato a (*numero totale delle pagine fisiche*)/2 con un massimo di 8192 pagine al secondo. Questa variazione di frequenza è illustrata nella Figura 10.19 (con *fastscan* fissato al massimo).

La distanza (in pagine) tra le lancette dell'orologio è determinata dal parametro di sistema *handspread*. L'intervallo tra l'azzeramento di un bit da parte della lancetta anteriore e l'esame del suo valore da parte della lancetta posteriore dipende sia da *scanrate* sia da *handspread*. Se il valore di *scanrate* è pari a 100 pagine al secondo e quello di *handspread* è pari a 1024 pagine, possono passare 10 secondi tra la scrittura di un bit della lancetta anteriore e la sua verifica da parte di quella posteriore. Tuttavia, visti i requisiti imposti a un sistema di memoria, non sono rari valori di *scanrate* di diverse migliaia di pagine al secondo. Questo significa che l'intervallo tra l'azzeramento e il controllo di un bit è spesso di pochi secondi.

Come si è descritto sopra, il processo pageout controlla la memoria quattro volte al secondo. Tuttavia, se la memoria libera scende sotto *desfree* (Figura 10.19) pageout sarà eseguito 100 volte al secondo con lo scopo di tenere una quantità di memoria libera almeno pari a *desfree*. Se il processo pageout non riesce a mantenere al valore *desfree* la quantità media di memoria libera calcolata in un intervallo di 30 secondi, il nucleo intraprende l'avvicendamento dei processi, liberando, in questo caso, tutte le pagine assegnate a un processo. In generale, il nucleo cerca i processi che sono rimasti inattivi per lunghi periodi. Infine, se il sistema non riesce a mantenere la quantità di memoria libera a *minfree*, invoca il processo pageout a ogni richiesta di una nuova pagina.

Le versioni recenti del nucleo del Solaris 2 hanno portato alcuni miglioramenti all'algoritmo di paginazione. Uno è il riconoscimento delle pagine che appartengono a librerie condivise da più processi: anche se sono potenzialmente richiedibili per la scansione, sono ignorate durante il processo d'esame delle pagine. Un altro miglioramento riguarda la capacità di distinguere le pagine assegnate ai processi dalle pagine assegnate ai file ordinari. Si tratta del meccanismo di **paginazione con priorità** descritto nel Paragrafo 12.6.2.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.



→ Si può stabilire in base:
- Dimensione del TLB
- frammentazione
- Località

Figura 10.19 Scansione delle pagine nel Solaris 2.

10.8 Altre considerazioni

In un sistema di paginazione, le decisioni più importanti riguardano la scelta di un algoritmo di sostituzione e il criterio di assegnazione. Si devono però fare anche molte altre considerazioni.

10.8.1 Prepaginazione

→ Si portano in memoria tutte le pag. richieste in 1 sola volta

Una caratteristica ovvia per un sistema di paginazione su richiesta pura consiste nell'alto numero di assenze di pagine che si verificano all'avvio di un processo. Questa situazione è dovuta al tentativo di portare la località iniziale nella memoria. La stessa situazione si può presentare anche in altri momenti. Ad esempio, quando si riavvia un processo scaricato nell'area d'avvicendamento, tutte le sue pagine si trovano nel disco e ognuna di esse deve essere reinserita nella memoria tramite la gestione di un'eccezione di pagina mancante. La prepaginazione rappresenta un tentativo di prevenire un livello così alto di paginazione iniziale. La strategia prevede di portare nella memoria in un'unica soluzione tutte le pagine richieste.

In un sistema che usi il modello dell'insieme di lavoro, ad esempio, a ogni processo si associa un elenco delle pagine contenute nel suo insieme di lavoro. Se occorre sospendere un processo a causa di un'attesa di I/O oppure dell'assenza di blocchi liberi, si memorizza il suo insieme di lavoro. Al momento di riprendere l'esecuzione del processo, al completamento dell'I/O oppure quando si raggiunge il numero di blocchi di memoria

sufficiente, prima di riavviare il processo, si riporta nella memoria il suo intero insieme di lavoro.

In alcuni casi la prepaginazione può essere vantaggiosa. La questione riguarda semplicemente il suo costo, che deve essere inferiore al costo del servizio delle eccezioni di pagine mancanti corrispondenti. Può accadere che molte pagine trasferite nella memoria dalla prepaginazione non siano usate. Si supponga che siano prepaginate s pagine e sia effettivamente usata una frazione αs di queste s pagine ($0 \le \alpha \le 1$). Occorre sapere se il costo delle αs eccezioni di pagine mancanti risparmiate sia maggiore o minore del costo di prepaginazione di $(1 - \alpha)s$ pagine non necessarie. Se il parametro α è prossimo allo 0, la prepaginazione non è conveniente; se α è prossimo a 1, la prepaginazione è certamente conveniente.

10.8.2 Dimensione delle pagine

È raro che chi progetta un sistema operativo per un calcolatore esistente possa scegliere le dimensioni delle pagine. Tuttavia, se si devono progettare nuovi calcolatori occorre stabilire quali siano le dimensioni migliori per le pagine. Come s'intuisce non esiste un'unica dimensione migliore, ma più fattori sono a sostegno delle diverse dimensioni. Le dimensioni delle pagine sono invariabilmente potenze di 2, in genere comprese tra 4096 (2^{12}) e 4.194.304 (2^{22}) byte.

Un fattore da considerare nella scelta delle dimensioni di una pagina è la dimensione della tabella delle pagine. Per un dato spazio di memoria virtuale, diminuendo la dimensione delle pagine aumenta il numero delle stesse e quindi la dimensione della tabella delle pagine. Per una memoria virtuale di 4 MB (2^{22}), ci sarebbero 4096 pagine di 1024 byte, ma solo 512 pagine di 8192 byte. Poiché ogni processo attivo deve avere la propria copia della tabella delle pagine, conviene che le pagine siano ampie.

D'altra parte, la memoria è utilizzata meglio se le pagine sono piccole. Se a un processo si assegna una porzione della memoria che comincia alla locazione 00000 e continua fino alla quantità di cui necessita, è molto probabile che il processo non termini esattamente sul limite di una pagina, lasciando inutilizzata (le pagine sono unità di assegnazione) una parte della pagina finale (frammentazione interna). Supponendo che le dimensioni del processo e delle pagine siano indipendenti è probabile che, in media, metà dell'ultima pagina di ogni processo sia sprecata. Questa perdita è di soli 256 byte per una pagina di 512 byte, ma di 4096 byte per una pagina di 8192 byte. Quindi, per ridurre la frammentazione interna occorrono pagine di piccole dimensioni.

Un altro problema è dovuto al tempo richiesto per leggere o scrivere una pagina. Il tempo di I/O è dato dalla somma dei tempi di posizionamento, latenza e trasferimento. Il tempo di trasferimento è proporzionale alla quantità trasferita, ossia alla dimensione delle pagine; tutto ciò farebbe supporre che siano preferibili pagine piccole. Si ricordi però, dal Capitolo 2, che il tempo di trasferimento è piccolo se è confrontato con il tempo di latenza e il tempo di posizionamento. A una velocità di trasferimento di 2 MB al secondo, per trasferire 512 byte s'impiegano 0,2 millisecondi. D'altra parte, il tempo di latenza è di circa 8 millisecondi e quello di posizionamento 20 millisecondi. Perciò, del tempo totale di I/O (28,2 millisecondi), l'1 per cento è attribuibile al trasferimento effettivo. Raddop-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

piando le dimensioni delle pagine, il tempo di I/O aumenta solo fino a 28,4 millisecondi. S'impiegano 28,4 millisecondi per leggere una sola pagina di 1024 byte, ma 56,4 millisecondi per leggere la stessa quantità di byte su due pagine di 512 byte l'una. Quindi, per ridurre il tempo di I/O occorre avere pagine di dimensioni maggiori.

Tuttavia, con pagine di piccole dimensioni si riduce l'I/O totale, poiché si migliorano le caratteristiche di località. Pagine di piccole dimensioni permettono di corrispondere con maggior precisione alla località del programma. Si consideri, ad esempio, un processo di 200 KB, dei quali solo la metà (100 KB) sono effettivamente usati durante l'esecuzione. Se si dispone di una sola ampia pagina, occorre inserire tutta la pagina, sicché vengono trasferiti e assegnati 200 KB. Disponendo di pagine di 1 byte, si possono invece inserire i soli 100 KB effettivamente usati, con trasferimento e assegnazione di quei soli 100 KB. Con pagine di piccole dimensioni è possibile avere una migliore risoluzione, che permette di isolare solo la memoria effettivamente necessaria. Se le dimensioni delle pagine sono maggiori, occorre assegnare e trasferire non solo quanto è necessario, ma tutto quel che è presente nella pagina, a prescindere dal fatto che sia o meno necessario. Quindi dimensioni più piccole danno come risultato una minore attività di I/O e una minore memoria totale assegnata.

D'altra parte occorre notare che con pagine di 1 byte si verifica un'assenza di pagina per *ciascun* byte. Un processo di 200 KB che usa solo metà di tale memoria accusa una sola assenza di pagina con una pagina di 200 KB, ma 102.400 assenze di pagine con le pagine di 1 byte. Ciascuna assenza di pagina causa un rilevante sovraccarico necessario a elaborare il segnale di eccezione, salvare i registri, sostituire una pagina, mettere in coda nell'attesa del dispositivo di paginazione e aggiornare le tabelle. Per ridurre il numero delle assenze di pagine sono necessarie pagine di grandi dimensioni.

Occorre considerare altri fattori, come la relazione tra la dimensione delle pagine e la dimensione dei settori del mezzo di paginazione. Non esiste una risposta ottimale al problema considerato. Alcuni fattori (frammentazione interna, località) sono a favore delle piccole dimensioni, mentre altri (dimensione delle tabelle, tempo di I/O) sono a favore delle grandi dimensioni. La tendenza è storicamente verso l'incremento delle dimensioni delle pagine. Nella prima edizione di questo testo (1983) si considerava un valore di 4096 byte come limite superiore alla dimensione delle pagine. Nel 1990 tale dimensione delle pagine era la più comune. I sistemi moderni possono impiegare pagine di dimensioni assai maggiori.

10.8.3 Portata del TLB

Il tasso di successi (*hit ratio*) di un TLB — trattato anche nel Paragrafo 9.4.2 — si riferisce alla percentuale di traduzioni di indirizzi virtuali risolte dal TLB anziché dalla tabella delle pagine. Il tasso di successi è evidentemente proporzionale al numero di elementi del TLB. Tuttavia, la memoria associativa che si usa per costruire i TLB è costosa e consuma molta energia.

Un parametro simile, detto portata del TLB (*TLB reach*), esprime la quantità di memoria accessibile dal TLB, ed è dato semplicemente dal numero di elementi (quindi è correlato al tasso di successi) moltiplicato per la dimensione delle pagine. Idealmente, il

TLB dovrebbe contenere l'insieme di lavoro di un processo; altrimenti, la necessità di ricorrere alla tabella delle pagine per la traduzione dei riferimenti alla memoria farà sì che il processo impieghi in quest'operazione assai più tempo di quello richiesto dal solo TLB. Se si raddoppia il numero di elementi del TLB, si raddoppia la sua portata; per alcune applicazioni che comportano un uso intensivo della memoria ciò potrebbe rivelarsi ancora insufficiente per la memorizzazione dell'insieme di lavoro.

Un altro metodo per aumentare la portata del TLB consiste nell'aumentare la dimensione delle pagine oppure nell'impiegare diverse dimensioni delle pagine. Se si aumenta la dimensione delle pagine, per esempio da 8 KB a 32 KB, si quadruplica la portata del TLB. Quest'aumento potrebbe però condurre a una maggiore frammentazione della memoria relativamente alle applicazioni che non richiedono pagine così grandi. L'UltraSPARC II è un esempio di architettura che consente diverse dimensioni delle pagine: 8 KB, 64 KB, 512 KB e 4 MB. Di queste possibili dimensioni delle pagine, il sistema operativo Solaris 2 impiega sia quella di 8 KB sia quella di 4 MB. Con un TLB a 64 elementi, la portata del TLB per il Solaris 2 varia da 512 KB, con tutte le pagine di 8 KB, a 256 MB, con tutte le pagine di 4 MB. Per la maggior parte delle applicazioni una dimensione delle pagina di 8 KB è sufficiente, sebbene il Solaris 2 associ i primi 4 MB del codice e dei dati del nucleo a due pagine di 4 MB. Il sistema operativo Solaris 2 permette anche alle applicazioni, come ad esempio i sistemi di gestione delle basi di dati, di trarre vantaggio dalle grandi pagine di 4 MB.

L'uso di diverse dimensioni delle pagine richiede però che la gestione del TLB sia svolta dal sistema operativo e non direttamente dall'architettura. Ad esempio, uno dei campi degli elementi del TLB deve indicare la dimensione della pagina fisica cui il contenuto di ciascun elemento fa riferimento. La gestione del TLB svolta dal sistema operativo e non esclusivamente dall'architettura comporta una penalizzazione delle prestazioni. Tuttavia, i vantaggi dovuti all'aumento del tasso di successi e della portata del TLB superano gli svantaggi che derivano dalla riduzione della rapidità di traduzione degli indirizzi. I recenti sviluppi indicano infatti un'evoluzione verso TLB gestiti dal sistema operativo e verso l'uso di pagine di diverse dimensioni. Le architetture UltraSPARC, MIPS, e Alpha adottano TLB che si gestiscono tramite il sistema operativo, mentre le architetture PowerPC e Pentium gestiscono i TLB direttamente, senza l'intervento del sistema operativo.

10.8.4 Tabella delle pagine invertita

Nel Paragrafo 9.4.4.3 si è introdotto il concetto di tabella delle pagine invertita come sistema di gestione delle pagine che consente di ridurre la quantità di memoria fisica necessaria per tenere traccia della corrispondenza tra gli indirizzi virtuali e gli indirizzi fisici. Tale riduzione si ottiene tramite una tabella con un elemento per ciascun blocco di memoria, indicizzato dalla coppia <*id-processo*, *numero di pagina*>.

Poiché contiene informazioni su quale pagina di memoria virtuale è memorizzata in ciascun blocco di memoria, la tabella delle pagine invertita riduce la quantità di memoria fisica necessaria a memorizzare tali informazioni. Tuttavia essa non contiene le informazioni complete sullo spazio di indirizzi logici di un processo che sono richieste se una pagina alla quale si è fatto riferimento non è correntemente presente nella memoria;

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

la paginazione su richiesta richiede tali informazioni per elaborare le eccezioni di pagine mancanti. Per disporre di tali informazioni è necessaria una tabella delle pagine esterna per ciascun processo. Queste tabelle sono simili alle ordinarie tabelle delle pagine di ciascun processo e contengono le informazioni relative alla locazione di ciascuna pagina virtuale.

Contrariamente a quel che potrebbe apparire, l'uso delle tabelle esterne delle pagine non è in contrasto con l'utilità della tabella delle pagine invertita, infatti a esse si fa riferimento solo alla presenza di un'assenza di pagina; quindi non è necessario che siano immediatamente disponibili ed esse stesse sono paginate dentro e fuori dalla memoria come è necessario. Sfortunatamente, in questo modo si può verificare un'assenza di qualche pagina dello stesso gestore della memoria virtuale, in tal caso si verifica un'altra assenza di pagina quando esso carica nella memoria la tabella esterna delle pagine per individuare la pagina virtuale nella memoria ausiliaria (*backing store*). Questo caso particolare richiede un'accurata gestione da parte del nucleo del sistema operativo e un ritardo nell'elaborazione della ricerca della pagina.

10.8.5 Struttura dei programmi

La paginazione su richiesta deve essere trasparente per il programma utente: spesso, l'utente non è neanche a conoscenza della natura paginata della memoria. In altri casi, però, le prestazioni del sistema si possono addirittura migliorare se l'utente (o il compilatore) è consapevole della sottostante presenza della paginazione su richiesta.

Come esempio limite, ma esplicativo, si considerino pagine di 128 parole. Si consideri il seguente frammento di programma scritto in Java la cui funzione è inizializzare a 0 ciascun elemento di una matrice di 128 × 128 elementi. È tipico il seguente codice:

```
int A[][] = new int[128][128];

for (int j = 0; j < A.lenghth; j++)
    for (int i = 0; i < A.lenghth; i++)
        A[i][j] = 0;
```

Occorre notare che la matrice è memorizzata per riga, vale a dire che è disposta nella memoria secondo l'ordine `A[0][0]`, `A[0][1]`, ..., `A[0][127]`, `A[1][0]`, `A[1][1]`, ..., `A[127][127]`. In pagine di 128 parole, ogni riga occupa una pagina, quindi il frammento di codice precedente azzera una parola per pagina, poi un'altra parola per pagina, e così via. Se il sistema operativo assegna meno di 128 blocchi di memoria a tutto il programma, allora la sua esecuzione causa 128 × 128 = 16.384 assenze di pagine. D'altra parte, cambiando il codice in

```
int A[][] = new int[128][128];

for (int i = 0; i < A.lenghth; i++)
    for (int j = 0; j < A.lenghth; j++)
        A[i][j] = 0;
```


si azzerano tutte le parole di una pagina prima che si inizi la pagina successiva, riducendo a 128 il numero di assenze di pagine.

Un'attenta scelta delle strutture di dati e delle strutture di programmazione può aumentare la località e quindi ridurre la frequenza delle assenze di pagine e il numero di pagine dell'insieme di lavoro. Una buona località è quella di una pila, poiché l'accesso avviene sempre alla sua parte superiore. Una tabella hash, invece, è progettata proprio per diffondere i riferimenti, causando una località non buona. Naturalmente, la località dei riferimenti rappresenta soltanto una misura dell'efficienza d'uso di una struttura di dati. Altri fattori rilevanti sono la rapidità di ricerca, il numero totale dei riferimenti alla memoria e il numero totale delle pagine coinvolte.

In uno stadio successivo, anche il compilatore e il caricatore possono avere un effetto notevole sulla paginazione. La separazione di codice e dati e la generazione di un codice rientrante significano che le pagine di codice si possono soltanto leggere e quindi non vengono modificate. Nel sostituire le pagine che non sono state modificate, non occorre scriverle nella memoria ausiliaria. Il caricatore può evitare di collocare procedure lungo i limiti delle pagine, sistemando ogni procedura completamente all'interno di una pagina. Le procedure che si invocano a vicenda più volte si possono 'impaccare' nella stessa pagina. Questa forma di impaccamento è una variante del problema del *bin-packing* della ricerca operativa: cercare di impaccare i segmenti di dimensione variabile in pagine di dimensione fissa, in modo da ridurre al minimo i riferimenti tra pagine diverse. Un metodo di questo tipo è utile soprattutto per pagine di grandi dimensioni.

Anche la scelta del linguaggio di programmazione può influire sulla paginazione. Con il Linguaggio C e il C++, ad esempio, si fa spesso uso dei puntatori, che tendono a distribuire in modo casuale gli accessi alla memoria, diminuendo così la località di un processo. Alcuni studi hanno mostrato che anche i programmi scritti in linguaggi orientati agli oggetti tendono ad avere una scarsa località dei riferimenti. D'altra parte, poiché il Java non fornisce in modo esplicito i puntatori, i programmi scritti in Java hanno, nei sistemi con memoria virtuale, una località dei riferimenti migliore di quella dei programmi scritti in Linguaggio C o in C++.

10.8.6 Vincolo di I/O

Quando si usa la paginazione su richiesta, talvolta occorre permettere che alcune pagine si possano vincolare alla memoria. Una situazione di questo tipo si presenta quando l'I/O si esegue verso o dalla memoria d'utente (virtuale). Spesso il sistema di I/O comprende una specifica unità d'elaborazione; al controllore di un'unità a nastro, ad esempio, generalmente si indica il numero di byte da trasferire e un indirizzo di memoria per la lettura o la scrittura di tali byte; completato il trasferimento, la CPU riceve un segnale d'interruzione (si vedano a questo proposito la Figura 10.20 e la discussione sull'accesso diretto alla memoria — Paragrafi 2.2.2. e 13.2.3).

Occorre essere certi che non si verifichi la seguente successione di eventi: un processo emette una richiesta di I/O ed è messo in coda per il relativo dispositivo. Nel frattempo si assegna la CPU ad altri processi che accusano assenze di pagine, e, usando un algoritmo di sostituzione globale per uno di questi, si sostituisce la pagina contenente

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

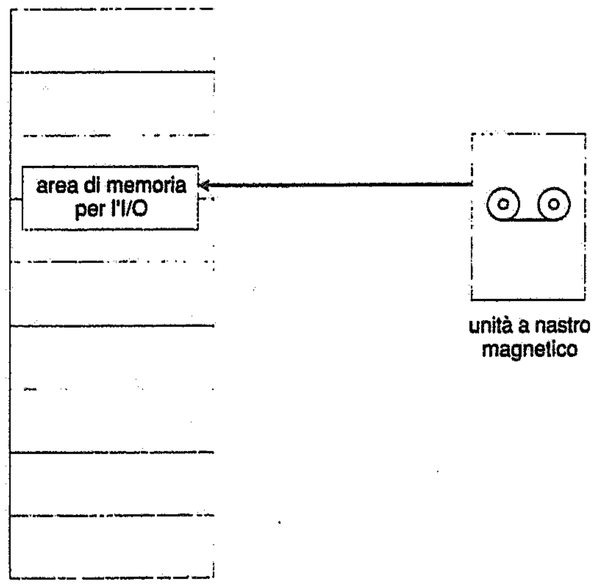


Figura 10.20 Ragione per i cui i blocchi di memoria usati per l'I/O devono essere presenti nella memoria.

l'indirizzo di memoria per l'operazione di I/O attesa dal primo processo; la pagina viene scaricata dalla memoria. Qualche tempo dopo, quando la richiesta di I/O raggiunge la prima posizione della coda d'attesa per il dispositivo, l'operazione di I/O avviene all'indirizzo specificato, ma questo blocco di memoria è ora impiegato per una pagina appartenente a un altro processo. Questo problema si può risolvere in due modi. Una soluzione prevede di non eseguire operazioni di I/O nella memoria d'utente, ma di copiare i dati sempre tra la memoria di sistema e la memoria d'utente. In questo modo l'I/O avviene solo tra la memoria di sistema e il dispositivo di I/O. Per scrivere dati in un nastro, occorre prima copiarli nella memoria di sistema, quindi trasferirli all'unità a nastro.

Quest'ulteriore copiatura può causare un sovraccarico inaccettabile. Un'altra soluzione consiste nel permettere che le pagine siano vincolate alla memoria. A ogni blocco di memoria si associa un bit di vincolo; se tale bit è attivato, la pagina contenuta in tale blocco di memoria non può essere selezionata per la sostituzione. Seguendo questo metodo, per scrivere dati in un nastro occorre vincolare alla memoria le pagine contenenti tali dati, quindi il sistema può continuare come di consueto. Le pagine vincolate non si possono sostituire. Completato l'I/O, si rimuove il vincolo.

Spesso il nucleo del sistema operativo, o una sua parte, è vincolato alla memoria. La maggior parte dei sistemi non può tollerare l'assenza di una pagina relativa al nucleo: si considerino le conseguenze di una procedura di sostituzione delle pagine che accusa un'assenza di pagina.

Un altro uso del bit di vincolo riguarda la normale sostituzione di pagine. Si consideri la seguente successione d'eventi: un processo a bassa priorità subisce un'assenza di pagina. Selezionando un blocco di memoria per la sostituzione, il sistema di paginazione carica nella memoria la pagina necessaria. Pronto per continuare, il processo con priorità bassa entra nella coda dei processi pronti per l'esecuzione e attende l'assegnazione della CPU. Giacché si tratta di un processo con bassa priorità, può non essere selezionato dallo scheduler della CPU per un certo tempo. Mentre il processo con priorità bassa attende, un processo ad alta priorità accusa un'assenza di pagina. Durante la ricerca per la sostituzione, il sistema di paginazione individua una pagina nella memoria alla quale non sono stati fatti riferimenti o modifiche; si tratta della pagina che il processo con bassa priorità ha appena caricato. Questa pagina sembra una sostituzione perfetta: non è stata modificata, non è necessario scriverla nella memoria secondaria e, apparentemente, non è stata usata da molto tempo.

Stabilire se la pagina del processo con bassa priorità si debba sostituire a vantaggio del processo con alta priorità è una questione di scelta di un criterio. Semplicemente, si ritarda il processo con bassa priorità a vantaggio di quello con alta priorità. D'altra parte, però, si spreca il lavoro fatto per trasferire nella memoria la pagina del processo con bassa priorità. Per evitare che una pagina appena caricata sia sostituita prima che sia usata almeno una volta si può usare il bit di vincolo. Se una pagina è selezionata per la sostituzione, si attiva il suo bit di vincolo, tale bit rimane attivato fino a che si esegue nuovamente il processo che ha accusato l'assenza di pagina.

Comunque, l'uso dei bit di vincolo può essere pericoloso: se un bit non viene mai disattivato, ad esempio a causa di un baco del sistema operativo, il blocco di memoria relativo alla pagina vincolata diventa inutilizzabile. Il sistema operativo Macintosh fornisce un meccanismo di vincolo delle pagine. Essendo un sistema per utente singolo, l'abuso di tale meccanismo può causare danni soltanto allo stesso utente. Ciò non si può consentire nei sistemi multiutente. Il sistema operativo Solaris 2, ad esempio, consente l'impiego di 'suggerimenti' di vincolo delle pagine, che si possono però trascurare se l'insieme delle pagine libere diviene troppo piccolo o se un singolo processo richiede che troppe pagine siano vincolate alla memoria.

10.8.7 Elaborazione in tempo reale

La discussione proposta in questo capitolo si è focalizzata su come sia possibile ottenere il miglior utilizzo complessivo di un sistema di calcolo tramite l'ottimizzazione dell'uso della memoria. Sfruttando la memoria per i dati attivi e trasferendo i dati inattivi nei dischi si finisce per incrementare la produttività complessiva del sistema. D'altra parte, i singoli processi possono soffrire di questa gestione, poiché durante la loro esecuzione possono essere soggetti a un numero maggiore di assenze di pagine.

Basti considerare un processo in tempo reale; un processo di questo tipo dovrebbe ottenere il controllo della CPU e quindi completare la propria elaborazione con un ritardo minimo. La memoria virtuale rappresenta l'antitesi dell'elaborazione in tempo reale,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

poiché durante il trasferimento delle pagine nella memoria può introdurre ritardi inattesi e a lungo termine nell'elaborazione di un processo. Per questa ragione i sistemi in tempo reale non sono quasi mai dotati della memoria virtuale.

Nel caso del Solaris 2, le intenzioni dei progettisti della Sun Microsystems sono state quelle di consentire, nello stesso sistema, elaborazioni sia a partizione del tempo sia in tempo reale. Per risolvere il problema delle assenze di pagine, Solaris 2 consente a un processo di notificare al sistema quali pagine sono importanti per il processo stesso. Oltre ad ammettere i 'suggerimenti' sull'uso delle pagine, il sistema operativo consente a utenti privilegiati di richiedere pagine che saranno vincolate alla memoria. L'abuso di questo meccanismo potrebbe relegare i restanti processi fuori del sistema. È necessario consentire ai processi in tempo reale di disporre di una limitata e bassa latenza di dispatch.

10.9 Sommario

Può essere necessario riuscire a eseguire un processo il cui spazio d'indirizzi logici sia maggiore dello spazio d'indirizzi fisici disponibile. Il programmatore può rendere eseguibile un processo di tal fatta strutturando il programma in sezioni sovrapponibili (*overlay*), ma generalmente si tratta di un compito di programmazione piuttosto difficile. La memoria virtuale è una tecnica che consente di mettere in corrispondenza uno spazio d'indirizzi logici molto ampio con una memoria fisica più piccola. Permette di eseguire processi molto grandi e di aumentare il grado di multiprogrammazione, incrementando l'utilizzo della CPU. Inoltre, grazie a tale tecnica, i programmatori di applicazioni non devono più preoccuparsi della disponibilità di memoria.

La paginazione su richiesta pura non carica mai una pagina fino a che non le si sia fatto riferimento. Il primo riferimento causa l'invio di un segnale d'eccezione di pagina mancante al nucleo residente del sistema operativo, quest'ultimo consulta una tabella interna per stabilire la locazione della pagina nella memoria ausiliaria, quindi individua un blocco di memoria libero e vi trasferisce la pagina prelevandola dalla memoria ausiliaria. La tabella delle pagine viene aggiornata per riflettere tale modifica e si riavvia l'istruzione che aveva causato l'eccezione di pagina mancante. Questo metodo permette l'esecuzione di un processo anche se nella memoria centrale non è interamente presente la sua immagine di memoria. Finché la frequenza delle assenze di pagine rimane ragionevolmente bassa, le prestazioni si considerano accettabili.

La paginazione su richiesta si può usare per ridurre il numero di blocchi di memoria assegnati a un processo. Questo metodo può aumentare il grado di multiprogrammazione, permettendo che più processi siano disponibili per l'esecuzione in un dato momento e, almeno in teoria, può aumentare il grado di utilizzo della CPU. Inoltre, consente l'esecuzione di processi i cui requisiti di spazio di memoria superano la memoria fisica disponibile. Tali processi si eseguono nella memoria virtuale.

Se i requisiti di spazio di memoria superano la memoria fisica, può essere necessaria la sostituzione di pagine presenti nella memoria allo scopo di liberare blocchi di memoria per nuove pagine. Gli algoritmi usati per la sostituzione delle pagine sono diversi: la sostituzione di tipo FIFO è facile da programmare, ma soffre dell'anomalia di Belady; la sostituzione ottimale delle pagine richiede la conoscenza dei futuri riferimenti alla memoria; la sostituzione delle pagine LRU è quasi ottimale, ma può essere di difficile realizzazione. Quasi tutti gli algoritmi di sostituzione delle pagine, come l'algoritmo con seconda chance, sono approssimazioni della sostituzione LRU.

Oltre un algoritmo di sostituzione delle pagine, occorre un criterio di assegnazione dei blocchi di memoria. L'assegnazione può essere statica, indicando una sostituzione di pagine locale, oppure dinamica, con una sostituzione di pagine globale. Il modello dell'insieme di lavoro presuppone che i processi siano eseguiti in località. L'insieme di lavoro è l'insieme delle pagine nella località corrente. Di conseguenza, a ogni processo si possono assegnare blocchi di memoria sufficienti al suo corrente insieme di lavoro. Se un processo non ha spazio di memoria sufficiente per il proprio insieme di lavoro, si ha una degenerazione dell'attività di paginazione (*thrashing*). Se a ogni processo si devono fornire blocchi di memoria sufficienti per evitare tale degenerazione, sono necessarie le attività d'avvicendamento (*swapping*) e scheduling dei processi.

Oltre a risolvere i problemi più importanti connessi alla sostituzione delle pagine e all'assegnazione dei blocchi di memoria, nella progettazione di un sistema di paginazione è necessario definire anche la dimensione delle pagine, gestire le operazioni di I/O, di vincolo delle pagine alla memoria, la prepaginazione, la creazione dei processi, la struttura dei programmi, la paginazione degenere e altri elementi ancora. La memoria virtuale si può pensare come uno dei livelli della gerarchia di memorie di un calcolatore; ogni livello ha il proprio tempo d'accesso, la propria dimensione e i propri parametri di costo.

10.10 Esercizi

10.1 Elencate le situazioni nelle quali si verificano le assenze di pagine. Descrivete le azioni intraprese dal sistema operativo quando si presenta un'eccezione di pagina mancante.

10.2 Supponete di avere una successione dei riferimenti alle pagine per un processo con m blocchi di memoria (inizialmente vuoti). La successione dei riferimenti ha lunghezza p con n numeri di pagine distinti. Per ogni algoritmo di sostituzione delle pagine, indicate quel che segue:

a) un limite inferiore per il numero di assenze di pagine;
b) un limite superiore per il numero di assenze di pagine.

http://www.adultpdf.com
Created by TIFF To PDF trial version, please visit www.tiffpdf.com to remove this watermark, please register this software.

10.3 Un certo calcolatore offre ai suoi utenti uno spazio di memoria virtuale di 2^{32} byte. Il calcolatore ha 2^{18} byte di memoria fisica. La memoria virtuale è realizzata attraverso la paginazione, e la dimensione delle pagine è di 4096 byte. Un processo utente genera l'indirizzo virtuale 11123456. Spiegate come il sistema stabilisce la locazione fisica corrispondente, distinguendo tra le operazioni svolte dal sistema operativo e le operazioni svolte dall'architettura del calcolatore.

10.4 Dite quali tra le seguenti tecniche e strutture di programmazione favoriscono un ambiente adatto alla paginazione su richiesta e quali no:

a) pila;
b) tabella dei simboli di tipo hash;
c) ricerca sequenziale;
d) ricerca binaria;
e) codice puro;
f) operazioni su vettori;
g) riferimenti indiretti.

Spiegate la vostra risposta.

10.5 Supponete di avere una memoria con paginazione su richiesta. La tabella delle pagine è contenuta in registri. Per servire un'eccezione di pagina mancante occorrono 8 millisecondi, se è disponibile una pagina vuota oppure la pagina sostituita non è modificata; occorrono 20 millisecondi, se la pagina sostituita è modificata. Il tempo d'accesso alla memoria è di 100 nanosecondi.

Supponete che nel 70 per cento dei casi la pagina da sostituire sia modificata. Dite qual è la massima frequenza accettabile delle assenze di pagine per un tempo d'accesso effettivo non superiore a 200 nanosecondi.

10.6 Ordinate i seguenti algoritmi di sostituzione delle pagine dal peggiore al migliore rispetto alla frequenza delle assenze di pagine:

a) sostituzione LRU;
b) sostituzione FIFO;
c) sostituzione ottimale;
d) sostituzione con seconda chance.

Distinguete gli algoritmi che soffrono dell'anomalia di Belady da quelli che non ne soffrono.

10.7 La realizzazione della memoria virtuale in un sistema di calcolo implica costi e vantaggi. Elencateli. È possibile che i costi superino i vantaggi. Indicate le misure che si possono intraprendere per evitare che ciò accada.

10.8 Un sistema operativo dispone di una memoria virtuale paginata, usando una CPU con un tempo di ciclo di un microsecondo; il costo d'accesso a una pagina diversa da quella corrente è di un altro microsecondo. Le pagine hanno 1000 parole e il dispositivo di paginazione è un disco che ruota a 3000 giri al minuto, trasferendo un milione di parole al secondo. Dal sistema sono state ottenute le seguenti misure statistiche:

- l'1 per cento di tutte le istruzioni eseguite ha avuto accesso a una pagina diversa da quella corrente;
- l'80 per cento delle istruzioni che hanno avuto accesso a un'altra pagina lo hanno fatto su una pagina già presente nella memoria;
- quando si richiedeva una pagina nuova, la pagina sostituita era modificata nel 50 per cento dei casi.

Calcolate il tempo effettivo di istruzione per questo sistema, supponendo che il sistema esegua un solo processo e che la CPU sia inattiva durante le operazioni di trasferimento da parte del disco.

10.9 Considerate un sistema con paginazione su richiesta con il seguente utilizzo temporale:

utilizzo della CPU	20 per cento;
disco di paginazione	97,7 per cento;
altri dispositivi di I/O	5 per cento.

Indicate, fra le seguenti operazioni, quelle che consentono (o è probabile che consentano) di migliorare l'utilizzo della CPU:

a) installazione di una CPU più veloce;
b) installazione di un disco di paginazione più grande;
c) aumento del grado di multiprogrammazione;
d) riduzione del grado di multiprogrammazione;
e) installazione di una maggiore quantità di memoria centrale;
f) installazione di un disco più veloce o di più controllori di unità con dischi multipli;
g) aggiunta della prepaginazione agli algoritmi di prelievo delle pagine;
h) aumento della dimensione delle pagine.

Spiegate le risposte.

10.10 Considerate la matrice `A`:

```
int A[][] = new int[100][100];
```

dove `A[0][0]` si trova alla locazione 200, in un sistema con memoria paginata e dimensione delle pagine 200. La manipolazione della matrice `A` è effettuata da un piccolo processo caricato nella pagina 0, alle locazioni da 0 a 199; quindi ogni prelievo di un'istruzione avverrà dalla pagina 0.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Dati tre blocchi di memoria, dite quante eccezioni di pagine mancanti sono causate dai seguenti cicli di inizializzazione della matrice, usando la sostituzione LRU, e supponendo che il blocco di memoria 1 contenga il processo e gli altri due siano inizialmente vuoti:

a)
```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
      A[i][j] = 0;
```

b)
```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
      A[i][j] = 0;
```

10.11 Considerate la seguente successione di riferimenti alle pagine:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Dite quante assenze di pagine avrebbero luogo, usando uno, due, tre, quattro, cinque, sei o sette blocchi di memoria, con i seguenti algoritmi di sostituzione:

- sostituzione LRU;
- sostituzione FIFO;
- sostituzione ottimale.

Ricordate che tutti i blocchi di memoria sono inizialmente vuoti, quindi per ciascuna delle prime singole pagine si ha un'assenza di pagina.

10.12 Supponete di usare un algoritmo di paginazione che richiede un bit di riferimento, ad esempio una sostituzione con seconda chance oppure un modello di insieme di lavoro, ma che l'architettura del sistema non disponga di tali bit. Schematizzate come questi si possono simulare e calcolatene il costo.

10.13 Avete escogitato un nuovo algoritmo di sostituzione delle pagine che potrebbe essere ottimale. In alcuni casi astrusi si verifica l'anomalia di Belady. Stabilite se il nuovo algoritmo è ottimale e spiegate la risposta.

10.14 Supponete che il criterio di sostituzione, in un sistema paginato, consista in un esame regolare di ogni pagina e nella rimozione della pagina che non è stata usata dall'esame precedente. Indicate i guadagni e le perdite ottenuti seguendo tale criterio anziché l'algoritmo di sostituzione LRU o con seconda chance.

10.15 La segmentazione è simile alla paginazione, ma usa 'pagine' di dimensione variabile dette segmenti. Definite due algoritmi di sostituzione dei segmenti basati sugli schemi di sostituzione delle pagine FIFO e LRU. Ricordate che, poiché i segmenti non hanno la stessa dimensione, il segmento da sostituire può non essere sufficientemente grande da lasciare abbastanza locazioni per il segmento richiesto. Descrivete le strategie per i sistemi in cui non è possibile rilocare i segmenti, e le strategie per i sistemi in cui è possibile.

10.16 Un algoritmo di sostituzione delle pagine dovrebbe ridurre al minimo il numero delle assenze di pagine. Questa minimizzazione si può ottenere distribuendo in modo uniforme su tutta la memoria le pagine maggiormente usate, anziché lasciarle competere per un piccolo numero di blocchi di memoria. A ogni blocco di memoria si può associare un contatore del numero delle pagine relative a quel blocco di memoria. Quindi, per sostituire una pagina, si cerca il blocco di memoria con il contatore più basso.

1. Definite un algoritmo di sostituzione delle pagine che si avvalga di questa idea di base. Affrontate in modo specifico i seguenti problemi:
   a) qual è il valore iniziale dei contatori;
   b) quando si incrementano i contatori;
   c) quando si decrementano i contatori;
   d) come si sceglie la pagina da sostituire.
2. Se sono disponibili quattro blocchi di memoria, dite quante assenze di pagine avvengono per l'algoritmo indicato, in relazione alla seguente successione di riferimenti:

   1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.
3. Calcolate il numero minimo di assenze di pagine per una strategia di sostituzione delle pagine ottimale per la successione di riferimenti del punto b), con quattro blocchi di memoria.

10.17 Considerate un sistema di paginazione su richiesta con un disco di paginazione che abbia un tempo medio d'accesso e di trasferimento di 20 millisecondi. Gli indirizzi sono tradotti per mezzo di una tabella delle pagine che si trova nella memoria centrale, con un tempo d'accesso di un microsecondo per ogni accesso alla memoria. Quindi, ogni riferimento alla memoria per mezzo della tabella delle pagine richiede due accessi. Per migliorare questo tempo, è stata aggiunta una memoria associativa che riduce il tempo d'accesso a un riferimento alla memoria, se l'elemento della tabella delle pagine si trova nella memoria associativa.

Supponete che, per l'80 per cento degli accessi, l'elemento relativo si trovi nella memoria associativa e che il 10 per cento dei restanti (cioè il 2 per cento del totale) causi un'assenza di pagina. Calcolate il tempo effettivo d'accesso alla memoria.

10.18 Considerate un calcolatore con paginazione su richiesta nel quale il grado di multiprogrammazione sia attualmente fissato a quattro. Sul sistema sono state recentemente fatte misure per stabilire l'utilizzo della CPU e del disco di paginazione. Il risultato può essere uno dei tre seguenti livelli di utilizzo:

a) 13 per cento della CPU, 97 per cento del disco;
b) 87 per cento della CPU, 3 per cento del disco;
c) 13 per cento della CPU, 3 per cento del disco.

Dite cosa accade in ciascuno di questi casi, e se si può aumentare il grado di multiprogrammazione per incrementare l'utilizzo della CPU. Stabilite anche se la paginazione può essere d'aiuto.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

10.19 Si dispone di un sistema operativo per un'architettura che impiega registri di base e di limite, ma l'architettura è stata modificata per offrire una tabella delle pagine. Dite se, e in che modo, le tabelle delle pagine si possono impostare per simulare i registri di base e di limite, motivando la risposta.

10.20 Descrivete la causa dell'attività di paginazione degenere e i metodi adottati dal sistema per individuare tale degenerazione e, successivamente, eliminarla.

10.21 Scrivete un programma che codifichi gli algoritmi di sostituzione delle pagine FIFO e LRU descritti in questo capitolo. Generate una successione di riferimenti casuale, nella quale i numeri delle pagine siano compresi tra 0 e 9. Applicate ciascun algoritmo a tale successione e registrate i rispettivi numeri delle assenze di pagine. Codificate gli algoritmi di sostituzione delle pagine in modo che il numero di blocchi di memoria sia compreso tra 1 e 7. Assumete l'impiego della paginazione su richiesta.

10.11 Note bibliografiche

La paginazione su richiesta è stata usata per la prima volta nel sistema operativo Atlas, realizzato per il calcolatore MUSE della Manchester University intorno al 1960 [Kilburn et al. 1961]. Un altro tra i primi sistemi di paginazione su richiesta è stato il MULTICS, per il calcolatore GE 645 [Organick 1972].

[Belady et al. 1969] sono stati i primi ricercatori a osservare che la strategia di sostituzione FIFO poteva presentare l'anomalia che porta il nome di Belady. In [Mattson et al. 1970] si dimostra che gli algoritmi a pila non sono soggetti all'anomalia di Belady. L'algoritmo di sostituzione ottimale è dovuto a [Belady 1966]. In [Mattson et al. 1970] si trova la dimostrazione che esso è ottimale. L'algoritmo ottimale di Belady si usa per l'assegnazione statica; [Prieve e Fabry 1976] hanno proposto un algoritmo ottimale per situazioni in cui l'assegnazione può variare.

L'algoritmo con seconda chanche migliorato è discusso in [Carr e Hennessy 1981]; è usato nel sistema di memoria virtuale del sistema operativo Macintosh ed è descritto in [Goldman 1989].

Il modello dell'insieme di lavoro è stato sviluppato da [Denning 1968]. Discussioni relative al modello dell'insieme di lavoro sono presenti in [Denning 1980].

Lo schema di controllo della frequenza di assenze di pagine è stato sviluppato da [Wulf 1969], il quale ha applicato con successo la propria tecnica al calcolatore Burroughs B5500. [Gupta e Franklin 1978] fornisce un confronto delle prestazioni tra lo schema dell'insieme di lavoro e lo schema di sostituzione basato sulla frequenza delle assenze di pagine.

[Solomon 1998] descrive come la memoria virtuale è stata realizzata per il sistema operativo Windows NT. [Mauro e McDougall 2001] discute la memoria virtuale del Solaris 2. Le tecniche di memoria virtuale nei sistemi operativi LINUX e BSD UNIX sono descritte, rispettivamente, in [Bovet e Cesati 2001] e [McKusick et. al. 1996]

I dettagli del sistema operativo OS/2 e la segmentazione su richiesta sono descritti in [Iacobucci 1988]. [Ganapathy e Schimmel 1998] tratta le caratteristiche dei sistemi per pagine di dimensioni diverse. In [Intel 1986] si trova una buona trattazione dell'architettura di paginazione dell'Intel 80386 e in [Motorola 1989b] di quella del Motorola 68030. La gestione della memoria virtuale nel sistema operativo VAX/VMS è discussa da [Levy e Lipman 1982]. [Hagmann 1989] presenta i sistemi operativi e la memoria virtuale delle stazioni di lavoro. Una comparazione dei sistemi di memoria virtuale tra le architetture MIPS, PowerPC e Pentium si trova in [Jacob e Mudge 1998b]. Un articolo correlato, [Jacob e Mudge 1998a], descrive gli elementi dell'architettura di sistema necessari alla realizzazione della memoria virtuale in sei diversi sistemi, tra i quali l'UltraSPARC.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 11

Interfaccia del file system

Per la maggior parte degli utenti il file system è l'aspetto più visibile di un sistema operativo. Esso fornisce il meccanismo per la registrazione e l'accesso in linea a dati e programmi appartenenti al sistema operativo e a tutti gli utenti del sistema di calcolo. Il file system consiste di due parti distinte: un insieme di *file*, ciascuno dei quali contenente dati correlati, e una *struttura di directory*, che organizza tutti i file nel sistema e fornisce informazioni su di essi. Alcuni file system hanno un terzo componente: le *partizioni*, usate per separare fisicamente o logicamente grandi gruppi di directory. In questo capitolo si considerano i vari aspetti dei file e i principali tipi di strutture di directory. Si discute inoltre della gestione della *protezione dei file*, necessaria in un ambiente in cui più utenti hanno accesso ai file, e dove si vuole controllare chi e in che modo vi ha accesso. Infine è trattata la semantica dei file condivisi da più processi, utenti e calcolatori.

11.1 Concetto di file

I calcolatori possono memorizzare le informazioni in diversi mezzi di memorizzazione, come dischi magnetici, nastri magnetici e dischi ottici. Per rendere agevole l'uso del calcolatore, il sistema operativo offre una visione logica uniforme della memorizzazione delle informazioni; astrae dalle caratteristiche fisiche dei propri dispositivi di memoria per definire un'unità di memoria logica, il *file*. Il sistema operativo associa i file ai dispositivi fisici di memorizzazione, di solito non volatili in modo che il loro contenuto non vada perduto a causa delle interruzioni dell'alimentazione elettrica e dei riavvii del sistema.

Un file è un insieme di informazioni, correlate e registrate nella memoria secondaria, cui è stato assegnato un nome. Dal punto di vista dell'utente, un file è la più piccola porzione di memoria secondaria logica; i dati si possono cioè scrivere nella memoria secondaria soltanto all'interno di un file. Di solito i file rappresentano programmi, in forma sorgente e oggetto, e dati. I file di dati possono essere numerici, alfabetici, alfanume-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this watermark, please register this software.

rici o binari; e non possedere un formato specifico, come i file di testo; oppure essere rigidamente formattati. In genere un file è formato da una sequenza di bit, byte, righe o *record* il cui significato è definito dal creatore e dall'utente del file stesso. Il concetto di file è quindi estremamente generale.

Le informazioni contenute in un file sono definite dal suo creatore e possono essere di molti tipi: programmi sorgente, programmi oggetto, dati numerici, testo, dati contabili, immagini, registrazioni sonore, e così via. Un file ha una **struttura** definita secondo il tipo: un file di *testo* è formato da una sequenza di caratteri organizzati in righe, e probabilmente pagine; un file *sorgente* è formato da una sequenza di procedure e funzioni, ciascuna delle quali è a sua volta organizzata in dichiarazioni seguite da istruzioni eseguibili; un file *oggetto* è formato da una sequenza di byte, organizzati in blocchi, comprensibile al modulo di collegamento del sistema; un file *eseguibile* consiste di una serie di sezioni di codice che il caricatore può caricare nella memoria ed eseguire.

11.1.1 Attributi dei file

Per comodità degli utenti umani, ogni file ha un nome che si usa come riferimento. Un nome, di solito, è una sequenza di caratteri come `esempio.c`. Alcuni sistemi, nella composizione dei nomi, distinguono le lettere maiuscole dalle minuscole, altri le considerano equivalenti. Una volta ricevuto il nome, il file diviene indipendente dal processo, dall'utente, e anche dal sistema da cui è stato creato. Ad esempio, un utente potrebbe creare il file `esempio.c` e un altro utente potrebbe modificarlo specificandone il nome. Il proprietario del file potrebbe registrare il file in un dischetto, inviarlo per posta elettronica, o copiarlo attraverso la rete, ed esso potrebbe ancora chiamarsi `esempio.c` nel sistema di destinazione.

Un file ha altri attributi che possono variare secondo il sistema operativo, ma che tipicamente comprendono i seguenti:

- **Nome.** Il nome simbolico del file è l'unica informazione in forma umanamente leggibile.
- **Identificatore.** Si tratta di un'etichetta unica, di solito un numero, che identifica il file all'interno del file system; è il nome impiegato dal sistema per il file.
- **Tipo.** Questa informazione è necessaria ai sistemi che gestiscono tipi di file diversi.
- **Locazione.** Si tratta di un puntatore al dispositivo e alla locazione del file in tale dispositivo.
- **Dimensione.** Si tratta della dimensione corrente del file (in byte, parole o blocchi) ed eventualmente della massima dimensione consentita.
- **Protezione.** Le informazioni di controllo degli accessi controllano chi può leggere, scrivere o far eseguire il file.
- **Ora, data e identificazione dell'utente.** Queste informazioni possono essere relative alla creazione, l'ultima modifica e l'ultimo uso. Questi dati possono essere utili ai fini della protezione e per controllarne l'uso.

Le informazioni sui file sono conservate nella struttura di directory, che risiede a sua volta nella memoria secondaria. Di solito un elemento di directory consiste di un nome di file e di un identificatore unico, che a sua volta individua gli altri attributi del file. Un elemento di directory può richiedere più di un kilobyte per contenere queste informazioni per ciascun file. In un sistema con molti file, la dimensione della stessa directory può essere dell'ordine dei megabyte. Poiché le directory, come i file, devono essere non volatili, si devono registrare nella memoria secondaria e caricare nella memoria centrale un po' per volta, secondo le necessità.

11.1.2 Operazioni sui file

Un file è un **tipo di dato astratto.** Per definire adeguatamente un file è necessario considerare le operazioni che si possono eseguire su di esso. Il sistema operativo può offrire chiamate del sistema per creare, scrivere, leggere, spostare, cancellare e troncare un file. Le successive considerazioni su ciò che deve fare un sistema operativo per ciascuna di queste sei operazioni di base dovrebbero rendere più semplice osservare come si possano realizzare altre operazioni simili, ad esempio la ridenominazione di un file.

- **Creazione di un file.** Per creare un file è necessario compiere due passaggi. In primo luogo si deve trovare lo spazio per il file nel file system; la discussione sui criteri di assegnazione dei file è rimandata al Capitolo 12. Secondariamente, per il file si deve creare un nuovo elemento nella directory in cui registrare il nome del file, la sua posizione nel file system ed eventualmente altre informazioni.
- **Scrittura di un file.** Per scrivere in un file è indispensabile una chiamata del sistema che specifichi il nome del file e le informazioni che si vogliono scrivere. Dato il nome del file, il sistema cerca la sua posizione nella directory. Il file system deve mantenere un puntatore di *scrittura* alla locazione nel file in cui deve avvenire la prossima operazione di scrittura. Il puntatore si deve aggiornare ogniqualvolta si esegue una scrittura.
- **Lettura di un file.** Per leggere da un file è necessaria una chiamata del sistema che specifichi il nome del file e la posizione nella memoria dove collocare il successivo blocco del file. Anche in questo caso si cerca l'elemento corrispondente nella directory e il sistema deve mantenere un puntatore di *lettura* alla locazione nel file in cui deve avvenire la successiva operazione di lettura. Una volta completata la lettura, si aggiorna il puntatore. Di solito un processo o legge o scrive in un file, e la posizione corrente è mantenuta come un **puntatore alla posizione corrente del file.** Sia le operazioni di lettura sia quelle di scrittura adoperano lo stesso puntatore, risparmiando spazio e riducendo la complessità del sistema.
- **Riposizionamento in un file.** Si ricerca l'elemento appropriato nella directory e si assegna un nuovo valore al puntatore alla posizione corrente nel file. Il riposizionamento non richiede alcuna operazione di I/O. Questa operazione è anche nota come *posizionamento* (*seek*) nel file.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- **Cancellazione di un file.** Per cancellare un file si cerca l'elemento della directory associato al file designato, si rilascia lo spazio associato al file (in modo che possa essere adoperato per altri file) e si elimina l'elemento della directory.

- **Troncamento di un file.** Si potrebbe voler cancellare il contenuto di un file ma mantenere i suoi attributi. Invece di forzare gli utenti a cancellare il file e quindi ricrearlo, questa funzione consente di mantenere immutati gli attributi (a esclusione della lunghezza del file) pur azzerando la lunghezza del file e rilasciando lo spazio occupato.

Queste sei operazioni di base comprendono sicuramente l'insieme minimo delle operazioni richieste per i file. Altre operazioni comuni comprendono l'*aggiunta* di nuove informazioni alla fine di un file esistente e la *ridenominazione* di un file esistente. Queste operazioni primitive si possono combinare per compiere altre operazioni. Ad esempio, per creare una *copia* di un file o per copiare il file in un altro dispositivo di I/O, come una stampante o un video, è sufficiente creare un nuovo file, leggere i dati dal file vecchio e scriverli nel nuovo. Sono inoltre necessarie operazioni che consentano a un utente di leggere e impostare i vari attributi di un file. Ad esempio, si potrebbe dover ricorrere a un'operazione che consenta all'utente di determinare lo stato di un file, come la lunghezza, e che consenta di definire gli attributi di un file, come il proprietario.

La maggior parte delle operazioni sopra citate richiede una ricerca dell'elemento associato al file specificato nella directory. Per evitare questa continua ricerca, molti sistemi richiedono l'impiego di una chiamata del sistema open la prima volta che si adopera un file in maniera attiva. Il sistema operativo mantiene una piccola tabella contenente informazioni riguardanti tutti i file aperti (detta, per l'appunto, **tabella dei file aperti**). Quando si richiede un'operazione su un file, questo viene individuato tramite un indice in tale tabella, in questo modo si evita qualsiasi ricerca. Quando il file non è più attivamente usato viene *chiuso* dal processo, e il sistema operativo rimuove l'elemento a esso associato dalla tabella dei file aperti.

Alcuni sistemi aprono implicitamente un file al primo riferimento e lo chiudono automaticamente quando il processo che lo ha aperto termina. A ogni modo, la maggior parte dei sistemi esige che il programmatore richieda l'apertura del file in modo esplicito per mezzo di una chiamata del sistema open prima che sia possibile adoperarlo. L'operazione open riceve il nome del file, lo cerca nella directory e copia l'elemento a esso associato nella tabella dei file aperti. La chiamata del sistema open può accettare anche informazioni sui modi d'accesso: creazione, sola lettura, lettura e scrittura, sola aggiunta, ecc. Si controllano i permessi relativi al file, e se il modo d'accesso richiesto è consentito, si apre il file. La chiamata del sistema open riporta di solito un puntatore all'elemento nella tabella dei file aperti; questo puntatore si adopera al posto dell'effettivo nome del file in tutte le operazioni di I/O, evitando così successive operazioni di ricerca e semplificando l'interfaccia delle chiamate del sistema.

La realizzazione delle operazioni open e close in un ambiente multiutente, come lo UNIX, è più complicata poiché più utenti possono aprire un file contemporaneamente; di solito il sistema operativo introduce due livelli di tabelle interne: una tabella per ciascun processo e una tabella di sistema. La tabella di un processo contiene i riferimenti a tutti i file aperti da quel processo; ad esempio, il puntatore alla posizione di ciascun file si trova in questa tabella e specifica la posizione nel file in cui opereranno le successive read o write. Si possono includere anche i diritti d'accesso al file e informazioni di contabilizzazione.

Ciascun elemento della tabella associata a ciascun processo punta a sua volta a una tabella di sistema dei file aperti, che contiene le informazioni indipendenti dai processi come la posizione dei file nei dischi, le date degli accessi e le dimensioni dei file. Quando un file è già stato aperto da un processo, una open eseguita da un altro processo comporta solamente l'aggiunta di un nuovo elemento nella tabella dei file aperti associata a quel processo, che punta al corrispondente elemento della tabella di sistema. Tipicamente, la tabella dei file aperti ha anche un *contatore delle aperture* associato a ciascun file, che indica il numero di processi che hanno aperto quel file. Ogni close decrementa questo *contatore*; quando raggiunge il valore zero il file non è più in uso e si elimina l'elemento corrispondente dalla tabella dei file aperti. Riassumendo, a ciascun file aperto sono associate diverse informazioni:

- **Puntatore al file.** Nei sistemi che non prevedono lo scostamento come parte delle chiamate del sistema read e write, il sistema deve tenere traccia dell'ultima posizione di lettura e scrittura sotto forma di un puntatore alla posizione corrente nel file. Questo puntatore è unico per ogni processo che opera sul file e quindi deve essere tenuto separato dagli attributi del file residenti nel disco.

- **Contatore dei file aperti.** A mano a mano che si chiudono i file, per evitare di esaurire lo spazio associato alla propria tabella dei file aperti, il sistema operativo deve riutilizzarne gli elementi. Poiché più processi possono aprire uno stesso file, prima di rimuovere l'elemento corrispondente, il sistema deve attendere l'ultima chiusura del file. Questo contatore tiene traccia del numero di open e close, e raggiunge il valore zero dopo l'ultima chiusura, momento in cui il sistema può rimuovere l'elemento della tabella.

- **Posizione nel disco del file.** La maggior parte delle operazioni richiede al sistema di modificare i dati contenuti nel file. L'informazione necessaria per localizzare il file nel disco è mantenuta nella memoria, per evitare di doverla prelevare dal disco a ogni operazione.

- **Diritti d'accesso.** Ciascun processo apre un file in uno dei modi d'accesso. Questa informazione è contenuta nella tabella del processo in modo che il sistema operativo possa permettere o negare le successive richieste di I/O.

Per condividere sezioni di file tra più processi (usando pagine condivise), alcuni sistemi operativi offrono funzioni per l'accesso bloccante di più processi a sezioni di un file aperto, e, nei sistemi con memoria virtuale, per associare alla memoria le sezioni di un file (Paragrafo 10.3.2).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

11.1.3 Tipi di file

Nella progettazione di un file system, ma anche dell'intero sistema operativo, si deve sempre considerare la possibilità o meno di quest'ultimo di riconoscere e gestire i tipi di file. Un sistema operativo che riconosce il tipo di un file ha la possibilità di trattare il file in modo ragionevole. Ad esempio, un errore abbastanza comune consiste nel tentativo, da parte degli utenti, di stampare un programma oggetto in forma binaria; di solito questo tentativo porta semplicemente a uno spreco di carta, ma si potrebbe impedire se il sistema operativo fosse informato del fatto che il file è un programma oggetto in forma binaria.

Una tecnica comune per realizzare la gestione dei tipi di file consiste nell'includere il tipo nel nome del file. Il nome è suddiviso in due parti, un nome e un'*estensione*, di solito separate da un punto (Figura 11.1); in questo modo l'utente e il sistema operativo possono risalire al tipo del file semplicemente esaminandone il nome. Nell'MS-DOS, ad esempio, un nome può essere composto da non più di otto caratteri seguiti da un punto e terminati da un'estensione di al più tre caratteri. Il sistema usa l'estensione per stabilire il *tipo* del file e le operazioni che si possono eseguire su tale file. Ad esempio, si possono eseguire solamente i file con estensione `.com`, `.exe` o `.bat`; i file con estensione `.com` e `.exe` sono due formati di file eseguibili, mentre i file con estensione `.bat` sono file (*batch*) contenenti una sequenza di comandi, scritti in formato ASCII, diretti al sistema operativo. L'MS-DOS riconosce un numero limitato di estensioni, che però anche i programmi d'applicazione possono usare per individuare i tipi di file cui sono interessati: gli assemblatori si aspettano che i file sorgente siano caratterizzati dall'estensione `.asm`; l'elaboratore di testi WordPerfect si aspetta che i propri file terminino con l'estensione `.wp`. Queste estensioni non sono necessarie, ma la loro presenza consente a un utente di ridurre il numero delle battute specificando il nome del file senza estensione e lasciando all'applicazione il compito di cercare il file con il nome impostato e l'estensione attesa. Poiché queste estensioni non sono gestite dal sistema operativo, si possono considerare un 'suggerimento' rivolto alle applicazioni che operano su di esso.

Tipo di file	Usuale estensione	Funzione
Eseguibile	exe, com, bin, o nessuna	Programma, in linguaggio di macchina, eseguibile
Oggetto	obj, o	Compilato, in linguaggio di macchina, non collegato
Codice sorgente	c, cc, java, pas, asm, a	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi all'interprete dei comandi
Testo	txt, doc	Testi, documenti
Elaboratore di testi	wp, tex, rtf, doc	Vari formati per elaboratori di testi
Libreria	lib, a, so, dll, mpeg, mov, rm	Librerie di procedure per programmatori
Stampa o visualizzazione	ps, pdf, dvi, gif	File ASCII o binari in formato per stampa o visione
Archivio	arc, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, rm	File binari contenenti informazioni audio o A/V

Figura 11.1 Comuni tipi di file.

Un altro esempio dell'utilità dei tipi di file viene dal sistema operativo TOPS-20. Se l'utente cerca di eseguire un programma oggetto il cui file sorgente è stato aggiornato dopo la creazione dell'oggetto, il file sorgente sarà automaticamente ricompilato. Questa funzione consente agli utenti di eseguire sempre una versione aggiornata del programma oggetto. Affinché questa funzione sia possibile il sistema operativo deve poter distinguere il file sorgente dal file oggetto, controllare il momento di creazione o di ultima modifica di entrambi e determinare il linguaggio nel quale è stato scritto il programma sorgente (al fine di adoperare il compilatore corretto).

Nel sistema operativo Apple Macintosh ogni file ha il proprio tipo, come `text` o `pict`, e ciascun file possiede anche un attributo di creazione contenente il nome del programma che lo ha creato. Questo attributo è impostato dal sistema operativo durante la chiamata del sistema `create`, quindi la sua presenza è forzata e gestita dal sistema operativo. Ad esempio, un file prodotto da un elaboratore di testi avrà il nome dell'elaboratore di testi come attributo di creazione. Quando un utente apre il file, con un doppio clic del mouse sull'icona che lo rappresenta, si attiva automaticamente l'elaboratore di testi che apre il file, pronto per essere letto e modificato.

Il sistema operativo UNIX non fornisce una funzione di questo tipo, ma si limita a memorizzare un codice (noto come *magic number*) all'inizio di alcuni tipi di file allo scopo di indicarne in modo generico il tipo: eseguibili, sequenze di comandi (noti come *shell script*), PostScript e così via. Non tutti i file possiedono tale codice, quindi il sistema non può affidarsi unicamente a questo tipo d'informazione; inoltre, non memorizza il nome del programma che ha creato il file. UNIX consente di sfruttare le estensioni come suggerimento del tipo di file; queste non vengono però imposte né dipendono dal sistema operativo; il loro compito consiste principalmente nell'aiutare gli utenti a riconoscere il tipo di contenuto del file. Un'applicazione può usare o ignorare le estensioni; dipende dalle scelte dei programmatori.

11.1.4 Struttura dei file

I tipi di file si possono anche adoperare per indicare la struttura interna dei file. Come si è accennato nel Paragrafo 11.1.3, i file sorgente e i file oggetto hanno una struttura corrispondente a ciò che il programma che dovrà leggerli si attende. Inoltre alcuni file devono rispettare una determinata struttura comprensibile al sistema operativo. Ad esempio, il sistema operativo può richiedere che un file eseguibile abbia una struttura specifica che

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

consenta di determinare dove caricare il file nella memoria e qual è la locazione della prima istruzione. Alcuni sistemi operativi estendono questa idea a un insieme di strutture di file gestite dal sistema, con un insieme di operazioni specifiche per la manipolazione dei file con queste strutture. Ad esempio, il sistema operativo DEC VMS è dotato di un file system che gestisce tre strutture di file.

La discussione precedente porta a considerare uno degli svantaggi dei sistemi operativi che gestiscono più strutture di file: la dimensione risultante del sistema operativo è ingombrante. Se definisce cinque strutture di file differenti, il sistema operativo deve contenere il codice per gestirle tutte; inoltre qualsiasi file potrebbe dover essere definito come uno dei tipi gestiti dal sistema operativo, con la conseguenza di introdurre notevoli problemi per le applicazioni che richiedono una strutturazione dei propri dati in modi non previsti dal sistema operativo.

Ad esempio, si supponga che un sistema operativo preveda due tipi di file: file di testo (composti da caratteri ASCII separati da caratteri di ritorno del carrello e avanzamento di riga) e file binari eseguibili. Un utente che volesse definire un file cifrato per proteggere i propri dati da letture non autorizzate potrebbe scoprire che nessuna delle due strutture si adatta al problema: non è un file di righe di testo ASCII, ma un insieme di bit (apparentemente casuali), e sebbene possa sembrare un file binario, non è eseguibile. Queste limitazioni impongono all'utente di raggirare o usare in modo scorretto il meccanismo dei tipi dei file definito dal sistema operativo, oppure di abbandonare lo schema di codifica.

Alcuni sistemi operativi impongono (e gestiscono) un numero minimo di strutture di file. Questo orientamento è stato seguito dallo UNIX, dall'MS-DOS e altri. Lo UNIX considera ciascun file come una sequenza di byte, senza alcuna interpretazione. Questo schema garantisce la massima flessibilità, ma il minimo sostegno. Qualsiasi programma d'applicazione deve contenere il proprio codice per interpretare in modo appropriato la struttura di un file. A ogni modo, per poter caricare ed eseguire i programmi, tutti i sistemi operativi devono prevedere almeno un tipo di struttura, quella dei file eseguibili.

Anche il sistema operativo Macintosh gestisce un numero ridotto di strutture di file, prevede che i file eseguibili consistano di due parti, *resource fork* e *data fork*. La prima contiene le informazioni che interessano l'utente, ad esempio le etichette dei pulsanti dell'interfaccia del programma. (Per tradurre in un'altra lingua le etichette dei pulsanti, si possono adoperare gli strumenti messi a disposizione dal sistema operativo per la modifica delle informazioni contenute nella prima parte del file.) La seconda contiene il codice del programma o dati: l'usuale contenuto dei file. Per ottenere i medesimi risultati nello UNIX o nell'MS-DOS, il programmatore dovrebbe modificare e ricompilare il codice sorgente, a meno che non abbia creato il proprio tipo di file di dati modificabile dall'utente. Evidentemente è utile che un sistema operativo gestisca le strutture che si adoperano spesso, ciò risparmia molto lavoro ai programmatori. Un numero eccessivamente limitato di strutture rende scomoda la programmazione, mentre troppe strutture appesantiscono il sistema operativo e confondono i programmatori.

11.1.5 Struttura interna dei file

Per il sistema operativo la localizzazione di uno scostamento all'interno di un file può essere complicata. I dischi hanno una dimensione dei blocchi ben definita, stabilita secondo la dimensione di un settore (Capitolo 2). Tutti gli I/O su disco si eseguono in unità di un blocco (record fisico), e tutti i blocchi hanno la stessa dimensione. È improbabile che la dimensione del record fisico corrisponda esattamente alla lunghezza del record logico desiderato, che può anche essere variabile. Una soluzione diffusa per questo tipo di problema consiste nell'**impaccamento** di un certo numero di record logici in blocchi fisici.

Il sistema operativo UNIX, ad esempio, definisce tutti i file semplicemente come un flusso di byte. A ciascun byte si può accedere in modo individuale tramite il suo scostamento a partire dall'inizio, o dalla fine, del file. In questo caso il record logico è un byte. Il file system impacca e deimpacca automaticamente i byte in blocchi fisici (ad esempio 512 byte per blocco) come è necessario.

La dimensione dei record logici, quella dei blocchi fisici e la tecnica d'impaccamento determinano il numero dei record logici all'interno di ogni blocco fisico. L'impaccamento può essere fatto dal programma d'applicazione dell'utente oppure dal sistema operativo.

In entrambi i casi il file si può considerare come una sequenza di blocchi. Tutte le funzioni di I/O di base operano in termini di blocchi. La conversione da record logici a blocchi fisici è un problema di programmazione relativamente semplice.

Poiché lo spazio del disco è sempre assegnato in blocchi, una parte dell'ultimo blocco di ogni file in genere è sprecata. Se ogni blocco è composto di 512 byte, allora a un file di 1949 byte si assegnano quattro blocchi, 2048 byte; gli ultimi 99 byte sono sprecati. I byte sprecati, assegnati per la gestione in multipli di blocchi invece che di byte, costituiscono la frammentazione interna. Tutti i file system soffrono di *frammentazione interna*; maggiore è la dimensione dei blocchi, maggiore è la frammentazione interna.

11.2 Metodi d'accesso

I file memorizzano informazioni; al momento dell'uso è necessario accedere a queste informazioni e trasferirle nella memoria. Esistono molti metodi per accedere alle informazioni dei file; alcuni sistemi consentono un solo metodo d'accesso ai file, altri, come quelli dell'IBM, offrono diversi metodi d'accesso; la scelta del metodo giusto per una particolare applicazione è un importante problema di progettazione.

11.2.1 Accesso sequenziale

Il più semplice metodo d'accesso è l'**accesso sequenziale**: le informazioni del file si elaborano ordinatamente, un record dopo l'altro; questo metodo d'accesso è di gran lunga il più comune, ed è usato, ad esempio, dagli editor e dai compilatori.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le più comuni operazioni che si compiono sui file sono le letture e le scritture: un'operazione di lettura legge la prima porzione e fa avanzare automaticamente il puntatore del file che tiene traccia della locazione di I/O; analogamente, un'operazione di scrittura fa un'aggiunta in coda al file e avanza fino alla fine delle informazioni appena scritte, che costituisce la nuova fine del file. Un file siffatto si può reimpostare sull'inizio e, in alcuni sistemi, un programma può riuscire ad andare avanti o indietro di n record, con n intero e alcune volte solo per $n = 1$. L'accesso sequenziale è illustrato nella Figura 11.2. L'accesso sequenziale è basato su un modello di file che si rifà al nastro, e funziona nei dispositivi ad accesso sequenziale così come nei dispositivi ad accesso diretto.

11.2.2 Accesso diretto

Un altro metodo è l'**accesso diretto** (o **accesso relativo**). Un file è formato da elementi logici (*record*) di lunghezza fissa, ciò consente ai programmi di leggere e scrivere rapidamente tali elementi senza un ordine particolare. Il metodo ad accesso diretto si fonda su un modello di file che si rifà al disco: i dischi permettono, infatti, l'accesso diretto a ogni blocco di file. Il file si considera come una sequenza numerata di blocchi o record che si possono leggere o scrivere in modo arbitrario: si può ad esempio leggere il blocco 14, quindi il blocco 53 e poi scrivere il blocco 7. Non esistono limiti all'ordine di lettura o scrittura di un file ad accesso diretto.

I file ad accesso diretto sono molto utili quando è necessario accedere immediatamente a grandi quantità di informazioni. Spesso le basi di dati sono di questo tipo: quando si presenta un'interrogazione riguardante un oggetto particolare, occorre stabilire quale blocco contiene la risposta alla richiesta e quindi leggere direttamente quel blocco, ottenendo così le informazioni richieste.

In un sistema di prenotazione di volo, ad esempio, si possono registrare tutte le informazioni su un particolare volo, ad esempio il 713, nel blocco identificato da tale numero di volo. Quindi, il numero di posti disponibili per il volo 713 si memorizza nel blocco 713 del file di prenotazione. Per registrare informazioni riguardanti un gruppo più grande, ad esempio una popolazione, si può eseguire la ricerca calcolando una funzione hash sui nomi delle persone, oppure usando un piccolo indice per determinare il blocco da leggere.

Per il metodo ad accesso diretto, si devono modificare le operazioni sui file per inserire il numero del blocco in forma di parametro. Quindi, si hanno `read` n, dove n è il numero del blocco, al posto di `read next`, e `write` n, invece che `write next`. Un metodo alternativo prevede di mantenere `read next` e `write next`, come nell'accesso sequenziale, e di aggiungere un'operazione `position to` n, dove n è il numero del blocco. Quindi a un'operazione `read` n corrispondono una `position to` n e una `read next`.

Il numero del blocco fornito dall'utente al sistema operativo è normalmente un **numero di blocco relativo**. Si tratta di un indice relativo all'inizio del file, quindi il primo blocco relativo del file è 0, il successivo è 1 e così via, anche se l'indirizzo assoluto nel disco del blocco può essere 14703 per il primo blocco e 3192 per il secondo. L'uso dei numeri di blocco relativi permette al sistema operativo di decidere dove posizionare il file (si tratta del *problema dell'assegnazione* trattato nel Capitolo 12) e aiuta a impedire che l'utente acceda a porzioni del file system che possono non fare parte del suo file. Alcuni sistemi iniziano la numerazione dei blocchi relativi da 0, altri da 1.

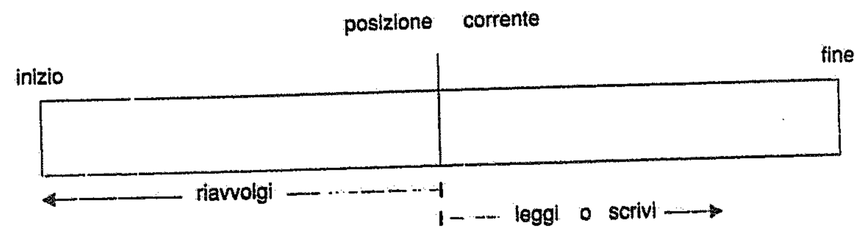


Figura 11.2 File ad accesso sequenziale.

Dato un record logico di lunghezza l, una richiesta per il record r determina una richiesta di I/O per l byte alla locazione $l \times (r - 1)$ all'interno del file. La lettura, scrittura e cancellazione di un record sono rese più semplici dalla sua dimensione fissa.

Non tutti i sistemi operativi gestiscono ambedue i tipi di accesso; alcuni permettono il solo accesso sequenziale, altri solo quello diretto. Alcuni sistemi richiedono che si definisca il tipo d'accesso al file al momento della sua creazione; a tale file si può accedere soltanto nel modo definito. Tuttavia si può facilmente simulare l'accesso sequenziale a un file ad accesso diretto mantenendo una variabile pc che, come illustra la Figura 11.3, definisce la posizione corrente. D'altra parte è estremamente goffo e inefficiente simulare l'accesso diretto a un file che di per sé è ad accesso sequenziale.

11.2.3 Altri metodi d'accesso

Sulla base di un metodo d'accesso diretto se ne possono costruire altri, che implicano generalmente la costruzione di un indice per il file. L'**indice** contiene puntatori ai vari blocchi; per trovare un elemento del file occorre prima cercare nell'indice, e quindi usare il puntatore per accedere direttamente al file e trovare l'elemento desiderato.

Si consideri, ad esempio, un file contenente prezzi al dettaglio; questo può contenere un elenco dei codici universali dei prodotti (*universal product codes* — UPC), a ciascuno dei quali è associato un prezzo. Dato un elemento di 16 byte, questo è composto da un codice UPC a 10 cifre e un prezzo a 6 cifre. Se il disco usato ha 1024 byte per blocco, in ogni blocco si possono memorizzare 64 elementi. Un file di 120.000 elementi occupa circa 2000 blocchi (2 milioni di byte). Ordinando il file secondo il codice UPC si può definire un indice composto dal primo codice UPC di ogni blocco. Tale indice è costituito di 2000 elementi di 10 cifre ciascuno (20.000 byte) e quindi può essere tenuto nella memoria. Per trovare il prezzo di un oggetto specifico si può fare una ricerca (binaria) nell'indice, che permette di sapere esattamente quale blocco contiene l'elemento de-

http://www.adobe.com/ Created by TIFF To PDF trial version, to remove this mark, please register this software.

Accesso sequenziale	Realizzazione nel caso di accesso diretto
`reset`	`cp = 0;`
`read next`	`read cp;` `cp = cp+1;`
`write next`	`write cp;` `cp = cp+1;`

Figura 11.3 Simulazione dell'accesso sequenziale a un file ad accesso diretto.

siderato e quindi accedere a quel blocco. Questa struttura permette di compiere ricerche in file molto lunghi limitando il numero di operazioni di I/O.

Nel caso di file molto lunghi, lo stesso file indice può diventare troppo lungo perché sia tenuto nella memoria. Una soluzione a questo problema è data dalla creazione di un indice per il file indice. Il file indice principale contiene puntatori ai file indice secondari, i quali puntano agli effettivi elementi di dati.

Il metodo ad accesso sequenziale indicizzato dell'IBM (*indexed sequential access method* — ISAM), ad esempio, usa un piccolo indice principale che punta ai blocchi del disco di un indice secondario, e i blocchi dell'indice secondario puntano ai blocchi del file effettivo. Il file è ordinato rispetto a una chiave definita. Per trovare un particolare elemento, si fa inizialmente una ricerca binaria nell'indice principale, che fornisce il numero del blocco dell'indice secondario. Questo blocco viene letto e sottoposto a una seconda ricerca binaria che individui il blocco contenente l'elemento richiesto. Infine, si fa una ricerca sequenziale sul blocco. In questo modo si può localizzare ogni elemento tramite il suo codice con al più due letture ad accesso diretto. La Figura 11.4 mostra uno schema simile, come è realizzato nel VMS con indici e relativi file.

11.3 Struttura di directory

Il file system di un calcolatore può essere molto ampio. Alcuni sistemi registrano milioni di file in terabyte di spazio dei dischi. Per gestire tutti questi dati è necessario che essi siano organizzati; di solito si suddivide il file system in una o più *partizioni*, note anche come *minidischi* nell'ambiente IBM e *volumi* tra gli utenti di PC e Macintosh; ciascuna partizione contiene le informazioni sui file in essa contenuti. Tipicamente, ciascun disco di un sistema contiene almeno una partizione, che è una struttura a basso livello nella quale risiedono file e directory. Talvolta le partizioni si adoperano per fornire aree separate all'interno di un unico disco, ciascuna delle quali è trattata come un diverso dispositivo di memorizzazione. Altri sistemi consentono la definizione di partizioni più grandi di un disco

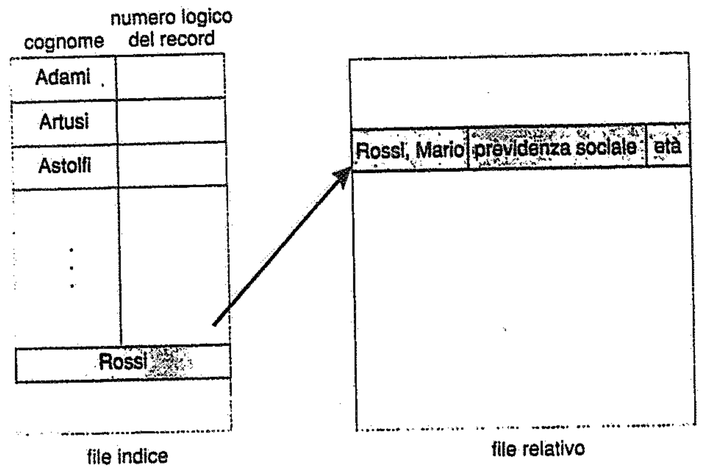


Figura 11.4 Esempio di indice e relativo file.

per raggruppare più dischi in un'unica struttura logica. In questo modo l'utente deve considerare unicamente la struttura logica di file e directory e può ignorare completamente i problemi dell'assegnazione fisica dello spazio per i file. Per questa ragione è possibile considerare le partizioni come dischi virtuali. Le partizioni possono anche contenere più sistemi operativi, consentendo l'avviamento e l'esecuzione di ciascuno di essi.

Le informazioni sui file contenuti in ciascuna partizione sono mantenute negli elementi della **directory del dispositivo** (*device directory*) o **tabella dei contenuti del volume**. La directory di dispositivo (più nota semplicemente come *directory*) registra le informazioni (come nome, posizione, dimensione e tipo) di tutti i file della partizione. La Figura 11.5 mostra l'usuale organizzazione di un file system.

La directory si può considerare come una tabella di simboli che traduce i nomi dei file negli elementi in essa contenuti. Da questo punto di vista, si capisce che la stessa directory si può organizzare in molti modi diversi; deve essere possibile inserire nuovi elementi, cancellarne di esistenti, cercare un elemento ed elencare tutti gli elementi della directory. Nel Capitolo 12 si affronta la discussione sulle appropriate strutture di dati utilizzabili per la realizzazione delle directory. In questo paragrafo si esaminano i vari schemi per la definizione della struttura logica del sistema di directory. Nel considerare una particolare struttura di directory si deve tenere presente l'insieme delle operazioni che si possono eseguire su una directory:

- **Ricerca di un file.** Deve esserci la possibilità di scorrere una directory per trovare l'elemento associato a un particolare file. Poiché i file possono avere nomi simbolici, e poiché nomi simili possono indicare relazioni tra file, deve esistere la possibilità di trovare tutti i file il cui nome soddisfi una particolare espressione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

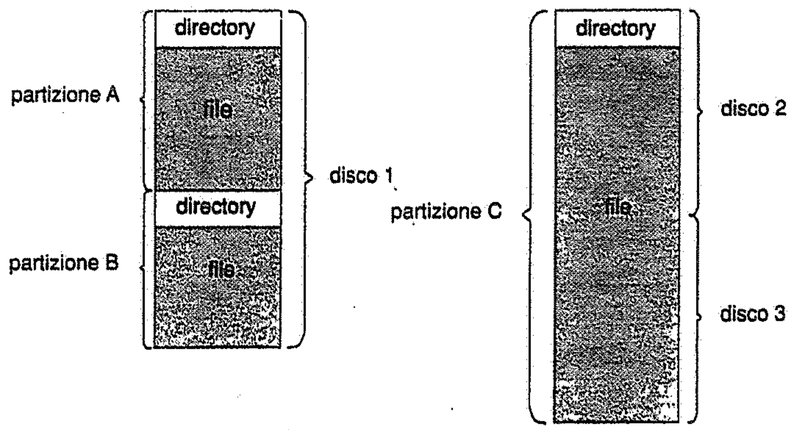


Figura 11.5 Tipica organizzazione di un file system.

- **Creazione di un file.** Deve essere possibile creare nuovi file e aggiungerli alla directory.
- **Cancellazione di un file.** Quando non serve più, si deve poter rimuovere un file dalla directory.
- **Elencazione di una directory.** Deve esistere la possibilità di elencare tutti i file di una directory, e il contenuto degli elementi della directory associati ai rispettivi file nell'elenco.
- **Ridenominazione di un file.** Poiché il nome di un file rappresenta il contenuto del file ai suoi utenti, questo nome deve poter essere modificato quando cambia il contenuto o l'uso del file. La ridenominazione di un file potrebbe comportare la variazione della posizione del file nella directory.
- **Attraversamento del file system.** Si potrebbe voler accedere a ogni directory e a ciascun file contenuto in una directory. Per motivi di affidabilità, è opportuno salvare il contenuto e la struttura dell'intero file system a intervalli regolari. Questo salvataggio consiste nella copiatura di tutti i file in un nastro magnetico; tale tecnica consente di avere una copia di riserva (*backup*) che sarebbe utile nel caso in cui si dovesse verificare un guasto nel sistema o più semplicemente se un file non è più in uso. In quest'ultimo caso si può liberare lo spazio da esso occupato nel disco che può essere, quindi, riutilizzato per altri file.

Nei Paragrafi dall'11.3.1 all'11.3.5 sono descritti gli schemi più comuni per la definizione della struttura logica di una directory.

11.3.1 Directory a singolo livello

La struttura più semplice per una directory è quella a singolo livello. Tutti i file sono contenuti nella stessa directory, facilmente gestibile e comprensibile (Figura 11.6).

Una directory a livello singolo presenta però limiti notevoli che si manifestano all'aumentare del numero dei file in essa contenuti, oppure se il sistema è usato da più utenti. Poiché si trovano tutti nella stessa directory, i file devono avere nomi unici; se due utenti attribuiscono lo stesso nome al loro file di dati, ad esempio prova, si vìola la regola del nome unico. Anche se i nomi dei file generalmente si scelgono in modo da riflettere il contenuto del file stesso, spesso hanno una lunghezza limitata (il sistema operativo MS-DOS permette nomi di non più di 11 caratteri, lo UNIX permette lunghezze di 255 caratteri).

Anche per un solo utente, con una directory a livello singolo, diventa difficile ricordare i nomi dei file, con l'aumentare del loro numero. Non è affatto raro che un utente abbia centinaia di file in un calcolatore e altrettanti file in un altro sistema. In un tale ambiente, sarebbe terribile dover ricordare tanti nomi di file.

11.3.2 Directory a due livelli

Una directory a livello singolo spesso causa la confusione dei nomi dei file tra diversi utenti. La soluzione più ovvia prevede la creazione di una directory *separata* per ogni utente.

Nella struttura a due livelli, ogni utente dispone della propria **directory d'utente** (*user file directory* — UFD). Tutte le directory d'utente hanno una struttura simile, ma in ciascuna sono elencati solo i file del proprietario. Quando comincia l'elaborazione di un lavoro d'utente, oppure un utente inizia una sessione di lavoro, si fa una ricerca nella **directory principale** (*master file directory* — MFD) del sistema. La directory principale viene indirizzata con il nome dell'utente o il numero che lo rappresenta, e ogni suo elemento punta alla relativa directory d'utente (Figura 11.7).

Quando un utente fa un riferimento a un file particolare, il sistema operativo esegue la ricerca solo nella directory di quell'utente. In questo modo utenti diversi possono avere file con lo stesso nome, purché tutti i nomi di file all'interno di ciascuna directory d'utente siano unici. Per creare un file per un utente, il sistema operativo controlla che non ci sia un altro file con lo stesso nome soltanto nella directory di tale utente. Per cancellare un file il sistema operativo limita la propria ricerca alla directory d'utente locale,

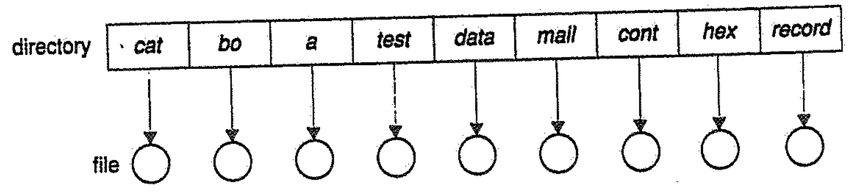


Figura 11.6 Directory a livello singolo.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

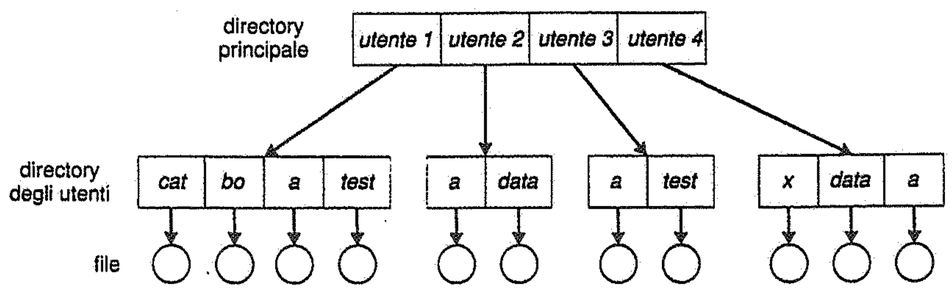


Figura 11.7 Struttura di directory a due livelli.

quindi non può cancellare per errore un file con lo stesso nome che appartenga a un altro utente.

Le stesse directory d'utente devono essere create e cancellate come è necessario; a tale scopo si esegue uno speciale programma di sistema con nome d'utente e dati contabili adeguati. Il programma crea una nuova directory d'utente e aggiunge l'elemento a essa corrispondente nella directory principale. L'esecuzione di questo programma può essere limitata all'amministratore del sistema. L'assegnazione dello spazio nei dischi per le directory d'utente può essere gestita con le tecniche descritte per i file nel Capitolo 12.

Sebbene risolva il problema delle collisioni dei nomi, la struttura di directory a due livelli presenta ancora dei problemi. In effetti, questa struttura isola un utente dagli altri utenti. Questo isolamento può essere un vantaggio quando gli utenti sono completamente indipendenti, ma è uno svantaggio quando gli utenti *vogliono* cooperare e accedere a file di altri utenti. Alcuni sistemi non permettono l'accesso a file di utenti locali da parte di altri utenti.

Se l'accesso è autorizzato, un utente deve avere la possibilità di riferirsi al nome di un file che si trova nella directory di un altro utente. Per attribuire un nome unico a un particolare file di una directory a due livelli, occorre indicare sia il nome dell'utente sia il nome del file. Una directory a due livelli si può pensare come un albero, o almeno un albero rovesciato, di altezza 2. La radice dell'albero è la directory principale, i suoi diretti discendenti sono le directory d'utente, da cui discendono i file che sono le foglie dell'albero. Specificando un nome d'utente e un nome di file si definisce un percorso che parte dalla radice (la directory principale) e arriva a una specifica foglia (il file specificato). Quindi, un nome d'utente e un nome di file definiscono un *nome di percorso* (*path name*). Ogni file del sistema ha un nome di percorso. Per attribuire un nome unico a un file, un utente deve conoscere il nome di percorso del file desiderato.

Se, ad esempio, l'utente *A* desidera accedere al proprio file chiamato prova, è sufficiente che faccia riferimento a prova. Invece, per accedere al file denominato prova dell'utente *B*, con nome di elemento della directory utenteB, l'utente *A* deve fare riferimento a /utenteB/prova. Ogni sistema ha la propria sintassi per riferirsi ai file delle directory diverse da quella dell'utente.

Per specificare la partizione cui appartiene un file occorrono ulteriori regole sintattiche. Nell'MS-DOS, ad esempio, una partizione è indicata da una lettera seguita dai due punti. Quindi l'indicazione di un file potrebbe essere del tipo C:\utenteB\prova. Alcuni sistemi vanno oltre e separano: la partizione, il nome della directory, e il nome del file. Nel VMS, ad esempio, il file login.com potrebbe essere indicato come u:[sst.jdeck]login.com;1, dove u è il nome della partizione, sst è il nome della directory, jdeck è il nome della sottodirectory e 1 è il numero della versione. Altri sistemi trattano il nome della partizione semplicemente come parte del nome della directory. Il primo elemento è quello della partizione, il resto è composto dalla directory e dal file. Ad esempio, /u/pbg/prova potrebbe indicare la partizione u, la directory pbg e il file prova.

Un caso particolare di questa situazione riguarda i file di sistema. I programmi forniti come elementi integranti del sistema, come caricatori, assemblatori, compilatori, strumenti, librerie e così via, sono infatti definiti come file. Quando si impartiscono i comandi appropriati al sistema operativo, il caricatore legge questi file che poi vengono eseguiti. Molti interpreti di comandi operano semplicemente trattando il comando come il nome di un file da caricare ed eseguire. Poiché il sistema delle directory è già stato definito, questo nome di file viene cercato nella directory d'utente locale. Una soluzione prevede la copiatura dei file di sistema in ciascuna directory d'utente. Tuttavia, con la copiatura di tutti i file di sistema si spreca un'enorme quantità di spazio. Se i file di sistema occupano 5 MB, con 12 utenti si avrebbe un'occupazione di spazio pari a 5 × 12 = 60 MB, solo per le copie dei file di sistema.

La soluzione standard prevede una leggera complicazione della procedura di ricerca; si definisce una speciale directory d'utente contenente i file di sistema, ad esempio la directory d'utente 0. Ogni volta che si indica un file da caricare, il sistema operativo lo cerca innanzi tutto nella directory d'utente locale, e, se lo trova, lo usa; se non lo trova, il sistema cerca automaticamente nella speciale directory d'utente che contiene i file di sistema. La sequenza delle directory in cui è stata fatta la ricerca avviata dal riferimento a un file è detta **percorso di ricerca** (*search path*). Tale idea si può estendere in modo che il percorso di ricerca contenga un elenco illimitato di directory nelle quali fare le ricerche quando si dà il nome di un comando. Questo metodo è il più usato nello UNIX e nell'MS-DOS.

11.3.3 Directory con struttura ad albero

La corrispondenza strutturale tra directory a due livelli e albero a due livelli, permette di generalizzare facilmente il concetto estendendo la struttura di directory a un albero di altezza arbitraria (Figura 11.8). Questa generalizzazione permette agli utenti di creare proprie sottodirectory e di organizzare i file di conseguenza. Il file system dell'MS-DOS, ad esempio, è strutturato ad albero, si tratta infatti del più comune tipo di struttura delle directory. L'albero ha una directory radice (*root directory*), e ogni file del sistema ha un unico nome di percorso. Un nome di percorso descrive il percorso che parte dalla radice, passa attraverso tutte le sottodirectory e arriva a un file specifico.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

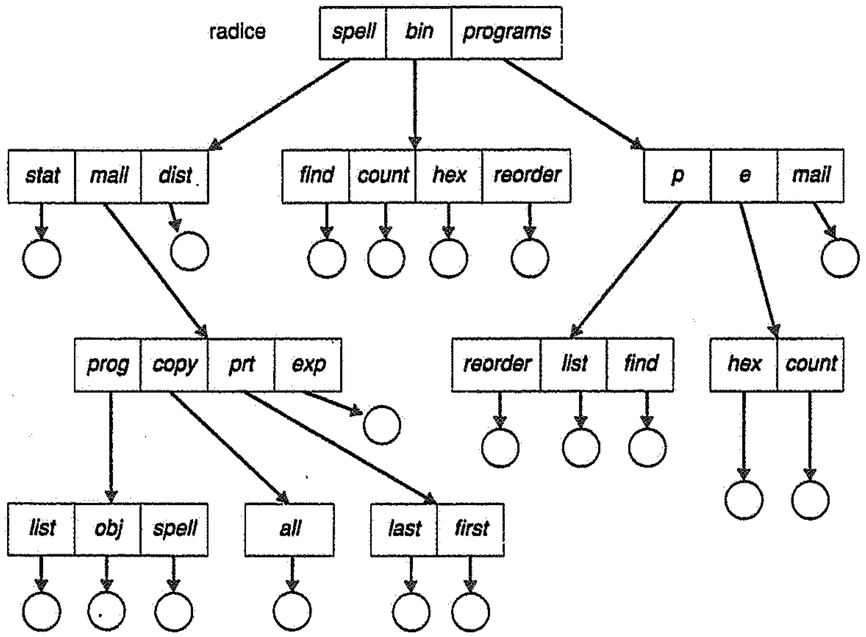


Figura 11.8 Struttura di directory ad albero.

Una directory, o una sottodirectory, contiene un insieme di file o sottodirectory. Le directory sono semplicemente file, che sono però trattati in modo speciale. Tutte le directory hanno lo stesso formato interno. La distinzione tra file e directory è data dal bit, rispettivamente 0 e 1, di ogni elemento della directory. Per creare e cancellare le directory si adoperano speciali chiamate del sistema.

Normalmente, ogni utente dispone di una directory corrente. La directory corrente deve contenere la maggior parte dei file di interesse corrente. Quando si fa un riferimento a un file, si esegue una ricerca nella directory corrente; se il file non si trova in tale directory, l'utente deve specificare un nome di percorso oppure cambiare la directory corrente facendo diventare tale la directory che contiene il file desiderato. Per cambiare directory corrente si fa uso di una chiamata del sistema che preleva un nome di directory come parametro e lo usa per ridefinire la directory corrente. Quindi, l'utente può cambiare la propria directory corrente ogni volta che lo desidera. Da una chiamata del sistema `change directory` alla successiva, tutte le chiamate del sistema `open` cercano i file specificati nella directory corrente.

La directory corrente iniziale di un utente è stabilita all'avvio del lavoro d'elaborazione dell'utente, oppure quando quest'ultimo inizia una sessione di lavoro; il sistema operativo cerca nel file di contabilizzazione, o in qualche altra locazione predefinita, l'elemento relativo a questo utente. Nel file di contabilizzazione è memorizzato un puntatore alla (oppure il nome della) directory iniziale dell'utente. Tale puntatore viene copiato in una variabile locale per l'utente che specifica la sua directory corrente iniziale.

I nomi di percorso possono essere di due tipi: **nomi di percorso assoluti** e **nomi di percorso relativi**. Un nome di percorso assoluto comincia dalla radice dell'albero di directory e segue un percorso che lo porta fino al file specificato indicando i nomi delle directory che costruiscono le tappe del suo cammino. Un nome di percorso relativo definisce un percorso che parte dalla directory corrente. Ad esempio, nel file system con struttura ad albero della Figura 11.8, se la directory corrente è `root/spell/mail`, allora il nome di percorso relativo `prt/first` si riferisce allo stesso file cui si riferisce il nome di percorso assoluto `root/spell/mail/prt/first`.

Se si permette all'utente di definire le proprie sottodirectory, gli si consente anche di dare una struttura ai suoi file. Questa struttura può presentare directory distinte per file associati a soggetti diversi; ad esempio, si può creare una sottodirectory contenente il testo di questo libro, oppure diversi tipi di informazioni, ad esempio la directory `programmi` può contenere programmi sorgente; la directory `bin` può contenere tutti i programmi eseguibili.

Una decisione importante relativa alla strutturazione ad albero delle directory riguarda il modo di gestire la cancellazione di una directory. Se una directory è vuota, è sufficiente cancellare l'elemento che la designa nella directory che la contiene. Tuttavia se la directory da cancellare non è vuota, ma contiene file oppure sottodirectory, è possibile procedere in due modi. Alcuni sistemi, come l'MS-DOS, non cancellano una directory a meno che non sia vuota; per cancellarla l'utente deve prima cancellare i file in essa contenuti. Se esiste qualche sottodirectory, questa procedura si deve applicare anche alle sottodirectory. Questo metodo può richiedere una discreta quantità di lavoro.

In alternativa, come nel comando `rm` dello UNIX, si può avere un'opzione che, alla richiesta di cancellazione di una directory, cancelli anche tutti i file e tutte le sottodirectory in essa contenuti. Entrambi i criteri sono abbastanza facili da realizzare; si tratta soltanto di stabilire quale seguire. Il secondo criterio è più comodo, anche se più pericoloso, poiché si può rimuovere un'intera struttura di directory con un solo comando. Se si eseguisse tale comando per sbaglio sarebbe necessario ripristinare un gran numero di file e directory dalle copie di riserva.

Con un sistema di directory strutturato ad albero anche l'accesso ai file di altri utenti è di facile realizzazione. Ad esempio, l'utente *B* può accedere ai file dell'utente *A* specificando i nomi di percorso assoluti oppure relativi. In alternativa, l'utente B può far sì che la propria directory corrente sia quella dell'utente *A* e accedere ai file usando direttamente i loro nomi. Alcuni sistemi permettono anche agli utenti di definire i propri percorsi di ricerca. In questo caso, l'utente *B* può definire il proprio percorso di ricerca come (1) directory locale, (2) file directory dei file di sistema e (3) directory dell'utente *A*. Fintanto che il nome di un file dell'utente *A* non entra in conflitto con il nome di un file locale o di un file di sistema, si può continuare a fare riferimento al file con il suo nome.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un percorso per un file in una directory strutturata ad albero può ovviamente essere più lungo di quello in una directory a due livelli. Per consentire agli utenti di accedere ai programmi senza dover ricordare tali lunghi percorsi, il sistema operativo Macintosh cerca automaticamente i programmi eseguibili attraverso un file detto `Desktop file` contenente il nome e la locazione di tutti i programmi eseguibili a esso noti. Quando si aggiunge al sistema un nuovo disco o dischetto, o si accede a una rete di comunicazione, il sistema operativo attraversa la struttura delle directory del dispositivo alla ricerca dei programmi eseguibili e registra le informazioni relative. Questo meccanismo consente la funzione del doppio clic descritta precedentemente. Un doppio clic sul nome di un file, o sull'icona che lo rappresenta, determina la lettura del suo attributo di creazione e la ricerca dell'elemento corrispondente nel `Desktop file`. Una volta trovato tale elemento, si avvia il programma eseguibile associato che apre il file su cui si è fatto il doppio clic. La famiglia di sistemi operativi Microsoft Windows (95/98/*me* e NT/2000/XP) mantiene una struttura di directory a due livelli estesa; con lettere di unità assegnate a dispositivi e partizioni (Paragrafo 11.4).

11.3.4 Directory con struttura a grafo aciclico

Si considerino due programmatori che lavorano a un progetto comune. I file associati a quel progetto si possono memorizzare in una sottodirectory, separandoli da altri progetti e file dei due programmatori, ma poiché entrambi i programmatori hanno le stesse responsabilità sul progetto, ciascuno richiede che la sottodirectory si trovi nelle proprie directory. La sottodirectory comune deve essere *condivisa*. Nel sistema esiste quindi una directory condivisa o un file condiviso in due, o più, posizioni alla volta.

La struttura ad albero non ammette la condivisione di file o directory. Un **grafo aciclico** permette alle directory di avere sottodirectory e file condivisi (Figura 11.9). Lo *stesso* file o la *stessa* sottodirectory possono essere in due directory diverse. Un grafo aciclico, cioè senza cicli, rappresenta la generalizzazione naturale dello schema delle directory con struttura ad albero.

Il fatto che un file sia condiviso, o che una directory sia condivisa, non significa che ci siano due copie del file: con due copie ciascun programmatore potrebbe vedere la copia presente nella propria directory e non l'originale; se un programmatore modifica il file le modifiche non appaiono nell'altra copia. Se invece il file è condiviso esiste *un* solo file effettivo, perciò tutte le modifiche sono immediatamente visibili. La condivisione è di particolare importanza se applicata alle sottodirectory; un nuovo file appare automaticamente in tutte le sottodirectory condivise.

Quando più persone lavorano insieme, tutti i file da condividere si possono inserire in una directory comune. Ciascuna directory d'utente di tutti i membri del gruppo contiene questa directory di file condivisi in forma di sottodirectory. Anche nel caso di un singolo utente, il metodo di gestione dei file di tale utente può richiedere che alcuni file siano inseriti in più sottodirectory. Ad esempio, un programma scritto per un progetto particolare deve trovarsi sia nella directory di tutti i programmi sia nella directory di quel progetto.

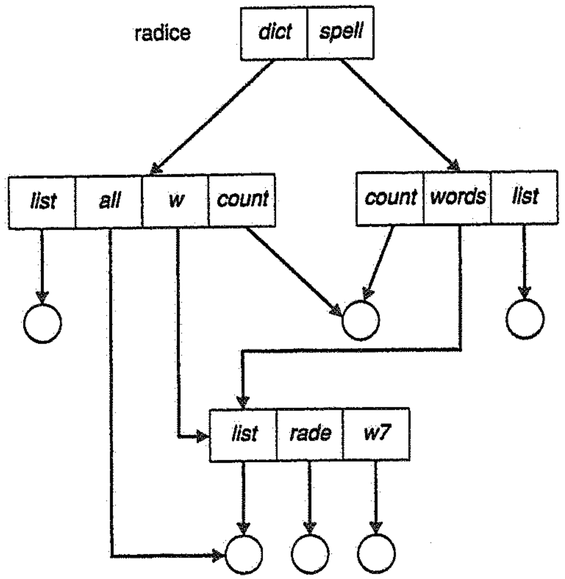


Figura 11.9 Struttura di directory a grafo aciclico.

I file e le sottodirectory condivisi si possono realizzare in molti modi. Un metodo diffuso, esemplificato da molti tra i sistemi UNIX, prevede la creazione di un nuovo elemento di directory, chiamato collegamento. Un **collegamento** (*link*) è un puntatore a un altro file o un'altra directory. Ad esempio, un collegamento si può realizzare come un nome di percorso assoluto o relativo. Quando si fa riferimento a un file, si compie una ricerca nella directory, l'elemento cercato è contrassegnato come collegamento e riporta il nome di percorso del file, o della directory, reale. Quindi si *risolve* il collegamento usando il nome di percorso per localizzare il file reale. I collegamenti si identificano facilmente tramite il loro formato nell'elemento della directory (o, nei sistemi che gestiscono i tipi, dal tipo speciale), e sono infatti chiamati puntatori indiretti. Durante l'attraversamento degli alberi delle directory il sistema operativo ignora questi collegamenti, preservando così la struttura aciclica della struttura.

Un altro comune metodo per la realizzazione dei file condivisi prevede semplicemente la duplicazione di tutte le informazioni relative ai file in entrambe le directory di condivisione, quindi i due elementi sono identici. Un collegamento è chiaramente diverso dagli altri elementi della directory. Duplicando gli elementi della directory, invece, la copia e l'originale sono resi indistinguibili: sorge allora il problema di mantenere la coerenza se il file viene modificato.

http://www.adultpdf.com
Created by TIFF To PDF trial version, please visit www.tiffpdf.com to remove this watermark, please register this software.

Una struttura di directory a grafo aciclico è più flessibile di una semplice struttura ad albero, ma anche più complessa. Si devono prendere in considerazione parecchi problemi. Un file può avere più nomi di percorso assoluti, quindi nomi diversi possono riferirsi allo stesso file. Questa situazione è simile al problema dell'uso degli *alias* nei linguaggi di programmazione. Quando si percorre tutto il file system — per trovare un file, per raccogliere dati statistici su tutti i file o per fare le copie di riserva di tutti i file — il problema diviene più grave poiché non si devono attraversare più di una volta le strutture condivise.

Un altro problema riguarda la cancellazione, poiché è necessario stabilire in quali casi è possibile riassegnare e riutilizzare lo spazio assegnato a un file condiviso. Una possibilità prevede che a ogni operazione di cancellazione segua l'immediata rimozione del file, quest'azione può però lasciare puntatori a un file che ormai non esiste più. Sarebbe ancora più grave se i puntatori contenessero gli indirizzi effettivi del disco e lo spazio fosse poi riutilizzato per altri file, poiché i puntatori potrebbero puntare in mezzo a questi altri file.

In un sistema dove la condivisione è realizzata da collegamenti simbolici la gestione di questa situazione è relativamente semplice. La cancellazione di un collegamento non influisce sul file originale, poiché si rimuove solo il collegamento. Se si cancella il file, si libera lo spazio corrispondente lasciando in sospeso il collegamento; a questo punto è possibile cercare tutti questi collegamenti e rimuoverli, ma se in ogni file non esiste un elenco dei collegamenti associati al file stesso questa ricerca può essere abbastanza onerosa. In alternativa, si possono lasciare i collegamenti finché non si tenta di usarli, quindi si 'scopre' che il file con il nome dato dal collegamento non esiste e non si riesce a determinare il collegamento rispetto al nome; l'accesso si tratta proprio come qualsiasi altro nome di file irregolare. In questo caso, il progettista del sistema deve decidere attentamente cosa si debba fare quando si cancella un file e si crea un altro file con lo stesso nome, prima che sia stato usato un collegamento simbolico al file originario. Nello UNIX, quando si cancella un file, i collegamenti simbolici restano, è l'utente che deve rendersi conto che il file originale è scomparso o è stato sostituito. Nella famiglia di sistemi operativi Microsoft Windows si segue lo stesso criterio.

Un altro criterio prevede la conservazione del file fino a che non siano stati cancellati tutti i riferimenti a esso. In questo caso è necessario disporre di un meccanismo che determini la cancellazione dell'ultimo riferimento a quel file; è possibile tenere un elenco di tutti i riferimenti a un file (elementi di directory o collegamenti simbolici). Quando si crea un collegamento, oppure una copia dell'elemento della directory, si aggiunge un nuovo elemento all'elenco dei riferimenti al file; quando si cancella un collegamento oppure un elemento della directory, si elimina dall'elenco l'elemento corrispondente. Se il suo elenco di riferimenti è vuoto si cancella il file.

Questo metodo presenta, però, un problema: la dimensione dell'elenco dei riferimenti al file può essere variabile e potenzialmente grande. Tuttavia, non è realmente necessario mantenere l'intero l'elenco, è sufficiente un contatore del *numero* di riferimenti. Un nuovo collegamento o un nuovo elemento della directory incrementa il numero dei riferimenti; la cancellazione di un collegamento o un elemento decrementa questo numero. Quando il contatore è uguale a 0 si può cancellare il file, poiché non ci sono più

riferimenti a tale file. Il sistema operativo UNIX usa questo metodo per i collegamenti non simbolici, o **collegamenti effettivi** (*hard link*); il contatore dei riferimenti è tenuto nel descrittore di file (o *inode*, si veda il Paragrafo A.7.2). Impedendo che si facciano più riferimenti a una directory, si può mantenere una struttura a grafo aciclico.

Per evitare questi problemi alcuni sistemi non consentono la condivisione delle directory né i collegamenti. Nell'MS-DOS, ad esempio, la struttura di directory è una struttura ad albero anziché a grafo aciclico.

11.3.5 Directory con struttura a grafo generale

Un serio problema connesso all'uso di una struttura a grafo aciclico consiste nell'assicurare che non vi siano cicli. Iniziando con una directory a due livelli e permettendo agli utenti di creare sottodirectory si crea una directory con struttura ad albero. Aggiungendo nuovi file e nuove sottodirectory alla directory con struttura ad albero, la natura di quest'ultima persiste. Tuttavia, quando si aggiungono dei collegamenti a una directory con struttura ad albero, tale struttura si trasforma in una semplice struttura a grafo, come quella illustrata nella Figura 11.10.

Il vantaggio principale di un grafo aciclico è dato dalla semplicità degli algoritmi necessari per attraversarlo e per determinare quando non ci sono più riferimenti a un file. È preferibile evitare un duplice attraversamento di sezioni condivise di un grafo aciclico, soprattutto per motivi di prestazioni. Se un file particolare è stato appena cercato in una sottodirectory condivisa, ma non è stato trovato, è preferibile evitare una seconda ricerca nella stessa sottodirectory, che costituirebbe solo una perdita di tempo.

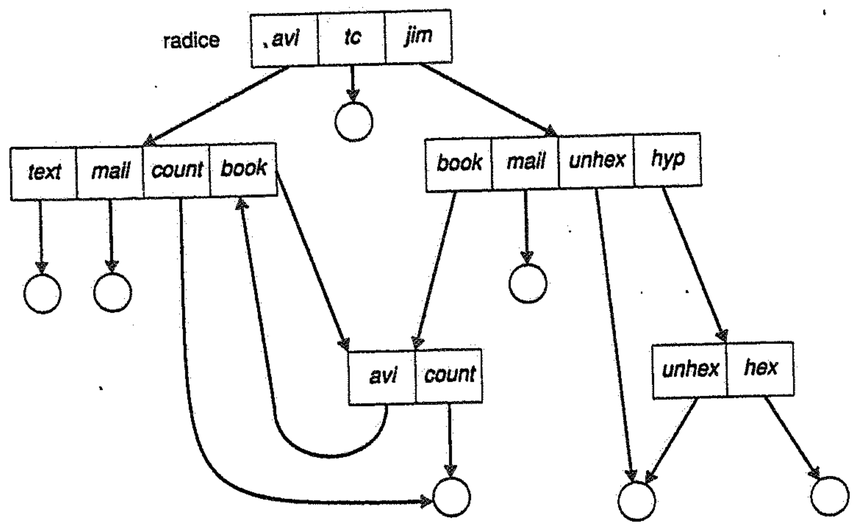


Figura 11.10 Directory a grafo generale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Se si permette che nella directory esistano cicli, è preferibile evitare una duplice ricerca di un elemento, per motivi di correttezza e di prestazioni. Un algoritmo mal progettato potrebbe causare un ciclo infinito di ricerca. Una soluzione è quella di limitare arbitrariamente il numero di directory cui accedere durante una ricerca.

Un problema analogo si presenta al momento di stabilire quando sia possibile cancellare un file. Come con le strutture delle directory a grafo aciclico, la presenza di uno 0 nel contatore dei riferimenti significa che non esistono più riferimenti al file o alla directory, e quindi il file può essere cancellato. Tuttavia, se esistono cicli, è possibile che il contatore dei riferimenti possa essere non nullo, anche se non è più possibile fare riferimento a una directory o a un file. Questa anomalia è dovuta alla possibilità di autoriferimento (ciclo) nella struttura delle directory. In questo caso è generalmente necessario usare un metodo di 'ripulitura' (*garbage collection*) per stabilire quando sia stato cancellato l'ultimo riferimento e quando sia possibile riassegnare lo spazio dei dischi. Tale metodo implica l'attraversamento del file system, durante il quale si contrassegna tutto ciò che è accessibile; in un secondo passaggio si raccoglie in un elenco di blocchi liberi tutto ciò che non è contrassegnato. Una procedura di marcatura analoga si può usare per assicurare che un attraversamento o una ricerca coprano tutto quel che si trova nel file system una sola volta. L'applicazione di questo metodo a un file system basato su dischi richiede però molto tempo, perciò viene tentata solo di rado. Inoltre, poiché è necessaria solo a causa della presenza dei cicli, è molto più conveniente lavorare con una struttura a grafo aciclico.

La difficoltà consiste nell'evitare i cicli quando si aggiungono nuovi collegamenti alla struttura. Per sapere quando un nuovo collegamento ha completato un ciclo si possono impiegare gli algoritmi che permettono di individuare la presenza di cicli nei grafi. Dal punto di vista del calcolo, però, questi algoritmi sono onerosi, soprattutto quando il grafo si trova nella memoria secondaria. Nel caso particolare di directory e collegamenti, un semplice algoritmo prevede di evitare i collegamenti durante l'attraversamento delle directory. Si evitano i cicli senza alcun carico ulteriore.

11.4 Montaggio di un file system

Così come si deve *aprire* un file per poterlo usare, per essere reso accessibile ai processi di un sistema, un file system deve essere *montato*. La struttura delle directory può ad esempio essere composta di più partizioni, che devono essere montate affinché siano disponibili nello spazio dei nomi di un file system.

La procedura di montaggio è molto semplice: si fornisce al sistema operativo il nome del dispositivo e la sua locazione (detta **punto di montaggio**) nella struttura di file e directory alla quale agganciare il file system. Di solito, un punto di montaggio è una directory vuota cui sarà agganciato il file system che deve essere montato. Ad esempio, in

un sistema UNIX, un file system che contiene le directory iniziali degli utenti si potrebbe montare come `/home`; quindi la directory iniziale dell'utente *gianna* avrebbe il percorso `/home/gianna`. Se lo stesso file system si montasse come `/users` il percorso per quella directory sarebbe `/users/gianna`.

Il passo successivo consiste nella verifica da parte del sistema operativo della validità del file system contenuto nel dispositivo. La verifica si compie chiedendo al driver del dispositivo di leggere la directory di dispositivo e controllando che tale directory abbia il formato previsto. Infine, il sistema operativo annota nella sua struttura di directory che un certo file system è montato al punto di montaggio specificato. Questo schema permette al sistema operativo di attraversare la sua struttura di directory, passando da un file system all'altro secondo le necessità.

Per illustrare l'operazione di montaggio, si consideri il file system rappresentato nella Figura 11.11, in cui i triangoli rappresentano sottoalberi di directory rilevanti. La Figura 11.11(a), mostra un file system esistente, mentre nella Figura 11.11(b), è raffigurata una partizione non ancora montata che risiede in `/device/dsk`. A questo punto, si può accedere solo ai file del file system esistente. Nella Figura 11.12, si possono vedere gli effetti dell'operazione di montaggio della partizione residente in `/device/dsk` al punto di montaggio `/users`. Se si smonta la partizione, il file system ritorna alla situazione rappresentata nella Figura 11.11.

I sistemi operativi impongono una semantica a queste operazioni per rendere più chiare le loro funzioni. Ad esempio, un sistema potrebbe vietare il montaggio in una directory che contiene file, o rendere disponibile il file system montato in tale directory e nascondere i file preesistenti nella directory fino a quando non si *smonta* il file system,

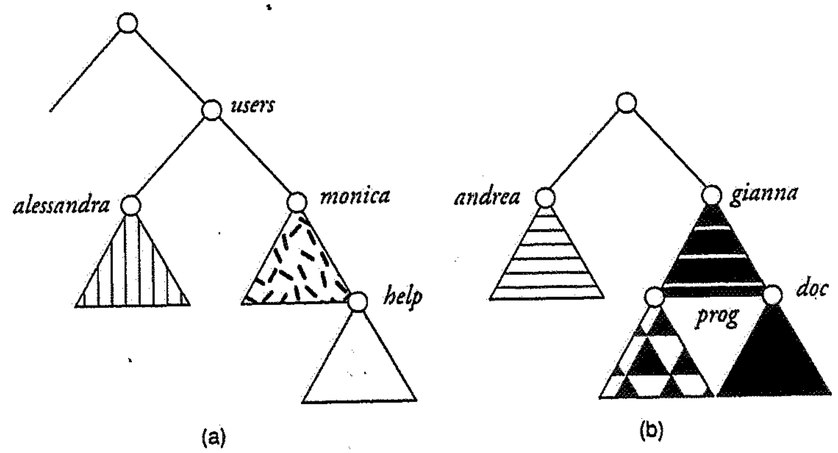


Figura 11.11 File system. (a) Esistente; (b) partizione non montata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

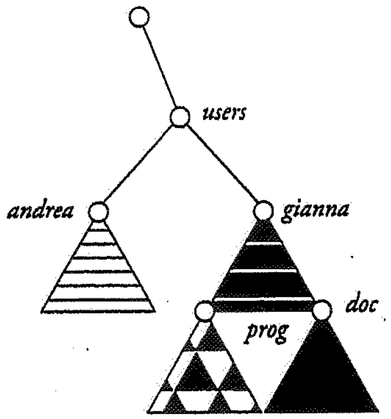


Figura 11.12 Punto di montaggio.

concludendone l'uso e permettendo l'accesso ai file originariamente presenti in tale directory. Come ulteriore esempio, un sistema potrebbe permettere il montaggio ripetuto dello stesso file system in diversi punti di montaggio, o potrebbe imporre una sola possibilità di montaggio per ciascun file system.

Si considerino le azioni del sistema operativo Macintosh. Ogni volta che il sistema rileva per la prima volta un disco (i dischi sono rilevati nella fase d'avviamento, mentre i dischetti sono rilevati quando sono inseriti nella relativa unità), il sistema operativo Macintosh cerca un file system nel dispositivo. Se ne trova uno, esegue automaticamente il montaggio del file system al livello della radice, aggiungendo un'icona di cartella sulla scrivania virtuale etichettata con il nome del file system (secondo quel che è memorizzato nella directory di dispositivo). A questo punto l'utente può selezionare l'icona con il mouse e quindi vedere il contenuto del nuovo file system appena montato.

La famiglia di sistemi operativi Microsoft Windows (95, 98, NT, 2000 e XP) mantiene una struttura di directory a due livelli estesa, con una lettera di unità associata a dispositivi e partizioni. Le partizioni hanno una struttura di directory a grafo generale associata a una lettera di unità. Il percorso completo per uno specifico file è quindi della forma `lettera_di_unità:\percorso\file`. Questi sistemi operativi rilevano automaticamente tutti i dispositivi ed eseguono il montaggio di tutti i file system rilevati nella fase d'avviamento del sistema. In alcuni sistemi, come lo UNIX, i comandi di montaggio sono espliciti. Un file di configurazione di sistema contiene un elenco di dispositivi e di punti di montaggio per il montaggio automatico nella fase d'avviamento, ma le altre operazioni di montaggio di solito si eseguono manualmente.

Altri aspetti del montaggio dei file system sono discussi nei Paragrafi 12.2.2 e A.7.5.

11.5 Condivisione di file

Nei paragrafi precedenti sono presentate le motivazioni alla base della necessità di condivisione dei file ed alcune difficoltà che si incontrano nel permettere che diversi utenti possano condividere file. Tale possibilità è particolarmente utile agli utenti che vogliono collaborare e per ridurre le risorse richieste per raggiungere un certo obiettivo di calcolo. Quindi, nonostante le difficoltà inerenti alla condivisione, i sistemi operativi orientati agli utenti devono soddisfare questa esigenza.

In questo paragrafo si considerano diversi aspetti della condivisione dei file; innanzi tutto l'aspetto relativo alla molteplicità degli utenti e ai diversi metodi di condivisione possibili. Una volta che più utenti possono condividere file, l'obiettivo diventa estendere la condivisione a più file system, compresi i file system remoti. Infine, ci possono essere diverse interpretazioni delle azioni conflittuali intraprese su file condivisi. Ad esempio, se più utenti stanno scrivendo nello stesso file, ci si può chiedere se il sistema dovrebbe permettere tutte le operazioni di scrittura, oppure dovrebbe proteggere le azioni di ciascun utente da quelle degli altri. La semantica della coerenza è discussa nel Paragrafo 11.5.3.

11.5.1 Utenti multipli

Se un sistema operativo permette l'uso del sistema da parte di più utenti, diventano particolarmente rilevanti i problemi relativi alla condivisione dei file, alla loro identificazione tramite nomi e alla loro protezione. Data una struttura di directory che permette la condivisione di file da parte degli utenti, il sistema deve poter mediare questa condivisione. Il sistema può permettere a ogni utente, in modo predefinito, di accedere ai file degli altri utenti, oppure può richiedere che un utente debba esplicitamente concedere i permessi di accesso ai file. Questi aspetti sono alla base dei temi del controllo degli accessi e della protezione, trattati nel seguito.

Per realizzare i meccanismi di condivisione e protezione, il sistema deve memorizzare e gestire più attributi di directory e file rispetto a un sistema che consente un singolo utente. Sebbene storicamente siano stati proposti molti metodi per la realizzazione di questi meccanismi, la maggior parte dei sistemi ha adottato i concetti di *proprietario* (o utente) e *gruppo* relativamente a ciascun file o directory. Il proprietario è l'utente che può cambiare gli attributi di un file o directory, concedere l'accesso e che, in generale, ha il maggior controllo sul file o directory. L'attributo di gruppo di un file si usa per definire il sottoinsieme di utenti autorizzati a condividere l'accesso al file. Ad esempio, il proprietario di un file in un sistema UNIX può fare qualsiasi operazione sul file, mentre i membri del gruppo possono compiere un sottoinsieme di queste operazioni e il resto degli utenti un altro sottoinsieme. Il proprietario del file può definire l'esatto insieme di operazioni che i membri del gruppo e gli altri utenti possono eseguire. Maggiori dettagli sugli attributi che regolano i permessi sono trattati nel Paragrafo 11.6.2.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La maggior parte dei sistemi realizza gli attributi di proprietà gestendo un elenco di nomi di utenti e di **identificatori di utenti** (*user ID*) a essi associati. Nel contesto del sistema operativo Windows NT, questi identificatori si chiamano **identificatori di protezione** (*Security ID* — SID). Questi identificatori numerici sono unici, uno per ciascun utente. Quando un utente si collega al sistema, il modulo di autenticazione determina l'appropriato identificatore d'utente che sarà associato a tutti i processi e thread dell'utente. Nei casi in cui devono essere leggibili dall'utente, sono convertiti nel nome dell'utente usando l'elenco dei nomi degli utenti. Allo stesso modo la funzione concernente il gruppo si può realizzare come un elenco esteso a tutto il sistema di nomi di gruppi e relativi identificatori di gruppi. Ogni utente può appartenere a uno o più gruppi, secondo i criteri scelti nella progettazione di ciascun sistema operativo. Gli identificatori relativi ai gruppi dell'utente sono anch'essi inclusi in ogni processo e thread associato all'utente.

Gli identificatori del gruppo e del proprietario di un certo file o directory sono memorizzati insieme con gli altri attributi del file. Quando un utente richiede di compiere un'operazione su un file, si può confrontare l'identificatore d'utente con l'attributo che identifica il proprietario per verificare se l'utente richiedente è il proprietario del file. Analogamente si confrontano gli identificatori di gruppo. Ne risultano i permessi che l'utente ha sul file, e che il sistema considera per consentire o impedire l'operazione richiesta.

In un processo, l'informazione relativa all'utente si può usare anche per altri scopi. Un processo potrebbe cercare di interagire con un altro processo e l'informazione relativa all'utente potrebbe determinarne il risultato, secondo i criteri di progettazione del sistema operativo. Ad esempio, un processo potrebbe cercare di far terminare, far eseguire in sottofondo, o abbassare la priorità di un altro processo. Se il proprietario di ciascun processo è lo stesso, allora il comando dovrebbe essere ammesso, altrimenti dovrebbe essere respinto. Ma potrebbe anche essere accettato qualora l'utente proprietario del primo processo fosse, ad esempio, l'amministratore del sistema.

Molti sistemi operativi hanno più file system locali, incluse partizioni di un unico disco, o più partizioni in diversi dischi connessi al sistema. In questi casi, la verifica degli identificatori e il confronto dei permessi si possono fare facilmente dopo l'operazione di montaggio dei file system.

11.5.2 File system remoti

L'introduzione delle reti (Capitolo 15) ha permesso la comunicazione tra calcolatori separati da grandi distanze. Le reti permettono la condivisione di risorse sparse nell'area di un *campus* universitario o addirittura in diversi luoghi del mondo. Un'ovvia risorsa da condividere sono i dati, nella forma di file; i metodi con i quali i file si condividono in una rete sono cambiati molto, seguendo l'evoluzione della tecnologia delle reti e dei file. Un primo metodo consiste nel trasferimento dei file richiesto in modo esplicito dagli utenti, attraverso programmi come l'`ftp`. Un secondo metodo, molto diffuso, è quello del **file system distribuito** (*distributed file system* — DFS), che permette la visibilità nel calcolatore locale delle directory remote. Il terzo metodo, il **World Wide Web**, è, da un

certo punto di vista, il contrario del primo metodo. Si usa un programma di consultazione per accedere ai file remoti e operazioni distinte — essenzialmente un involucro (*wrapper*) per l'`ftp` — per trasferirli.

L'`ftp` si usa sia per l'accesso anonimo sia per quello autenticato. L'**accesso anonimo** permette di trasferire file senza essere utenti accreditati nel sistema remoto. Il World Wide Web usa quasi esclusivamente lo scambio di file anonimo. Un DFS comporta un'integrazione molto più stretta tra il calcolatore che accede ai file remoti e il calcolatore che fornisce i file.

11.5.2.1 Modello client-server

I file system remoti permettono il montaggio di uno o più file system di uno o più calcolatori remoti in un calcolatore locale. In questo caso, il calcolatore che contiene i file si chiama *server*, mentre il calcolatore che richiede l'accesso ai file si chiama *client*. La relazione tra client e server è piuttosto comune tra i calcolatori di una rete. In generale, il server dichiara che determinate risorse sono disponibili ai client, specificando esattamente quali risorse (in questo caso, quali file) ed esattamente a quali client. I file di solito si specificano rispetto a una partizione o a un livello di directory. Un server può gestire richieste provenienti da più client, e un client può accedere a più server, secondo i dettagli del particolare sistema client-server.

L'identificazione certa dei client è più difficile; proprio perché può avvenire facilmente tramite i relativi nomi simbolici di rete, o tramite altri identificatori, si può altrettanto facilmente **ingannare** un server imitando l'identificatore di un client accreditato (si tratta del cosiddetto *spoofing*). Un client non accreditato o privo di determinate autorizzazioni può ingannare un server presentandosi con l'identità di un altro client che possiede le autorizzazioni che il client impostore vuole ottenere. Tra le soluzioni più sicure ci sono quelle che prevedono l'autenticazione reciproca dei client e dei server tramite chiavi di cifratura. Sfortunatamente, l'introduzione di tecniche per la sicurezza introduce nuovi problemi, ad esempio la necessità della compatibilità tra client e server (si devono impiegare gli stessi algoritmi di cifratura) e dello scambio sicuro delle chiavi di cifratura (l'intercettazione delle chiavi può permettere accessi non autorizzati). Questi problemi sono sufficientemente difficili da far sì che nella maggioranza dei casi si usino metodi di autenticazione insicuri. Nel caso dello UNIX e del suo file system di rete (*network file system* — NFS), l'autenticazione avviene, ordinariamente, tramite le informazioni di connessione relative al client. In questo schema, gli identificatori che identificano l'utente devono coincidere nel client e nel server; diversamente, il server non può determinare i diritti d'accesso ai file.

Si consideri ad esempio un utente con un identificatore uguale a 1000 nel client e a 2000 nel server. Una richiesta per uno specifico file dal client al server, non potrà essere gestita correttamente, perché il server cercherà di determinare se l'utente 1000 ha i permessi d'accesso al file, invece di usare il *reale* identificatore dell'utente che è 2000. L'accesso sarà concesso o negato secondo un'informazione di autenticazione sbagliata. Il server deve fidarsi del client e assumere che quest'ultimo gli presenti l'identificatore corretto. I protocolli NFS permettono relazioni da molti a molti; cioè più server possono for-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nire file a più client. Infatti, un calcolatore può comportarsi sia da server per altri client NFS, sia da client di altri server NFS.

Una volta che il file system remoto è stato montato, le richieste delle operazioni su file sono inviate al server, attraverso la rete, per conto dell'utente, usando il protocollo DFS. Normalmente, una richiesta di apertura di file si invia insieme con l'identificatore dell'utente richiedente. Il server quindi applica i normali controlli d'accesso per determinare se l'utente ha le credenziali per accedere al file nel modo richiesto e se tali controlli hanno esito positivo, riporta una 'maniglia' d'accesso al file (*file handle*) all'applicazione client, che la usa per eseguire sul file operazioni di lettura, scrittura e altro. Il client chiude il file quando non deve più accedervi. Il sistema operativo può applicare una semantica simile a quella adottata per il montaggio di un file system locale, oppure una semantica diversa.

11.5.2.2 Sistemi informativi distribuiti

Per semplificare la gestione dei servizi client-server, i **sistemi informativi distribuiti**, noti anche come **servizi di nominazione distribuiti**, sono stati concepiti per fornire un accesso unificato alle informazioni necessarie per il calcolo remoto. Il **sistema dei nomi di dominio** (*domain name system* — DNS) fornisce le traduzioni dai nomi dei calcolatori agli indirizzi di rete per l'intera Internet (compreso il World Wide Web). Prima che s'inventasse il DNS e che si diffondesse capillarmente nella rete, si scambiavano tra i calcolatori, per posta elettronica o `ftp`, file contenenti le stesse informazioni. Questo metodo non poteva adattarsi dinamicamente all'aumento delle dimensioni della rete Internet. Il DNS è trattato ulteriormente nel Paragrafo 15.4.1.

Altri sistemi informativi distribuiti forniscono uno spazio identificato da *nome d'utente*/*parola d'ordine*/*identificatore d'utente*/*identificatore di gruppo* per un servizio distribuito. I sistemi UNIX hanno adottato un'ampia varietà di metodi per l'informazione distribuita. La Sun Microsystems ha introdotto il sistema *yellow pages* (poi ribattezzato *network information service* — NIS), adottato da una gran parte dell'industria. Questo servizio centralizza la memorizzazione dei nomi degli utenti e dei calcolatori, delle informazioni sulle stampanti e altro. Sfortunatamente, usa metodi di autenticazione insicuri, ad esempio l'invio di parole d'ordine d'utente non cifrate (*in chiaro*) e l'identificazione dei calcolatori attraverso gli indirizzi IP. Il sistema NIS+ della Sun è una versione del NIS più sicura, ma è anche molto più complessa e non ha avuto una grande diffusione.

Nel caso delle reti della Microsoft (CIFS), le informazioni di rete si usano insieme con gli elementi di autenticazione dell'utente (nome dell'utente e parola d'ordine) per creare un **nome d'utente di rete** (*network login*) che il server usa per decidere se permettere o negare l'accesso a un file system richiesto. Affinché questa autenticazione sia valida, i nomi d'utente devono essere uguali nel calcolatore client e nel server (come per l'NFS). La Microsoft usa due strutture di nominazione distribuita per fornire un unico spazio di nomi per gli utenti: i **domìni** costituiscono la vecchia tecnologia di nominazione; la nuova tecnologia, disponibile nel sistema operativo Windows 2000 e successivi, è l'**active directory**. Una volta impostata, la funzione di nominazione è usata per autenticare gli utenti da tutti i client e da tutti i server.

L'industria si sta orientando verso il protocollo LDAP (*lightweight directory-access protocol*) come meccanismo sicuro per la nominazione distribuita. Lo stesso *active directory* è basato sull'LDAP. Il sistema Solaris 8 della Sun Microsystems permette l'uso del protocollo LDAP per l'autenticazione degli utenti e per altri servizi di ricerca di informazioni al livello dell'intero sistema, ad esempio tutte le stampanti disponibili. Se la convergenza sull'uso dell'LDAP avrà successo, allora un'organizzazione potrà usare una singola directory LDAP distribuita per memorizzare le informazioni su tutti gli utenti e le risorse di tutti i calcolatori dell'organizzazione stessa. Si avrebbe un **unico punto d'accesso sicuro** per gli utenti, che inserirebbero una sola volta le proprie informazioni di autenticazione per avere accesso a tutti i calcolatori dell'organizzazione. Questa soluzione semplificherebbe anche i compiti degli amministratori di sistema, concentrando in un unico punto informazioni che sono ora sparse in vari file in ciascun sistema o in diversi servizi di informazione distribuiti.

11.5.2.3 Malfunzionamenti

I file system locali possono presentare malfunzionamenti per varie cause: problemi dei dischi che li contengono, alterazione dei dati relativi alle strutture delle directory o a informazioni necessarie alla gestione dei dischi (chiamate collettivamente **metadati**), malfunzionamenti dei controllori dei dischi, problemi ai cavi di connessione, o agli adattatori. Anche il comportamento involontario degli utenti o dell'amministratore di sistema può causare la perdita di file, d'intere directory o addirittura la cancellazione di partizioni. Molte di queste cause di malfunzionamento portano al crollo del sistema (*crash*), all'emissione di una condizione di errore e alla necessità di un intervento umano per risolvere il problema.

Alcuni malfunzionamenti non causano la perdita di dati o la perdita della disponibilità dei dati. L'impiego di **batterie ridondanti di dischi** (*redundant arrays of inexpensive* [o *independent*] *disks* — RAID), basata sull'uso di più dischi secondo un'opportuna strategia di distribuzione delle informazioni, può prevenire la perdita di dati nel caso del malfunzionamento di uno dei dischi (Paragrafo 14.5).

L'uso di file system remoti implica maggiori possibilità di malfunzionamenti; a causa della complessità dei sistemi di rete e della necessità di interazioni tra calcolatori remoti, i problemi che possono interferire con il corretto funzionamento dei file system remoti sono infatti molto più numerosi. Nel caso delle reti, si possano verificare interruzioni del collegamento tra due calcolatori, dovute a malfunzionamenti o a improprie configurazioni dell'architettura, oppure a questioni relative alla realizzazione degli aspetti di rete in uno dei calcolatori coinvolti. Sebbene alcune reti includano meccanismi di tolleranza ai guasti, compresi cammini multipli tra ogni coppia di calcolatori, altre non li prevedono. Qualsiasi malfunzionamento potrebbe interrompere il flusso dei comandi del DFS.

Si consideri un client mentre usa un file system remoto. Il client ha il file system remoto montato nel proprio file system e potrebbe avere qualche file remoto aperto; tra le varie attività potrebbe richiedere elenchi dei file nelle directory remote per aprire quelli necessari, svolgere operazioni di lettura e scrittura e chiudere i file. Si consideri ora un

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

malfunzionamento della rete, un crollo del server remoto, oppure anche uno spegnimento programmato di quel server, tali da determinare l'inaccessibilità del file system remoto. Questo scenario è piuttosto comune, quindi il client non dovrebbe comportarsi come nel caso di una perdita del file system locale. Piuttosto, il sistema dovrebbe terminare tutte le operazioni sul server non più raggiungibile, oppure posticiparle fino a quando il server sarà nuovamente disponibile. Questa semantica di trattamento dei malfunzionamenti si definisce e si realizza come parte del protocollo di un file system remoto. La terminazione di tutte le operazioni può portare alla perdita di dati (e della pazienza) da parte degli utenti. La maggior parte dei protocolli DFS impone o permette la posticipazione delle operazioni sul file system remoto, con la speranza che il calcolatore remoto diventi nuovamente disponibile in breve tempo.

Per questo tipo di recupero dai malfunzionamenti, è necessario mantenere alcune informazioni di *stato* sia sui client sia sui server. Nel caso di un server malfunzionante, ma che deve comunque riconoscere che ha esportato file system che sono stati montati in sistemi remoti, e che alcuni suoi file sono stati aperti, l'NFS segue un criterio semplice realizzando un DFS **senza stato**. Sostanzialmente, assume che una richiesta di un client per la lettura o scrittura di un file non sia avvenuta, sempre che il file system non sia stato montato in modo remoto e il file in questione aperto prima della richiesta. Il protocollo NFS trasferisce tutte le informazioni necessarie per localizzare il file appropriato e per svolgere l'operazione richiesta sul file. Allo stesso modo, non tiene traccia di quali client hanno montato le sue partizioni esportate, assumendo anche in questo caso che se perviene una richiesta, deve essere legittima. Sebbene questo metodo senza stato renda l'NFS tollerante i guasti e sia piuttosto facile da realizzare, lo rende insicuro. Un server NFS potrebbe ad esempio permettere richieste contraffatte di lettura o scrittura da un client non autorizzato anche se la necessaria operazione di montaggio e il controllo dei permessi richiesti non sono avvenuti.

11.5.3 Semantica della coerenza

La **semantica della coerenza** è un importante criterio per la valutazione di qualsiasi file system che consenta la condivisione dei file. Si tratta di una caratterizzazione del sistema che specifica la semantica delle operazioni in cui più utenti accedono contemporaneamente a un file condiviso. In particolare, questa semantica deve specificare quando le modifiche ai dati apportate da un utente possano essere osservate da altri utenti. La semantica è tipicamente realizzata come codice facente parte del codice del file system.

La semantica della coerenza è direttamente correlata agli algoritmi di sincronizzazione dei processi del Capitolo 7. Tuttavia, i complessi algoritmi descritti in tale capitolo di solito non s'impiegano per l'I/O su file a causa delle lunghe latenze e delle basse velocità di trasferimento dei dischi e delle reti. Ad esempio, l'esecuzione di una transazione atomica su dischi remoti può coinvolgere molte comunicazioni di rete e molte letture e scritture nei dischi. I sistemi che tentano una così completa serie di funzioni tendono ad avere scarse prestazioni. Una realizzazione riuscita di semantica della condivisione si trova nel file system del sistema Andrew (Paragrafo 16.6).

Nella discussione che segue si suppone che una serie d'accessi, cioè letture e scritture, tentati da un utente allo stesso file sia sempre compresa tra una coppia di operazioni `open` e `close`. Tale serie d'accessi si chiama **sessione di file**. Per illustrare questo concetto si descrivono sommariamente qui di seguito alcuni esempi di semantica della coerenza.

11.5.4 Semantica UNIX

Il file system dello UNIX, descritto nel Capitolo 16, usa la seguente semantica della coerenza:

- Le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno aperto contemporaneamente lo stesso file.
- Esiste un metodo di condivisione in cui gli utenti condividono il puntatore alla locazione corrente nel file, quindi l'avanzamento del puntatore da parte di un utente influisce su tutti gli utenti che condividono il file. In questo caso il file ha una singola immagine e tutti gli accessi si alternano (intercalandosi) a prescindere dalla loro origine.

Un file è associato a una singola immagine fisica alla quale si può accedere come a una risorsa esclusiva. La contesa di quest'immagine singola determina il differimento dei processi utenti.

11.5.5 Semantica delle sessioni

Il file system Andrew (*Andrew file system* — AFS), descritto nel Paragrafo 16.6, usa la seguente semantica della coerenza:

- le scritture in un file aperto da un utente non sono visibili immediatamente ad altri utenti che hanno aperto contemporaneamente lo stesso file;
- una volta chiuso il file, le modifiche apportate sono visibili solo nelle sessioni che iniziano successivamente. Le istanze del file già aperte non riportano queste modifiche.

Secondo questa semantica, un file può essere temporaneamente associato a parecchie immagini, probabilmente diverse. Di conseguenza, più utenti possono eseguire accessi concorrenti di lettura o scrittura sulla rispettiva immagine del file senza subire ritardi. Non si impone quasi alcun vincolo nella gestione degli accessi.

11.5.6 Semantica dei file condivisi immutabili

Un altro metodo è quello dei **file condivisi immutabili**; una volta che un file è stato dichiarato *condiviso* dal suo creatore, non può essere modificato. Un file immutabile presenta due caratteristiche chiave: il suo nome non si può riutilizzare e il suo contenuto non può essere modificato. Quindi, il nome di un file immutabile designa il contenuto fissato del file, piuttosto che un file come contenitore di informazioni variabili. Come si descrive nel Capitolo 16, la realizzazione di questa semantica in un sistema distribuito è semplice; infatti la condivisione è molto disciplinata poiché consente la sola lettura.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

11.6 Protezione

La salvaguardia delle informazioni contenute in un sistema di calcolo dai danni fisici (*affidabilità*) e da accessi impropri (*protezione*) è fondamentale.

Generalmente l'affidabilità è assicurata da più copie dei file. Molti calcolatori hanno programmi di sistema che copiano i file dai dischi ai nastri a intervalli regolari, ad esempio una volta al giorno, alla settimana o al mese; quest'operazione di copiatura può essere automatica, o controllata dall'intervento di un operatore. Lo scopo è quello di conservare copie di riserva utili nei casi in cui il file system andasse distrutto a causa di problemi dei dispositivi: errori di lettura o scrittura, cali o cadute della tensione elettrica, rotture delle testine, sporcizia, temperature estreme, atti vandalici, ecc. I file possono inoltre essere cancellati accidentalmente, e anche errori di programmazione possono causare la perdita del contenuto dei file. L'affidabilità è trattata con maggiori dettagli nel Capitolo 14.

La protezione si può ottenere in molti modi. Per un piccolo sistema monoutente, la protezione si ottiene rimovendo fisicamente i dischetti e chiudendoli in un cassetto della scrivania oppure in un armadietto. In un sistema multiutente sono necessari altri metodi.

11.6.1 Tipi d'accesso

La necessità di proteggere i file deriva direttamente dalla possibilità di accedervi. I sistemi che non permettono l'accesso ai file di altri utenti non richiedono protezione; quindi si può ottenere una completa protezione proibendo l'accesso. In alternativa si può permettere un accesso totalmente libero senza alcuna protezione. Questi orientamenti sono entrambi eccessivi, quindi non si possono applicare in generale; ciò che serve in realtà è un **accesso controllato**.

Il controllo offerto dai meccanismi di protezione si ottiene limitando i possibili tipi d'accesso. Gli accessi si permettono o si negano secondo diversi fattori, innanzi tutto i tipi d'accesso richiesti. Si possono controllare diversi tipi di operazioni:

- **Lettura.** Lettura da file.
- **Scrittura.** Scrittura o riscrittura di file.
- **Esecuzione.** Caricamento di file nella memoria ed esecuzione.
- **Aggiunta.** Scrittura di nuove informazioni in coda ai file.
- **Cancellazione.** Cancellazione di file e liberazione del relativo spazio per un possibile riutilizzo.
- **Elencazione.** Elencazione del nome e degli attributi dei file.

Si possono controllare anche altre operazioni, come la ridenominazione, la copiatura o la modifica dei file. Tuttavia, in molti sistemi queste funzioni di livello superiore si possono realizzare tramite un programma di sistema che compie alcune chiamate del sistema di livello inferiore, quindi è sufficiente garantire la protezione al livello inferiore. Ad esempio, la copiatura di un file si può realizzare semplicemente con una sequenza di richieste di lettura; in questo caso un utente con accesso per la lettura di un file può richiederne la copiatura, la stampa o altro.

Sono stati proposti molti meccanismi di protezione. Come sempre, ogni meccanismo presenta vantaggi e svantaggi, e deve essere appropriato alla particolare applicazione che richiede la protezione. Un piccolo calcolatore usato soltanto da pochi membri di un gruppo di ricerca non richiede la stessa protezione del sistema di calcolo di una grande società, usato per operazioni di ricerca, finanza e per il lavoro del personale. Il problema della protezione è trattato nel Capitolo 18.

11.6.2 Controllo degli accessi

Il problema della protezione comunemente si affronta rendendo l'accesso dipendente dall'identità dell'utente. Più utenti possono richiedere diversi tipi d'accesso a un file o a una directory. Lo schema più generale per realizzare gli accessi dipendenti dall'identità consiste nell'associare un **elenco di controllo degli accessi** (*access-control list* — ACL) a ogni file e directory; in tale elenco sono specificati i nomi degli utenti e i relativi tipi d'accesso consentiti. Quando un utente richiede un accesso a un file specifico il sistema operativo esamina l'elenco di controllo degli accessi associato a quel file; se tale utente è presente nell'elenco si autorizza l'accesso, altrimenti si verifica una violazione della protezione e si nega l'accesso al file.

Questo sistema ha il vantaggio di permettere complessi metodi d'accesso. Il problema maggiore degli elenchi di controllo degli accessi è la loro lunghezza: per permettere a tutti di leggere un file, l'elenco deve contenere tutti gli utenti con accesso per la lettura. Questa tecnica comporta due inconvenienti:

- la costruzione di un elenco di questo tipo può essere un compito noioso e non gratificante, soprattutto se l'elenco degli utenti del sistema non è già noto;
- l'elemento della directory, precedentemente di dimensione fissa, richiede di essere di dimensione variabile, quindi anche la gestione dello spazio è più complicata.

Questi problemi si possono risolvere introducendo una versione condensata dell'elenco di controllo degli accessi.

Per condensare la lunghezza dell'elenco, molti sistemi raggruppano gli utenti di ogni file in tre classi distinte:

- **Proprietario.** È l'utente che ha creato il file.
- **Gruppo.** O gruppo di lavoro, si tratta di un insieme di utenti che condividono il file e richiedono tipi di accesso simili.
- **Universo.** Tutti gli altri utenti del sistema.

Il più comune orientamento recente prevede la combinazione degli elenchi di controllo degli accessi con lo schema di controllo degli accessi per proprietario, gruppo e universo (più facile da realizzare). Il sistema operativo Solaris 2.6 (e le sue versioni successive) impiega le tre categorie d'accesso in modo predefinito ma, se si vuole una maggiore selettività del controllo degli accessi, permette l'attribuzione di elenchi di controllo degli accessi a specifici file e directory.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Si consideri, ad esempio, una persona, Donatella, che sta scrivendo un nuovo libro. Donatella ha assunto tre studenti, Giulia, Paolo e Carlo, per aiutarla a lavorare al progetto. Il testo del libro è tenuto in un file chiamato `libro`. La protezione associata a tale file prevede quel che segue:

- Donatella può compiere tutte le operazioni sul file;
- Giulia, Paolo e Carlo possono solo leggere e scrivere il file ma non possono cancellarlo;
- tutti gli altri utenti possono leggere, ma non scrivere, il file (Donatella ha interesse che il libro sia letto dal maggior numero possibile di persone, in modo da ottenere pareri competenti).

Per ottenere tale protezione si deve creare un nuovo gruppo composto da Giulia, Paolo e Carlo, e si deve associare il nome del gruppo, ad esempio `testo`, al file `libro` con i diritti d'accesso conformi al criterio descritto.

Si consideri un ospite cui Donatella vorrebbe concedere un accesso temporaneo al Capitolo 1. L'ospite non si può aggregare al gruppo `testo` poiché ciò gli darebbe accesso a tutti i capitoli, e considerato che i file possono essere in un solo gruppo, non si può associare un altro gruppo al Capitolo 1. L'aggiunta della funzione degli elenchi di controllo degli accessi, permette di inserire l'ospite nell'elenco di controllo degli accessi del Capitolo 1.

Affinché questo schema funzioni correttamente, è necessario uno stretto controllo dei permessi e degli elenchi di controllo degli accessi, che si può fare in diversi modi. Nel sistema UNIX ad esempio solo un utente con compiti di gestione (o un *superuser*) può creare e modificare i gruppi, quindi questo controllo si ottiene con la partecipazione umana. Nel sistema VMS, il proprietario del file può creare e modificare tali elenchi. Gli elenchi di controllo degli accessi sono trattati anche nel Paragrafo 18.4.2.

Per definire la protezione, data questa più limitata classificazione, occorrono solo tre campi. Ogni campo è formato di un insieme di bit, ciascuno dei quali permette o impedisce l'accesso che gli è associato. Nel sistema UNIX, ad esempio, sono definiti tre campi di tre bit ciascuno: `rwx`, dove `r` controlla l'accesso per la lettura, `w` quello per la scrittura e `x` per l'esecuzione. Un campo distinto è riservato al proprietario del file, al gruppo proprietario e a tutti gli altri utenti. In questo schema, per registrare le informazioni di protezione sono necessari nove bit per file. Così, nell'esempio, i campi di protezione per il file `libro` s'impostano come segue: per Donatella, il proprietario, tutti e tre i bit; per il gruppo `testo` i bit `r` e `w`; per l'universo il solo bit `r`.

Nel combinare i due metodi si presenta una difficoltà nell'interfaccia d'utente; gli utenti devono poter indicare l'impostazione opzionale dei permessi nell'elenco di controllo degli accessi per un file. Nell'esempio del Solaris, un segno + aggiunge un permesso d'accesso:

```
19 -rw-r--r--+ 1 alessandra gruppo 130 May 25 22:13 file1
```

Una specifica serie di comandi `setfacl` e `getfacl` si usa per gestire gli elenchi di controllo degli accessi. Un'altra difficoltà s'incontra nell'assegnazione delle precedenze quando ci

sono conflitti tra i permessi e gli elenchi di controllo degli accessi. Ad esempio, se Andrea è in un gruppo di file, che ha il permesso di lettura, ma il file ha un elenco di controllo degli accessi che contiene i permessi di lettura e scrittura per Andrea, si pone il problema della concessione del permesso di scrittura. Nel sistema operativo Solaris si attribuiscono i permessi contenuti negli elenchi di controllo degli accessi; sono più selettivi e non sono predefiniti. Si segue il principio che la maggiore specificità deve essere prioritaria.

11.6.3 Altri metodi di protezione

Un altro metodo di protezione consiste nell'associazione di una parola d'ordine a ciascun file. Proprio come l'accesso al sistema di calcolo è spesso controllato da una parola d'ordine, anche l'accesso a ogni file può avere lo stesso tipo di protezione. Se le parole d'ordine sono scelte a caso e si cambiano spesso, questo schema può essere efficace, poiché limita l'accesso a un file agli utenti che conoscono la parola d'ordine.

Questo schema ha comunque diversi svantaggi: innanzi tutto, il numero di parole d'ordine da ricordare può diventare molto alto, rendendo tale metodo impraticabile; secondariamente, se si impiega la stessa parola d'ordine per tutti i file, la sua scoperta li rende tutti accessibili. Per risolvere questo problema alcuni sistemi, ad esempio il TOPS-20, permettono a un utente di associare una parola d'ordine a una directory anziché a un singolo file. Il sistema operativo IBM VM/CMS consente di associare tre parole d'ordine ai minidischi: una per la lettura, una per la scrittura, e una per gli accessi multipli per la scrittura. Inoltre, di solito, a tutti i file di un utente è associata una sola parola d'ordine, quindi la protezione è basata sul principio del 'o tutto o niente'. Per offrire protezione a un livello più dettagliato occorre usare più parole d'ordine.

Una limitata protezione dei file è disponibile anche nei sistemi operativi monoutente come l'MS-DOS e le vecchie versioni del Macintosh. Questi sistemi operativi furono progettati ignorando i problemi relativi alla protezione, ma il collegamento in rete di questi sistemi e le relative funzioni di condivisione e comunicazione hanno determinato la necessità di meccanismi di protezione.

Progettare una caratteristica di un nuovo sistema operativo è quasi sempre più facile che aggiungerla a uno esistente. Inoltre, tali aggiornamenti sono di solito meno efficaci e si integrano meno bene nel sistema.

In una struttura di directory a più livelli è necessario proteggere non solo i singoli file, ma anche gruppi di file contenuti in directory; è cioè necessario disporre di un meccanismo per la protezione delle directory. Le operazioni riguardanti le directory che devono essere protette sono piuttosto diverse dalle operazioni sui file. Esse sono la creazione e la cancellazione dei file in una directory; probabilmente va anche controllata la possibilità, per un utente, di determinare l'esistenza di un file in una directory. Talvolta la conoscenza dell'esistenza di un file e del suo nome può essere di per sé significativa, perciò l'elencazione del contenuto di una directory dev'essere un'operazione protetta. Quindi, affinché un nome di percorso possa fare riferimento a un file in una certa directory, all'utente deve essere consentito l'accesso sia al file sia alla directory. Nei sistemi dove i file possono avere numerosi nomi di percorso (come quelli con struttura a grafo aciclico o a grafo generale) un certo utente può avere diversi diritti d'accesso secondo il nome di percorso di cui fa uso.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

11.6.4 Un esempio: UNIX

Nel sistema UNIX la protezione delle directory è gestita in modo simile alla protezione dei file. In altre parole, a ciascuna directory sono associati tre campi (relativi al proprietario, al gruppo e all'universo), ciascuno composto dei tre bit `rwx`. Quindi, un utente può elencare il contenuto di una directory solamente se il bit `r` è asserito nel campo appropriato. Analogamente, un utente può modificare la propria directory corrente in un altra directory (ad esempio `foo`) solo se è asserito, nel campo appropriato, il bit `x` associato a `foo`.

Nella Figura 11.13 è illustrato un esempio di elenco del contenuto di una directory nell'ambiente UNIX. Il primo campo descrive le protezioni di file e directory, il carattere `d` presente all'inizio del campo contraddistingue le directory; inoltre, l'elenco contiene il numero di collegamenti relativi al file, il nome del proprietario e del gruppo, la dimensione del file in byte, la data di creazione e infine il nome del file (con l'eventuale estensione).

11.7 Sommario

Un file è un tipo di dati astratto definito e realizzato dal sistema operativo. È una sequenza di elementi logici (o *record*), ciascuno dei quali può essere un byte, una riga di lunghezza fissa o variabile, oppure un elemento di dati più complesso. Il sistema operativo può gestire in modo specifico diversi tipi di elementi logici o può lasciare tale gestione al programma d'applicazione.

Il compito più importante del sistema operativo consiste nell'associare i file ai dispositivi fisici di memorizzazione, ad esempio dischi o nastri magnetici. Poiché le dimensioni dei blocchi fisici dei dispositivi possono non coincidere con quelle dei record logici, può essere necessario riunire un certo numero di record logici in modo da ottenere una dimensione pari a quella di un blocco fisico. Anche questo compito può essere gestito dal sistema operativo, oppure lasciato al programma d'applicazione.

Ciascun dispositivo di un file system conserva una tabella dei contenuti del volume o una directory di dispositivo che elenca la locazione dei file presenti nel dispositivo. Inoltre, è utile creare directory per permettere di organizzare i file. Una directory a livello singolo in un sistema multiutente causa problemi di nominazione, poiché ogni file deve avere un nome unico. Una directory a due livelli limita questo problema all'ambito relativo a ciascun utente creando una directory distinta per ciascun utente; ciascun utente dispone di una directory che contiene i suoi file. Le directory contengono l'elenco dei file che le costituiscono, insieme con informazioni a essi associate: locazione nei dischi, lunghezza, tipo, ora di creazione, ora dell'ultimo uso, e così via.

La naturale generalizzazione del concetto di directory a due livelli è la directory con struttura ad albero. Tale tipo di struttura permette a un utente di creare sottodirectory in cui organizzare i file. Le strutture di directory a grafo aciclico permettono la condivisione di sottodirectory e file, ma complicano le funzioni di ricerca e cancellazione. Una struttura a grafo generale permette la massima flessibilità nella condivisione dei file e delle directory, ma talvolta richiede operazioni di 'ripulitura' (*garbage collection*) per recuperare lo spazio inutilizzato nei dischi.


```
-rw-rw-r--   1 pbg  staff      31200  set   3 08:30   intro.ps
drwx------   5 pbg  staff        512  lug   8 09:33   privato/
drwxrwxr-x   2 pbg  staff        512  lug   8 09:35   doc/
drwxrwx--    2 pbg  studente     512  ago   3 14:13   studente-prog/
-rw-r--r--   1 pbg  staff       9423  feb  24  1993   program.c
-rwxr-xr-x   1 pbg  staff      20471  feb  24  1993   program
drwx--x--x   4 pbg  facoltà      512  lug  31 10:31   lib/
drwx------   3 pbg  staff       1024  ago  29 06:52   mail/
drwxrwxrwx   3 pbg  staff        512  lug   8 09:35   test/
```

Figura 11.13 Esempio di elenco del contenuto di una directory.

I dischi sono suddivisi in una o più partizioni, ciascuna contenente un file system o priva di struttura (*'raw' partition*). Per essere resi disponibili, i file system si possono montare nelle strutture di nominazione del sistema. Lo schema di nominazione varia da sistema a sistema. Una volta che una partizione è stata montata, i file in essa contenuti sono disponibili per l'uso. I file system si possono smontare per disabilitarne l'accesso o per attività di manutenzione.

La condivisione dei file dipende dalla semantica definita dal sistema. I file possono avere più lettori, più scrittori, o limiti alla condivisione. I file system distribuiti consentono ai sistemi client di montare partizioni o directory di server, a mano a mano che vi accedono tramite una rete. I file system remoti pongono delle sfide in termini di affidabilità, prestazioni e sicurezza. I sistemi informativi distribuiti mantengono informazioni su utenti, calcolatori e accessi in modo che i client e i server condividano le informazioni di stato per la gestione dell'uso e degli accessi.

Poiché i file rappresentano il principale meccanismo di memorizzazione delle informazioni, è necessario che siano dotati di un sistema di protezione. Ogni tipo d'accesso ai file si può controllare separatamente: lettura, scrittura, esecuzione, aggiunta, elencazione del contenuto di directory, e così via. La protezione dei file si può ottenere con parole d'ordine, liste d'accesso, oppure tecniche appositamente predisposte per le situazioni specifiche.

11.8 Esercizi

11.1 Considerate un file system in cui si può cancellare un file e reclamare il suo spazio di memoria secondaria mentre esistono ancora collegamenti (*link*) a esso. Dite quale problema si può presentare se si crea un nuovo file nella stessa area di memoria o con lo stesso nome di percorso assoluto. Spiegate come si possono evitare tali problemi.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

11.2 Alcuni sistemi cancellano automaticamente tutti i file d'utente quando un utente termina una sessione di lavoro o un lavoro d'elaborazione, sempre che l'utente non richieda esplicitamente che i file siano mantenuti. Altri sistemi mantengono tutti i file, sempre che l'utente non li cancelli esplicitamente. Discutete i vantaggi di ciascuno di questi due criteri.

11.3 Spiegate perché alcuni sistemi tengono conto del tipo dei file, mentre altri lasciano agli utenti tale compito o semplicemente non prevedono più tipi di file; e spiegate qual è il sistema migliore.

11.4 Alcuni sistemi gestiscono molti tipi di strutture per i dati in un file, mentre altri gestiscono semplicemente un flusso di byte. Discutete i vantaggi e gli svantaggi dei due schemi.

11.5 Spiegate quali sono i vantaggi e gli svantaggi della registrazione, tra gli attributi di un file, del nome del programma che crea il file stesso, come avviene nel sistema operativo Macintosh.

11.6 Dite se è possibile simulare una struttura di directory a più livelli con una struttura di directory a livello singolo nella quale si possono usare nomi di lunghezza arbitraria. Nel caso affermativo spiegate come ciò sia possibile e confrontate questo schema con quello della directory a più livelli. Nel caso contrario, spiegate che cosa impedisce il successo di questa simulazione. Dite in che modo cambierebbe la risposta se la lunghezza dei nomi dei file fosse limitata a sette caratteri.

11.7 Spiegate lo scopo delle operazioni `open` e `close`.

11.8 Alcuni sistemi aprono automaticamente un file al primo riferimento e lo chiudono al termine del lavoro. Discutete i vantaggi e gli svantaggi di questo schema, confrontandolo con quello più tradizionale nel quale l'utente deve aprire e chiudere esplicitamente il file.

11.9 Fornite un esempio di applicazione in cui è necessario accedere ai dati di un file nel seguente ordine:

a) sequenziale;
b) diretto.

11.10 Alcuni sistemi consentono la condivisione dei file usando una singola copia di ogni file; altri sistemi impiegano più copie, una per ciascun utente che condivide il file. Discutete i vantaggi di ciascun metodo.

11.11 In alcuni sistemi una directory può essere letta e scritta da un utente autorizzato, proprio come accade con i file.

a) Descrivete i problemi di protezione che possono insorgere.
b) Suggerite uno schema per la gestione di ciascuno dei problemi di protezione impliciti nella risposta a).

11.12 Considerate un sistema con 5000 utenti. Supponete di voler permettere a 4990 di questi utenti di accedere a un file.

a) Spiegate come si può specificare questo schema di protezione nel sistema operativo UNIX.
b) Suggerite un altro schema di protezione che si possa usare più efficacemente dello schema offerto dallo UNIX.

11.13 Alcuni ricercatori hanno suggerito che, invece di avere un elenco d'accesso associato a ciascun file (indicante quali utenti possono accedere al file e come) si può avere un **elenco di controllo degli utenti** associato a ogni utente, indicante a quali file, e in che modo, un determinato utente può accedere. Discutete i vantaggi di ciascuno di questi due schemi.

11.9 Note bibliografiche

Discussioni generali sui file system si trovano in [Grosshans 1986]; [Golden e Pechura 1986] descrive la struttura dei file system dei microcalcolatori. I sistemi di gestione delle basi di dati e le loro strutture di file sono ampiamente descritti in [Korth e Silberschatz 2001].

La struttura di directory a più livelli è stata realizzata per la prima volta nel sistema MULTICS [Organick 1972]. Attualmente la maggior parte dei sistemi dispone di una struttura di directory a più livelli, ad esempio il sistema operativo UNIX [Ritchie e Thompson 1974], l'Apple Macintosh [Apple 1991] e l'MS-DOS [Microsoft 1991].

Il file system dell'MS-DOS è descritto in [Norton e Wilton 1988]. Il VAX VMS è trattato in [Kenah et al. 1988] e [DEC 1981]. Il Network File System (NFS), progettato dalla Sun Microsystems permette di distribuire le strutture di directory sui calcolatori di una rete. Discussioni riguardanti l'NFS sono presentate in [Sandberg et al. 1985], [Sandberg 1987] e [Sun Microsystems 1990]. L'NFS è descritto nel Capitolo 16. La semantica dei file condivisi immutabili è descritta in [Schroeder et al. 1985].

Inizialmente proposto da [Su 1982], il DNS è passato attraverso diverse revisioni; con [Mockapetris 1987] è stato esteso con importanti caratteristiche. Più recentemente, [Eastlake 1999] ha proposto estensioni relative alla sicurezza per dotarlo delle chiavi di sicurezza.

Il protocollo LDAP, noto anche come X.509, è un sottoinsieme del protocollo di directory distribuita X.500. È stato definito da [Yeong et al. 1995] ed è stato incorporato in molti sistemi operativi.

Sono in corso interessanti ricerche nell'area delle interfacce dei file system, in particolare sui problemi relativi alla nominazione dei file e ai loro attributi. Ad esempio, il sistema operativo Plan 9 dei Bell Laboratories (Lucent Technology) rende tutti gli oggetti simili a file system. In tal modo, per ottenere l'elenco dei processi nel sistema, un utente deve semplicemente leggere il contenuto della directory `/proc`; per vedere l'ora si deve leggere il file `/dev/time`.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 12

Realizzazione del file system

Come si spiega nel Capitolo 11, il file system fornisce il meccanismo per la memorizzazione e l'accesso al contenuto dei file, compresi dati e programmi. Il file system risiede permanentemente nella *memoria secondaria*, progettata per contenere in modo permanente grandi quantità di dati. Questo capitolo riguarda principalmente i problemi connessi alla memorizzazione e all'accesso ai file nel più comune mezzo di memoria secondaria, il disco. Si esaminano vari modi di strutturare l'uso dei file, per l'assegnazione dello spazio dei dischi, il recupero dello spazio liberato, la registrazione delle locazioni dei dati, e per l'interfaccia di altri componenti del sistema operativo alla memoria secondaria. Nel corso della trattazione si considerano anche i problemi riguardanti le prestazioni.

12.1 Struttura del file system

I dischi costituiscono la maggior parte della memoria secondaria in cui si conserva il file system. Hanno due caratteristiche importanti che ne fanno un mezzo conveniente per la memorizzazione dei file:

1. si possono riscrivere localmente; si può leggere un blocco dal disco, modificarlo e quindi scriverlo nella stessa posizione;
2. è possibile accedere direttamente a qualsiasi blocco di informazioni del disco, quindi risulta semplice accedere a qualsiasi file, sia in modo sequenziale sia in modo diretto, e passare da un file all'altro spostando le testine di lettura e scrittura e attendendo la rotazione del disco.

La struttura dei dischi è discussa nei dettagli nel Capitolo 14.

Anziché trasferire un byte alla volta, per migliorare l'efficienza dell'I/O, i trasferimenti tra la memoria centrale e i dischi si eseguono per *blocchi*. Ciascun blocco è composto da uno o più settori. Secondo l'unità a disco, la dimensione dei settori è compresa fra 32 byte e 4096 byte; di solito è pari a 512 byte.

http://www.odultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per fornire un efficiente e conveniente accesso al disco, il sistema operativo fa uso di uno o più **file system** che consentono di memorizzare, individuare e recuperare facilmente i dati. Un file system presenta due problemi di progettazione piuttosto diversi. Il primo problema riguarda la definizione dell'aspetto del file system agli occhi dell'utente. Questo compito implica la definizione di un file e dei suoi attributi, delle operazioni permesse su un file e della struttura delle directory per l'organizzazione dei file. Il secondo problema riguarda la creazione di algoritmi e strutture di dati che permettano di far corrispondere il file system logico ai dispositivi fisici di memoria secondaria.

Lo stesso file system è generalmente composto da molti livelli distinti. La struttura illustrata nella Figura 12.1 è un esempio di struttura stratificata. Ogni livello si serve delle funzioni dei livelli inferiori per creare nuove funzioni che sono impiegate dai livelli superiori.

Il livello più basso, il *controllo dell'I/O*, costituito dei **driver dei dispositivi** e dei gestori dei segnali d'interruzione, si occupa del trasferimento delle informazioni tra la memoria centrale e la memoria secondaria. Un driver di dispositivo si può pensare come un traduttore che riceve comandi ad alto livello, come "recupera il blocco 123", ed emette specifiche istruzioni di basso livello per i dispositivi, usate dal controllore che fa da interfaccia tra i dispositivi di I/O e il resto del sistema. Un driver di dispositivo di solito scrive specifiche configurazioni di bit in specifiche locazioni della memoria del controllore di I/O per indicare quali azioni il dispositivo di I/O debba compiere, e in quali locazioni. I dettagli dei driver dei dispositivi e le strutture per l'I/O sono trattati nel Capitolo 13.

Il **file system di base** deve soltanto inviare dei generici comandi all'appropriato driver di dispositivo per leggere e scrivere blocchi fisici nel disco. Ogni blocco fisico si identifica col suo indirizzo numerico nel disco, ad esempio unità 1, cilindro 73, superficie 2, settore 10.

Il **modulo di organizzazione dei file** è a conoscenza dei file e dei loro blocchi logici, così come dei blocchi fisici dei dischi. Conoscendo il tipo di assegnazione dei file usato e la locazione dei file, può tradurre gli indirizzi dei blocchi logici, che il file system di base deve trasferire, negli indirizzi dei blocchi fisici. I blocchi logici di ciascun file sono numerati da 0 (o 1) a *n*; i blocchi fisici contenenti tali dati di solito non corrispondono ai numeri dei blocchi logici; per individuare ciascun blocco è quindi necessaria una traduzione. Il modulo di organizzazione dei file comprende anche il gestore dello spazio libero, che registra i blocchi non assegnati e li mette a disposizione del modulo di organizzazione dei file quando sono richiesti.

Infine, il **file system logico** gestisce i metadati; si tratta di tutte le strutture del file system, eccetto gli effettivi dati (il contenuto dei file). Il file system logico gestisce la struttura di directory per fornire al modulo di organizzazione dei file le informazioni di cui necessita, dato un nome simbolico di file. Mantiene le strutture di file tramite i **descrittori di file** (*file descriptor* — FD), noti anche come **blocchi di controllo dei file** (*file control block* — FCB), che contengono informazioni sui file, come la proprietà, i permessi, e la posizione del contenuto. Come si discute nel Capitolo 11 e poi nel Capitolo 18, il file system logico è responsabile anche della protezione e della sicurezza.

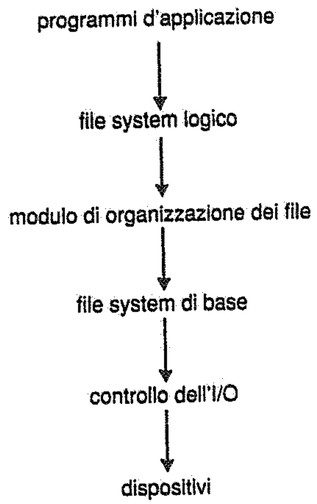


Figura 12.1 File system stratificato.

Esistono molti tipi di file system, e la maggior parte dei sistemi operativi ne impiega più di uno. La maggior parte dei CD-ROM ad esempio è scritta nel formato *High Sierra*, definito tramite un accordo fra produttori. Senza un tale formato standard, ci sarebbe una scarsa o nulla possibilità di scambio tra sistemi per mezzo dei CD-ROM. Oltre ai file system dei mezzi rimovibili, i sistemi operativi dispongono di uno o più file system basati su dischi. Il sistema operativo UNIX usa lo UNIX **file system** (UFS) come file system di base. Il sistema Windows 2000 gestisce i formati di file system per dischi FAT, FAT32 e NTFS (*Windows NT file system*), così come i formati per CD-ROM, DVD, e dischetti. L'uso di una struttura stratificata nella realizzazione di un file system consente di ridurre al minimo la duplicazione del codice: il codice del controllo dell'I/O e talvolta del file system di base si può usare per più file system. Ciascun file system deve avere il proprio file system logico e i propri moduli di organizzazione dei file.

12.2 Realizzazione del file system

I sistemi operativi offrono le chiamate del sistema `open` e `close` per permettere ai processi di richiedere l'accesso al contenuto dei file (Paragrafo 11.1.2). In questo paragrafo si approfondiscono le strutture di dati e le operazioni usate per realizzare le funzioni del file system.

http://www.adultpdf.com
Created by TIFF To PDF trial version, please visit www.tiffpdf.com to remove this watermark, please register this software.

12.2.1 Introduzione

Per realizzare un file system si usano parecchie strutture di dati, sia nei dischi sia nella memoria. Queste strutture variano secondo il sistema operativo e il file system, ma esistono dei princìpi generali. Nei dischi, il file system tiene informazioni su come eseguire l'avviamento di un sistema operativo memorizzato nei dischi stessi, il numero totale di blocchi, il numero e la locazione dei blocchi liberi, la struttura delle directory e i singoli file. Molte di queste strutture sono analizzate nei dettagli nel resto di questo capitolo.

Fra le strutture presenti nei dischi ci sono le seguenti:

- Il **blocco di controllo dell'avviamento** (*boot control block*), che contiene le informazioni necessarie al sistema per l'avviamento di un sistema operativo da quella partizione; se il disco non contiene un sistema operativo, tale blocco è vuoto. Di solito è il primo blocco di una partizione. Nell'UFS, si chiama **blocco d'avviamento** (*boot block*); nell'NTFS, **settore d'avviamento della partizione** (*partition boot sector*).
- I **blocchi di controllo delle partizioni** (*partition control block*); ciascuno di essi contiene i dettagli riguardanti la relativa partizione, come il numero e la dimensione dei blocchi nel disco, il contatore dei blocchi liberi e i relativi puntatori, il contatore di descrittori di file liberi e i relativi puntatori. Nell'UFS si chiama **superblocco**; nell'NTFS si chiama **tabella principale dei file** (*master file table* — MFT).
- Le **strutture di directory** che si usano per organizzare i file.
- I **descrittori di file**, che contengono molti dettagli dei file, compresi i permessi d'accesso ai relativi file, i proprietari, le dimensioni e le locazioni dei blocchi di dati. Nell'UFS si chiamano *inode*; nell'NTFS, queste informazioni sono memorizzate all'interno della tabella principale dei file, che si serve di una struttura di base di dati relazionale, con una riga per ciascun file.

Le informazioni tenute nella memoria servono sia per la gestione del file system sia per migliorare le prestazioni attraverso l'uso di cache. Le strutture che contengono queste informazioni comprendono:

- la **tabella delle partizioni** (*partition table*) contenente informazioni su ciascuna delle partizioni montate;
- la struttura di directory, tenuta nella memoria, contenente le informazioni relative a tutte le directory cui i processi hanno avuto accesso di recente (per le directory che costituiscono dei punti di montaggio, può essere presente un puntatore alla tabella delle partizioni);
- la tabella generale dei file aperti, contenente una copia del descrittore di file per ciascun file aperto, insieme con altre informazioni;
- la tabella dei file aperti per ciascun processo, contenente un puntatore all'appropriato elemento della tabella generale dei file aperti, insieme con altre informazioni.

Un'applicazione, per creare un nuovo file, fa una chiamata al file system logico, che conosce il formato delle strutture di directory e, per creare il nuovo file, assegna un nuovo descrittore di file, carica nella memoria la directory appropriata, l'aggiorna con il nuovo nome di file e il relativo descrittore e la riscrive nella memoria secondaria. Una tipica struttura per il descrittore di file è illustrata nella Figura 12.2.

Alcuni sistemi operativi, compreso UNIX, trattano le directory esattamente come i file, le distinguono con un campo per il tipo che indica che si tratta di una directory. Altri, tra cui il sistema operativo Windows NT, dispongono di chiamate del sistema distinte per i file e le directory e trattano le directory come entità separate dai file. Indipendentemente da questioni strutturali, il file system logico può basarsi sul modulo che si occupa dell'organizzazione dei file per far corrispondere l'I/O su directory a numeri di blocchi di disco, che poi si passano al file system di base e al sistema per il controllo dell'I/O. Il modulo che si occupa dell'organizzazione dei file assegna anche i blocchi necessari per memorizzare i dati dei file.

Una volta che un file è stato creato, per essere usato per operazioni di I/O deve essere *aperto*. La chiamata del sistema open passa un nome di file al file system affinché questo cerchi il nome richiesto nella struttura delle directory. Alcune porzioni della struttura delle directory sono di solito tenute nella memoria per accelerare le operazioni sulle directory. Una volta che il file è stato trovato, si copia il suo descrittore di file nella tabella generale dei file aperti, tenuta nella memoria. Questa tabella non solo contiene il descrittore di file, ma anche il numero di processi che in quel momento hanno il file aperto.

Successivamente, si crea un elemento nella tabella dei file aperti del processo che ha richiesto l'esecuzione della open, con un puntatore alla tabella generale e con alcuni altri campi. Questi altri campi possono comprendere un puntatore alla posizione corrente nel file (per successive operazioni di lettura o scrittura) e il tipo d'accesso richiesto all'apertura del file. La open riporta un puntatore all'elemento appropriato nella tabella dei file aperti del processo sicché tutte le operazioni sul file si svolgeranno usando questo puntatore. Il nome del file potrebbe non essere contenuto nella tabella dei file aperti, visto che,

permessi per il file
data e ora di creazione, di ultimo accesso e di ultima scrittura
proprietario del file, gruppo, ACL
dimensione del file
blocchi di dati del file

Figura 12.2 Tipico descrittore di file.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

una volta che il descrittore di file appropriato è stato individuato nei dischi, il sistema non ne ha bisogno. Il nome dato all'elemento della tabella si chiama **descrittore di file** (*file descriptor*) nello UNIX, e **maniglia di file** (*file handle*) nel Windows 2000. Finché un file non viene chiuso, tutte le operazioni si compiono sulla tabella dei file aperti usando questo elemento.

Quando un processo chiude il file, si cancella il relativo elemento nella tabella dei file aperti del processo e si decrementa il contatore associato al file nella tabella generale. Se tutti i processi che avevano aperto il file lo hanno chiuso, si riscrive l'informazione aggiornata sul file nella struttura delle directory nei dischi e si cancella il relativo elemento nella tabella generale dei file aperti.

In realtà, la chiamata del sistema open inizialmente verifica, cercando nella tabella generale dei file aperti, se un file è già usato da un altro processo. Se lo è, semplicemente si crea un elemento nella tabella dei file aperti del processo, puntando all'elemento relativo a quel file nella tabella generale. Questo algoritmo permette di risparmiare molto carico nell'apertura dei file già aperti.

Alcuni sistemi complicano ulteriormente lo schema descritto, usando il file system come interfaccia per altri aspetti del sistema, ad esempio la comunicazione in rete. Ad esempio, nell'UFS, la tabella generale dei file aperti contiene gli *inode* e altre informazioni su file e directory, ma contiene anche informazioni simili per le connessioni di rete e i dispositivi. In questo modo si può usare un unico meccanismo per gestire diversi aspetti del sistema.

Le questioni concernenti l'uso delle cache per queste strutture non vanno però trascurate; usando questo schema tutta l'informazione su un file aperto, eccetto i suoi effettivi blocchi di dati, è tenuta nella memoria. Il sistema UNIX BSD è noto per il suo uso di cache ovunque sia possibile risparmiare su operazioni di I/O nei dischi. La sua frequenza media di successi nella cache, pari all'85 per cento, dimostra l'utilità di queste tecniche. Il sistema UNIX BSD è descritto nei dettagli nell'Appendice A. La Figura 12.3 riassume le strutture che si usano nella realizzazione di un file system.

12.2.2 Partizioni e montaggio

Un disco si può configurare in vari modi, secondo il sistema operativo che lo gestisce. Si può suddividere in più partizioni, oppure una partizione può comprendere più dischi. Il secondo caso si può considerare un caso particolare di organizzazione RAID ed è trattato nel Paragrafo 14.5.

Una partizione è priva di struttura logica (*raw partition*) se non contiene alcun file system. Se nessun file system è appropriato, si usa un **disco privo di struttura logica** (*raw disk*). Il sistema operativo UNIX impiega una partizione senza file system per l'area d'avvicendamento dei processi; per questo scopo usa un formato specifico. Allo stesso modo alcuni sistemi di gestione delle basi di dati usano dischi privi di un'ordinaria struttura logica e formattano i dati secondo le proprie necessità. Un disco privo di struttura logica può anche contenere informazioni necessarie per sistemi RAID di gestione dei dischi, ad esempio le mappe di bit che indicano quali blocchi sono duplicati in altri dischi, e quali

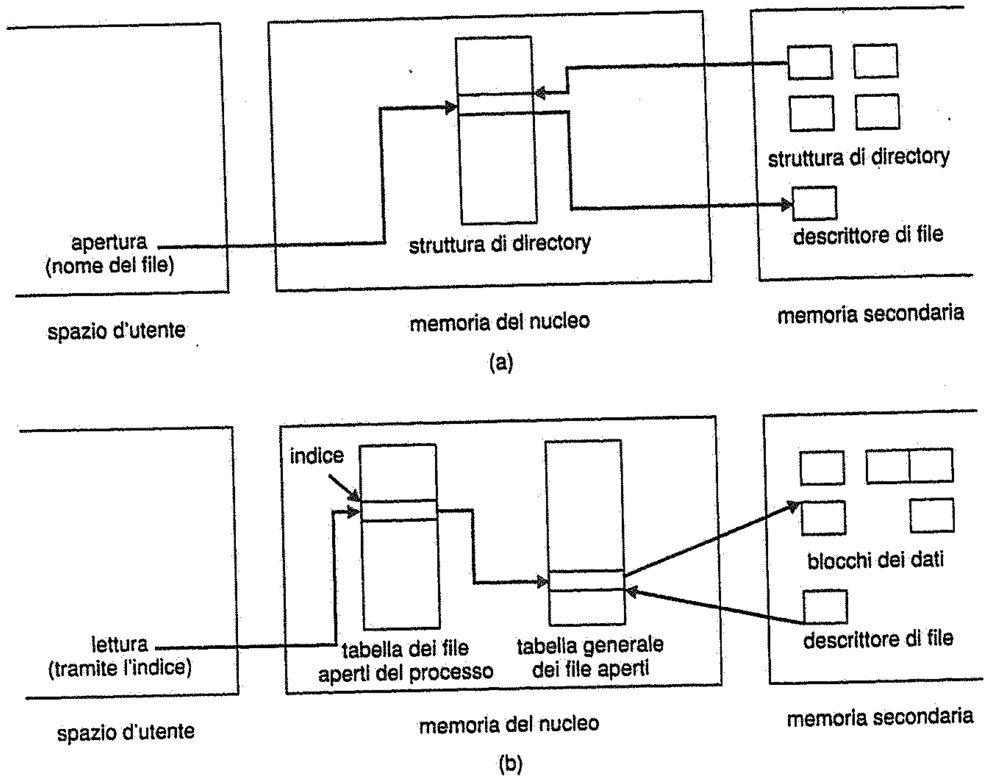


Figura 12.3 Strutture del file system che si mantengono nella memoria: (a) apertura di file; (b) lettura di file.

sono stati modificati e si devono aggiornare negli altri dischi. Analogamente, può contenere una piccola base di dati di informazioni sulla configurazione RAID, ad esempio, quali dischi appartengono a ciascun insieme RAID. Il Paragrafo 14.3.1 affronta altri aspetti concernenti l'uso dei dischi privi di struttura logica.

Le informazioni relative all'avviamento del sistema si possono registrare in un'apposita partizione, che anche in questo caso ha un proprio formato, poiché nella fase d'avviamento il sistema non ha ancora caricato i driver di dispositivo del file system e quindi non può interpretarne il formato. Questa partizione consiste piuttosto in una serie sequenziale di blocchi, che si carica nella memoria come un'immagine. L'esecuzione dell'immagine comincia a una locazione prefissata, ad esempio il primo byte. L'immagine d'avviamento può contenere più informazioni di quelle che servono per un singolo sistema operativo. I PC, ad esempio, e altri sistemi si possono configurare per l'installazione di più sistemi operativi. In questo caso l'area d'avviamento può contenere un modulo,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

detto caricatore d'avviamento (*boot loader*), capace di interpretare diversi file system e diversi sistemi operativi. Una volta caricato, può avviare uno dei sistemi operativi disponibili nei dischi. Ciascun disco può avere più partizioni, ognuna contenente un diverso tipo di file system e un diverso sistema operativo.

Nella fase di caricamento del sistema operativo, si esegue il montaggio della **partizione radice** (*root partition*), che contiene il nucleo del sistema operativo e in alcuni casi altri file di sistema. Secondo il sistema operativo, il montaggio delle altre partizioni avviene automaticamente in questa fase oppure si può compiere successivamente in modo esplicito. Durante l'operazione di montaggio, il sistema verifica che il dispositivo contenga un file system valido chiedendo al dispositivo di leggere la directory di dispositivo e verificando che la directory abbia il formato corretto. Se così non fosse, è necessaria una verifica della coerenza della partizione e una eventuale correzione, con o senza l'intervento dell'utente. Infine, il sistema annota nella struttura della tabella di montaggio nella memoria che un file system è stato montato insieme al tipo di file system. I dettagli di questa funzione dipendono dal sistema operativo.

I sistemi basati sui sistemi operativi Microsoft Windows eseguono il montaggio di ogni partizione in uno spazio di nomi separato, identificato da una lettera seguita dai due punti (`:`). Ad esempio, per memorizzare che un file system è stato montato in `F:`, il sistema operativo introduce un puntatore al file system in un campo della struttura di dispositivo corrispondente a `F:`. Quando un processo specifica la lettera dell'unità, il sistema operativo trova il puntatore al file system appropriato e attraversa la struttura delle directory in quel dispositivo per trovare lo specifico file o directory.

Nello UNIX, l'operazione di montaggio di un file system si può compiere in qualsiasi directory. Questa funzione si realizza impostando un indicatore nella copia dell'*inode* tenuta nella memoria di quella directory, che segnala che la directory è un punto di montaggio. Un campo dell'*inode* punta a un elemento nella tabella di montaggio, che indica quale dispositivo è montato in quella posizione. L'elemento della tabella di montaggio contiene un puntatore al superblocco del file system in quel dispositivo. Questo schema permette al sistema operativo di attraversare facilmente la sua struttura delle directory, passando da un file system all'altro secondo le necessità.

12.2.3 File system virtuali

Nel Paragrafo 12.1 si sottolinea il fatto che i sistemi operativi moderni devono gestire contemporaneamente diversi tipi di file system. Per capire come si può realizzare questa funzione occorre considerare il modo in cui un sistema operativo può consentire l'integrazione di diversi tipi di file system in un'unica struttura di directory, in modo da permettere agli utenti di spostarsi senza problemi da un tipo di file system all'altro mentre percorrono lo spazio del file system complessivo.

Un metodo ovvio ma non ottimale per realizzare più tipi di file system è di scrivere procedure di gestione di file e directory separate per ciascun tipo di file system. Al contrario, la maggior parte dei sistemi operativi, compreso UNIX, impiega tecniche orientate agli oggetti per semplificare e organizzare in maniera modulare la soluzione. L'uso di

queste tecniche rende possibile la realizzazione, nella stessa struttura, di tipi di file system molto diversi tra loro, compresi i file system di rete, come l'NFS. Gli utenti possono accedere a file contenuti in diversi file system nei dischi locali, o anche in file system disponibili tramite la rete.

Per isolare le funzioni di base delle chiamate del sistema dai dettagli di realizzazione si adoperano apposite strutture di dati. In questo modo la realizzazione del file system si articola in tre strati principali, riportati in modo schematico nella Figura 12.4. Il primo strato è l'interfaccia del file system, basata sulle chiamate del sistema `open`, `read`, `write` e `close` e sui descrittori di file.

Il secondo strato si chiama strato del **file system virtuale** (*virtual file system* — VFS) e svolge due funzioni importanti:

1. Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme. Nello stesso calcolatore possono coesistere più interfacce VFS, che permettono un accesso trasparente a diversi tipi di file system montati localmente.

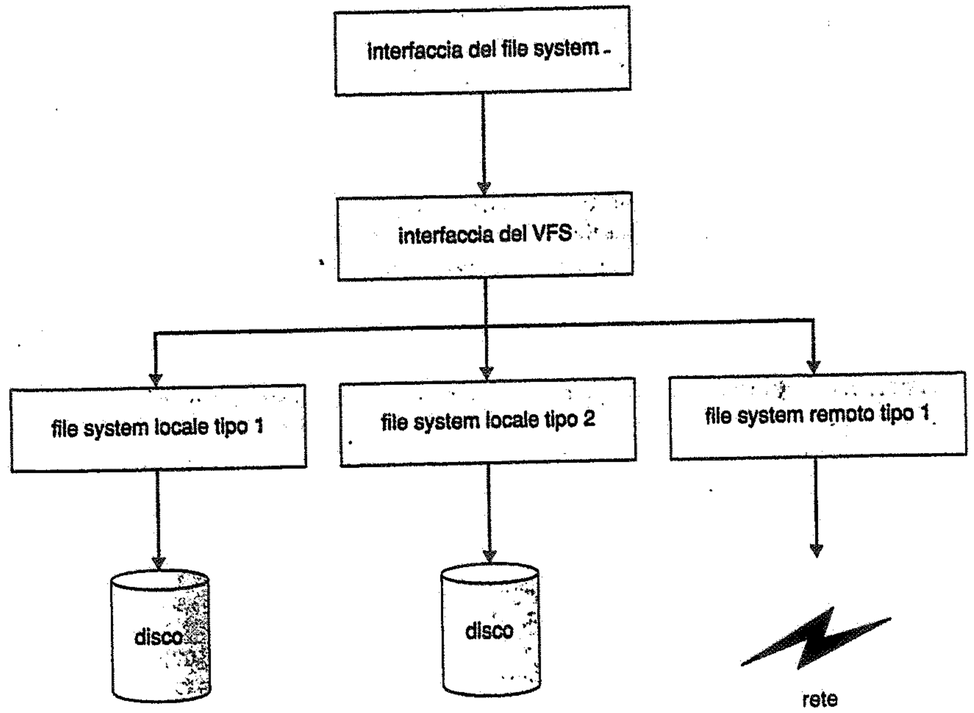


Figura 12.4 Schema di un file system virtuale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

2. Il VFS è basato su una struttura di rappresentazione dei file detta *vnode* che contiene un indicatore numerico unico per tutta la rete per ciascun file. (Gli *inode* dello UNIX sono unici solo all'interno di un singolo file system.) Tale unicità per tutta la rete è richiesta per la gestione del file system di rete. Il nucleo contiene una struttura *vnode* per ciascun nodo attivo, sia che si tratti di un file sia che si tratti di una directory.

Quindi, il VFS distingue i file locali da quelli remoti, e distingue i file locali secondo i relativi tipi di file system.

Il VFS attiva le operazioni specifiche del file system per gestire le richieste locali secondo i tipi di file system, e invoca le procedure del protocollo NFS per le richieste remote. Le maniglie di file si costruiscono secondo i *vnode* relativi e s'inviano a queste procedure come argomenti. Lo strato che realizza il protocollo NFS è il più basso dell'architettura. Una spiegazione del funzionamento dell'NFS si trova nel Paragrafo 12.9.

12.3 Realizzazione delle directory

La selezione degli algoritmi di assegnazione e degli algoritmi di gestione delle directory ha un grande effetto sull'efficienza, le prestazioni e l'affidabilità del file system. Per tale ragione è necessario comprendere i vari aspetti di questi algoritmi.

12.3.1 Lista lineare

Il più semplice metodo di realizzazione di una directory è basato sull'uso di una lista lineare contenente i nomi dei file con puntatori ai blocchi di dati. L'individuazione di un elemento particolare in una lista lineare di elementi di directory richiede una ricerca lineare. Questo metodo è di facile programmazione, ma la sua esecuzione è onerosa in termini di tempo. Per creare un nuovo file occorre prima esaminare la directory per essere sicuri che non esista già un file con lo stesso nome, quindi aggiungere un nuovo elemento alla fine della directory. Per cancellare un file occorre cercare nella directory il file con quel nome, quindi rilasciare lo spazio che gli era assegnato. Esistono vari metodi per riutilizzare un elemento della directory: si può contrassegnare l'elemento come non usato (attribuendogli un nome speciale, come un nome vuoto, oppure un bit d'uso in ogni elemento), oppure può essere aggiunto a una lista di elementi di directory liberi; una terza possibilità prevede la copiatura dell'ultimo elemento della directory in una locazione liberata e la diminuzione della lunghezza della directory. Per ridurre il tempo di cancellazione di un file si può usare anche una lista concatenata.

Il vero svantaggio dato da una lista lineare di elementi di directory è dato dalla ricerca lineare di un file. Le informazioni sulla directory vengono usate frequentemente, e gli utenti avvertirebbero una gestione lenta dell'accesso a tali informazioni. In effetti, molti sistemi operativi impiegano una cache per memorizzare le informazioni sulla directory usata più recentemente. La presenza nella cache delle informazioni richieste ne evita la continua rilettura dai dischi. Una lista ordinata permette una ricerca binaria e ri-

duce il tempo medio di ricerca, tuttavia il requisito dell'ordinamento può complicare la creazione e la cancellazione di file, poiché, per tenere ordinata la lista, può essere necessario spostare quantità notevoli di informazioni sulla directory. In questo caso, può essere d'aiuto una struttura di dati più raffinata come un B-albero. Un vantaggio della lista ordinata è che consente di produrre l'elenco ordinato del contenuto della directory senza una fase d'ordinamento separata.

12.3.2 Tabella hash

Un'altra struttura di dati che si usa per realizzare le directory è la **tabella hash**. In questo metodo una lista lineare contiene gli elementi di directory, ma si usa anche una struttura di dati hash. La tabella hash riceve un valore calcolato usando come operando il nome del file e riporta un puntatore al nome del file nella lista lineare. Questa struttura di dati può diminuire notevolmente il tempo di ricerca nella directory. L'inserimento e la cancellazione sono abbastanza semplici, anche se occorre prendere provvedimenti per evitare le **collisioni**, cioè situazioni in cui da due nomi di file si ottiene un riferimento alla stessa locazione. Le maggiori difficoltà legate a una tabella hash sono la sua dimensione, che in genere è fissa, e la dipendenza della funzione hash da tale dimensione.

Si supponga, ad esempio, di realizzare una tabella hash di 64 elementi; la funzione hash converte i nomi di file in interi da 0 a 63, ad esempio, usando il resto di una divisione per 64. Per creare in un secondo tempo un sessantacinquesimo file occorre allungare la tabella hash della directory, ad esempio fino a 128 elementi. Occorre quindi una nuova funzione hash per associare i nomi di file all'intervallo compreso tra 0 e 127, e gli elementi esistenti nella directory si devono riorganizzare in modo da riflettere i loro nuovi valori della funzione hash. Alternativamente, ciascun elemento della tabella hash, anziché un singolo valore, può essere una lista concatenata; ciò consente di risolvere le collisioni aggiungendo il nuovo elemento alla lista concatenata. Le ricerche vengono alquanto rallentate, poiché la ricerca per nome può richiedere l'attraversamento della lista concatenata degli elementi in collisione della tabella hash; probabilmente tale metodo è comunque più veloce di una ricerca lineare nell'intera directory.

12.4 Metodi di assegnazione

La natura ad accesso diretto dei dischi permette una certa flessibilità nella realizzazione dei file. In quasi tutti i casi, molti file si memorizzano nello stesso disco. Il problema principale consiste dunque nell'assegnare lo spazio per questi file in modo che lo spazio nel disco sia usato efficientemente e l'accesso ai file sia rapido. Esistono tre metodi principali per l'assegnazione dello spazio di un disco; può essere infatti contigua, concatenata o indicizzata. Ciascuno di questi metodi presenta vantaggi e svantaggi. Alcuni sistemi, come l'RDOS della Data General per la linea di calcolatori Nova, dispongono di tutti e tre i metodi. Generalmente, però, un sistema usa un unico metodo per tutti i file.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

12.4.1 Assegnazione contigua

Per usare il metodo di **assegnazione contigua**, ogni file deve occupare un insieme di blocchi contigui del disco. Gli indirizzi del disco definiscono un ordinamento lineare nel disco stesso. Con questo ordinamento l'accesso al blocco $b + 1$ dopo il blocco b non richiede normalmente alcuno spostamento della testina. Se la testina deve essere spostata dall'ultimo settore di un cilindro al primo settore del cilindro successivo lo spostamento avviene su una sola traccia. Quindi, il numero dei posizionamenti richiesti per accedere a file il cui spazio è assegnato in modo contiguo è trascurabile, così com'è trascurabile il tempo di posizionamento, quando quest'ultimo è necessario. Il sistema operativo IBM VM/CMS usa l'assegnazione contigua poiché consente tali buone prestazioni.

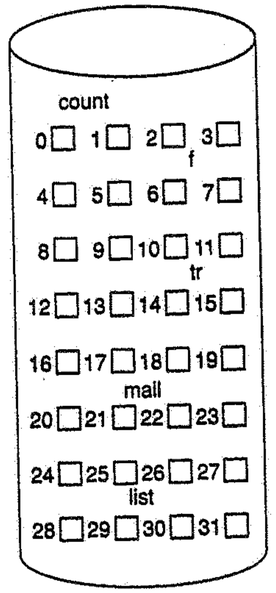
L'assegnazione contigua dello spazio per un file è definita dall'indirizzo del primo blocco (inteso come numero di blocco) e dalla lunghezza (espressa in blocchi). Se il file è lungo n blocchi e comincia dalla locazione b, allora occupa i blocchi b, $b + 1$, $b + 2$, ..., $b + n - 1$. L'elemento di directory per ciascun file indica l'indirizzo del blocco d'inizio e la lunghezza dell'area assegnata per questo file (Figura 12.5).

Accedere a un file il cui spazio è assegnato in modo contiguo è facile. Quando si usa un accesso sequenziale, il file system memorizza l'indirizzo dell'ultimo blocco cui è stato fatto riferimento e, se è necessario, legge il blocco successivo. Nel caso di un accesso diretto al blocco i di un file che comincia al blocco b si può accedere immediatamente al blocco $b + i$. Quindi, sia l'accesso sequenziale sia quello diretto si possono gestire con l'assegnazione contigua.

L'assegnazione contigua presenta però alcuni problemi; una difficoltà riguarda l'individuazione dello spazio per un nuovo file. La realizzazione del sistema di gestione dello spazio libero, discusso nel Paragrafo 12.5, determina il modo in cui tale compito viene eseguito. Si può usare ogni sistema di gestione, anche se alcuni sono più lenti di altri.

Il problema dell'assegnazione contigua dello spazio dei dischi si può considerare un'applicazione particolare del problema generale dell'**assegnazione dinamica della memoria**, trattato nel Paragrafo 9.3; il problema generale è, infatti, quello di soddisfare una richiesta di dimensione n data una lista di buchi liberi. I più comuni criteri di scelta di un buco libero da un insieme di buchi disponibili sono quelli del primo buco abbastanza grande (*first-fit*) e del più piccolo tra i buchi abbastanza grandi (*best-fit*). Simulazioni hanno dimostrato che questi due criteri sono più efficienti di quello di scelta del buco più grande (*worst-fit*) sia in termini di tempo sia d'uso della memoria. Nessuno dei due, è palesemente migliore rispetto all'uso della memoria, ma la scelta del primo buco abbastanza grande è generalmente più rapida.

Questi algoritmi soffrono della **frammentazione esterna**: assegnando e liberando lo spazio per i file, lo spazio libero dei dischi viene frammentato in tanti piccoli pezzi. La frammentazione esterna si ha ogniqualvolta lo spazio libero è suddiviso in pezzi, e diviene un problema quando il più grande di tali pezzi contigui non è sufficiente a soddisfare una richiesta; la memoria viene frammentata in tanti buchi, nessuno dei quali è abbastanza grande da contenere i dati. Secondo la capacità dei dischi e la dimensione media dei file, la frammentazione esterna può essere un problema più o meno grave.



directory

file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figura 12.5 Assegnazione contigua dello spazio dei dischi.

Alcuni vecchi microcalcolatori adoperavano l'assegnazione contigua per i dischetti. Per impedire lo spreco di quantità notevoli di spazio a causa della frammentazione esterna, l'utente doveva eseguire una procedura di compattazione che copiava l'intero file system in un altro dischetto o in un nastro. Quindi si liberava completamente il primo dischetto creando un ampio spazio libero contiguo. La procedura provvedeva poi a copiare nuovamente i file nel dischetto, assegnando tale spazio contiguo. Questo schema compatta efficacemente tutto lo spazio libero in uno spazio contiguo, risolvendo il problema della frammentazione. Il costo di questa compattazione è rappresentato dal tempo necessario; ed è particolarmente pesante per i dischi di grande capacità che impiegano l'assegnazione contigua; in essi, compattare lo spazio può richiedere ore e può essere necessario eseguire tale operazione settimanalmente. Durante questo 'tempo morto' il normale funzionamento del sistema non è possibile, quindi tale compattazione va evitata a tutti i costi per i calcolatori in attività.

Un altro problema che riguarda l'assegnazione contigua è la determinazione della quantità di spazio necessaria per un file. Quando si crea un file, occorre trovare e assegnare lo spazio di cui necessita. Esiste il problema di conoscere la dimensione del file da creare; in alcuni casi questa dimensione si può stabilire in modo abbastanza semplice, ad esempio quando si copia un file esistente; in generale, comunque, non è facile stimare la dimensione di un file che deve contenere dati emessi da un programma.

http://www.addtpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Se un file riceve poco spazio, può essere impossibile estenderlo: soprattutto nel caso in cui si adoperi il criterio di assegnazione del più piccolo tra i buchi abbastanza grandi, lo spazio oltre le due estremità del file può essere già in uso, quindi non è possibile ampliare il file in modo contiguo. Esistono allora due possibilità. La prima è che il programma utente si possa terminare con un idoneo messaggio d'errore. L'utente deve allora assegnare più spazio ed eseguire di nuovo il programma. Queste esecuzioni ripetute possono essere onerose; per prevenire tale circostanza, normalmente l'utente sovrastima la quantità di spazio necessaria, sprecandone parecchio.

L'altra possibilità consiste nel trovare un buco più grande, copiare il contenuto del file nel nuovo spazio e rilasciare lo spazio precedente. Queste operazioni si possono ripetere finché esiste spazio, anche se ciò può far perdere tempo. In questo caso comunque non è necessario informare esplicitamente l'utente su cosa stia succedendo; anche se più lentamente, il sistema prosegue le attività nonostante il problema.

Anche se si conosce in anticipo la quantità di spazio necessaria per un file, l'assegnazione preventiva può in ogni modo essere inefficiente. A un file che cresce lentamente in un periodo di tempo lungo (mesi o anni) si deve assegnare spazio sufficiente per la sua dimensione finale, anche se molto di quello spazio può rimanere inutilizzato per parecchio tempo. Il file ha perciò un'estesa frammentazione interna.

Per ridurre al minimo questi inconvenienti, alcuni sistemi operativi fanno uso di uno schema di assegnazione contigua modificato: inizialmente si assegna una porzione di spazio contiguo, e se questa non è abbastanza grande si aggiunge un'altra porzione di spazio, un'*estensione*. La locazione dei blocchi dei file si registra come una locazione e un numero dei blocchi, insieme con l'indirizzo del primo blocco dell'estensione seguente. In alcuni sistemi il proprietario del file può impostare la dimensione dell'estensione, e tale possibilità, se il proprietario è impreciso, può causare inefficienze. La frammentazione interna può ancora essere un problema se le estensioni sono troppo grandi; si possono presentare problemi dovuti alla frammentazione esterna quando si assegnano e si rilasciano estensioni di dimensione variabile. Il file system commerciale Veritas impiega le estensioni per ottimizzare le prestazioni; è un sostituto ad alte prestazioni dell'ordinario UFS.

12.4.2 Assegnazione concatenata

L'**assegnazione concatenata** risolve tutti i problemi dell'assegnazione contigua. Con questo tipo di assegnazione, infatti, ogni file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso. La directory contiene un puntatore al primo e all'ultimo blocco del file. Ad esempio, un file di cinque blocchi può cominciare dal blocco 9, continuare al blocco 16, quindi al blocco 1, al blocco 10 e infine terminare al blocco 25 (Figura 12.6). Ogni blocco contiene un puntatore al blocco successivo. Questi puntatori non sono disponibili all'utente, quindi se ogni blocco è formato di 512 byte e un indirizzo del disco (il puntatore) richiede 4 byte, l'utente vede blocchi di 508 byte.

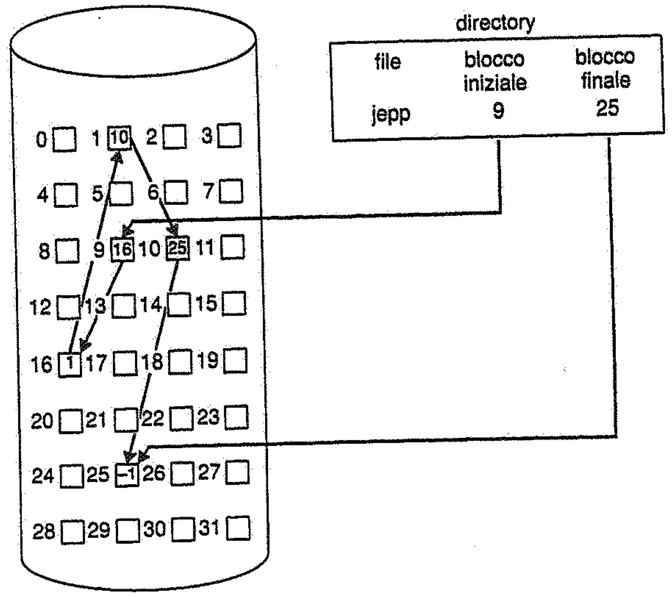


Figura 12.6 Assegnazione concatenata dello spazio dei dischi.

Per creare un nuovo file si crea semplicemente un nuovo elemento nella directory. Con l'assegnazione concatenata, ogni elemento della directory ha un puntatore al primo blocco del file. Questo puntatore s'inizializza a `nil` (valore del puntatore di fine lista) a indicare un file vuoto; anche il campo della dimensione s'imposta a 0. Un'operazione di scrittura nel file determina la ricerca di un blocco libero attraverso il sistema di gestione dello spazio libero, la scrittura in tale blocco, e la concatenazione di tale blocco alla fine del file. Per leggere un file occorre semplicemente leggere i blocchi seguendo i puntatori da un blocco all'altro. Con l'assegnazione concatenata non esiste frammentazione esterna e per soddisfare una richiesta si può usare qualsiasi blocco libero della lista. Inoltre non è necessario dichiarare la dimensione di un file al momento della sua creazione. Un file può continuare a crescere finché sono disponibili blocchi liberi, di conseguenza non è mai necessario compattare lo spazio del disco.

L'assegnazione concatenata presenta comunque alcuni svantaggi. Il problema principale riguarda il fatto che può essere usata in modo efficiente solo per i file ad accesso sequenziale. Per trovare l'*i*-esimo blocco di un file occorre partire dall'inizio del file e seguire i puntatori finché non si raggiunge l'*i*-esimo blocco. Ogni accesso a un puntatore implica una lettura del disco, e talvolta un posizionamento della testina. Di conseguenza, per file il cui spazio è assegnato in modo concatenato, la funzione d'accesso diretto è inefficiente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un altro svantaggio dell'assegnazione concatenata riguarda lo spazio richiesto per i puntatori. Se un puntatore richiede 4 byte di un blocco di 512 byte, allora lo 0,78 per cento del disco è usato per i puntatori anziché per le informazioni: ogni file richiede un po' più spazio di quanto ne richiederebbe altrimenti.

La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi contigui in gruppi (*cluster*), e nell'assegnare i gruppi di blocchi anziché i blocchi. Ad esempio, il file system può definire gruppi di 4 blocchi e operare nel disco soltanto per gruppi di blocchi. Così i puntatori usano una quantità di spazio di disco che si riduce in modo proporzionale al numero di blocchi di un gruppo. Questo metodo permette che la corrispondenza tra blocchi logici e blocchi fisici rimanga semplice, ma migliora la produttività del disco: si hanno meno posizionamenti della testina del disco e diminuisce lo spazio necessario per l'assegnazione dei blocchi e la gestione dell'elenco dei blocchi liberi. Il costo di questo metodo è dato da un incremento della frammentazione interna, poiché se un gruppo di blocchi è parzialmente pieno si spreca più spazio di quanto se ne sprecherebbe con un solo blocco parzialmente pieno. I gruppi di blocchi si possono usare per ottimizzare l'accesso ai dischi in molti altri algoritmi, quindi s'impiegano nella maggior parte dei sistemi operativi.

Un altro problema riguarda l'affidabilità. Poiché i file sono tenuti insieme da puntatori sparsi per tutto il disco, s'immagini cosa accadrebbe se un puntatore andasse perduto o danneggiato. Un errore di programmazione del sistema operativo oppure un errore di un'unità a disco potrebbero causare il prelevamento del puntatore errato. Questo errore, a sua volta, potrebbe causare il collegamento all'elenco dei blocchi liberi oppure a un altro file. Una soluzione parziale a tale problema consiste nell'usare liste doppiamente concatenate oppure nel memorizzare il nome del file e il relativo numero di blocco in ogni blocco; questi schemi però sono ancora più onerosi per ogni file.

Una variante importante del metodo di assegnazione concatenata consiste nell'uso della **tabella di assegnazione dei file** (*file allocation table* — FAT). Tale metodo di assegnazione, semplice ma efficiente, dello spazio dei dischi è usato nei sistemi operativi MS-DOS e OS/2. Per contenere tale tabella si riserva una sezione del disco all'inizio di ciascuna partizione; la FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco; si usa essenzialmente come una lista concatenata. L'elemento di directory contiene il numero del primo blocco del file. L'elemento della tabella indicizzato da quel numero di blocco contiene a sua volta il numero del blocco successivo del file. Questa catena continua fino all'ultimo blocco, che ha come elemento della tabella un valore speciale di fine del file. I blocchi inutilizzati sono indicati nella tabella da un valore 0. L'assegnazione di un nuovo blocco a un file implica semplicemente la localizzazione del primo elemento della tabella con valore 0 e la sostituzione del valore di fine del file precedente con l'indirizzo del nuovo blocco; lo 0 è quindi sostituito con il valore di fine del file. Un esempio esplicativo di tale metodo è dato dalla struttura della FAT della Figura 12.7, dove il file in questione è formato dei blocchi 217, 618 e 339.

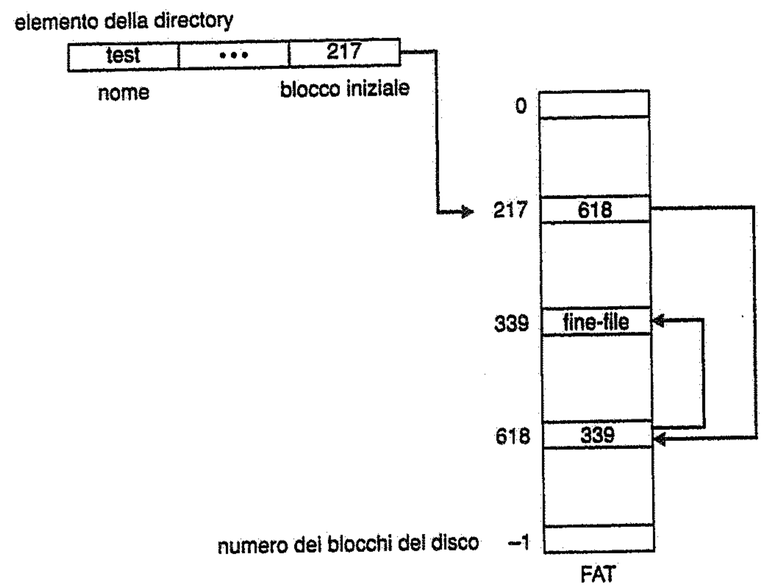


Figura 12.7 Tabella di assegnazione dei file.

Lo schema di assegnazione basato sulla FAT, se non si usa una cache, può causare un significativo numero di posizionamenti della testina. La testina del disco deve spostarsi all'inizio della partizione per leggere la FAT e trovare la locazione del blocco in questione, quindi raggiungere la locazione del blocco stesso; nel caso peggiore sono necessari ambedue i movimenti per ciascun blocco. Un vantaggio è dato dall'ottimizzazione del tempo d'accesso diretto, poiché la testina del disco può trovare la locazione di ogni blocco leggendo le informazioni contenute nella FAT.

12.4.3 Assegnazione indicizzata

L'assegnazione concatenata risolve il problema della frammentazione esterna e quello della dichiarazione delle dimensioni dei file, entrambi presenti nell'assegnazione contigua. Tuttavia, in mancanza di una FAT, l'assegnazione concatenata non è in grado di sostenere un efficiente accesso diretto, poiché i puntatori ai blocchi sono sparsi, con i blocchi stessi, per tutto il disco e si devono recuperare in ordine. **L'assegnazione indicizzata** risolve questo problema, raggruppando tutti i puntatori in una sola locazione: il **blocco indice**.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

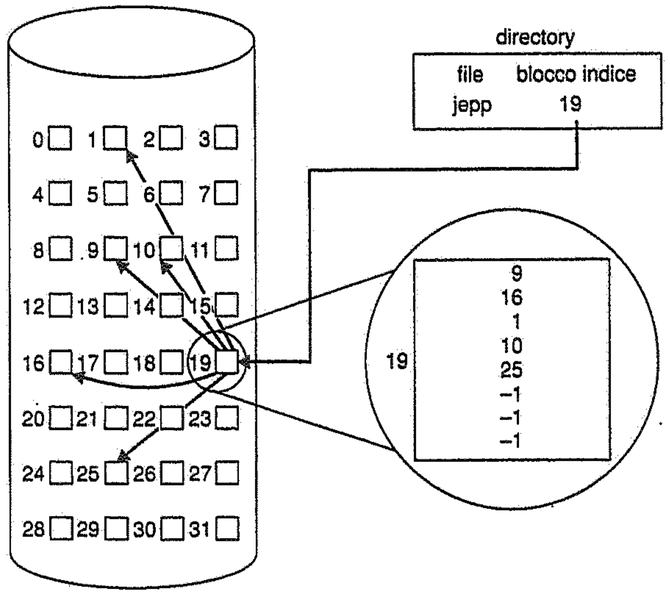


Figura 12.8 Assegnazione indicizzata dello spazio dei dischi.

Ogni file ha il proprio blocco indice: si tratta di un vettore d'indirizzi di blocchi del disco. L'*i*-esimo elemento del blocco indice punta all'*i*-esimo blocco del file. La directory contiene l'indirizzo del blocco indice, com'è illustrato nella Figura 12.8. Per leggere l'*i*-esimo blocco occorre usare il puntatore che si trova nell'*i*-esimo elemento del blocco indice, per poi localizzare e leggere il blocco desiderato. Questo schema è simile a quello della paginazione descritto nel Capitolo 9.

Una volta creato il file, tutti i puntatori del blocco indice sono impostati a `nil`. Quando si scrive l'*i*-esimo blocco per la prima volta, il gestore dei blocchi liberi fornisce un blocco; l'indirizzo di questo blocco viene inserito nell'*i*-esimo elemento del blocco indice. Poiché ogni blocco libero del disco può soddisfare una richiesta di maggiore spazio, l'assegnazione indicizzata consente l'accesso diretto senza soffrire di frammentazione esterna.

Lo spazio aggiuntivo richiesto dai puntatori del blocco indice è generalmente maggiore dello spazio aggiuntivo necessario per l'assegnazione concatenata. Si consideri il comune caso di un file con uno o due blocchi; con l'assegnazione concatenata si perde il solo spazio di un puntatore per blocco, complessivamente uno o due puntatori; con l'assegnazione indicizzata occorre assegnare un intero blocco indice, anche se solo uno o due puntatori sono diversi da `nil`.

Questo punto solleva la questione della dimensione del blocco indice. Ogni file deve avere un blocco indice, quindi è auspicabile che questo sia quanto più piccolo è possibile; ma se il blocco indice è troppo piccolo non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni, quindi è necessario disporre di un meccanismo per gestire questa situazione:

- **Schema concatenato.** Un blocco indice è formato normalmente di un solo blocco di disco; perciò, ciascun blocco indice può essere letto e scritto esattamente con una operazione. Per permettere la presenza di lunghi file è possibile collegare tra loro parecchi blocchi indice. Ad esempio, un blocco indice può contenere una piccola intestazione nella quale sono riportati il nome del file e l'insieme dei primi 100 indirizzi del blocco di disco. L'indirizzo successivo, vale a dire l'ultima parola del blocco indice, è `nil` (per un file piccolo) oppure è un puntatore a un altro blocco indice (per un file lungo).
- **Indice a più livelli.** Una variante della rappresentazione concatenata consiste nell'impiego di un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello che, a loro volta, puntano ai blocchi dei file. Per accedere a un blocco, il sistema operativo usa l'indice di primo livello, con il quale individua il blocco indice di secondo livello, e con esso trova il blocco di dati richiesto. Questo metodo può continuare fino a un terzo o quarto livello, secondo la massima dimensione desiderata del file. Con blocchi di 4096 byte si possono memorizzare 1024 puntatori di 4 byte in un blocco indice. Due livelli di indici consentono 1.048.576 blocchi di dati, che permettono di avere file sino a 4 GB.
- **Schema combinato.** Un'altra possibilità, è la soluzione adottata nell'UFS, consiste nel tenere i primi 15 puntatori del blocco indice nell'*inode* del file. I primi 12 di questi 15 puntatori puntano a **blocchi diretti**, cioè contengono direttamente gli indirizzi di blocchi che contengono dati del file. Quindi, i dati per piccoli file (non più di 12 blocchi) non richiedono un blocco indice distinto. Se la dimensione dei blocchi è di 4 KB, è possibile accedere direttamente fino a 48 KB di dati. Gli altri tre puntatori puntano a **blocchi indiretti**. Il primo puntatore di blocco indiretto è l'indirizzo di un **blocco indiretto singolo**; si tratta di un blocco indice che non contiene dati, ma indirizzi di blocchi che contengono dati. Quindi c'è un puntatore di **blocco indiretto doppio** che contiene l'indirizzo di un blocco che a sua volta contiene gli indirizzi di blocchi contenenti puntatori agli effettivi blocchi di dati. L'ultimo puntatore contiene l'indirizzo di un **blocco indiretto triplo**. Con questo metodo, il numero dei blocchi che si possono assegnare a un file supera la quantità di spazio che possono indirizzare i puntatori a file di 4 byte usati da molti sistemi operativi. Un puntatore a file di 32 bit consente di arrivare a soli 2^{32} byte, 4 GB. Molte versioni del sistema operativo UNIX, tra le quali il Solaris 2 e l'IBM AIX ora gestiscono puntatori a file sino a 64 bit. Puntatori di questa dimensione permettono di avere file e file system di dimensioni dell'ordine dei terabyte. Un *inode* è mostrato nella Figura 12.9.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

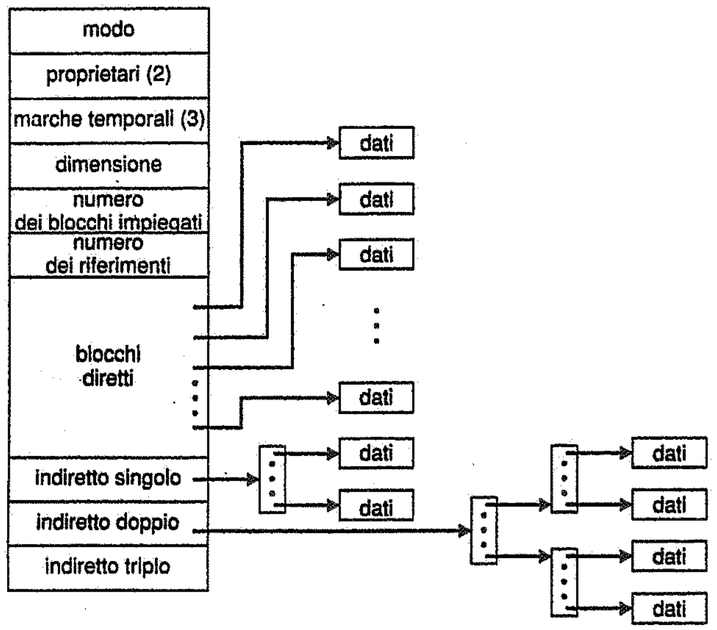


Figura 12.9 *Inode* dello UNIX.

Gli schemi d'assegnazione indicizzata soffrono di alcuni degli stessi problemi di prestazioni che riguardano l'assegnazione concatenata. In particolare, i blocchi indice si possono caricare nella memoria, ma i blocchi dei dati possono essere sparsi per un'intera partizione.

12.4.4 Prestazioni

I metodi d'assegnazione discussi hanno diversi livelli di efficienza di memorizzazione e diversi tempi d'accesso ai blocchi di dati; entrambi i fattori sono importanti nella scelta del metodo o dei metodi d'assegnazione più adatti da impiegare in un sistema operativo.

Prima di scegliere un metodo di assegnazione, è necessario determinare il modo in cui si usano i sistemi: un sistema con una prevalenza di accessi sequenziali farà uso di un metodo differente da quello di un sistema con una prevalenza di accessi diretti. Per qualsiasi tipo d'accesso, l'assegnazione contigua richiede un solo accesso per ottenere un blocco. Poiché è facile tenere l'indirizzo iniziale del file nella memoria, si può calcolare immediatamente l'indirizzo del disco dell'*i*-esimo blocco, oppure del blocco successivo, e leggerlo direttamente.

Con l'assegnazione concatenata si può tenere nella memoria anche l'indirizzo del blocco successivo e leggerlo direttamente. Questo metodo è valido per l'accesso sequenziale, mentre, per quel che riguarda l'accesso diretto, un accesso all'*i*-esimo blocco può richiedere *i* letture del disco. Questo spiega perché l'assegnazione concatenata non si dovrebbe usare per un'applicazione che richiede accessi diretti.

Da tutto ciò segue che alcuni sistemi gestiscono i file ad accesso diretto usando l'assegnazione contigua, e i file ad accesso sequenziale tramite l'assegnazione concatenata. Per questi sistemi, il tipo d'accesso si deve dichiarare al momento della creazione del file. Un file creato per l'accesso sequenziale è un file concatenato e non si può usare per l'accesso diretto. Un file creato per l'accesso diretto è contiguo e consente entrambi i tipi d'accesso, purché se ne dichiari la lunghezza massima al momento della sua creazione. In questo caso, il sistema operativo deve avere strutture di dati idonee e algoritmi capaci di gestire *entrambi* i metodi di assegnazione. I file si possono convertire da un tipo all'altro creando un nuovo file del tipo desiderato, nel quale si copia il contenuto del vecchio file; quest'ultimo si può quindi cancellare e il nuovo file rinominare.

L'assegnazione indicizzata è più complessa. Se il blocco indice è già nella memoria, allora l'accesso può essere diretto. Tuttavia, per tenere il blocco indice nella memoria occorre una quantità di spazio considerevole. Se questo spazio di memoria non è disponibile, occorre leggere prima il blocco indice e quindi il blocco di dati desiderato. Per un indice a due livelli possono essere necessarie due letture del blocco indice. Se un file è estremamente grande, per compiere l'accesso a un blocco che si trovi vicino alla fine del file, prima di leggere il blocco dei dati occorre leggere tutti i blocchi indice per seguire la catena dei puntatori. Quindi le prestazioni dell'assegnazione indicizzata dipendono dalla struttura dell'indice, dalla dimensione del file e dalla posizione del blocco desiderato.

Alcuni sistemi combinano l'assegnazione contigua con l'assegnazione indicizzata, usando quella contigua per i file piccoli (fino a tre o quattro blocchi) e passando automaticamente a quella indicizzata per i file grandi. Poiché generalmente i file sono piccoli, e in questo caso l'assegnazione contigua è efficiente, le prestazioni medie possono risultare abbastanza buone.

Nel 1991, ad esempio, la versione dello UNIX della Sun Microsystems è stata modificata per migliorare le prestazioni dell'algoritmo di assegnazione del file system. Alcune misure delle prestazioni hanno mostrato che la massima produttività del disco in una tipica stazione di lavoro (SPARCstation1, da 12 MIPS) richiedeva il 50 per cento d'impegno della CPU e produceva un'ampiezza di banda di soli 1,5 MB al secondo (56 KB era la massima dimensione del trasferimento per DMA di allora). Per migliorare le prestazioni, la Sun ha apportato alcune modifiche per assegnare lo spazio in gruppi di blocchi di 56 KB ogniqualvolta fosse possibile. Questo metodo di assegnazione riduceva la frammentazione esterna e i tempi di posizionamento e di latenza. Inoltre le procedure di lettura dei dischi sono state ottimizzate per leggere in questi grandi gruppi di blocchi. La struttura dell'*inode* non è stata modificata. Queste modifiche insieme con l'uso della lettura anticipata e del rilascio indietro (discussi nel Paragrafo 12.6.2) hanno prodotto una diminuzione del 25 per cento dell'impegno della CPU e una produttività notevolmente migliorata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Sono possibili e si usano effettivamente anche molte altre ottimizzazioni. Data la disparità tra la velocità della CPU e la velocità dei dischi, non è irragionevole aggiungere al sistema operativo migliaia di istruzioni solo per risparmiare alcuni movimenti della testina. Col passare del tempo tale disparità aumenta a tal punto che, per ottimizzare i movimenti della testina, si possono ragionevolmente usare centinaia di migliaia di istruzioni.

12.5 Gestione dello spazio libero

Poiché la quantità di spazio dei dischi è limitata, è necessario riutilizzare lo spazio lasciato dai file cancellati per scrivere, se è possibile, nuovi file (i dischi ottici a una sola scrittura permettono una sola scrittura in qualsiasi settore e quindi il riutilizzo è fisicamente impossibile). Per tenere traccia dello spazio libero in un disco, il sistema conserva un **elenco dei blocchi liberi**; vi sono registrati tutti i blocchi *liberi*, cioè non assegnati ad alcun file o directory. Per creare un file occorre cercare nell'elenco dei blocchi liberi la quantità di spazio necessaria e assegnarla al nuovo file, quindi rimuovere questo spazio dall'elenco dei blocchi liberi. Quando si cancella un file, si aggiungono all'elenco dei blocchi liberi i blocchi di disco a esso assegnati.

12.5.1 Vettore di bit

Spesso l'elenco dei blocchi liberi si realizza come una **mappa di bit**, o **vettore di bit**. Ogni blocco è rappresentato da un bit: se il blocco è libero il bit è 1, se il blocco è assegnato il bit è 0.

Si consideri, ad esempio, un disco dove i blocchi 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 e 27 sono liberi e gli altri sono assegnati. La mappa di bit dello spazio libero è la seguente:

001111001111110001100000011100000...

I vantaggi principali che derivano da questo metodo sono la sua relativa semplicità ed efficienza nel trovare il primo blocco libero o *n* blocchi liberi consecutivi nel disco; in effetti, molti calcolatori forniscono istruzioni di manipolazione dei bit che si possono usare con efficacia a tale scopo. Ad esempio, la famiglia di CPU Intel a partire dall'80386 e la famiglia di CPU Motorola a partire dal 68020 (rispettivamente, nei PC e nei vecchi Macintosh) hanno istruzioni che riportano lo scostamento del primo bit con valore 1 contenuto in una parola. Infatti il sistema operativo Apple Macintosh usava il metodo del vettore di bit per assegnare lo spazio dei dischi. Per trovare il primo blocco libero, il sistema operativo Macintosh controllava in modo sequenziale ogni parola nella mappa di bit per verificare che il valore non fosse 0, poiché una parola con valore 0 ha tutti i bit a 0 e rappresenta un insieme di blocchi assegnati. La prima parola non 0 viene scandita alla ricerca del primo bit 1, che indica la locazione del primo blocco libero. Il numero del blocco è dato dalla seguente espressione:

(*numero di bit per parola*) × (*numero di parole di valore 0*) + *scostamento del primo bit* 1.

Anche in questo caso le caratteristiche dell'architettura guidano le funzioni del sistema operativo. Sfortunatamente, i vettori di bit sono efficienti solo se tutto il vettore è mantenuto nella memoria centrale, e viene di tanto in tanto scritto nella memoria secondaria allo scopo di consentire eventuali operazioni di ripristino; è possibile tenere il vettore nella memoria centrale se i dischi sono piccoli, come quelli usati nei microcalcolatori; tale soluzione non è applicabile ai dischi più grandi. Un disco di 1,3 GB con blocchi di 512 byte richiederebbe una mappa di bit di oltre 332 KB per tenere traccia dei suoi blocchi liberi. Il raggruppamento di quattro blocchi riduce questo numero a 83 KB per disco.

12.5.2 Lista concatenata

Un altro metodo di gestione dei blocchi liberi consiste nel collegare tutti i blocchi liberi, tenere un puntatore al primo di questi in una speciale locazione del disco e caricarlo nella memoria. Questo primo blocco contiene un puntatore al successivo blocco libero, e così via. Nell'esempio considerato (Paragrafo 12.5.1) si deve tenere un puntatore al blocco 2, che è il primo blocco libero. Il blocco 2 contiene un puntatore al blocco 3 che a sua volta punta al blocco 4 e questo al blocco 5 e così via (Figura 12.10). Questo schema non è comunque efficiente: per attraversare la lista occorre leggere ogni blocco, e l'operazione richiede un notevole tempo di I/O. Fortunatamente, l'attraversamento dell'elenco dei blocchi liberi non è un'operazione frequente. Di solito il sistema operativo ha semplice-

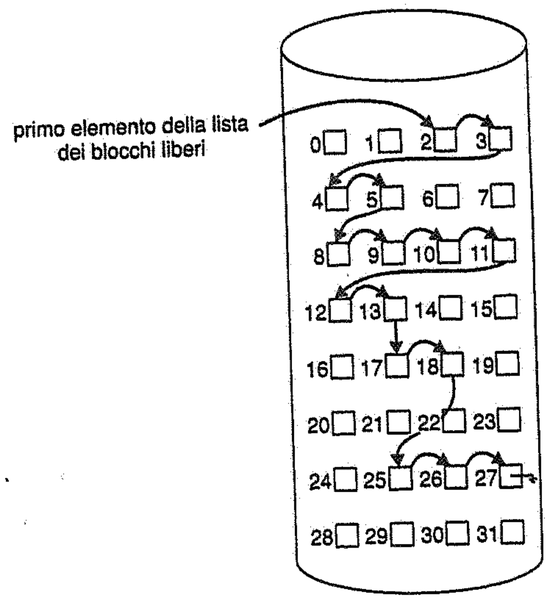


Figura 12.10 Lista concatenata dei blocchi liberi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, get complete version at www.tiffpdf.com/mark, please register this software.

mente bisogno di un blocco libero perché possa assegnarlo a un file, quindi si usa il primo blocco dell'elenco dei blocchi liberi. Il metodo che fa uso della FAT include il conteggio dei blocchi liberi nella struttura di dati per l'assegnazione; non è necessario un metodo separato.

12.5.3 Raggruppamento

Una possibile modifica del metodo dell'elenco dei blocchi liberi prevede la memorizzazione degli indirizzi di n blocchi liberi nel primo di questi. I primi $n - 1$ di questi blocchi sono effettivamente liberi; l'ultimo blocco contiene gli indirizzi di altri n blocchi liberi, e così via. L'importanza di questo metodo, diversamente dall'ordinaria lista concatenata, è data dalla possibilità di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi.

12.5.4 Conteggio

Un altro orientamento sfrutta il fatto che, generalmente, più blocchi contigui si possono assegnare o liberare contemporaneamente, soprattutto quando lo spazio si assegna usando l'algoritmo di assegnazione contigua o attraverso l'uso di gruppi di blocchi. Quindi, anziché tenere un elenco di n indirizzi liberi, è sufficiente tenere l'indirizzo del primo blocco libero e il numero n di blocchi liberi contigui che seguono il primo blocco. Ogni elemento dell'elenco dei blocchi liberi è formato da un indirizzo del disco e un contatore. Anche se ogni elemento richiede più spazio di quanto ne richieda un semplice indirizzo del disco, se il contatore è generalmente maggiore di 1 l'elenco complessivo è più corto.

12.6 Efficienza e prestazioni

Dopo avere descritto le opzioni di assegnazione dei blocchi e di gestione delle directory, è possibile considerare i loro effetti sulle prestazioni e l'efficienza d'uso dei dischi. I dischi tendono di solito a essere la principale strozzatura per le prestazioni di un sistema, essendo i più lenti tra i componenti principali di un calcolatore. In questo paragrafo si considerano diverse tecniche utili per migliorare l'efficienza e le prestazioni della memoria secondaria.

12.6.1 Efficienza

L'uso efficiente di un disco dipende fortemente dagli algoritmi usati per l'assegnazione del disco e la gestione delle directory. Ad esempio, gli *inode* dello UNIX sono assegnati preventivamente in una partizione. Anche un disco 'vuoto' impiega una certa percentuale del suo spazio per gli *inode*. D'altra parte, l'assegnazione preventiva degli *inode* e la loro distribuzione per la partizione migliorano le prestazioni del file system. Queste migliori prestazioni sono il risultato degli algoritmi di assegnazione e di gestione dei blocchi

liberi adottati dallo UNIX, i quali cercano di mantenere i blocchi di dati di un file vicini al blocco che ne contiene l'*inode* allo scopo di ridurre il tempo di posizionamento.

Come ulteriore esempio, si consideri lo schema che fa uso dei gruppi di blocchi discusso nel Paragrafo 12.4.2, che migliora le prestazioni di posizionamento in un file e trasferimento di un file al costo di una maggiore frammentazione interna. Per ridurre questa frammentazione, il BSD UNIX varia la dimensione del gruppo di blocchi al crescere della dimensione del file. Gruppi più grandi si usano dove possono essere riempiti, mentre gruppi di blocchi più piccoli si usano per l'ultimo gruppo di blocchi di un file e i file di piccole dimensioni; questo sistema è descritto nell'Appendice A (on-line).

Anche il tipo di dati normalmente contenuti in un elemento di una directory (o di un *inode*) si devono tenere in considerazione. Di solito si memorizza la *data dell'ultima scrittura* per fornire informazioni all'utente e per determinare se per il file occorre la creazione o l'aggiornamento di una copia di riserva. Alcuni sistemi mantengono anche la *data dell'ultimo accesso* per consentire all'utente di risalire all'ultima volta che un file è stato letto. Per mantenere queste informazioni, ogniqualvolta si legge un file, si deve aggiornare un campo della directory. Questa modifica richiede la lettura nella memoria del blocco, la modifica della sezione e la riscrittura del blocco nel disco, poiché sui dischi si può operare solamente per blocchi (o gruppi di blocchi). Quindi, ogni volta che si apre un file per la lettura, si deve leggere e scrivere anche l'elemento della directory a esso associato. Ciò può essere inefficiente per file cui si accede frequentemente, quindi nella fase della progettazione del file system è necessario confrontare i benefici con i costi rispetto alle prestazioni. In generale, è necessario considerare l'influenza sull'efficienza e sulle prestazioni di *ogni* informazione che si vuole associare a un file.

A titolo d'esempio, si consideri come l'efficienza sia influenzata dalle dimensioni dei puntatori usati per l'accesso ai dati. La maggior parte dei sistemi usa puntatori di 16 o 32 bit ovunque all'interno del sistema operativo. Questa dimensione limita la lunghezza di un file a 2^{16} (64 KB) o 2^{32} byte (4 GB). Alcuni sistemi impiegano puntatori di 64 bit per portare il limite a 2^{64} byte, un numero effettivamente molto grande. Comunque, i puntatori di 64 bit richiedono più spazio per la loro memorizzazione e di conseguenza fanno sì che i metodi di assegnazione e di gestione dello spazio libero (liste concatenate, indici, e così via) impieghino più spazio.

Una delle difficoltà nella scelta della dimensione dei puntatori, o di qualsiasi altra dimensione di assegnazione fissa all'interno di un sistema operativo, è la pianificazione degli effetti provocati dal cambiamento della tecnologia. Basti considerare che il primo IBM PC XT aveva un disco di 10 MB e che il file system dell'MS-DOS poteva gestire solamente 32 MB. (Ciascun elemento della FAT era di 12 bit e puntava a un gruppo di blocchi di 8 KB.) Con la crescita della capacità dei dischi, i dischi più grandi si dovevano suddividere in partizioni di 32 MB, poiché il file system non poteva tenere traccia di blocchi disposti oltre i 32 MB. Quando divennero comuni dischi di capacità superiore ai 100 MB, si dovettero modificare le strutture di dati e gli algoritmi usati dall'MS-DOS per gestire i dischi in modo da consentire file system più grandi. (La dimensione di ciascun elemento della FAT fu portata a 16 bit e più tardi a 32 bit.) La decisione iniziale fu presa per mo-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

tivi di efficienza; comunque, con l'avvento della Versione 4 dell'MS-DOS milioni di utenti si trovarono a disagio quando dovettero passare ai nuovi, più grandi file system.

Come altro esempio, si consideri l'evoluzione del sistema operativo Solaris della Sun Microsystems. Originariamente molte strutture di dati avevano lunghezza fissa ed erano assegnate all'avviamento del sistema. Queste strutture comprendevano la tabella dei processi e quella dei file aperti. Una volta riempite queste tabelle, non si potevano più creare nuovi processi o aprire nuovi file, sicché il sistema non riusciva nel proprio compito di fornire un servizio agli utenti. L'unico modo di aumentare le dimensioni di queste tabelle era la ricompilazione del nucleo e il riavvio del sistema. Dall'uscita del Solaris 2 quasi tutte le strutture di dati del nucleo si assegnano in modo dinamico, eliminando questi limiti artificiali alle prestazioni del sistema. Naturalmente, gli algoritmi che manipolano queste tabelle sono ora più complessi e il sistema operativo è un po' più lento dovendo assegnare e rilasciare dinamicamente gli elementi delle tabelle, ma questo è il prezzo da pagare per la generalizzazione delle funzioni.

12.6.2 Prestazioni

Dopo aver scelto gli algoritmi fondamentali del file system si possono migliorare le prestazioni in diversi modi. Come si osserva nel Capitolo 2, alcuni controllori di unità a disco contengono una quantità di memoria locale sufficiente per la creazione di una cache interna al controllore che può essere sufficientemente grande da memorizzare un'intera traccia del disco alla volta. Eseguito il posizionamento della testina, si legge la traccia nella cache del controllore del disco a partire dal settore sotto cui si viene a trovare la testina (riducendo il tempo di latenza). Il controllore trasferisce quindi al sistema operativo tutte le richieste di settori. Quando i blocchi sono trasferiti dal controllore del disco alla memoria centrale, il sistema operativo ha la possibilità di inserirli in una propria cache nella memoria centrale.

Alcuni sistemi riservano una sezione separata della memoria centrale come cache del disco, dove tenere i blocchi in previsione di un loro riutilizzo entro breve tempo. Altri sistemi impiegano una **cache delle pagine** per i file; si tratta di una soluzione che impiega tecniche di memoria virtuale per la gestione dei dati dei file come pagine anziché come blocchi di file system; l'uso degli indirizzi virtuali è molto più efficiente dell'uso dei blocchi fisici di disco. Diversi sistemi, compreso il Solaris, alcune nuove versioni del LINUX e Windows NT e 2000, usano le cache delle pagine, sia per le pagine relative ai processi sia per i dati dei file. Questo metodo è noto come **memoria virtuale unificata**. Il sistema operativo Solaris usa sia una cache basata su blocchi sia una basata su pagine: la prima per i metadati del file system (come gli *inode*); la seconda per tutti i dati del file system.

Alcune versioni dello UNIX prevedono la cosiddetta *buffer cache unificata*. Si considerino le due possibilità per aprire un file e accedervi: l'uso dell'associazione alla memoria (Paragrafo 10.3.2) e l'uso delle ordinarie chiamate del sistema `read` e `write`. Senza una *buffer cache unificata*, si verifica una situazione simile a quella illustrata nella Figura

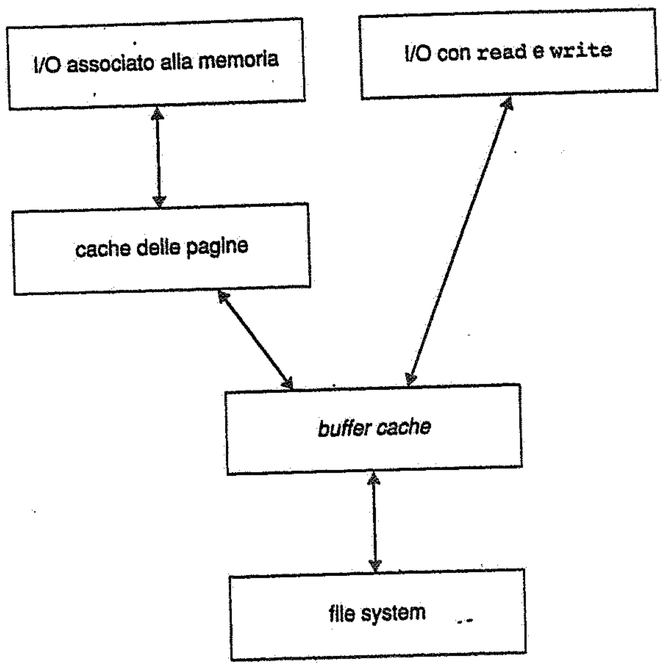


Figura 12.11 I/O senza una *buffer cache* unificata.

12.11. In questo caso, le chiamate del sistema `read` e `write` passano attraverso la *buffer cache*. La chiamata con associazione alla memoria richiede l'uso di due cache, la cache delle pagine e la *buffer cache*. L'associazione alla memoria prevede la lettura dei blocchi di disco dal file system e la loro memorizzazione nella *buffer cache*. Poiché il sistema di memoria virtuale non può interfacciarsi con la *buffer cache*, si deve copiare nella cache delle pagine il contenuto del file presente nella *buffer cache*. Questa situazione è nota come *double caching* proprio perché i dati del file system richiedono un doppio passaggio di cache. Non solo ciò comporta uno spreco di memoria, ma anche uno spreco di cicli della CPU e di I/O dovuti a un ulteriore trasferimento di dati nella memoria del sistema. Inoltre, eventuali incoerenze tra le due cache possono generare errori nella memorizzazione dei dati nei file. Con una *buffer cache unificata*, sia l'associazione alla memoria sia le chiamate del sistema `read` e `write` usano la stessa cache delle pagine, col vantaggio di evitare il *double caching* e di permettere al sistema di memoria virtuale di gestire dati del file system. La Figura 12.12 illustra l'uso della *buffer cache unificata*.

Indipendentemente dalla gestione delle cache per blocchi di disco oppure per pagine, l'algoritmo LRU è in generale ragionevole per la sostituzione dei blocchi o delle pagine. Tuttavia, l'evoluzione degli algoritmi di gestione delle cache delle pagine usati dal sistema operativo Solaris rivela le difficoltà nella scelta di un algoritmo ottimale. Tale

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this watermark, please register this software.

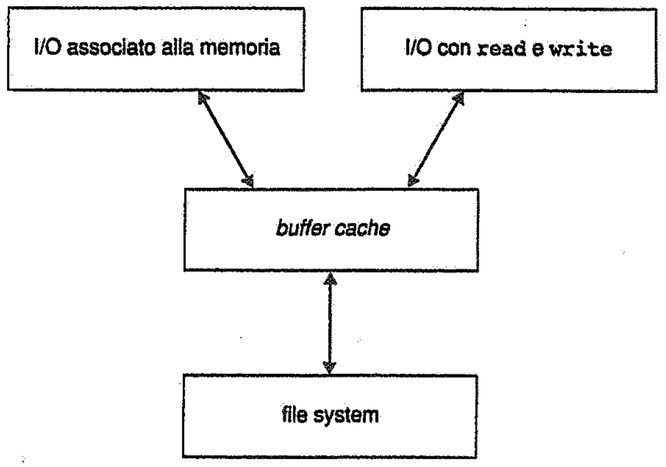


Figura 12.12 I/O con una *buffer cache unificata.*

sistema operativo permette ai processi e alla cache delle pagine di condividere la memoria inutilizzata; prima della versione 2.5.1, non si facevano distinzioni tra l'assegnazione delle pagine a un processo o alla cache delle pagine, con la conseguenza che un sistema che eseguiva molte operazioni di I/O usava la maggior parte della memoria disponibile per la cache delle pagine. A causa dell'alta frequenza delle operazioni di I/O, quando la memoria libera diventa troppo esigua, il modulo di scansione delle pagine (Paragrafo 10.7.2) sottrae pagine ai processi anziché alla cache delle pagine. Nel Solaris 2.6 e nel Solaris 7 è stata realizzata in forma opzionale la tecnica di *paginazione con priorità*, secondo la quale il modulo di scansione delle pagine dà la priorità alle pagine dei processi rispetto a quelle della cache delle pagine. Il sistema operativo Solaris 8 ha aggiunto un limite prefissato tra pagine dei processi e cache delle pagine per il file system, impedendo a ciascun meccanismo di sottrarre totalmente la memoria all'altro.

La cache delle pagine, il file system e i driver dei dischi presentano interazioni interessanti. Quando si scrivono dati in un file presente in un disco, si memorizzano transitoriamente le pagine nella cache e il driver del disco ordina la sua coda di emissione dei dati secondo gli indirizzi del disco. Queste due azioni permettono al driver di disco di ridurre al minimo il tempo di ricerca per la testina del disco, e di scrivere i dati secondo una sequenza che sfrutti la rotazione del disco stesso. Sempre che non siano richieste scritture sincrone, un processo che scrive in un disco in realtà scrive semplicemente nella cache, successivamente il sistema scrive i dati nel disco in modo asincrono, quando è più conveniente. Il processo utente percepisce quindi scritture molto rapide. Quando i dati sono letti da un file presente in un disco, il sistema di I/O compie alcune letture anticipate; tuttavia, nelle scritture c'è un maggior grado di asincronismo che nelle letture. Per questo motivo, per i trasferimenti di grandi dimensioni, contrariamente a quel che si

potrebbe intuire, le scritture nei dischi attraverso il file system sono spesso assai più rapide delle letture.

Le **scritture sincrone** avvengono nell'ordine in cui le riceve il sottosistema per la gestione del disco e non subiscono la memorizzazione transitoria. Quindi la procedura chiamante prima di proseguire deve attendere che i dati raggiungano l'unità a disco. Nella maggior parte dei casi si usano **scritture asincrone**. Nelle scritture asincrone si memorizzano i dati nella cache e si restituisce immediatamente il controllo alla procedura chiamante. Le scritture dei metadati, tra le altre, possono essere sincrone. I sistemi operativi spesso includono un indicatore nella chiamata del sistema open per permettere a un processo di richiedere che le operazioni di scrittura si eseguano in modo sincrono. I sistemi di gestione delle basi di dati ad esempio usano questa funzione per realizzare le transazioni atomiche, in modo da assicurare che i dati raggiungano la memoria stabile nell'ordine richiesto.

Alcuni sistemi ottimizzano la cache delle pagine adottando, secondo il tipo d'accesso ai file, differenti algoritmi di sostituzione. Le pagine relative a un file da leggere o scrivere in modo sequenziale non si dovrebbero sostituire nell'ordine LRU, infatti la pagina usata più di recente sarà usata nuovamente per ultima, o forse mai. Gli accessi sequenziali si potrebbero invece ottimizzare con tecniche note come rilascio indietro e lettura anticipata. Il **rilascio indietro** (*free-behind*) rimuove una pagina dalla memoria di transito non appena si verifica una richiesta della pagina successiva; le pagine precedenti con tutta probabilità non saranno più usate e quindi sprecano spazio nella memoria di transito. Con la **lettura anticipata** (*read-ahead*) si leggono e si mettono nella cache la pagina richiesta e parecchie pagine successive: è probabile che queste pagine siano richieste una volta terminata l'elaborazione della pagina corrente. Il recupero di questi dati dal disco con un unico trasferimento e la memorizzazione nella cache consentono di risparmiare una quantità di tempo considerevole. La presenza nel controllore di una cache per le tracce non elimina la necessità di adottare la tecnica di lettura anticipata in un sistema multiprogrammato, ciò a causa dell'elevata latenza e del sovraccarico determinato dai tanti piccoli trasferimenti dalla cache per le tracce alla memoria centrale.

Nei PC si adotta comunemente un'altra tecnica per migliorare le prestazioni. Si riserva e si gestisce una sezione della memoria come un **disco virtuale** o **disco RAM**; in questo caso il driver di un disco RAM accetta tutte le operazioni ordinarie dei dischi, eseguendole però in questa sezione della memoria invece che su un disco. Tutte le operazioni su disco si possono eseguire su questo disco RAM senza che gli utenti se ne accorgano, se non per la velocità eccezionalmente elevata. Sfortunatamente i dischi RAM sono utili solamente come mezzo di memorizzazione temporanea, poiché la mancanza di alimentazione elettrica o il riavvio del sistema di solito ne cancellano il contenuto; in essi di solito si memorizzano i file temporanei, come i file intermedi di compilazione.

La differenza tra un disco RAM e la cache di un disco è che il contenuto del primo è totalmente controllato dall'utente, mentre quello della cache è sotto il controllo del sistema operativo. Ad esempio, un disco RAM resterà vuoto finché un utente (o un programma su richiesta dell'utente) non vi creerà un file. La Figura 12.13 illustra le possibili collocazioni delle cache in un sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

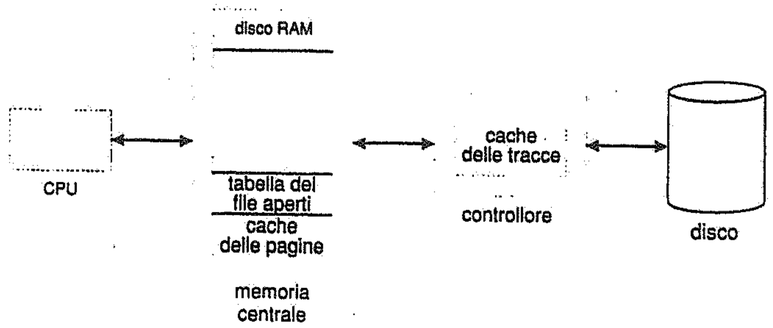


Figura 12.13 Diverse locazioni di cache per i dischi.

12.7 Ripristino

Poiché i file e le directory sono mantenuti sia nella memoria centrale sia nei dischi, è necessario aver cura di assicurare che il verificarsi di un malfunzionamento nel sistema non comporti la perdita di dati o la loro incoerenza.

12.7.1 Verifica della coerenza

Come si discute nel Paragrafo 12.3, una parte delle informazioni contenute nelle directory è mantenuta nella memoria centrale (cache) allo scopo di accelerarne gli accessi. Queste informazioni sono generalmente più aggiornate delle corrispondenti informazioni presenti nella memoria secondaria, poiché la scrittura nei dischi dei dati contenuti nella cache non si verifica necessariamente nell'istante in cui occorrono le modifiche. Si considerino i possibili effetti di un crollo del sistema; in questo caso la tabella dei file aperti va generalmente persa e con essa qualsiasi modifica alle directory alle quali i file aperti appartengono. Questo evento può lasciare il file system in uno stato di incoerenza: l'effettivo stato di alcuni file non equivale a quello descritto dalle directory. Di solito, durante il riavvio del sistema si esegue uno speciale programma che ha il compito di individuare e correggere le incoerenze presenti nei dischi.

Il **verificatore della coerenza** confronta i dati delle directory con quelli contenuti nei blocchi dei dischi, tentando di correggere ogni incoerenza. Gli algoritmi di assegnazione e di gestione dello spazio libero determinano il genere di problemi che questo programma può riconoscere e con quanto successo riuscirà a risolverli. Ad esempio, se si adotta uno schema di assegnazione concatenata con un puntatore da ciascun blocco al successivo, si può ricostruire l'intero file e ricreare il corrispondente elemento nella directory analizzando i blocchi di dati. Diversamente, la perdita di un elemento di una directory in un sistema ad assegnazione indicizzata potrebbe essere disastrosa, ogni blocco di dati non contiene alcuna informazione sugli altri blocchi di dati. Per questo motivo, lo UNIX gestisce tramite cache gli elementi delle directory per le letture, mentre qualsiasi operazione di scrittura di dati che produca l'assegnazione di spazio, o altre modifiche dei metadati, è svolta in modo sincrono, prima della scrittura dei corrispondenti blocchi di dati.

12.7.2 Copie di riserva e recupero dei dati

Poiché si possono verificare malfunzionamenti e perdite di dati anche nei dischi magnetici, è necessario provvedere affinché i dati non vadano persi definitivamente. A questo scopo si possono usare programmi di sistema che consentano di fare delle **copie di riserva** (*backup*) dei dati residenti nei dischi in altri dispositivi di registrazione di dati, come i dischetti, i nastri magnetici o i dischi ottici. Il ripristino della situazione antecedente la perdita di un singolo file, o del contenuto di un intero disco, richiederà il **recupero** (*restore*) dei dati dalle copie di riserva.

Al fine di ridurre al minimo la quantità di dati da copiare, è possibile sfruttare le informazioni contenute nell'elemento della directory associato a ogni file. Ad esempio, se il programma di creazione delle copie di riserva sa quando è stata eseguita l'ultima copia di riserva di un file, e se la data di ultima scrittura di quel file, registrata nella directory, indica che il file da quel momento non ha subìto variazioni, non sarà necessario copiare nuovamente il file. Quella che segue è una tipica sequenza di gestione delle copie di riserva:

- **Giorno 1.** Copiatura nel mezzo di registrazione delle copie di riserva di tutti i file contenuti nel disco; detta **copiatura completa**.
- **Giorno 2.** Copiatura su un altro mezzo di tutti i file modificati dal Giorno 1; si tratta di una **copiatura incrementale**.
- **Giorno 3.** Copiatura su un altro mezzo di tutti i file modificati dal Giorno 2.

  .

  .

  .

- **Giorno *n*.** Copiatura su un altro mezzo di tutti i file modificati dal Giorno $n - 1$. Ritorno al Giorno 1.

Il nuovo ciclo può comportare la scrittura delle nuove copie di riserva nel primo insieme di mezzi di registrazione, oppure in un nuovo insieme; in questo modo si ha la possibilità di recuperare il contenuto dell'intero disco iniziando le operazioni di recupero dalla copia di riserva completa e proseguendo con le copie di riserva incrementali. Naturalmente, più grande è n, maggiore sarà il numero di nastri o dischi da leggere per un completo recupero. Un ulteriore vantaggio di questo ciclo di creazione di copie di riserva è la possibilità di recuperare qualsiasi file accidentalmente cancellato durante il ciclo, recuperandolo dalle copie del giorno precedente. La lunghezza del ciclo è un compromesso

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

tra la quantità di mezzi di registrazione delle copie di riserva necessari e il numero di giorni addietro da cui si può compiere un'operazione di recupero.

Un utente potrebbe accorgersi dopo molto tempo che si è perso o si è danneggiato un particolare file. Per questa ragione si è soliti pianificare di tanto in tanto una copiatura completa che sarà conservata in modo permanente; quindi il mezzo che la contiene non sarà riutilizzato. È inoltre opportuno conservare tali copie di riserva permanenti lontano dalle copie ordinarie per proteggerle dai vari pericoli, ad esempio un incendio che può distruggere il calcolatore e tutte le copie di riserva. Se il ciclo di creazione delle copie di riserva prevede il reimpiego dei mezzi che le contengono, è anche necessario aver cura di non usarli troppe volte: se dovessero logorarsi, potrebbe essere impossibile ripristinare i dati in essi contenuti.

12.8 File system con annotazione delle modifiche

Spesso nell'informatica si adottano algoritmi e tecnologie anche in aree diverse da quelle per le quali sono stati progettati. È il caso degli algoritmi per il ripristino, sviluppati nell'area dei sistemi di gestione delle basi di dati, basati sulla registrazione delle modifiche, descritti nel Paragrafo 7.9.2. Questi algoritmi sono stati applicati con successo al problema della verifica della coerenza, realizzando i **file system orientati alle transazioni e basati sull'annotazione delle modifiche** (*log-based transaction-oriented file system*), noti anche come **file system annotati** (*journaling file system*).

Si ricordi che le strutture di dati del file system che risiedono nei dischi, come le strutture di directory, i puntatori ai blocchi liberi, i puntatori ai descrittori di file liberi, possono diventare incoerenti nel caso di un crollo del sistema. Prima dell'uso nei sistemi operativi delle tecniche basate sulla registrazione delle modifiche, i cambiamenti si applicavano direttamente a queste strutture. Un'operazione comune, come la creazione di un file, può implicare diversi cambiamenti strutturali nel file system di un disco: si modificano le strutture di directory, si assegnano descrittori di file, blocchi di dati e si decrementano i corrispondenti contatori degli elementi liberi. Questi cambiamenti possono essere interrotti da un crollo del sistema, rendendo le strutture incoerenti. Ad esempio, il contatore dei descrittori di file liberi potrebbe indicare che un descrittore di file è stato assegnato, ma la struttura di directory potrebbe non avere un puntatore a quel descrittore di file. Se non ci fosse una fase di verifica della coerenza, quest'ultimo si perderebbe.

Il metodo che consente l'incoerenza delle strutture per poi correggere gli errori in una fase di ripristino presenta diversi problemi. Uno di questi è che l'incoerenza potrebbe non essere risolvibile; il sistema di verifica della coerenza potrebbe non riuscire a ripristinare le strutture, con il risultato della perdita di file o addirittura di directory. La verifica della coerenza può richiedere l'intervento dell'amministratore di sistema per risolvere i conflitti, e questo può diventare un aspetto critico se l'amministratore non è disponibile. Il sistema può rimanere inutilizzabile fino a quando l'amministratore non interviene indicando al sistema come procedere. La verifica della coerenza richiede anche molto tempo e molte risorse; ad esempio, la verifica di terabyte di dati può richiedere alcune ore.

La soluzione a questo problema consiste nell'applicare agli aggiornamenti dei metadati relativi al file system metodi di ripristino basati sulla registrazione delle modifiche. Sia il file system NTFS sia il Veritas usano questo metodo, che è anche opzionale rispetto al file system UFS nel Solaris 7 e nelle versioni successive. In realtà, sta diventando un metodo comune in molti sistemi operativi.

Fondamentalmente, tutte le modifiche dei metadati si annotano in modo sequenziale in un file di registrazione, il *giornale delle modifiche*. Ogni insieme di operazioni che esegue uno specifico compito si chiama **transazione**. Una volta che le modifiche sono riportate nel file di registrazione, le operazioni si considerano portate a termine con successo (*committed*) e la chiamata del sistema può restituire il controllo al processo utente, permettendogli di proseguire la sua esecuzione. Nel frattempo, si applicano alle effettive strutture del file system le operazioni scritte nel giornale delle modifiche, e a mano a mano che si eseguono si aggiorna un puntatore che indica quali azioni sono state completate e quali sono ancora incomplete. Quando un'intera transazione è stata completata, se ne rimuovono le annotazioni dal giornale delle modifiche, che è in realtà una struttura di dati circolare. Il giornale delle modifiche si potrebbe mantenere in una sezione separata del file system, o anche in un disco separato. È più efficiente, anche se è più complesso, averlo sotto testine di lettura e scrittura separate poiché si riducono le situazioni di contesa della testina e i tempi di ricerca.

Se si verifica un crollo del sistema, nel giornale delle modifiche ci potranno essere zero o più transazioni. Le transazioni presenti non sono mai state ultimate nel file system, anche se il sistema operativo le definisce portate a termine con successo, e quindi si devono completare. Le transazioni si possono eseguire a partire dalla posizione corrente del puntatore fino al completamento, e le strutture del file system rimangono coerenti. L'unico problema che si può presentare è il caso in cui una transazione sia fallita (*aborted*), cioè non sia stata dichiarata terminata con successo prima del crollo del sistema. In questo caso, si devono annullare tutti i cambiamenti che erano stati applicati al file system dalla transazione, di nuovo mantenendo la coerenza del file system. Questo ripristino è tutto ciò che è necessario fare dopo un crollo del sistema, eliminando tutti i problemi concernenti la verifica della coerenza.

Un vantaggio indiretto dell'uso dell'annotazione in un disco degli aggiornamenti dei metadati è che gli aggiornamenti sono molto più rapidi di quelli che si applicano direttamente alle strutture di dati nei dischi. La ragione di questo miglioramento sta nel vantaggio, dal punto di vista delle prestazioni, dell'I/O ad accesso sequenziale rispetto a quello ad accesso diretto. Le onerose operazioni di scrittura dei metadati ad accesso diretto e sincrono si sostituiscono con molto meno gravose operazioni di scrittura sincrone ma ad accesso sequenziale nell'area di registrazione delle modifiche di un file system con annotazione delle modifiche. I cambiamenti determinati da quelle operazioni si riportano successivamente in modo asincrono nelle strutture appropriate nei dischi attraverso operazioni di scrittura ad accesso diretto. Il risultato complessivo è un significativo guadagno in termini di prestazioni per le operazioni orientate ai metadati, come la creazione e la cancellazione dei file.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

12.9 NFS

I file system di rete sono di particolare interesse; tipicamente integrano l'intera struttura di directory e l'interfaccia del sistema client. L'NFS è un buon esempio di file system di rete client-server ampiamente usato e ben realizzato, che in questo paragrafo s'impiega per esplorare i particolari che caratterizzano la realizzazione dei file system di rete.

L'NFS è sia una realizzazione sia una definizione di un sistema per l'accesso a file remoti attraverso LAN, o anche WAN. Fa parte dell'ONC+, adottato dalla maggior parte dei distributori del sistema operativo UNIX e in alcuni sistemi operativi per PC. La versione qui descritta fa parte del sistema operativo Solaris, che è a sua volta una versione modificata dello UNIX SVR4, delle stazioni di lavoro Sun e altre architetture. Esso usa i protocolli TCP/IP o i protocolli UDP/IP (secondo la rete di comunicazione). La definizione e realizzazione dell'NFS, nella sua descrizione fornita in questa sede, si accavallano: per ogni descrizione dettagliata si fa riferimento alla versione realizzata dalla Sun; mentre ogni descrizione sufficientemente generale vale anche per la sua definizione.

12.9.1 Generalità

Nel contesto dell'NFS si considera un insieme di stazioni di lavoro interconnesse come un insieme di calcolatori indipendenti con file system indipendenti. Lo scopo è quello di permettere un certo grado di condivisione tra questi file system, su richiesta esplicita, in modo trasparente. La condivisione è basata su una relazione client-server. Un calcolatore può essere, come spesso accade, sia un client sia un server. La condivisione è ammessa tra ogni coppia di calcolatori, anziché essere limitata ai calcolatori aventi la specifica funzione di server. Per assicurare l'indipendenza dei calcolatori, la condivisione di un file system remoto ha effetto esclusivamente sul calcolatore client.

Affinché una directory remota sia accessibile in modo trasparente a un calcolatore particolare, ad esempio C_1, un client di quel calcolatore deve prima eseguire un'operazione di montaggio. La semantica dell'operazione consiste nel fatto che una directory remota si monta in corrispondenza di una directory di un file system locale. Una volta completata l'operazione di montaggio, la directory montata assume l'aspetto di un sottoalbero integrante del file system locale, e sostituisce il sottoalbero che discende dalla directory locale; questa, a sua volta, rappresenta la radice della directory appena montata. La directory remota si specifica come argomento dell'operazione di montaggio in modo esplicito; si deve fornire la locazione (o il nome del calcolatore) della directory remota. Tuttavia, da questo momento in poi gli utenti del calcolatore C_1 possono accedere ai file della directory remota in modo del tutto trasparente.

Per illustrare il montaggio dei file system, si consideri il file system riportato nella Figura 12.14, dove i triangoli rappresentano i sopraccitati sottoalberi di directory. La figura illustra tre file system di calcolatori indipendenti chiamati U, S_1 e S_2. A questo punto, in ogni calcolatore si può accedere solo a file locali. Nella Figura 12.15(a), sono riportati gli effetti del montaggio di `S1:/usr/shared` in `U:/usr/local`. In questa figura è illustrata la visione che gli utenti di U hanno del loro file system. Occorre osservare che, una volta completato il montaggio, essi possono accedere a qualsiasi file che si trovi

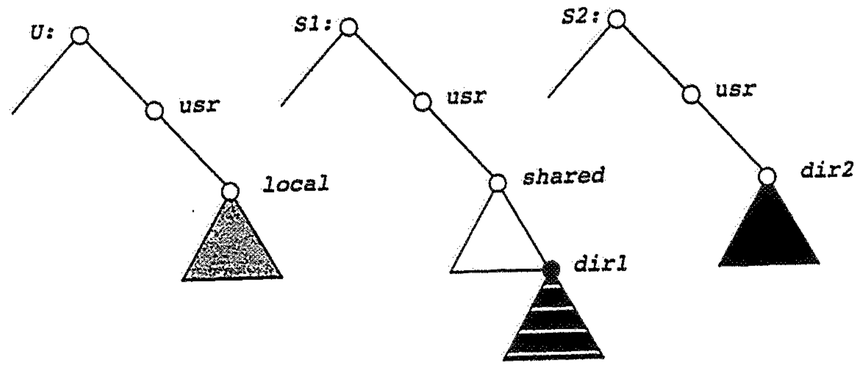


Figura 12.14 Tre file system indipendenti.

nella directory `dir1`, ad esempio usando il prefisso `/usr/local/dir1` in U. La directory originale `/usr/local` di quel calcolatore non è più visibile.

Potenzialmente ogni file system o ogni directory in un file system, nel rispetto dei diritti d'accesso, si possono montare in modo remoto in cima a qualsiasi directory locale. Le stazioni di lavoro prive di dischi possono montare i propri file system prelevandoli dai server.

In alcune versioni dell'NFS sono permessi anche i montaggi a cascata; ciò significa che un file system si può montare in corrispondenza di un altro file system montato in modo remoto. Un calcolatore è tuttavia sottoposto ai soli montaggi da esso richiesti.

Montando un file system remoto, il client non acquisisce l'accesso ai file system che erano montati sopra il primo file system; così, il meccanismo di montaggio non ha la proprietà transitiva. Nella Figura 12.15(b) sono riportati i montaggi a cascata relativi all'e-

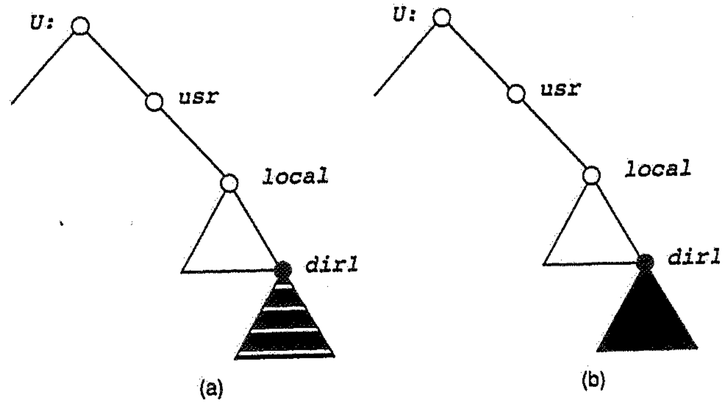


Figura 12.15 Montaggio nell'NFS: (a) montaggio; (b) montaggi a cascata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sempio precedente. Nella figura è riportato il risultato del montaggio di `S2:/usr/dir2` in `U:/usr/local/dir1`, che è già stato montato in modo remoto da S_1. Gli utenti di U possono accedere ai file di `dir2` usando il prefisso `/usr/local/dir1`. Se si monta un file system condiviso in corrispondenza delle directory iniziali degli utenti di tutti i calcolatori di una rete, un utente può aprire una sessione in qualsiasi stazione di lavoro e prelevare il proprio ambiente di lavoro iniziale. Questa proprietà permette la **mobilità dell'utente**.

Uno degli scopi nella progettazione dell'NFS era quello di operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete. La definizione dell'NFS è indipendente da questi mezzi e quindi incoraggia altre realizzazioni. Questa indipendenza si ottiene usando primitive RPC costruite su un protocollo di rappresentazione esterna dei dati (*external data representation* — XDR) usato tra due interfacce indipendenti dalla realizzazione. Quindi, se il sistema è formato da calcolatori e file system eterogenei adeguatamente interfacciati all'NFS, si possono montare file system di diversi tipi, sia localmente sia in modo remoto.

La definizione dell'NFS distingue tra i servizi offerti da un meccanismo di montaggio e gli effettivi servizi d'accesso ai file remoti. Di conseguenza, per questi servizi si definiscono due protocolli distinti: un protocollo di montaggio e un protocollo per gli accessi ai file remoti, il **protocollo NFS**. I protocolli sono definiti come insiemi di RPC, che sono gli elementi di base usati per realizzare, in modo trasparente, l'accesso remoto ai file.

12.9.2 Protocollo di montaggio

Il **protocollo di montaggio** stabilisce la connessione logica iniziale tra un server e un client. Nella versione della Sun Microsystems ogni calcolatore ha un processo server, esterno al nucleo, che esegue le funzioni del protocollo.

Un'operazione di montaggio comprende il nome della directory remota da montare e il nome del calcolatore server in cui tale directory è memorizzata. La richiesta di montaggio si associa alla RPC corrispondente e s'invia al server di montaggio in esecuzione nello specifico calcolatore server. Il server conserva una **lista di esportazione** (la `/etc/dfs/dfstab` nel sistema Solaris, che può essere modificata soltanto da un *superuser*) che specifica i file system locali esportati per il montaggio e i nomi dei calcolatori ai quali è permessa tale operazione. La lista può comprendere anche i diritti d'accesso, come la sola scrittura. Per semplificare la manutenzione delle liste di esportazione e delle tabelle di montaggio, si può usare uno schema distribuito di nominazione per contenere queste informazioni e renderle disponibili agli appropriati client.

Occorre ricordare che qualsiasi directory che si trovi all'interno di un file system esportato si può montare in modo remoto da un calcolatore accreditato. Di conseguenza, un'unità componente è rappresentata da una directory di questo tipo. Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client una **maniglia di file** (*file handle*) da usare come chiave per ulteriori accessi ai file che si trovano all'interno del file system montato. La maniglia di file contiene tutte le informazioni di cui ha bisogno il server per gestire un proprio file. Nei termini dell'ambiente UNIX, la maniglia di file è composta da un identificatore di file system e di un numero di *inode* per identificare la directory montata all'interno del file system esportato.

Il server contiene anche un elenco dei calcolatori client e delle corrispondenti directory correntemente montate. Questo elenco si usa soprattutto per scopi amministrativi, ad esempio per informare i client che un server sta andando fuori servizio. L'aggiunta o la cancellazione di elementi da questa lista sono gli unici modi in cui il protocollo dell'operazione di montaggio può modificare lo stato del server.

Generalmente un sistema ha una configurazione di montaggio predefinita che si stabilisce nella fase d'avviamento (`/etc/vfstab` nel Solaris); tale configurazione si può comunque modificare. Oltre alla procedura di montaggio effettiva, il protocollo di montaggio comprende numerose altre procedure, come lo smontaggio e la restituzione della lista d'esportazione.

12.9.3 Protocollo NFS

Il protocollo NFS offre un insieme di RPC per operazioni su file remoti che svolgono le seguenti operazioni:

- ricerca di un file in una directory;
- lettura di un insieme di elementi di una directory;
- manipolazione di collegamenti e di directory;
- accesso ad attributi di file;
- lettura e scrittura di file.

Queste procedure si possono invocare soltanto dopo aver stabilito una maniglia di file per la directory montata in modo remoto.

L'omissione delle operazioni `open` e `close` è intenzionale. Una caratteristica importante dei server NFS è l'*assenza dell'informazione di stato*. I server non conservano informazioni sui loro client da un accesso all'altro. Non esistono parallelismi con la tabella dei file aperti o le strutture di file dello UNIX da parte del server, quindi ogni richiesta deve fornire un insieme completo di argomenti, tra cui un identificatore unico di file e uno scostamento assoluto all'interno del file per svolgere le operazioni appropriate. La struttura che ne deriva è robusta; non si devono prendere misure speciali per ripristinare un server dopo un guasto. Per tale ragione, le operazioni sui file devono essere idempotenti. Ciascuna richiesta dell'NFS ha un numero di sequenza, che permette al server di determinare la duplicazione o la mancanza di richieste.

La presenza della suddetta lista di client sembra violare la proprietà dell'assenza di informazione di stato del server. Tuttavia, essa non è essenziale ai fini del corretto funzionamento del client o del server, quindi non è necessario recuperare tale lista dopo il crollo di un server; tale lista può contenere anche dati incoerenti e si considera come un semplice suggerimento.

Un'ulteriore implicazione della filosofia dei server senza informazione di stato e un risultato della sincronia di una RPC consiste nel fatto che i dati modificati, tra cui blocchi indiretti e di stato, si devono riscrivere nei dischi del server prima che i risultati siano riportati al client. Un client può cioè ricorrere a cache per i blocchi di scrittura, ma quan-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

do li invia al server, assume che abbiano raggiunto i dischi del server, che deve scrivere tutti i dati dell'NFS in modo sincrono. In questo modo il crollo di un server e il successivo ripristino saranno invisibili al client; tutti i blocchi che il server gestisce per il client resteranno intatti. La conseguente perdita di prestazioni può essere rilevante poiché si perdono i vantaggi derivanti dall'impiego di una cache. Le prestazioni si possono incrementare impiegando mezzi di memoria secondaria con una propria cache non volatile (di solito si tratta di memorie alimentate da una pila), Il controllore del disco riporta che la scrittura nel disco è avvenuta quando la scrittura è avvenuta nella cache non volatile. Essenzialmente, il calcolatore 'vede' una scrittura sincrona molto rapida. Questi blocchi restano intatti anche dopo un crollo del sistema, e periodicamente vengono trasferiti da tale memoria stabile al disco.

Di una singola chiamata di procedura di scrittura dell'NFS sono garantite l'atomicità e la non interferenza con altre chiamate di scrittura nello stesso file. Tuttavia, il protocollo NFS non fornisce meccanismi di controllo della concorrenza, e poiché ogni chiamata di scrittura o lettura dell'NFS può contenere non più di 8 KB di dati e i pacchetti UDP sono limitati a 1500 byte, può essere necessario dividere una chiamata del sistema `write` in diverse RPC di scrittura; quindi, due utenti che scrivono nello stesso file remoto possono riscontrare interferenze nei loro dati. Poiché la gestione di meccanismi di bloccaggio richiede informazioni di stato, si richiede che un servizio esterno all'NFS debba fornire tali meccanismi, come nel caso del Solaris. Gli utenti sanno che per coordinare l'accesso ai file condivisi devono usare meccanismi che sono oltre la portata dell'NFS.

L'NFS è integrato nel sistema operativo tramite un VFS. Per illustrarne l'architettura si può accennare al modo in cui si gestisce un'operazione su un file remoto già aperto (Figura 12.16). Il client inizia l'operazione con un'ordinaria chiamata del sistema. Lo strato del sistema operativo fa corrispondere tale chiamata del sistema a un'operazione del VFS sull'opportuno *vnode*. Lo strato del VFS identifica il file come remoto e invoca l'opportuna procedura dell'NFS. S'impiega una RPC allo strato di servizio dell'NFS del server remoto. Tale chiamata si reintroduce nello strato del VFS del sistema remoto, che riconosce essere locale e invoca l'appropriata operazione del file system. Questo cammino si ripercorre per riportare il risultato. Un vantaggio di questa architettura è che il client e il server sono identici; così un calcolatore può essere un client, un server o entrambi.

L'effettivo servizio in ciascun server si esegue tramite diversi processi del nucleo che offrono una temporanea sostituzione al meccanismo dei processi leggeri (o *thread*)

12.9.4 Traduzione dei nomi di percorso

La **traduzione dei nomi di percorso** si compie suddividendo il percorso stesso in nomi componenti ed eseguendo una chiamata `lookup` dell'NFS separata per ogni coppia formata da un nome componente e un *vnode* di directory. Quando s'incontra un punto di montaggio, la ricerca di un componente causa una RPC separata al server. Questo schema di attraversamento del nome di percorso è costoso ma necessario, poiché ogni client ha un'unica configurazione del proprio spazio di nomi logico, dettata dai montaggi che

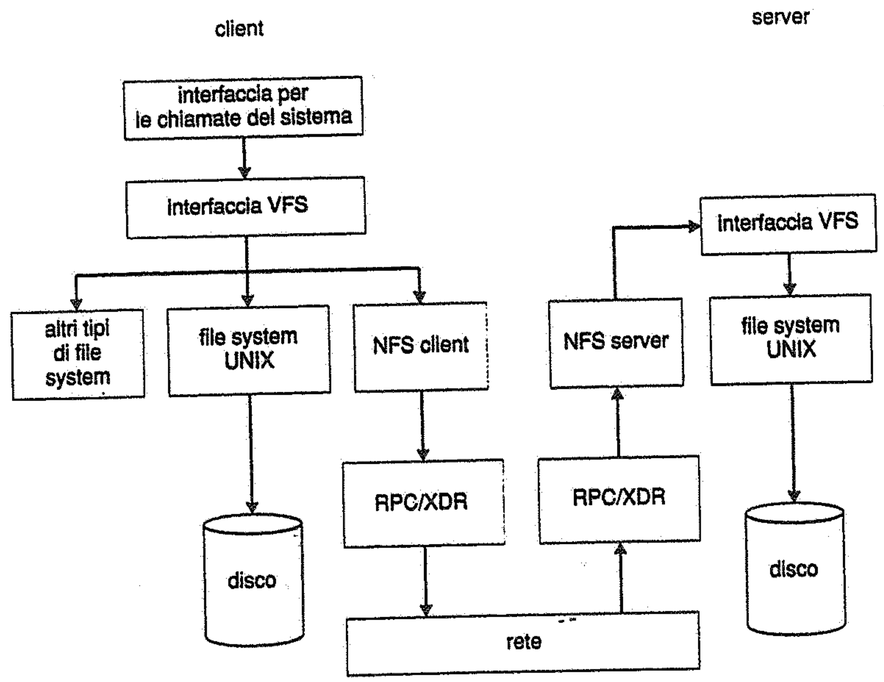


Figura 12.16 Schema dell'architettura dell'NFS.

ha eseguito. Sarebbe stato molto più efficiente consegnare un nome di percorso a un server e ricevere un *vnode* d'arrivo una volta incontrato un punto di montaggio, ma dovunque può essere presente un altro punto di montaggio per un particolare client sconosciuto al server senza informazione di stato.

Una cache per la ricerca dei nomi delle directory, nel sito del client, conserva i *vnode* per i nomi delle directory remote; in questo modo si accelerano i riferimenti ai file con lo stesso nome di percorso iniziale. Se gli attributi restituiti dal server non corrispondono agli attributi del *vnode* nella cache, si scarta il contenuto della cache della directory.

Occorre ricordare che nell'NFS è permesso il montaggio di un file system remoto in cima a un altro file system remoto già montato; si tratta del montaggio a cascata. Tuttavia, un server non può agire come intermediario tra un client e un altro server. Al contrario, un client deve stabilire un collegamento diretto client-server con il secondo server, montando direttamente la directory richiesta. Quando un client ha un montaggio a cascata, nel caso di un attraversamento di un nome di percorso può essere coinvolto più di un server. Tuttavia, ogni ricerca di componente si compie tra il client originale e alcuni server, perciò quando un client fa una ricerca in una directory sulla quale il server ha montato un file system, il client vede la directory sottostante e non la directory montata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

12.9.5 Funzionamento remoto

Eccetto che per l'apertura e la chiusura dei file, tra le normali chiamate del sistema dello UNIX per operazioni su file e le RPC del protocollo NFS esiste una corrispondenza quasi da uno a uno. Quindi, un'operazione remota su un file si può tradurre direttamente nella RPC corrispondente. Dal punto di vista concettuale l'NFS aderisce al paradigma del servizio remoto, ma in pratica si usano tecniche di memorizzazione transitoria e cache per migliorare le prestazioni. Non c'è una corrispondenza diretta tra un'operazione remota e una RPC, le RPC prelevano blocchi e attributi del file che memorizzano localmente nelle cache. Le successive operazioni remote usano i dati nella cache, soggetti a vincoli di coerenza.

Esistono due cache: la cache degli attributi dei file (informazioni sugli *inode*) e la cache dei blocchi di file. Su un file aperto, il nucleo fa un controllo rispetto al server remoto per stabilire se deve prelevare o riconvalidare gli attributi nella cache: i blocchi di file nella cache si usano solo se i corrispondenti attributi nella cache sono aggiornati. La cache degli attributi viene aggiornata ogni volta che arrivano nuovi attributi dal server. Dopo 60 secondi si scartano, in modo predefinito, gli attributi presenti nella cache. Tra il server e il client si usano le tecniche di lettura anticipata (*read-ahead*) e scrittura differita (*delayed-write*). I client non liberano i blocchi di scrittura differita finché il server non ha confermato che i dati sono stati scritti nei dischi. Contrariamente a quanto accade nello Sprite, la scrittura differita viene mantenuta anche quando si apre un file in concorrenza in modi conflittuali. Ne deriva che la semantica UNIX non si conserva.

Mettere a punto il sistema per migliorare le prestazioni rende difficile caratterizzare la semantica della coerenza dell'NFS. File nuovi creati in un calcolatore possono non essere visibili in altri calcolatori per 30 secondi. Non è stabilito se le scritture eseguite in un file in un sito siano visibili anche in altri siti che hanno aperto lo stesso file per la lettura. Le nuove aperture di quel file consentono di osservare solo le modifiche che sono già state inviate al server, quindi l'NFS non fornisce né una stretta emulazione della semantica UNIX, né la semantica delle sessioni di Andrew. Nonostante questi inconvenienti, l'utilità e le alte prestazioni del meccanismo ne fanno il sistema distribuito più usato.

12.10 Sommario

Il file system risiede permanentemente nella memoria secondaria, che è progettata per contenere, permanentemente, una grande quantità di dati. Il più comune mezzo di memoria secondaria è il disco.

I dischi si possono segmentare in partizioni allo scopo di controllarne l'uso e consentire più, anche diversi, file system per ogni disco. Questi file system si montano in un file system logico per renderli disponibili all'uso.

I file system spesso si realizzano secondo una struttura stratificata o modulare: i livelli più bassi hanno a che fare con le caratteristiche fisiche dei dispositivi di memorizzazione; i livelli più alti hanno a che fare con i nomi simbolici e le caratteristiche logiche dei file; i livelli intermedi fanno corrispondere le caratteristiche logiche dei file alle caratteristiche fisiche dei dispositivi.

Ogni tipo di file system può avere diverse strutture e algoritmi. Uno strato VFS consente ai livelli superiori di aver a che fare con ciascun tipo di file system in modo uniforme. Nella struttura di directory del sistema si possono integrare anche i file system remoti sui quali si agisce con le ordinarie chiamate del sistema tramite l'interfaccia del VFS

Ai file si può assegnare lo spazio dei dischi in tre modi: assegnazione contigua, concatenata e indicizzata. L'assegnazione contigua può risentire di frammentazione esterna. I file con accesso diretto non si possono gestire con l'assegnazione concatenata. L'assegnazione indicizzata, infine, può richiedere un notevole carico per il proprio blocco indice. Tali algoritmi si possono ottimizzare in molti modi. Lo spazio contiguo si può allargare attraverso delle estensioni allo scopo di aumentare la flessibilità e ridurre la frammentazione esterna. L'assegnazione indicizzata si può realizzare in gruppi di più blocchi per incrementare la produttività e ridurre il numero di elementi dell'indice necessari. L'indicizzazione in grossi gruppi di blocchi è analoga all'assegnazione contigua con estensioni.

I metodi di assegnazione dello spazio libero influenzano anche l'efficienza d'uso dello spazio dei dischi, le prestazioni del file system e l'affidabilità della memoria secondaria. I metodi usati comprendono i vettori di bit e le liste concatenate. Le ottimizzazioni comprendono il raggruppamento, il conteggio e la FAT, che colloca la lista concatenata in un'area contigua.

Le procedure di gestione delle directory devono tenere conto dell'efficienza, delle prestazioni e dell'affidabilità. La tabella hash è il metodo usato più spesso; è veloce ed efficiente. Sfortunatamente, il danneggiamento di una tabella o il crollo del sistema possono causare discordanze tra le informazioni contenute nelle directory e il contenuto del disco. Per riparare i danni si può usare un programma di sistema di verifica della coerenza, come `fsck` nello UNIX o `chkdsk` nell'MS-DOS. Gli strumenti di creazione di copie di riserva (*backup*) del sistema operativo consentono la copiatura nelle unità a nastro dei dati contenuti nei dischi allo scopo di poterli ripristinare in seguito a perdite dovute a malfunzionamenti dei dispositivi fisici, errori del sistema operativo, o a errori degli utenti.

I file system di rete, come l'NFS, usano un metodo client-server per permettere agli utenti di accedere a file e directory in calcolatori remoti come se fossero in file system locali. Si traducono le chiamate del sistema del client nei protocolli di rete, per poi ritradurle in operazioni del file system nel server. L'interconnessione in reti e gli accessi di più client costituiscono una sfida nei campi della coerenza dei dati e delle prestazioni.

A causa del ruolo fondamentale che i file system hanno nel funzionamento di un sistema, le loro prestazioni e affidabilità sono fondamentali. Tecniche come quelle che impiegano le cache e i giornali delle modifiche migliorano le prestazioni, mentre i giornali delle modifiche e le tecniche RAID migliorano l'affidabilità.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

12.11 Esercizi

12.1 Considerate un file formato da 100 blocchi. Supponete che il descrittore del file (e il blocco indice, nel caso dell'assegnazione indicizzata) sia già nella memoria. Calcolate quante operazioni di I/O nel disco sono richieste per le strategie di assegnazione contigua, concatenata e indicizzata (a singolo livello) se, per un blocco, vale quel che segue:

a) si aggiunge il blocco all'inizio;
b) si aggiunge il blocco nel mezzo;
c) si aggiunge il blocco alla fine;
d) si rimuove il blocco dall'inizio;
e) si rimuove il blocco dal mezzo;
f) si rimuove il blocco dalla fine.

Nel caso dell'assegnazione contigua, si suppone che all'inizio del file non ci sia spazio per l'aggiunta di nuovi elementi, ma che sia disponibile alla fine. Supponete che il blocco di informazioni da aggiungere al file sia presente nella memoria.

12.2 Considerate un sistema dove lo spazio libero sia contenuto in un elenco di blocchi liberi.

a) Supponete che il puntatore all'elenco dei blocchi liberi vada perduto. Dite se il sistema può ricostruire l'elenco dei blocchi liberi, e spiegate la risposta.
b) Suggerite uno schema che assicuri che il puntatore non vada mai perduto a causa di un guasto della memoria.

12.3 Dite quali problemi si possono verificare in un sistema che permette che si monti un file system contemporaneamente in più di una locazione.

12.4 Dite perché la mappa di bit per l'assegnazione dei file si deve trovare nella memoria secondaria, e non nella memoria centrale.

12.5 Considerate un sistema che gestisca i metodi di assegnazione contigua, concatenata e indicizzata. Dite quali criteri si devono usare per decidere qual è il miglior metodo per un file specifico.

12.6 Considerate un file system in un disco con dimensioni dei blocchi logici e fisici di 512 byte. Supponete che le informazioni su ciascun file siano già nella memoria. Per ciascuno dei tre metodi di assegnazione (contigua, concatenata e indicizzata) fornite le risposte alle seguenti domande:

a) dite come in questo sistema si fanno corrispondere gli indirizzi logici agli indirizzi fisici (per l'assegnazione indicizzata supponete che la lunghezza di un file sia sempre inferiore a 512 blocchi);
b) se l'ultimo accesso è stato fatto al blocco 10 (posizione corrente), dite quanti blocchi fisici si devono leggere dal disco per accedere al blocco logico 4.

12.7 Un problema connesso all'assegnazione contigua consiste nel fatto che l'utente deve assegnare in anticipo spazio sufficiente per ciascun file. Se il file cresce e diventa più grande dello spazio che gli è stato assegnato, è necessario intraprendere azioni specifiche. Una soluzione consiste nel definire una struttura di file formata di un'area iniziale contigua (di dimensione specificata). Se si riempie quest'area, il sistema operativo definisce automaticamente un'area di estensione collegata all'area contigua iniziale. Se si riempie anche l'area di estensione, si assegna un'altra area di estensione. Confrontate questo metodo con gli ordinari metodi di assegnazione contigua e concatenata.

12.8 In un dispositivo di memoria si può eliminare la frammentazione ricompattando le informazioni. I tipici dispositivi a disco non dispongono di registri di rilocazione o registri di base (come quelli usati per compattare la memoria), in questa situazione dite come si possono rilocare i file. Date tre motivi per i quali la ricompattazione e la rilocazione dei file vengono spesso evitate.

12.9 Spiegate in che modo le cache aiutano a migliorare le prestazioni, e perché i sistemi non usano un maggior numero di cache o cache più grandi, giacché sono così utili.

12.10 Dite in quali situazioni l'uso della memoria come disco RAM sarebbe più utile del suo uso come cache del disco.

12.11 Spiegate perché è vantaggioso per l'utente che un sistema operativo assegni dinamicamente le sue tabelle interne, e quali sono gli svantaggi per il sistema operativo nell'adottare questo metodo.

12.12 Spiegate perché l'annotazione degli aggiornamenti ai metadati assicura il ripristino di un file system dopo un crollo del sistema.

12.13 Spiegate in che modo lo strato del VFS permette a un sistema operativo di gestire facilmente più tipi di file system.

12.14 Considerate il seguente schema di creazione di copie di riserva:

- **Giorno 1.** Copiatura nel mezzo di registrazione delle copie di riserva di tutti i file contenuti nel disco.
- **Giorno 2.** Copiatura in un altro mezzo di tutti i file modificati dal giorno 1.
- **Giorno 3.** Copiatura in un altro mezzo di tutti i file modificati dal giorno 1.

Tale schema differisce dalla sequenza data nel Paragrafo 12.7.2 per il fatto che tutte le copiature riguardano tutti i file modificati dopo la prima copiatura completa. Dite quali sono i vantaggi di questo sistema rispetto a quello del Paragrafo 12.7.2 e se le operazioni di recupero sono più semplici o più difficili. Spiegate le risposte.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

12.12 Note bibliografiche

Lo schema di gestione dello spazio dei dischi dell'Apple Macintosh si trova in [Apple 1987] e [Apple 1991]. Il sistema FAT dell'MS-DOS è spiegato in [Norton e Wilton 1988] e la descrizione del sistema OS/2 si trova in [Iacobucci 1988]. Questi sistemi operativi usano rispettivamente la famiglia di CPU Motorola MC68000 [Motorola 1989a] e Intel 8086 [Intel 1985a], [Intel 1985b], [Intel 1986] e [Intel 1990]. I metodi di assegnazione dell'IBM sono descritti in [Deitel 1990]. L'organizzazione interna del sistema BSD UNIX è ampiamente trattata in [McKusick et al. 1996]. [McVoy e Kleiman 1991] presenta ottimizzazioni di questi metodi realizzate per il SunOS.

L'assegnazione dello spazio dei dischi per i file basata sul sistema *buddy* è discussa in [Koch 1987]. Uno schema di organizzazione dei file che garantisce il recupero dei dati in un accesso è trattato in [Larson e Kajla 1984].

L'uso delle memorie cache nella gestione dei dischi è trattato da [McKeon 1985] e [Smith 1985], per il sistema operativo sperimentale Sprite, è descritto in [Nelson et al. 1988]. Discussioni generali sulle tecnologie delle memorie di massa si trovano in [Chi 1982] e [Hoagland 1985]. [Folk e Zoellick 1987] tratta le strutture di file. [Silvers 2000] discute la realizzazione della cache delle pagine nel sistema operativo NetBSD.

[Sandberg et al. 1985], [Sandberg 1987], [Sun 1990] e [Callaghan 2000] trattano il file system di rete NFS. [Vahalia 1996] e [Mauro e McDougall 2001] descrivono l'NFS e il file system del sistema operativo UNIX, l'UFS. [Solomon 1998] descrive il file system NTFS del sistema operativo Windows NT. [Bovet e Cesati 2001] descrive il file system *ext2* del sistema operativo LINUX.

Parte quarta

Sistemi di I/O

Sistemi di I/O
Memoria secondaria e terziaria

I dispositivi che si connettono a un calcolatore differiscono in molti aspetti: trasferiscono un carattere o un blocco di caratteri per volta; alcuni forniscono modi d'accesso ai dati solo sequenziale, altri diretto; alcuni trasferiscono i dati in modo sincrono, altri in modo asincrono; alcuni sono riservati, altri sono condivisi; possono essere dispositivi di sola lettura o di lettura e scrittura, e avere notevoli differenze di velocità; per vari aspetti sono i più lenti fra i principali componenti dei calcolatori.

A causa di tutte queste varianti dei dispositivi, il sistema operativo deve fornire alle applicazioni un'ampia gamma di funzioni che consentano di controllare tutti gli aspetti dei dispositivi stessi. Uno degli scopi cruciali del sottosistema di I/O di un sistema operativo è quello di fornire l'interfaccia più semplice possibile al resto del sistema; e poiché i dispositivi di I/O sono una strozzatura per le prestazioni, un altro elemento cruciale è quello di ottimizzare l'I/O accrescendo il livello di concorrenza. Inizialmente si descrivono le innumerevoli varianti dei dispositivi di I/O e i modi in cui sono controllati dal sistema operativo. Più avanti sono trattati più complessi dispositivi di I/O usati come memorie secondarie e terziarie e si espongono le particolari precauzioni che il sistema operativo deve adottare per essi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 13

Sistemi di I/O

I due compiti principali di un calcolatore sono l'I/O e l'elaborazione. Spesso il compito principale è costituito dall'I/O mentre l'elaborazione è semplicemente accessoria: quando si consulta una pagina Web, o quando si modifica un file, ciò che più direttamente interessa l'utente è la lettura o l'immissione di informazioni, non l'elaborazione delle risposte che gli sono fornite.

Il ruolo di un sistema operativo nell'I/O di un calcolatore è quello di gestire e controllare le operazioni e i dispositivi di I/O. Sebbene in altri capitoli compaiano argomenti collegati, in questo capitolo sono raccolti tutti gli elementi utili alla composizione di un quadro d'insieme dell'I/O. Poiché il genere delle interfacce stabilisce i requisiti che le funzioni interne del sistema operativo devono possedere, si descrivono innanzi tutto i fondamenti dell'architettura di I/O. Quindi si discutono i servizi di I/O che il sistema operativo fornisce e come questi sono inclusi nell'interfaccia di I/O per le applicazioni; inoltre si spiega come il sistema operativo colma il divario tra le interfacce fisiche e le interfacce per le applicazioni. Si discute anche il meccanismo STREAMS dello UNIX System V, che consente a un'applicazione di comporre dinamicamente catene di codice di driver. Infine, si trattano gli aspetti riguardanti le prestazioni dell'I/O e i princìpi di progettazione dei sistemi operativi utili al miglioramento delle prestazioni dell'I/O.

13.1 Introduzione

Il controllo dei dispositivi connessi a un calcolatore è una delle questioni più importanti che riguardano i progettisti di sistemi operativi. Poiché i dispositivi di I/O sono così largamente diversi per funzioni e velocità (si considerino ad esempio un mouse, un disco e un *jukebox* di CD-ROM), altrettanto diversi devono essere i metodi di controllo. Tali metodi costituiscono il *sottosistema di I/O* del nucleo; questo sottosistema separa il resto del nucleo dalla complessità di gestione dei dispositivi di I/O.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La tecnologia dei dispositivi di I/O mostra due tendenze tra loro in conflitto. Da una parte, si osserva la crescente uniformazione a degli standard delle interfacce fisiche e logiche; e ciò semplifica l'introduzione nei calcolatori e nei sistemi operativi già esistenti di più avanzate generazioni di dispositivi. D'altra parte, però, si assiste a una crescente varietà di dispositivi di I/O; alcuni di essi sono tanto diversi dai dispositivi precedenti da rendere molto difficile il compito di integrarli nei calcolatori e nei sistemi operativi esistenti. Questo problema si affronta strutturando gli elementi di base dell'architettura di I/O — porte, bus e controllori di dispositivi —, in modo da potervi connettere un'ampia varietà di dispositivi di I/O, e strutturando il nucleo del sistema operativo in moduli di driver di dispositivi allo scopo di incapsulare i dettagli e le particolarità dei diversi dispositivi. I **driver dei dispositivi** offrono al sottosistema di I/O un'interfaccia uniforme per l'accesso ai dispositivi, così come le chiamate del sistema forniscono un'interfaccia uniforme tra le applicazioni e il sistema operativo.

13.2 Architetture e dispositivi di I/O

I calcolatori fanno funzionare un gran numero di tipi di dispositivi. La maggior parte rientra nella categoria dei dispositivi di memorizzazione secondaria e terziaria (dischi, nastri), dispositivi di trasmissione (schede di rete, modem), interfacce uomo-macchina (schermi, tastiere, mouse). Altri dispositivi sono più specializzati, come i dispositivi di pilotaggio di un caccia a reazione o di una navetta spaziale. In questi velivoli il pilota interagisce col calcolatore di bordo tramite la *cloche*, il calcolatore invia comandi per l'attivazione dei motori che azionano timoni, *flap* e propulsori.

Nonostante l'incredibile varietà dei dispositivi di I/O, bastano alcuni concetti per capire come i dispositivi sono connessi e come il sistema operativo li controlla.

Un dispositivo comunica con un sistema di calcolo inviando segnali attraverso un cavo o attraverso l'etere e comunica con il calcolatore tramite un punto di connessione (**porta**), ad esempio una porta seriale. Se uno o più dispositivi usano in comune un insieme di fili, la connessione è detta *bus*. Un **bus** è un insieme di fili e un protocollo rigorosamente definito che specifica l'insieme dei messaggi che si possono inviare attraverso i fili. In termini elettronici, i messaggi si inviano tramite configurazioni di livelli di tensione elettrica applicate ai fili con una definita scansione temporale. Quando un dispositivo *A* ha un cavo che si connette a un dispositivo *B* e il dispositivo *B* ha un cavo che si connette a un dispositivo *C* che a sua volta è collegato a una porta di un calcolatore, si ottiene il cosiddetto **collegamento a margherita** (*daisy chain*), che di solito funziona come un bus.

I bus sono ampiamente usati nell'architettura dei calcolatori. La Figura 13.1 mostra una tipica struttura di bus di PC; si tratta di un **bus PCI** (il comune bus di sistema dei PC) che connette il sottosistema CPU-memoria ai dispositivi veloci, e di un **bus di espansione** cui si connettono i dispositivi relativamente lenti come la tastiera e le porte seriali e parallele. Nella parte superiore destra della figura, quattro dischi sono collegati a un bus SCSI inserito nel relativo controllore.

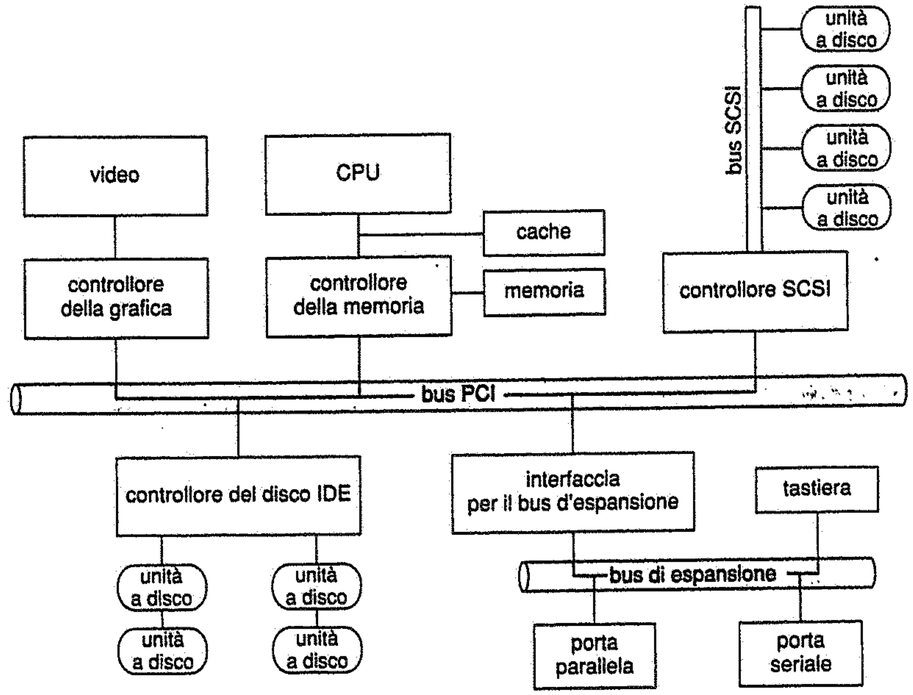


Figura 13.1 Tipica struttura del bus di un PC.

Un **controllore** è un insieme di componenti elettronici che può far funzionare una porta, un bus o un dispositivo. Un controllore di porta seriale è un semplice controllore di dispositivo; si tratta di un singolo circuito integrato (o di una sua parte) nel calcolatore che controlla i segnali presenti nei fili della porta seriale. Per contro un controllore SCSI non è semplice; poiché il protocollo SCSI è complesso, il controllore del bus SCSI è spesso realizzato come una scheda circuitale separata, **adattatore**, che s'inserisce nel calcolatore. Esso contiene tipicamente un'unità d'elaborazione, microcodice, e memoria privata che gli consentono di elaborare i messaggi del protocollo SCSI. Alcuni dispositivi sono dotati di propri controllori incorporati. Osservando un'unità a disco si vede, da un lato, una scheda elettronica a essa agganciata; si tratta del controllore del disco che attua la parte del protocollo di qualche tipo di connessione, ad esempio SCSI o IDE, relativa al disco. Ha un'unità d'elaborazione e microcodice per l'esecuzione di molti compiti, come la localizzazione dei settori difettosi, il prelievo anticipato, la memorizzazione transitoria, la gestione della cache.

L'unità d'elaborazione dà comandi e fornisce dati al controllore per portare a termine trasferimenti di I/O tramite uno o più registri per dati e segnali di controllo. La comunicazione col controllore avviene attraverso la lettura e la scrittura, da parte dell'unità d'elaborazione, di configurazioni di bit in questi registri. Un modo in cui questa comu-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nicazione può avvenire è tramite l'uso di speciali istruzioni di I/O che specificano il trasferimento di un byte o una parola a un indirizzo di porta di I/O. L'istruzione di I/O attiva le linee di bus per selezionare il giusto dispositivo e trasferire bit dentro o fuori dal registro di dispositivo. In alternativa, il controllore di dispositivo può disporre dell'I/O associato alla memoria. In questo caso i registri di controllo del dispositivo si fanno corrispondere a un sottoinsieme dello spazio d'indirizzi della CPU, che esegue le richieste di I/O usando le ordinarie istruzioni di trasferimento di dati per leggere e scrivere i registri di controllo del dispositivo.

Certi sistemi usano entrambe le tecniche. I PC ad esempio usano istruzioni di I/O per controllare alcuni dispositivi e l'I/O associato alla memoria per controllarne altri. Nella Figura 13.2 sono riportati gli usuali indirizzi delle porte di I/O dei PC. Il controllore della grafica ha alcune porte di I/O per le operazioni di controllo di base, ma dispone di un'ampia regione di memoria, detta memoria grafica, che serve a mantenere i contenuti dello schermo, associata alla memoria. Il processo scrive sullo schermo inserendo i dati nella regione associata alla memoria; il controllore genera l'immagine dello schermo sulla base del contenuto di questa regione di memoria. Questa tecnica è semplice da usare; inoltre la scrittura di milioni di byte nella memoria grafica è più veloce dell'invio di milioni di istruzioni di I/O. La facilità di scrittura in un controllore di I/O associato alla memoria è però controbilanciata da uno svantaggio: un comune errore di programmazione è la scrittura in una regione di memoria sbagliata causata da un errato puntatore. Ciò rende i registri dei dispositivi associati alla memoria vulnerabili ad accidentali modifiche. Naturalmente, le tecniche di protezione della memoria aiutano a ridurre tale rischio.

indirizzi per l'I/O (in esadecimale)	dispositivo
000-00F	controllore DMA
020-021	controllore delle interruzioni
040-043	temporizzatore
200-20F	controllore dei giochi
2F8-2FF	porta seriale (secondaria)
320-32F	controllore del disco
378-37F	porta parallela
3D0-3DF	controllore della grafica
3F0-3F7	controllore dell'unità a dischetti
3F8-3FF	porta seriale (principale)

Figura 13.2 Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale).

Una porta di I/O consiste tipicamente di quattro registri: `status`, `control`, `data-in` e `data-out`.

- Il registro `status` contiene alcuni bit che possono essere letti e indicano lo stato della porta; ad esempio indicano se è stata portata a termine l'esecuzione del comando corrente, se un byte è disponibile per essere letto dal registro `data-in`, se si è verificato un errore del dispositivo.
- Il registro `control` può essere scritto per attivare un comando o per cambiare il modo di funzionamento del dispositivo. Ad esempio, un certo bit nel registro `control` della porta seriale determina il tipo di comunicazione tra *half-duplex* e *full-duplex*, un altro abilita il controllo di parità, un terzo imposta la lunghezza delle parole a 7 o 8 bit, altri selezionano una tra le velocità che la porta seriale può sostenere.
- La CPU legge dal registro `data-in` per ricevere dati.
- La CPU scrive nel registro `data-out` per emettere dati.

La tipica dimensione dei registri di dati varia tra 1 e 4 byte. Certi controllori hanno circuiti integrati FIFO che possono contenere parecchi byte per l'immissione e l'emissione di dati, in modo da espandere la capacità del controllore oltre la dimensione del registro di dati. Un circuito integrato FIFO può contenere una piccola sequenza di dati finché il dispositivo o la CPU è in grado di riceverli.

13.2.1 Interrogazione ciclica

Il protocollo completo per l'interazione fra la CPU e un controllore può essere intricato, ma la fondamentale nozione di **negoziazione** (*handshaking*) è semplice, ed è illustrata con un esempio. Si assuma l'uso di due bit per coordinare la relazione di tipo produttore-consumatore fra il controllore e la CPU. Il controllore specifica il suo stato per mezzo del bit `busy` del registro `status`; pone a 1 il bit `busy` quando è impegnato in un'operazione, e lo pone a 0 quando è pronto a eseguire il comando successivo. La CPU comunica le sue richieste tramite il bit `command-ready` nel registro `command`: pone questo bit a 1 quando il controllore deve eseguire un comando. In questo esempio, la CPU scrive in una porta coordinandosi con un controllore per mezzo della negoziazione come segue:

1. La CPU legge ripetutamente il bit `busy` fino a che esso non vale 0.
2. La CPU pone a 1 il bit `write` del registro dei comandi e scrive un byte nel registro `data-out`.
3. La CPU pone a 1 il bit `command-ready`.
4. Quando il controllore si accorge del fatto che il bit `command-ready` è posto a 1, pone a 1 il bit `busy`.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

5. Il controllore legge il registro dei comandi e trova il comando `write`; legge il registro `data-out` per ottenere il byte da scrivere, e compie l'operazione di scrittura nel dispositivo.

6. Il controllore pone a 0 il bit `command-ready`, pone a 0 il bit `error` nel registro `status` per indicare che l'operazione di I/O ha avuto esito positivo, e pone a 0 il bit `busy` per indicare che l'operazione è terminata.

La sequenza appena descritta si ripete per ogni byte.

Durante l'esecuzione del passo 1, la CPU è in **attesa attiva** (*busy-waiting*) o in **interrogazione ciclica** (*polling*): itera la lettura del registro `status` fino a che il bit `busy` assume il valore 0. Se il controllore e il dispositivo sono veloci, questo metodo è ragionevole, ma se l'attesa rischia di prolungarsi, sarebbe probabilmente meglio se la CPU si dedicasse a un'altra operazione. In questo caso si pone il problema di come la CPU possa sapere quando il controllore è tornato inattivo. È necessario che la CPU serva certi tipi di dispositivi rapidamente, o si potrebbero perdere alcuni dati. Quando, ad esempio, i dati affluiscono in una porta seriale o dalla tastiera, la piccola memoria di transito del controllore diverrà presto piena, e se la CPU attende troppo a lungo prima di riprendere la lettura dei byte, si perderanno informazioni.

In molte architetture di calcolatori sono sufficienti tre cicli di istruzioni di CPU per interrogare ciclicamente un dispositivo: `read`, lettura di un registro del dispositivo; `logical-and`, congiunzione logica usata per estrarre il valore di un bit di stato; e `branch`, salto a un altro punto del codice se l'argomento è diverso da zero. Chiaramente, l'interrogazione ciclica è in sé un'operazione efficiente; tale tecnica diviene però inefficiente se le ripetute interrogazioni trovano raramente un dispositivo pronto per il servizio, mentre altre utili elaborazioni attendono la CPU. In tali casi, anziché richiedere alla CPU di eseguire un'interrogazione ciclica, può essere più efficiente far sì che il controllore comunichi alla CPU che il dispositivo è pronto. Il meccanismo dell'architettura che permette tale comunicazione si chiama **interruzione della CPU** o, più brevemente, **interruzione**.

13.2.2 Interruzioni

La CPU ha un contatto, detto **linea di richiesta dell'interruzione**, del quale la CPU controlla lo stato dopo l'esecuzione di ogni istruzione. Quando rileva il segnale di un controllore nella linea di richiesta dell'interruzione, la CPU memorizza una piccola quantità di informazioni sullo stato dell'elaborazione corrente — ad esempio il valore del puntatore alle istruzioni — e salta alla **procedura di gestione delle interruzioni** (*interrupt-handler routine*), che si trova a un indirizzo di memoria fissato. Questa procedura determina le cause dell'interruzione, porta a termine l'elaborazione necessaria ed esegue un'istruzione `return from interrupt` per far sì che la CPU ritorni nello stato in cui si trovava prima della sua interruzione. Il controllore del dispositivo *genera* un segnale d'interruzione della CPU lungo la linea di richiesta delle interruzioni, che la CPU *rileva* e *recapita* al gestore delle interruzioni, che a sua volta *evade* il compito corrispondente servendo il dispositivo. Nella Figura 13.3 è riassunto il ciclo di I/O indotto da un segnale d'interruzione della CPU.

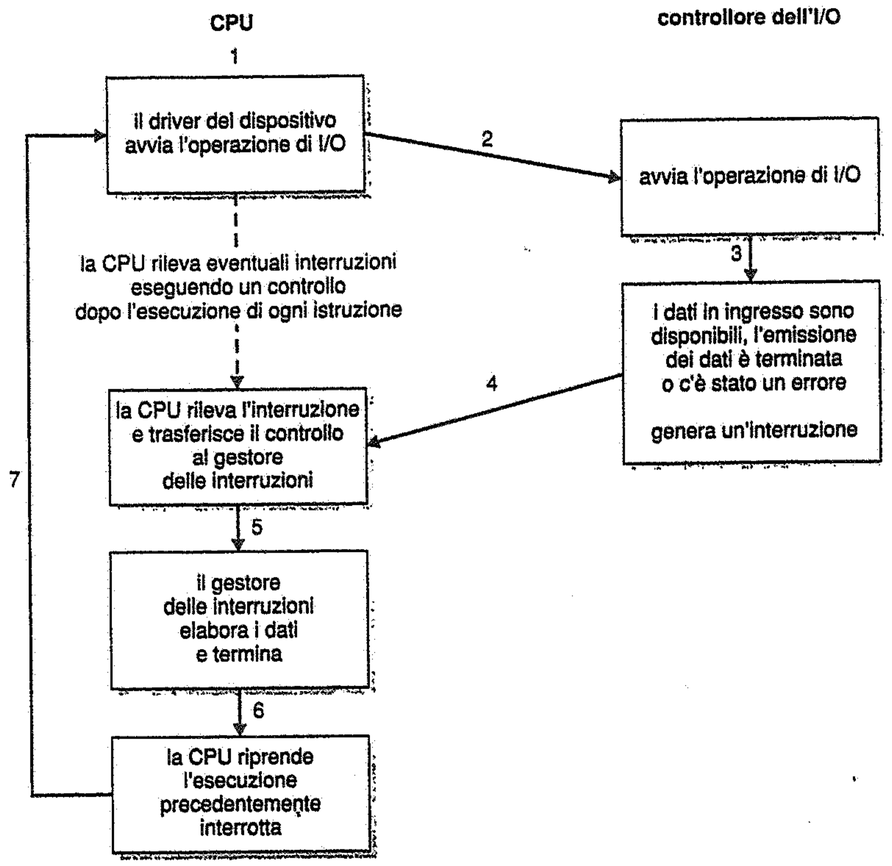


Figura 13.3 Ciclo di I/O basato sulle interruzioni.

Il meccanismo di base delle interruzioni permette alla CPU di rispondere a un evento asincrono, come quello di un controllore di un dispositivo che divenga pronto per essere servito. Nei sistemi operativi moderni sono necessarie capacità di gestione delle interruzioni più raffinate: innanzi tutto, è necessario poter differire la gestione delle interruzioni durante le elaborazioni critiche; inoltre, occorre disporre di un modo efficiente per recapitare un segnale d'interruzione al corretto gestore delle interruzioni relativo a un dispositivo, senza dover prima interrogare tutti i dispositivi per determinare quale abbia generato l'interruzione. Inoltre è necessario strutturare i segnali d'interruzione in diversi livelli, cosicché il sistema operativo possa distinguere tra interruzioni ad alta e a bassa priorità, e possa rispondere con l'appropriato grado di tempestività. In un calcolatore moderno queste tre caratteristiche sono fornite dalla CPU e dal **controllore delle interruzioni**.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La maggior parte delle CPU ha due linee di richiesta delle interruzioni. Una è quella delle **interruzioni non mascherabili**, che è riservata a eventi quali gli errori di memoria irrecuperabili. La seconda linea è quella delle **interruzioni mascherabili**: può essere disattivata dalla CPU prima dell'esecuzione di una sequenza critica di istruzioni che non deve essere interrotta. L'interruzione mascherabile è usata dai controllori dei dispositivi per richiedere un servizio.

Il meccanismo delle interruzioni accetta un **indirizzo** — un numero che seleziona da un insieme ristretto una specifica procedura di gestione delle interruzioni. Nella maggior parte delle architetture questo indirizzo è uno scostamento relativo a una tabella detta **vettore delle interruzioni**, che contiene gli indirizzi di memoria degli specifici gestori delle interruzioni. Lo scopo di un meccanismo vettoriale delle interruzioni è di ridurre la necessità che un singolo gestore debba individuare tutte le possibili fonti dei segnali d'interruzione per determinare quale di esse abbia richiesto un servizio. In pratica, tuttavia, i calcolatori hanno più dispositivi (e quindi, più gestori d'interruzioni) che elementi nel vettore delle interruzioni. Una maniera diffusa di risolvere questo problema consiste nell'usare gli elementi del vettore delle interruzioni come puntatori a una lista di gestori delle interruzioni. Quando si verifica un'interruzione, si chiamano uno alla volta i gestori nella lista corrispondente finché non se ne trova uno che può soddisfare la richiesta. Questa struttura è un compromesso fra lo svantaggio di una tabella delle interruzioni enorme e l'inefficienza dell'uso di un solo gestore delle interruzioni.

Nella Figura 13.4 è descritto il vettore delle interruzioni della CPU Intel Pentium. Gli eventi da 0 a 31, che non sono mascherabili, si usano per segnalare varie condizioni d'errore; quelli dal 32 al 255, mascherabili, si usano, ad esempio, per le interruzioni generate dai dispositivi.

Il meccanismo delle interruzioni realizza anche un sistema di **livelli di priorità delle interruzioni**. Esso permette alla CPU di differire la gestione delle interruzioni di bassa priorità senza mascherare tutte le interruzioni, e permette a un'interruzione di priorità alta di sospendere l'esecuzione della procedura di servizio di un'interruzione di priorità bassa.

Un sistema operativo moderno interagisce con il meccanismo delle interruzioni in vari modi. All'accensione della macchina esamina i bus per determinare quali dispositivi sono presenti, e installa gli indirizzi dei corrispondenti gestori delle interruzioni nel vettore delle interruzioni. Durante l'I/O, i vari controllori di dispositivi generano i segnali d'interruzione della CPU quando sono pronti per un servizio. Queste interruzioni significano che è stato completato un trasferimento di dati, o che sono disponibili dati in ingresso, o che un'operazione non è andata a buon fine. Il meccanismo delle interruzioni si usa anche per gestire un'ampia gamma di **eccezioni**, come la divisione per zero, l'accesso a indirizzi di memoria protetti o inesistenti o il tentativo di eseguire un'istruzione privilegiata nel modo d'utente. Gli eventi che producono segnali d'interruzione hanno una proprietà in comune: inducono la CPU a eseguire urgentemente una procedura autonoma.

Un sistema operativo può fare altri usi proficui di un efficiente meccanismo dell'architettura che memorizza una piccola quantità d'informazioni sullo stato della CPU e poi chiama una procedura del nucleo. Ad esempio, molti sistemi operativi usano il meccanismo delle interruzioni per la gestione della memoria virtuale. Un'eccezione di pagina mancante genera un'interruzione che sospende il processo corrente e trasferisce il controllo dell'esecuzione al relativo gestore nel nucleo. Tale gestore memorizza le informazioni sullo stato del processo, sposta il processo nella coda d'attesa, compie le necessarie operazioni di gestione della memoria virtuale, avvia un'operazione di I/O per prelevare la pagina giusta, stabilisce la ripresa dell'esecuzione di un altro processo e restituisce il controllo dell'esecuzione derivante dall'interruzione.

Un altro esempio è dato dall'esecuzione delle *chiamate del sistema*; si tratta di funzioni che le applicazioni invocano per usufruire di servizi del nucleo. Una chiamata del sistema controlla gli argomenti specificati dall'applicazione, costruisce una struttura di dati al fine di comunicare questi argomenti al nucleo quindi esegue una speciale istruzione detta **eccezione** (*trap*), che ha un operando che identifica il servizio del nucleo desiderato. Quando la chiamata del sistema esegue l'istruzione di eccezione, l'architettura delle interruzioni memorizza le informazioni riguardanti lo stato cui era giunta l'esecuzione del codice d'utente, passa al modo supervisore e recapita l'interruzione alla procedura del nucleo che realizza il servizio richiesto. All'eccezione si assegna una priorità di interruzione relativamente bassa rispetto a quelle date alle interruzioni dei dispositivi — eseguire una chiamata del sistema per conto di un'applicazione è meno urgente di quanto non sia servire un controllore prima che la sua coda FIFO trabocchi causando la perdita di informazioni.

indice del vettore	descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	devide not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19-31	(Intel reserved, do not use)
32-255	maskable interrupts

Figura 13.4 Vettore delle interruzioni della CPU Intel Pentium.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le interruzioni si possono inoltre usare per gestire il controllo del flusso all'interno del nucleo. Si consideri ad esempio l'elaborazione richiesta per completare una lettura da un disco. Un passo necessario è quello di copiare dati dalla regione di memoria usata dal nucleo all'area di memoria per l'I/O (*buffer*) dell'utente. Questa azione richiede tempo, ma non è urgente, e non dovrebbe bloccare la gestione delle interruzioni con priorità più alta. Un altro passo è quello di avviare l'evasione delle successive richieste di I/O relative a quell'unità a disco. Questo passo ha priorità più alta: se le unità a disco si devono usare in modo efficiente, è necessario avviare l'evasione della prossima richiesta di I/O non appena la precedente è stata soddisfatta. Di conseguenza una *coppia* di gestori di interruzioni realizza il codice del nucleo che compie le letture dai dischi. Il gestore ad alta priorità mantiene le informazioni sullo stato dell'I/O, risponde al segnale d'interruzione del dispositivo, avvia il prossimo I/O in attesa e genera un segnale d'interruzione a bassa priorità per completare il lavoro. Più tardi, in un momento in cui la CPU non è occupata in compiti ad alta priorità, si recapita l'interruzione a bassa priorità. Il gestore corrispondente completa l'I/O del livello d'utente copiando i dati dalla relativa area di memoria del nucleo a quello dell'applicazione, e chiamando poi lo scheduler per aggiungere l'applicazione alla coda dei processi pronti.

Un'architettura del nucleo basata su thread è adatta alla realizzazione di più livelli di priorità delle interruzioni, e a dare la precedenza alla gestione delle interruzioni rispetto alle elaborazioni in sottofondo delle procedure del nucleo e delle applicazioni. Questa osservazione si può esemplificare illustrando il nucleo del sistema operativo Solaris. In questo sistema i gestori delle interruzioni si eseguono come thread del nucleo cui si riserva una serie di alte priorità. Queste priorità garantiscono la precedenza dei gestori delle interruzioni rispetto al codice delle applicazioni e al lavoro ordinario del nucleo, e inoltre realizzano le necessarie relazioni di priorità fra i diversi gestori delle interruzioni. Esse portano lo scheduler dei thread del Solaris a sospendere i gestori delle interruzioni di bassa priorità a vantaggio di quelli di priorità più alta, e la realizzazione basata su thread permette alle architetture con più unità d'elaborazione di eseguire parallelamente diversi gestori delle interruzioni. Le architetture delle interruzioni dei sistemi UNIX e Windows 2000 sono descritte, rispettivamente, nell'Appendice A e nel Capitolo 21.

Riassumendo, i segnali d'interruzione sono usati diffusamente dai sistemi operativi moderni per gestire eventi asincroni e per eseguire nel modo supervisore le procedure del nucleo. Per far sì che i compiti più urgenti siano portati a termine per primi, i calcolatori moderni usano un sistema di priorità delle interruzioni. I controllori dei dispositivi, gli errori e le chiamate del sistema generano segnali d'interruzione al fine di innescare l'esecuzione di procedure del nucleo. Poiché le interruzioni sono usate in modo massiccio per affrontare situazioni in cui il tempo è un fattore critico, è necessario avere un'efficiente gestione delle interruzioni per ottenere buone prestazioni del sistema.

13.2.3 Accesso diretto alla memoria (DMA)

Quando un dispositivo compie trasferimenti di grandi quantità di dati, come nel caso di un'unità a disco, l'uso di una costosa CPU per il controllo dei bit di stato e per la scrittura di dati nel registro del controllore un byte alla volta, detto **I/O programmato** (*programmed I/O* — PIO), sembra essere uno spreco. In molti calcolatori si evita di sovraccaricare la CPU assegnando una parte di questi compiti a un'unità d'elaborazione specializzata, detta controllore dell'**accesso diretto alla memoria** (*direct memory-access* — DMA). Per dare avvio a un trasferimento DMA, la CPU scrive nella memoria un comando strutturato per il DMA. Esso contiene un puntatore alla locazione dei dati da trasferire, un altro puntatore alla destinazione dei dati, e il numero dei byte da trasferire. La CPU scrive l'indirizzo di questo comando strutturato nel controllore del DMA, e prosegue con l'esecuzione di altro codice. Il controllore DMA agisce quindi direttamente sul bus della memoria, presentando al bus gli indirizzi di memoria necessari per eseguire il trasferimento senza l'aiuto della CPU. Un semplice controllore DMA è un componente ordinario dei PC, e le schede di I/O dette *bus-mastering* di un PC includono di solito componenti DMA ad alta velocità.

La procedura di negoziazione tra il controllore del DMA e il controllore del dispositivo si svolge grazie a una coppia di fili detti `DMA-request` e `DMA-acknowledge`. Il controllore del dispositivo manda un segnale sulla linea `DMA-request` quando una parola di dati è disponibile per il trasferimento. Questo segnale fa sì che il controllore DMA prenda possesso del bus di memoria, presenti l'indirizzo desiderato ai fili d'indirizzamento della memoria e mandi un segnale lungo la linea `DMA-acknowledge`. Quando il controllore del dispositivo riceve questo segnale, trasferisce nella memoria la parola di dati e rimuove il segnale dalla linea `DMA-request`.

Quando l'intero trasferimento termina, il controllore del DMA interrompe la CPU. Nella Figura 13.5 è rappresentato questo processo. Quando il controllore del DMA prende possesso del bus di memoria, la CPU è temporaneamente impossibilitata ad accedere alla memoria centrale, sebbene abbia accesso ai dati contenuti nella sua cache primaria e secondaria. Questo fenomeno, noto come **sottrazione di cicli**, può rallentare le computazioni della CPU; ciononostante l'assegnamento del lavoro di trasferimento di dati a un controllore DMA migliora in generale le prestazioni complessive del sistema. In alcune architetture per realizzare la tecnica DMA si usano gli indirizzi della memoria fisica, mentre in altre s'impiega l'**accesso diretto alla memoria virtuale** (*direct virtual-memory access* — DVMA): in questo caso si usano indirizzi virtuali che poi si traducono in indirizzi fisici. La tecnica DVMA permette di compiere i trasferimenti di dati tra due dispositivi che eseguono I/O associato alla memoria senza far intervenire la CPU o accedere alla memoria centrale.

Quando il nucleo può operare in modo protetto, in genere il sistema operativo non permette ai processi di impartire direttamente istruzioni ai dispositivi. Ciò protegge i dati dalle violazioni dei controlli d'accesso e il sistema da un eventuale uso scorretto dei controllori dei dispositivi che potrebbe portare all'arresto del sistema stesso. Il sistema operativo mette a disposizione delle applicazioni funzioni di I/O che possono essere eseguite da processi sufficientemente privilegiati per accedere alle istruzioni di basso livello. Quando invece il nucleo non può garantire la protezione della memoria, i processi hanno accesso diretto ai controllori dei dispositivi. Questo accesso diretto si può strutturare in modo da ottenere buone prestazioni perché evita la comunicazione col nucleo, i cambi di contesto e l'interazione fra diversi livelli del nucleo. Purtroppo ha un impatto negativo sulla stabilità e la sicurezza del sistema. La tendenza comune per i sistemi operativi

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

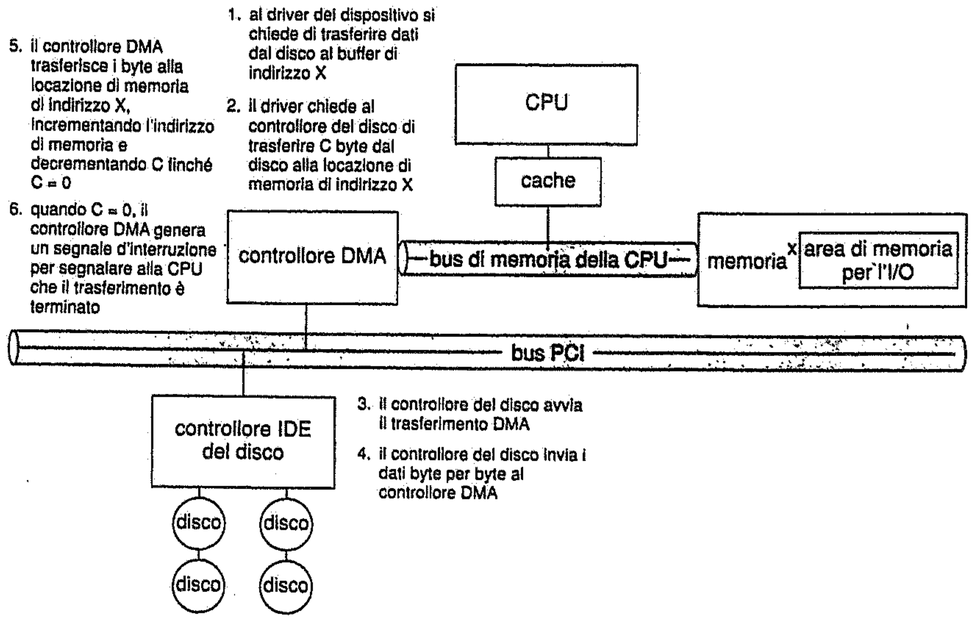


Figura 13.5 Passi di un trasferimento DMA.

d'uso generale è quella di proteggere la memoria e i dispositivi in modo da salvaguardarli da applicazioni accidentalmente o volutamente dannose.

Sebbene gli aspetti dell'I/O che riguardano i dispositivi siano complessi se si analizzano tanto dettagliatamente quanto farebbe un progettista elettronico, i concetti appena descritti sono sufficienti per comprendere molti aspetti dell'I/O per ciò che concerne i sistemi operativi. Ecco un sommario dei concetti principali:

- bus;
- controllore;
- porta di I/O e suoi registri;
- procedura di negoziazione tra la CPU e il controllore di un dispositivo;
- esecuzione della procedura di negoziazione per mezzo dell'interrogazione ciclica (*polling*) o delle interruzioni;
- delega dell'I/O a un controllore DMA nel caso di trasferimenti di grandi quantità di dati.

Nel Paragrafo 13.2.1 si è fornito un esempio della negoziazione che avviene tra un controllore di dispositivo e una macchina. In realtà, la grande varietà di dispositivi esi-

stenti pone un problema a chi voglia realizzare concretamente un sistema operativo. Ogni tipo di dispositivo ha le sue capacità operative, le sue definizioni dei bit di controllo, e il suo protocollo per l'interazione con la macchina — e tutto ciò varia da dispositivo a dispositivo. Il problema consiste nel modo in cui debba essere progettato un sistema operativo affinché sia possibile collegare al calcolatore nuovi dispositivi senza che sia necessario riscrivere il sistema operativo stesso. E inoltre, vista la grande varietà di dispositivi, nel modo in cui il sistema operativo possa fornire alle applicazioni un'interfaccia per l'I/O uniforme ed efficace.

13.3 Interfaccia di I/O per le applicazioni

In questo paragrafo si discutono le tecniche e le interfacce di un sistema operativo che permettono un trattamento uniforme dei dispositivi di I/O. Si spiega, ad esempio, come un'applicazione possa aprire un file residente in un disco senza sapere di che tipo di disco si tratti, e come si possano aggiungere al calcolatore nuove unità a disco e altri dispositivi senza che si debba modificare il sistema operativo.

I metodi qui esposti coinvolgono l'astrazione, l'incapsulamento e la stratificazione dei programmi. In particolare, si può compiere un procedimento di astrazione rispetto ai dettagli delle differenze tra i dispositivi per l'I/O identificandone alcuni tipi generali. A ognuno di questi tipi si accede per mezzo di un unico insieme di funzioni — un'interfaccia. Le differenze sono incapsulate in moduli del nucleo detti driver dei dispositivi che sono specializzati per gli specifici dispositivi, ma che comunicano con l'esterno per mezzo delle interfacce uniformi. Nella Figura 13.6 è illustrata la divisione in strati di quelle parti del nucleo che riguardano la gestione dell'I/O.

Lo scopo dello strato dei driver dei dispositivi è di nascondere al sottosistema di I/O del nucleo le differenze fra i controllori dei dispositivi, in modo simile a quello con cui le chiamate del sistema di I/O incapsulano il comportamento dei dispositivi in alcune classi generiche che nascondono le differenze alle applicazioni. Il fatto che così il sottosistema di I/O sia reso indipendente dalla struttura fisica semplifica il lavoro di chi sviluppa il sistema operativo, e va inoltre a vantaggio dei costruttori dei dispositivi. Questi, infatti, o progettano i nuovi dispositivi in modo tale che siano compatibili con un'interfaccia macchina-controllore già esistente (ad esempio la SCSI-2), oppure scrivono driver che permettano ai nuovi dispositivi di essere gestiti dai sistemi operativi più diffusi. In questo modo, i nuovi dispositivi sono utilizzabili da un calcolatore senza che occorra attendere lo sviluppo del codice relativo da parte del produttore del sistema operativo.

Sfortunatamente per i produttori di dispositivi, ogni tipo di sistema operativo ha le sue convenzioni riguardanti l'interfaccia dei driver dei dispositivi. Così, un dato dispositivo potrà essere venduto con molti driver diversi — ad esempio, driver per MS-DOS, Windows 95/98/*Me*, Windows NT/2000/XP e Solaris. I dispositivi (Figura 13.7) possono differire in molti aspetti:

- **Trasferimento a flusso di caratteri o a blocchi.** Un dispositivo del primo tipo trasferisce dati un byte alla volta, mentre uno del secondo tipo trasferisce un blocco di byte in un'unica soluzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

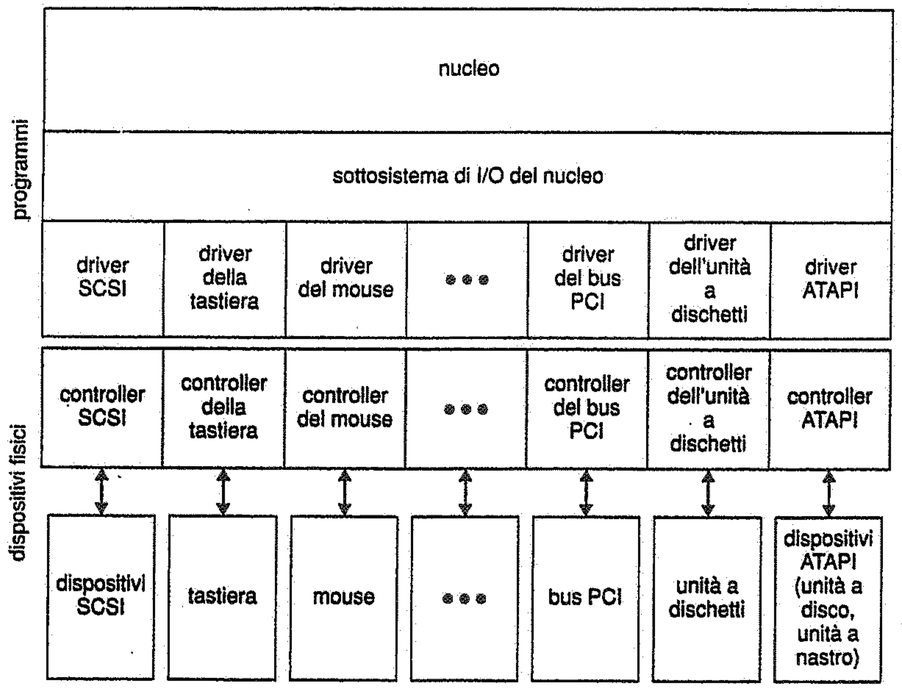


Figura 13.6 Struttura relativa all'I/O di un nucleo.

- **Accesso sequenziale o diretto.** Un dispositivo del primo tipo trasferisce dati secondo un ordine prestabilito e invariabile, mentre l'utente di un dispositivo ad accesso diretto può richiedere l'accesso a una qualunque delle possibili locazioni di memorizzazione.
- **Dispositivi sincroni o asincroni.** Un dispositivo sincrono trasferisce dati con un tempo di risposta prevedibile, mentre un dispositivo asincrono ha tempi di risposta irregolari o non prevedibili.
- **Condivisibili o riservati.** Un dispositivo condivisibile può essere usato in modo concorrente da diversi processi, mentre ciò è impossibile se il dispositivo è riservato.
- **Velocità di funzionamento.** Può variare da alcuni byte al secondo fino a qualche gigabyte al secondo.
- **Lettura e scrittura, sola lettura o sola scrittura.** Alcuni dispositivi possono emettere e ricevere dati, ma altri consentono solo una delle due possibilità.

aspetto	variazione	esempio
modo di trasferimento dei dati	a caratteri a blocchi	terminale unità a disco
modo d'accesso	sequenziale casuale	modem lettore di CD-ROM
prevedibilità dell'I/O	sincrono asincrono	unità a nastro tastiera
condivisione	dedicato condiviso	unità a nastro tastiera
velocità	latenza tempo di ricerca velocità di trasferimento attesa fra le operazioni	
direzione dell'I/O	solo lettura solo scrittura lettura e scrittura	lettore di CD-ROM controllore della grafica unità a disco

Figura 13.7 Caratteristiche dei dispositivi per l'I/O.

Per ciò che riguarda l'accesso delle applicazioni ai dispositivi, molte di queste differenze sono nascoste dal sistema operativo, e i dispositivi sono raggruppati in poche classi convenzionali. Si è riscontrato che i modi risultanti d'accesso ai dispositivi sono utili e largamente applicabili. Anche se la forma precisa delle chiamate del sistema può variare nei diversi sistemi operativi, le classi di dispositivi sono abbastanza regolari. Le convenzioni d'accesso principali includono l'I/O a blocchi, l'I/O a flusso di caratteri, l'accesso ai file associato alla memoria, e le socket di rete. I sistemi operativi forniscono anche chiamate del sistema speciali per l'accesso a qualche dispositivo aggiuntivo, ad esempio un orologio o un temporizzatore. Qualche sistema operativo mette a disposizione un insieme di chiamate del sistema per dispositivi video e audio.

La maggior parte dei sistemi ha anche una 'scappatoia' (*escape* o *back-door*) che permette il passaggio trasparente di comandi arbitrari da un'applicazione a un driver di dispositivo. Nello UNIX tale funzione è svolta dalla chiamata del sistema `ioctl` (che sta per *I/O control*) e consente a un'applicazione di impiegare qualsiasi servizio fornito da ogni driver di dispositivo, senza che per questo sia necessario creare nuove chiamate del sistema. Gli argomenti di `ioctl` sono tre: il primo è un descrittore di file che individua il driver desiderato facendo riferimento a un dispositivo gestito da quel driver; il secondo è un numero intero che seleziona uno dei comandi forniti dal driver; il terzo argomento, infine, è un puntatore a un'arbitraria struttura di dati nella memoria, tramite la quale l'applicazione e il driver si scambiano ogni necessaria informazione.

http://www.chultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

13.3.1 Dispositivi con trasferimento a blocchi o a caratteri

L'interfaccia per i **dispositivi a blocchi** sintetizza tutti gli aspetti necessari per accedere alle unità a disco e ad altri dispositivi basati sul trasferimento di blocchi di dati. Ci si aspetta che il dispositivo comprenda istruzioni come `read` e `write`, e anche, nel caso esso sia ad accesso casuale, un comando `seek` per specificare il prossimo blocco da trasferire. Di solito le applicazioni comunicano con questi dispositivi tramite un file system che funge da interfaccia. Il sistema operativo e certe applicazioni particolari come quelle per la gestione delle basi di dati possono trovare più conveniente trattare questi dispositivi come una semplice sequenza lineare di blocchi. In questo caso si parla di I/O **a basso livello** (*raw I/O*). Si vede come le istruzioni `read`, `write` e `seek` colgano il comportamento essenziale dei dispositivi a blocchi, cosicché le applicazioni rimangono isolate dalle differenze di basso livello presenti fra dispositivi diversi.

L'accesso ai file associato alla memoria può essere posto a un livello gerarchico immediatamente superiore a quello dei dispositivi a blocchi. Piuttosto che offrire funzioni di lettura e scrittura, un'interfaccia per l'accesso associato alla memoria fornisce la possibilità di usare un'unità a disco tramite un vettore di byte della memoria centrale. La chiamata del sistema che associa un file a una regione di memoria restituisce l'indirizzo di memoria virtuale di un vettore di caratteri che contiene una copia del file. Gli effettivi trasferimenti di dati sono eseguiti solo quando sono necessari per soddisfare una richiesta d'accesso all'immagine nella memoria. Poiché i trasferimenti si trattano nello stesso modo in cui si gestisce l'accesso su richiesta a una pagina di memoria virtuale, l'I/O associato alla memoria è efficiente. Esso è inoltre conveniente per i programmatori perché l'accesso a un file associato alla memoria è semplice tanto quanto la lettura e la scrittura nella memoria. I sistemi operativi capaci di gestire la memoria virtuale includono comunemente anche un'interfaccia di associazione alla memoria per i servizi del nucleo. Ad esempio, quando il sistema operativo deve eseguire un programma, associa il codice a un intervallo d'indirizzi di memoria, quindi trasferisce il controllo all'indirizzo iniziale. Questo tipo d'interfaccia è spesso usato anche per l'accesso del nucleo all'area d'avvicendamento dei processi nei dischi.

La tastiera è un esempio di dispositivo al quale si accede tramite un'interfaccia a **flusso di caratteri**. Le chiamate del sistema fondamentali per le interfacce di questo tipo permettono a un'applicazione di acquisire (`get`) o inviare (`put`) un carattere. Basandosi su quest'interfaccia è possibile costruire servizi aggiuntivi quali l'accesso riga per riga, la correzione (ad esempio, quando l'utente batte il tasto Backspace, si rimuove il carattere precedente dalla sequenza di caratteri da inserire). Questo tipo d'accesso è conveniente per dispositivi come le tastiere, i mouse, i modem, che producono dati 'spontaneamente', cioè in momenti che non possono essere sempre previsti dalle applicazioni. Lo stesso tipo d'accesso è adatto anche ai dispositivi che emettono dati organizzati in modo naturale come sequenza lineare di byte, ad esempio le stampanti o le schede audio.

13.3.2 Dispositivi di rete

Poiché i modi di indirizzamento e le prestazioni tipiche dell'I/O di rete sono notevolmente differenti da quelli dell'I/O delle unità a disco, la maggior parte dei sistemi operativi fornisce un'interfaccia per l'I/O di rete diversa da quella caratterizzata dalle operazioni `read`, `write` e `seek` usata per i dischi. Un'interfaccia disponibile in molti sistemi operativi, tra i quali UNIX e Windows NT, è l'interfaccia di rete *socket* (letteralmente, presa di corrente).

Si pensi a una presa di corrente elettrica a muro: vi si può collegare qualunque apparecchiatura elettrica; per analogia, le chiamate del sistema di un'interfaccia socket permettono a un'applicazione di creare una socket, collegare una socket locale all'indirizzo di un altro punto della rete (ciò ha l'effetto di collegare questa applicazione alla socket creata da un'altra applicazione), controllare se un'applicazione si sia inserita nella socket locale, e inviare o ricevere pacchetti di dati lungo la connessione. Per permettere lo sviluppo di server, l'interfaccia socket fornisce anche una funzione chiamata `select` che gestisce un insieme di socket. Essa restituisce informazioni sulle socket per le quali sono presenti pacchetti che attendono d'essere ricevuti, e su quelle che hanno spazio per accettare un pacchetto da inviare. L'uso della funzione `select` elimina l'interrogazione ciclica altrimenti necessaria per l'I/O di rete. Queste funzioni incapsulano il comportamento essenziale delle reti, facilitando notevolmente la creazione di applicazioni distribuite che possano sfruttare i dispositivi e i protocolli di rete.

Sono stati sviluppati molti altri metodi per affrontare il problema della comunicazione di rete e fra i processi. Il sistema operativo Windows NT, ad esempio, fornisce un'interfaccia per la scheda di rete e un'altra per i protocolli di rete (si veda il Paragrafo 21.6). Il sistema operativo UNIX, banco di prova storico delle tecnologie di rete, offre pipe *half-duplex*, code FIFO *full-duplex*, STREAMS *full-duplex*, code di messaggi e socket. Per maggiori informazioni sui servizi di rete dello UNIX si veda il Paragrafo A.9.

13.3.3 Orologi e temporizzatori

La maggior parte dei calcolatori ha temporizzatori e orologi che forniscono tre funzioni essenziali:

- segnare l'ora corrente;
- segnalare il tempo trascorso;
- regolare un temporizzatore in modo da avviare l'operazione x al tempo t.

Queste funzioni sono spesso usate sia dal sistema operativo sia da applicazioni per cui il tempo è un fattore importante. Purtroppo, le chiamate del sistema che realizzano queste funzioni non sono uniformi, ma variano da un sistema operativo all'altro.

Il dispositivo che misura la durata di un lasso di tempo e che può avviare un'operazione si chiama **temporizzatore programmabile**; si può regolare in modo da attendere un certo tempo e poi generare un segnale d'interruzione, e può anche ripetere questo processo un numero prefissato di volte, generando così interruzioni periodiche. Lo sche-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

duler usa questo meccanismo per generare un segnale d'interruzione che sospende un processo quando il suo quanto di tempo è scaduto. Il sottosistema dell'I/O delle unità a disco lo usa per riversare periodicamente nei dischi il contenuto della *buffer cache*, e il sottosistema di rete lo usa per annullare operazioni che procedono troppo lentamente a causa di congestionamenti o fallimenti. Il sistema operativo può inoltre fornire un'interfaccia per permettere ai processi utenti di usare i temporizzatori. In certi casi, simulando orologi virtuali, il sistema operativo può anche gestire un numero di richieste d'uso dei temporizzatori maggiore del numero dei temporizzatori fisici. Per far ciò il nucleo (o il driver del temporizzatore) mantiene un elenco ordinato cronologicamente delle interruzioni richieste dagli utenti e dalle proprie procedure, e imposta il temporizzatore per la prima scadenza. Quando il temporizzatore genera il segnale d'interruzione, il nucleo avvisa il richiedente, e reimposta il temporizzatore per la prossima scadenza.

In molti calcolatori, la frequenza delle interruzioni generate dall'orologio è fra le 18 e le 60 al secondo. Ciò costituisce un grado di precisione non sufficientemente fine per un calcolatore moderno che può eseguire centinaia di milioni di istruzioni al secondo. La precisione degli impulsi d'attivazione è limitata dalla bassa frequenza del temporizzatore, e dal costo aggiuntivo dato dal mantenimento di orologi virtuali. Inoltre, se lo stesso temporizzatore si usa per fornire l'ora corrente, questa sarà imprecisa. Nella maggior parte dei calcolatori, l'orologio è costruito sulla base di un contatore ad alta frequenza. In alcuni casi, è possibile leggere da un registro il valore di questo contatore, cosicché esso può essere visto come un orologio ad alta precisione. Sebbene non sia in grado di generare segnali d'interruzione, offre una misura accurata dello scorrere del tempo.

13.3.4 I/O bloccante e non bloccante

Un altro aspetto delle chiamate del sistema è la scelta fra I/O bloccante e non bloccante (asincrono). Quando un'applicazione impiega una chiamata del sistema **bloccante**, si sospende l'esecuzione dell'applicazione, che passa dalla coda dei processi pronti per l'esecuzione alla coda d'attesa del sistema. Quando la chiamata del sistema termina, l'applicazione è posta nuovamente nella coda dei processi pronti per l'esecuzione in modo che possa riprendere l'esecuzione; solo allora essa riceverà i valori riportati dalla chiamata del sistema. Le operazioni fisiche compiute dai dispositivi di I/O sono in genere asincrone — richiedono un tempo variabile o non prevedibile. Cionondimeno, la maggior parte dei sistemi operativi impiega chiamate del sistema bloccanti come interfaccia per le applicazioni, perché in questo caso il codice delle applicazioni è più facilmente comprensibile del corrispondente codice non bloccante.

Alcuni processi al livello dell'utente necessitano di una forma **non bloccante** di I/O. Un esempio è quello di un'interfaccia d'utente con cui s'interagisce col mouse e la tastiera mentre elabora dati e li mostra sullo schermo. Un altro esempio è un'applicazione per il video digitale che legge fotogrammi da un file in un'unità a disco e simultaneamente li decomprime e li mostra sullo schermo.

Uno dei modi in cui chi progetta un'applicazione può sovrapporre elaborazione e I/O è di scrivere un'applicazione a più thread. Alcuni di essi eseguono chiamate del sistema bloccanti, mentre altri continuano l'elaborazione. I progettisti del sistema Solaris

hanno usato questa tecnica per costruire una libreria di funzioni del livello d'utente per l'I/O asincrono, sollevando così da questo compito chi sviluppa applicazioni. Alcuni sistemi operativi forniscono chiamate del sistema non bloccanti per l'I/O. Una chiamata di questo tipo non arresta l'esecuzione dell'applicazione per un lasso di tempo significativo. Al contrario, essa restituisce rapidamente il controllo all'applicazione, fornendo un parametro che indica quanti byte di dati sono stati trasferiti.

Una possibile alternativa alle chiamate del sistema non bloccanti è costituita dalle chiamate del sistema asincrone. Esse restituiscono immediatamente il controllo al chiamante, senza attendere che l'I/O sia stato completato. L'applicazione continua a essere eseguita, e il completamento dell'I/O è successivamente comunicato all'applicazione per mezzo dell'impostazione del valore di una variabile nello spazio d'indirizzi dell'applicazione o tramite la generazione di un segnale, o ancora tramite una procedura di ritorno eseguita fuori del normale flusso lineare d'elaborazione dell'applicazione. La differenza fra chiamate del sistema non bloccanti e asincrone è che una `read` non bloccante restituisce immediatamente il controllo, fornendo i dati che è stato possibile leggere (l'intero numero di byte richiesti, parte di essi, o a volte anche nessun dato). Una chiamata `read` asincrona richiede un trasferimento di cui il sistema garantisce il completamento, ma solo in un momento successivo e non prevedibile

Un buon esempio di comportamento non bloccante è la chiamata del sistema `select` per le socket di rete. Essa include un argomento che specifica il tempo d'attesa massimo. Se questo valore è 0, l'applicazione può rilevare attività di rete senza arrestarsi. Tuttavia, l'uso della `select` introduce un carico aggiuntivo, perché essa può stabilire soltanto se sia possibile compiere dell'I/O: per un effettivo trasferimento di dati, la `select` deve essere seguita da istruzioni come `read` o `write`. Una variante di questo metodo, adottata ad esempio dal sistema Mach, è l'impiego di una chiamata del sistema bloccante per la lettura multipla. Essa specifica con una singola chiamata del sistema le desiderate richieste di lettura per diversi dispositivi, e termina non appena una di esse sia stata soddisfatta.

13.4 Sottosistema per l'I/O del nucleo

Il nucleo fornisce molti servizi riguardanti l'I/O, molti — scheduling del'I/O, memorizzazione transitoria, gestione delle cache, gestione delle code, uso esclusivo dei dispositivi e gestione degli errori — sono offerti dal sottosistema per l'I/O del nucleo, e sono realizzati a partire dai dispositivi e dai relativi driver.

13.4.1 Scheduling

Fare lo scheduling di un insieme di richieste di I/O significa stabilirne un ordine d'esecuzione efficace; l'ordine in cui si verificano le chiamate del sistema delle applicazioni è raramente la scelta migliore. Lo scheduling può migliorare le prestazioni complessive del sistema, distribuire equamente gli accessi dei processi ai dispositivi e ridurre il tempo d'attesa medio per il completamento di un'operazione di I/O. Ecco un semplice esempio

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

che illustra queste potenzialità. Si supponga che la testina di lettura di un'unità a disco sia vicina alla parte iniziale del disco, e che tre applicazioni impartiscano comandi di lettura bloccanti per quest'unità. L'applicazione 1 richiede la lettura di un blocco che si trova vicino alla parte finale del disco, l'applicazione 2 quella di un blocco vicino alla parte iniziale e l'applicazione 3 quella di un blocco situato nella zona centrale. Il sistema operativo può ridurre la distanza percorsa dalla testina del disco servendo le richieste nell'ordine 2, 3, 1. Simili riordinamenti delle sequenze di servizio delle richieste sono l'essenza dello scheduling dell'I/O.

I progettisti di sistemi operativi realizzano lo scheduling mantenendo una coda di richieste per ogni dispositivo. Quando un'applicazione richiede l'esecuzione di una chiamata del sistema di I/O bloccante, si aggiunge la richiesta alla coda relativa al dispositivo appropriato. Lo scheduler dell'I/O riorganizza l'ordine della coda per migliorare l'efficienza globale del sistema e il tempo medio d'attesa cui sono sottoposte le applicazioni. Il sistema operativo può anche tentare di essere equo, in modo che nessuna applicazione riceva un servizio carente, o può dare priorità a quelle richieste la cui corretta esecuzione potrebbe essere inficiata da un ritardo nel servizio. Ad esempio, le richieste del sottosistema per la memoria virtuale potrebbero necessitare di una priorità maggiore di quelle delle applicazioni. Parecchi algoritmi di scheduling per l'I/O delle unità a disco sono descritti dettagliatamente nel Paragrafo 14.2.

Lo scheduling dell'I/O è uno dei modi in cui il sottosistema per l'I/O migliora l'efficienza di un calcolatore; un altro è l'uso di spazio di memorizzazione nella memoria centrale o nei dischi, per tecniche di memorizzazione transitoria, uso di cache e di code per la gestione asincrona dell'I/O.

13.4.2 Memorizzazione transitoria

Una **memoria di transito** (*buffer*) è un'area di memoria che contiene dati mentre essi sono trasferiti fra due dispositivi o tra un'applicazione e un dispositivo. La memorizzazione transitoria si usa per tre ragioni. La prima è la necessità di gestire la differenza di velocità fra il produttore e il consumatore di un flusso di dati. Si supponga, ad esempio, di ricevere un file attraverso un modem e di volerlo memorizzare in un'unità a disco: il modem è circa mille volte più lento del disco, perciò conviene predisporre un'area della memoria centrale per contenere i byte che giungono dal modem. Quando tale area di memoria è piena, si trasferisce il suo contenuto nel disco con un'unica operazione. Poiché quest'operazione di scrittura non è istantanea e il modem ha bisogno di ulteriore spazio per memorizzare i dati in arrivo, è necessario impiegare due aree della memoria di questo tipo: quando la prima è piena, si richiede la scrittura nel disco del suo contenuto e il modem comincia a scrivere nella seconda area di memoria; la scrittura nel disco dovrebbe terminare prima che il modem possa riempirla, cosicché il modem potrà ricominciare a usare la prima area della memoria, mentre si trasferisce nel disco il contenuto della seconda. Questa **doppia memorizzazione transitoria** svincola il produttore dal consumatore, rendendo così meno critico il problema della loro sincronizzazione. La necessità di questa procedura è illustrata dalla Figura 13.8, che mostra le enormi differenze di velocità tra i dispositivi tipici di un calcolatore.

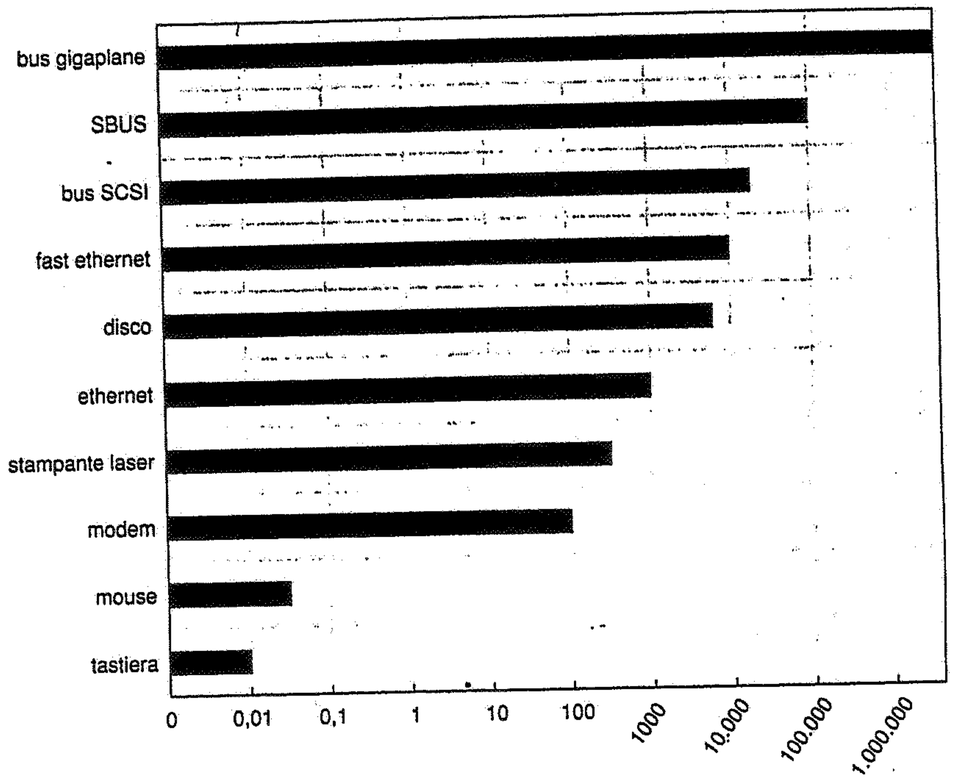


Figura 13.8 Velocità di trasferimento dei dispositivi di un Sun Enterprise 6000 (scala logaritmica).

Un secondo uso della memorizzazione transitoria riguarda la gestione dei dispositivi che trasferiscono dati in blocchi di dimensioni diverse. Queste disparità sono particolarmente comuni nelle reti di calcolatori, dove spesso è necessario frammentare e ricomporre messaggi. Quando un mittente spedisce un messaggio molto lungo, esso è spezzato in piccoli pacchetti che si spediscono attraverso la rete; il sistema destinatario provvede a ricostituire in un'apposita memoria di transito l'intero messaggio originario.

Il terzo modo in cui si può impiegare una memoria di transito è per la realizzazione della *semantica delle copie* nell'ambito dell'I/O delle applicazioni. Un esempio ne chiarirà il significato. Si supponga che un'applicazione disponga di un'area di memoria contenente dati da trasferire in un disco: essa richiederà l'esecuzione della chiamata del sistema `write`, fornendole un puntatore a tale area della memoria e un numero intero che specifica il numero di byte da trasferire. Ci si può chiedere cosa succede se, dopo che la chiamata del sistema restituisce il controllo all'applicazione, quest'ultima modifica il

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

contenuto dell'area di memoria. Ebbene, la **semantica delle copie** garantisce che la versione dei dati scritta nel disco sia conforme a quella contenuta nell'area di memoria al momento della chiamata del sistema, indipendentemente da ogni successiva modifica. Una semplice maniera di realizzare questa semantica consiste nel far sì che la chiamata del sistema `write` copi i dati forniti dall'applicazione in un'area di memoria del nucleo prima di restituire il controllo all'applicazione stessa. La scrittura nel disco si compie dalla memoria del nucleo, cosicché ogni successivo cambiamento nell'area di memoria per l'I/O dell'applicazione non impedirà che la versione dei dati trasferita nel disco sia quella corretta. In molti sistemi operativi si usa il metodo appena descritto: nonostante esso implichi una diminuzione dell'efficienza di certe operazioni di I/O, la sua semantica è chiara. Lo stesso effetto, tuttavia, si può ottenere più efficientemente tramite un uso intelligente della memoria virtuale e della protezione data dalla copiatura su scrittura delle pagine.

13.4.3 Cache

Una **cache** è una regione di una memoria veloce che serve per mantenere copie di certi dati: l'accesso a queste copie è più rapido dell'accesso agli originali. Ad esempio, le istruzioni di un processo correntemente in esecuzione sono memorizzate in un disco, copiate nella memoria fisica (che assume il ruolo di cache rispetto al disco) e copiate ulteriormente nelle cache primaria e secondaria della CPU. La differenza fra una memoria di transito e una cache consiste nel fatto che la prima può contenere dati di cui non esiste un'altra copia, mentre una cache, per definizione, mantiene su un mezzo più efficiente una copia di informazioni già memorizzate.

L'uso delle cache e l'uso delle memorie di transito sono due funzioni distinte, anche se a volte una stessa regione di memoria si può usare per entrambi gli scopi. Ad esempio, per realizzare la semantica delle copie e permettere uno scheduling efficiente delle operazioni di I/O riguardanti il disco, il sistema operativo impiega aree della memoria centrale che contengono copie di dati presenti nei dischi. Esse sono anche usate come cache per migliorare l'efficienza delle operazioni di I/O che coinvolgono file condivisi da più applicazioni o file per i quali gli accessi per lettura e scrittura si susseguano rapidamente. Quando riceve una richiesta di I/O relativa a un file, il nucleo controlla se la parte interessata del file è già presente nella cache: in questo caso è possibile evitare o differire l'accesso al disco. Inoltre, i dati da scrivere nel disco sono depositati nella cache per diversi secondi, cosicché il numero di trasferimenti fisici al disco è ridotto, e ogni trasferimento riguarda grandi quantità di dati: ciò permettere uno scheduling efficiente. La strategia consistente nel differire le scritture per migliorare l'efficienza dell'I/O è discussa nel Paragrafo 16.3.3, nel contesto dell'accesso ai file remoti.

13.4.4 Code e uso esclusivo dei dispositivi

Una coda di file da stampare (*spool*) è una memoria di transito contenente dati per un dispositivo che non può accettare flussi di dati intercalati, ad esempio una stampante. Sebbene una stampante possa servire una sola richiesta alla volta, diverse applicazioni devono poter richiedere simultaneamente la stampa di dati, senza che questi si mischino.

Il sistema operativo risolve questo problema filtrando tutti i dati per la stampante: i dati da stampare provenienti da ogni singola applicazione si registrano in uno specifico file in un disco, quando un'applicazione termina di emettere i dati da stampare, si aggiunge tale file alla coda di stampa; quest'ultima viene copiata sulla stampante, un file per volta. In certi sistemi operativi, questa funzione viene gestita da un processo di sistema specializzato (demone), in altri da un thread del nucleo. In ogni caso, il sistema operativo fornisce un'interfaccia di controllo che permette agli utenti e agli amministratori del sistema di esaminare la coda, eliminare elementi della coda prima che essi siano stampati, sospendere una stampa in corso, e così via.

Alcuni dispositivi, come le unità a nastro e le stampanti, non possono gestire autonomamente in modo efficiente più richieste concorrenti di I/O da parte di diverse applicazioni. L'accodamento è uno dei modi in cui il sistema operativo può coordinare le emissioni simultanee di dati; un altro è quello di fornire esplicite funzioni di coordinamento. Alcuni sistemi operativi, fra i quali il VMS, permettono di accedere a un dispositivo in modo esclusivo: un processo può accedere a un dispositivo che non sia già attivo, riservandosene l'uso; e restituirlo al sistema quando non ne ha più bisogno. Altri sistemi operativi impediscono l'apertura di più di una maniglia di file per un dato dispositivo. Molti sistemi operativi forniscono funzioni che permettono ai processi stessi di coordinare l'uso esclusivo dei dispositivi: il sistema Windows NT, ad esempio, mette a disposizione chiamate del sistema che permettono a un'applicazione di aspettare fino a che un certo dispositivo si liberi. Inoltre la sua chiamata del sistema open accetta un parametro che specifica il tipo d'accesso concesso ai thread concorrenti. In questi sistemi le applicazioni hanno la responsabilità di evitare le situazioni di stallo.

13.4.5 Gestione degli errori

Un sistema operativo che usi la protezione della memoria può proteggersi da molti tipi di errori dovuti ai dispositivi o alle applicazioni, cosicché il blocco completo del sistema non è l'ordinaria conseguenza di piccoli difetti tecnici. I dispositivi e i trasferimenti di I/O possono causare errori in molti modi, sia per motivi contingenti, come il sovraccarico di una rete di comunicazione, sia per ragioni 'permanenti', come nel caso in cui il controllore dell'unità a disco sia difettoso. I sistemi operativi sono spesso capaci di compensare efficacemente le conseguenze negative dovute a errori generati da cause contingenti: se ad esempio una chiamata del sistema `read` non ha successo, il sistema ritenterà la lettura; se una chiamata `send` provoca un errore, il protocollo di rete può richiedere l'esecuzione della chiamata del sistema `resend` per cercare di portare comunque a termine l'operazione originaria. Purtroppo, però, è improbabile che il sistema operativo riesca a compensare gli effetti di errori dovuti a disfunzioni permanenti di qualche componente importante.

Di norma, una chiamata del sistema per l'I/O riporta un bit d'informazione sullo stato d'esecuzione della chiamata, denotando con esso la riuscita o l'insuccesso dell'operazione richiesta. Il sistema operativo UNIX usa una variabile intera detta `errno` per codificare piuttosto genericamente il tipo d'errore avvenuto; i valori possibili sono un centinaio e denotano errori dovuti ad esempio a puntatori non validi, file non aperti o

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

argomenti oltre i limiti ammessi. Per contro, alcuni tipi di dispositivi possono fornire informazioni assai dettagliate sugli errori, sebbene molti sistemi operativi attuali non siano progettati per passare questi dati alle applicazioni. Ad esempio, l'insuccesso di un'operazione di un dispositivo SCSI è registrato dal protocollo SCSI usando un **codice di rilevazione** che identifica la natura generale dell'errore (ad esempio: errore fisico, o richiesta illecita); un secondo codice di rilevazione che descrive la categoria cui appartiene l'insuccesso (parametro del comando errato, o insuccesso della verifica di autovalutazione del funzionamento del dispositivo); e infine un terzo codice di rilevazione che fornisce informazioni ancora più dettagliate (quale parametro è errato, o quale componente del dispositivo non ha superato il procedimento di verifica). Inoltre, molti dispositivi SCSI mantengono alcune pagine di informazioni sugli errori avvenuti; queste pagine possono essere richieste dalla macchina, ma ciò accade raramente.

13.4.6 Strutture di dati del nucleo

Il nucleo ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di I/O, e usa a questo fine diverse strutture di dati interne, un esempio delle quali è la tabella dei file aperti descritta nel Paragrafo 12.2.1. Il nucleo usa molte strutture di questo tipo per tenere traccia dei collegamenti in rete, delle comunicazioni con i dispositivi a caratteri e di altre attività connesse all'I/O.

Il sistema operativo UNIX, per mezzo del file system, permette l'accesso a diversi oggetti: i file degli utenti, i dispositivi, lo spazio d'indirizzi dei processi, e altri ancora. Sebbene ognuno di questi oggetti possa eseguire una chiamata read, le semantiche sono diverse secondo i casi. Quando il nucleo, ad esempio, deve leggere un file d'utente, ha bisogno di controllare la *buffer cache* prima di decidere l'effettiva esecuzione di un'operazione di I/O su un disco. Per leggere un disco tramite un'interfaccia a basso livello, il nucleo deve accertarsi del fatto che la dimensione dell'insieme dei dati di cui è stato richiesto il trasferimento sia un multiplo della dimensione dei settori del disco e sia allineato con il settore interessato. Per leggere l'immagine di un processo, tutto ciò che occorre è copiare dati dalla memoria. UNIX incapsula queste differenze in una struttura uniforme usando una tecnica orientata agli oggetti. Il record dei file aperti, mostrato nella Figura 13.9, contiene una tabella di puntatori alle procedure appropriate secondo il tipo di file in questione.

Alcuni sistemi operativi applicano metodi orientati agli oggetti in misura più rilevante: il sistema Windows NT, ad esempio, usa per l'I/O un sistema basato sullo scambio di messaggi. Una richiesta di I/O si converte in un messaggio che s'invia tramite il nucleo al sottosistema per la gestione dell'I/O, quindi al driver del dispositivo; i contenuti del messaggio possono essere modificati a ogni passaggio intermedio. Quando l'operazione richiesta è di emissione di dati, il messaggio contiene i dati interessati; quando invece l'operazione richiesta è di immissione di dati, il messaggio contiene l'indirizzo dell'area di memoria (*buffer*) che si usa per ricevere i dati. Questo metodo può comportare una minore efficienza rispetto alle tecniche procedurali basate sulla condivisione delle strutture di dati, ma semplifica la progettazione e la struttura del sistema di I/O e permette una maggiore flessibilità.

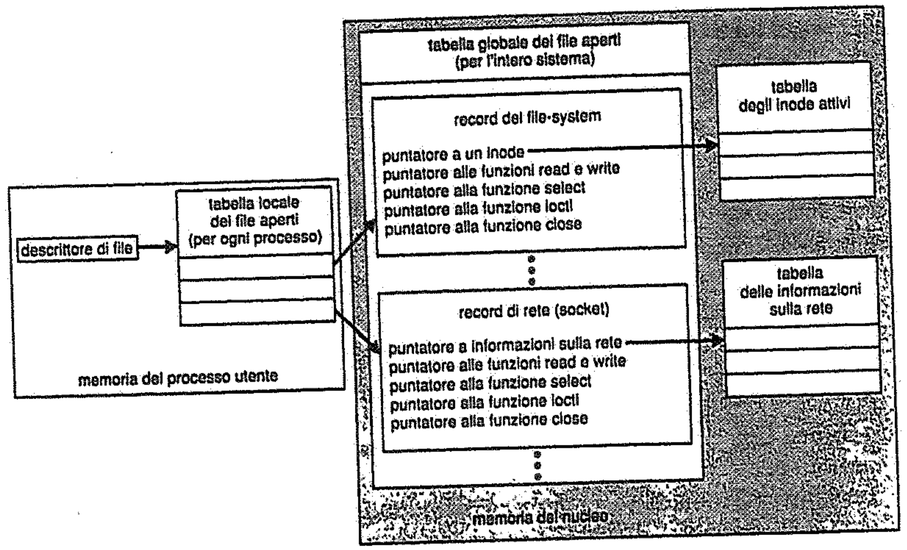


Figura 13.9 Strutture di dati del nucleo per la gestione dell'I/O nello UNIX.

Riassumendo, il sistema per l'I/O coordina un'ampia raccolta di servizi disponibili per le applicazioni e per altre parti del nucleo; in generale sovrintende alle seguenti funzioni:

- gestione dello spazio dei nomi per file e dispositivi;
- controllo dell'accesso ai file e ai dispositivi;
- controllo delle operazioni (ad esempio, un modem non può eseguire una chiamata seek);
- assegnazione dello spazio per il file system;
- assegnazione dei dispositivi;
- memorizzazione transitoria, gestione delle cache e delle code per la gestione asincrona dell'I/O;
- scheduling dell'I/O;
- controllo dello stato dei dispositivi, gestione degli errori e procedure di ripristino;
- configurazione e inizializzazione dei driver dei dispositivi.

I livelli superiori del sottosistema per la gestione dell'I/O accedono ai dispositivi per mezzo dell'interfaccia uniforme fornita dai driver.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

13.5 Trasformazione delle richieste di I/O in operazioni dei dispositivi

Il meccanismo di negoziazione tra un driver e un controllore di dispositivo è già stato illustrato; tuttavia, non si è ancora spiegato come il sistema operativo associ alla richiesta di un'applicazione un insieme di fili di rete o uno specifico settore di disco. Si consideri ad esempio la lettura di un file da un'unità a disco. L'applicazione fa riferimento ai dati per mezzo del nome del file: è compito del file system fornire il modo di giungere, attraverso la struttura delle directory, alla regione del disco appropriata, cioè quella dove i dati del file sono fisicamente residenti. Nell'MS-DOS, ad esempio, il nome del file è associato a un numero che individua un elemento della tabella d'accesso ai file, tale elemento identifica i blocchi del disco assegnati al file. Nello UNIX, il nome è associato a un elemento di una lista di oggetti detti numeri di *inode*, l'*inode* corrispondente contiene le informazioni necessarie per individuare lo spazio assegnato.

Per illustrare come si possa giungere dal nome del file al controllore del disco (all'indirizzo della sua porta o ai suoi registri associati allo spazio d'indirizzi di memoria), conviene innanzi tutto esaminare un sistema relativamente semplice come l'MS-DOS. La prima parte di un nome di file dell'MS-DOS, precisamente la parte che precede i due punti, identifica uno specifico dispositivo. Ad esempio, `C:` è la parte iniziale di ogni nome di file residente nell'unità a disco principale. Questa convenzione è codificata all'interno del sistema operativo: `C:` è associato a uno specifico indirizzo di porta per mezzo di una tabella dei dispositivi. Grazie all'uso dei due punti come separatore, lo spazio per i nomi dei dispositivi è distinto dallo spazio per i nomi dei file, ciò semplifica al sistema operativo l'associazione di funzioni aggiuntive ai dispositivi. Ad esempio, è facile filtrare per l'accodamento i file di cui è stata richiesta la stampa.

Se, invece, i nomi dei dispositivi sono inclusi nell'ordinario spazio dei nomi dei file, come nello UNIX, sono automaticamente disponibili i servizi del file system riguardanti i nomi dei file. Ad esempio, se il file system associa dei possessori ai nomi dei file e fornisce il controllo degli accessi a ogni nome di file, allora si potrà controllare anche l'accesso ai dispositivi, ed essi avranno un possessore. Visto che i file risiedono nei dispositivi, una tale interfaccia fornisce due livelli d'accesso al sistema d'I/O: i nomi si possono usare per accedere ai dispositivi stessi o ai file in essi contenuti.

UNIX rappresenta i nomi dei dispositivi all'interno dell'ordinario spazio dei nomi del file system. A differenza di un nome di file dell'MS-DOS, che include i due punti come separatore, in un nome di percorso dello UNIX non c'è alcuna chiara separazione fra il dispositivo interessato e il nome del file in senso proprio. In effetti, nessuna parte del nome di percorso di un file è il nome di un dispositivo; UNIX impiega una particolare tabella, detta **tabella di montaggio**, per associare i prefissi dei nomi di percorso ai corrispondenti nomi di dispositivi. Quando deve risolvere un nome di percorso, il sistema esamina la tabella per trovare il più lungo prefisso corrispondente: questo elemento della tabella indica il nome del dispositivo voluto. Anche questo nome, però, è rappresentato come un oggetto del file system: tuttavia, quando lo UNIX cerca questo nome nelle strutture di directory del file system, non trova il numero di un *inode*, ma una coppia di numeri *<principale, secondario>* (*<major, minor>*) che identifica un dispositivo. Il numero principale individua il driver di dispositivo che si deve usare per gestire l'I/O nel dispositivo in questione; mentre il numero secondario deve essere passato a questo driver affinché esso possa determinare, per mezzo di un'altra tabella, l'indirizzo della porta o l'indirizzo associato alla memoria del controllore del dispositivo interessato.

I moderni sistemi operativi riescono a ottenere un notevole grado di flessibilità grazie all'uso di tabelle di ricerca durante il processo che porta da una richiesta al controllore del dispositivo; questo processo, inoltre, è del tutto generale, cosicché non è necessario ricompilare il nucleo ogni volta che si aggiungono al calcolatore nuovi dispositivi e nuovi driver. In effetti, alcuni sistemi operativi hanno la capacità di caricare driver di dispositivi su richiesta: all'avviamento, il sistema sonda i bus per determinare quali dispositivi sono presenti; quindi carica i necessari driver, operazione che può anche essere rinviata fino alla prima richiesta di I/O.

La seguente descrizione del tipico svolgimento di una richiesta di lettura bloccante (Figura 13.10) indica che l'esecuzione di un'operazione di I/O richiede una grande quantità di passi, ciò implica l'uso di un enorme numero di cicli di CPU.

1. Un processo impartisce una chiamata del sistema `read` bloccante relativa a un descrittore di file di un file precedentemente *aperto*.
2. Il codice della chiamata del sistema all'interno del nucleo controlla la correttezza dei parametri. Se sono già presenti nella *buffer cache*, si passano i dati richiesti al processo chiamante e l'operazione è conclusa.
3. Altrimenti, è necessario eseguire un'operazione fisica di I/O, così si rimuove il processo dalla coda dei processi pronti per l'esecuzione per inserirlo nella coda d'attesa relativa al dispositivo interessato e si effettua lo scheduling della richiesta di I/O. Infine il sottosistema per l'I/O invia la richiesta al driver del dispositivo; secondo il sistema operativo, ciò avviene tramite la chiamata di una procedura, o per mezzo dell'emissione di un messaggio interno al nucleo.
4. Il driver del dispositivo assegna un'area di memoria nello spazio d'indirizzi del nucleo che serve per ricevere i dati immessi, ed esegue lo scheduling dell'I/O. Infine il driver impartisce comandi al controllore del dispositivo scrivendo nei suoi registri.
5. Il controllore aziona il dispositivo per compiere il trasferimento dei dati.
6. Il driver può eseguire un'interrogazione ciclica, o può aver predisposto un trasferimento DMA nella memoria del nucleo. Si assume che il trasferimento sia gestito dal controllore DMA, il quale genera un'interruzione al termine dell'operazione.
7. Tramite il vettore delle interruzioni, si attiva l'appropriato gestore dell'interruzione che, dopo aver memorizzato i dati necessari, avverte con un segnale il driver del dispositivo e restituisce il controllo.

http://www.adobe.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

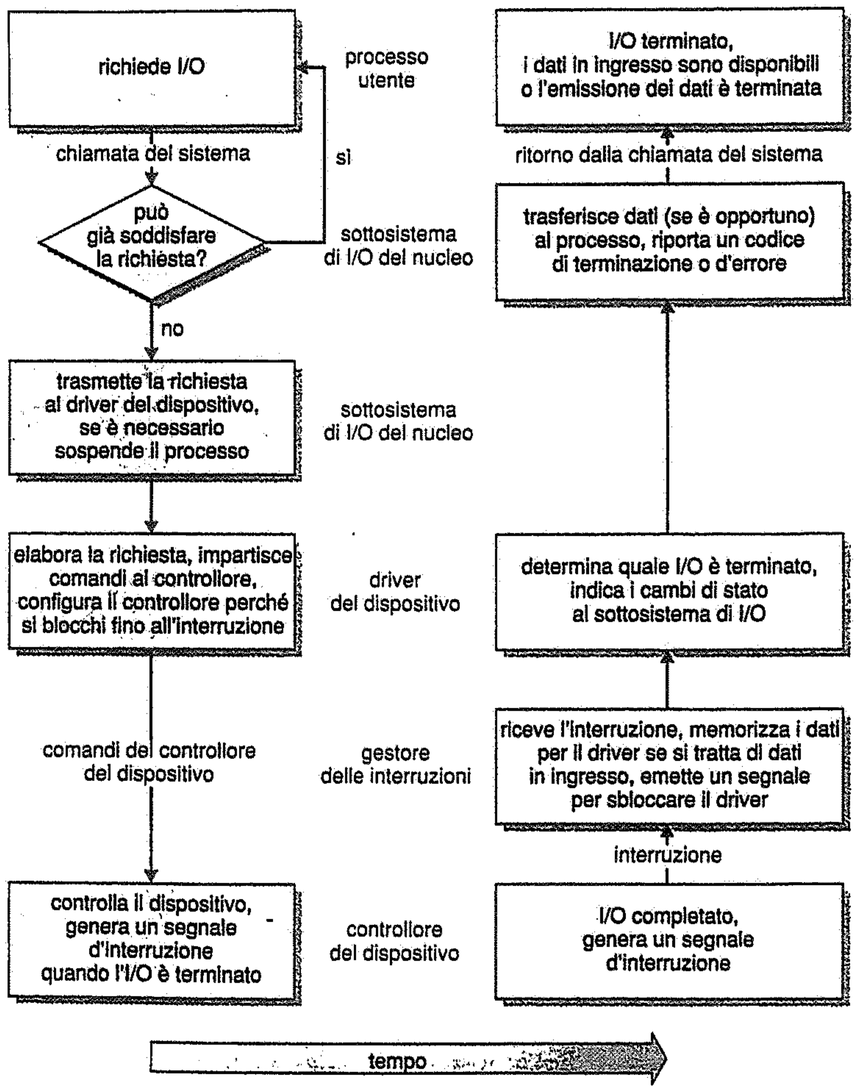


Figura 13.10 Schema d'esecuzione di una richiesta di I/O.

8. Il driver riceve il segnale, individua la richiesta di I/O, si accerta della riuscita o del fallimento dell'operazione e segnala al sottosistema per l'I/O del nucleo il completamento dell'operazione.

9. Il nucleo trasferisce dati (nel caso di successo) e/o codici di stato (per comunicare, ad esempio, la mancata riuscita dell'operazione) nello spazio degli indirizzi del processo chiamante, e sposta tale processo dalla coda d'attesa alla coda dei processi pronti per l'esecuzione.
10. Nel momento in cui è posto nella coda dei processi pronti per l'esecuzione, il processo non è più bloccato: quando lo scheduler gli assegnerà la CPU, esso riprenderà l'elaborazione. L'esecuzione della chiamata del sistema è completata.

13.6 STREAMS

Il sistema operativo UNIX System V offre un interessante meccanismo, chiamato STREAMS, che permette ad un'applicazione di comporre dinamicamente catene di codice di driver. Uno *stream* è una connessione *full-duplex* fra un driver di dispositivo e un processo utente, e consiste di un elemento iniziale d'interfaccia per il processo utente (*stream head*), un elemento terminale che controlla il dispositivo (*driver end*), ed eventualmente un certo numero di moduli intermedi fra questi due estremi (*stream modules*). Tutti questi elementi contengono una coppia di code, una di lettura e una di scrittura; per il trasferimento dei dati tra le due code s'impiega uno schema a scambio di messaggi. La Figura 13.11 mostra la struttura del meccanismo di STREAMS.

Le funzioni d'elaborazione di STREAMS sono offerte da moduli che si *inseriscono* in una catena di moduli attraverso la chiamata del sistema `ioctl`. Un processo può, ad esempio, accedere tramite questo meccanismo a un dispositivo collegato a una porta seriale, e può inserire un modulo per controllare la gestione dei dati immessi. Poiché i messaggi sono scambiati tra code di moduli adiacenti, la coda di un modulo potrebbe sconfinare nella coda di un modulo adiacente. Per prevenire questo problema, una coda può disporre di un meccanismo di **controllo di flusso**. Senza controllo di flusso, una coda accetta tutti i messaggi e li trasferisce immediatamente alla coda del modulo adiacente, senza memorizzazioni transitorie. Una coda che invece impiega il controllo di flusso memorizza i messaggi in una memoria di transito; se lo spazio disponibile in questa non è sufficiente, non si accettano messaggi. Il controllo di flusso si gestisce tramite scambi di messaggi di controllo tra code in moduli adiacenti.

Un processo utente usa le chiamate del sistema `write` o `putmsg` per scrivere dati in un dispositivo; la chiamata del sistema `write` scrive semplicemente i dati nello *stream*, mentre `putmsg` permette al processo utente di specificare un messaggio. Indipendentemente dalla chiamata del sistema adoperata dal processo utente, l'elemento iniziale d'interfaccia col processo utente copia i dati in un messaggio e li recapita alla coda del modulo successivo. Questa copiatura dei messaggi continua fino a quando il messaggio giunge all'elemento terminale di controllo del dispositivo e quindi al dispositivo. Analogamente, il processo utente legge i dati dall'elemento iniziale d'interfaccia usando la chiamata del sistema `read` oppure `getmsg`. Se si usa la `read`, l'elemento iniziale d'interfaccia preleva un messaggio dalla coda del modulo adiacente e riporta al processo dati or-

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

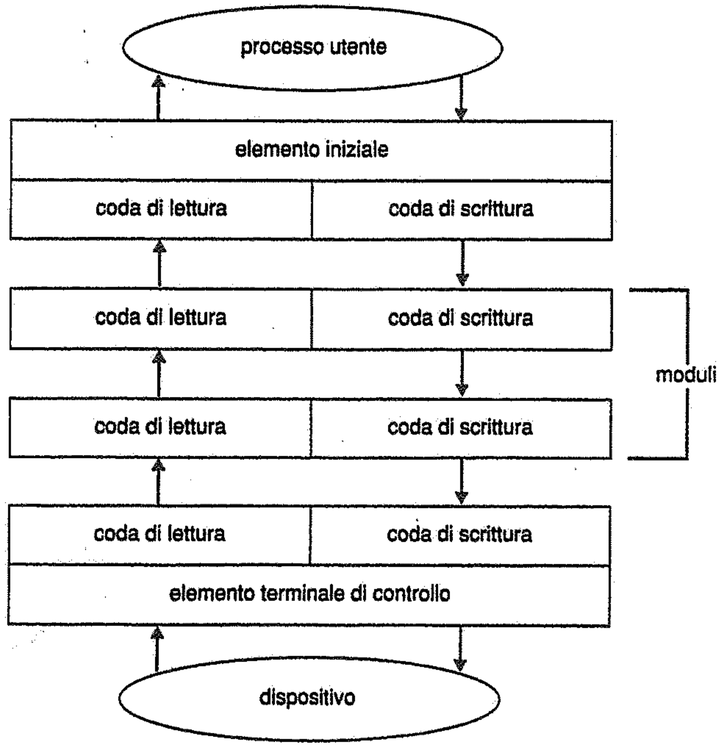


Figura 13.11 Struttura di STREAMS.

dinari (una sequenza non strutturata di byte). Se si usa la `getmsg`, l'elemento iniziale d'interfaccia riporta un messaggio al processo utente.

L'I/O per mezzo di STREAMS è asincrono (o non bloccante), con l'eccezione di quando il processo utente comunica con l'elemento iniziale d'interfaccia. Mentre scrive nello *stream*, il processo utente si blocca, assumendo che la coda successiva impieghi il controllo di flusso, fino a quando non c'è spazio sufficiente per copiarvi il messaggio. Analogamente, il processo utente si blocca durante la lettura dallo *stream* fino a quando non ci sono dati disponibili.

L'elemento terminale di controllo del dispositivo è simile all'elemento iniziale d'interfaccia o a un modulo intermedio nell'avere una coda di lettura e una di scrittura. Tuttavia deve poter rispondere a interruzioni come quelle generate quando un pacchetto di dati è pronto per essere letto da una rete. A differenza dell'elemento iniziale d'interfaccia che si può bloccare se non è possibile copiare un messaggio nella coda del modulo successivo, deve gestire tutti i dati in arrivo. I driver devono anche disporre del controllo di flusso. Tuttavia, se la memoria di transito di un dispositivo è piena, di solito s'ignorano i messaggi in entrata. Si consideri una scheda di rete la cui memoria di transito d'ingresso

sia piena; la scheda di rete deve semplicemente ignorare gli ulteriori messaggi fintantoché non vi sia sufficiente spazio per memorizzare i messaggi in arrivo.

Il vantaggio nell'utilizzo di STREAMS consiste nel disporre di un ambiente che permette uno sviluppo modulare e incrementale di driver di dispositivi e protocolli di rete.

I moduli possono essere usati da diversi STREAMS e quindi da diversi dispositivi. Ad esempio, un modulo di rete potrebbe essere adoperato sia da una scheda di rete Ethernet sia da una scheda di rete Token Ring. Inoltre, invece di trattare l'I/O di dispositivi a caratteri come una sequenza non strutturata di byte, STREAMS permette la gestione dei limiti dei messaggi e delle informazioni di controllo tra i diversi moduli. L'impiego di STREAMS è assai diffuso nella maggior parte dei sistemi UNIX, ed è il metodo più usato per la scrittura di protocolli e driver di dispositivi. Nello UNIX System V e nel Solaris, ad esempio, il meccanismo delle socket è realizzato tramite STREAMS.

13.7 Prestazioni

L'I/O è uno tra i principali fattori che influiscono sulle prestazioni di un sistema: richiede un notevole impegno della CPU per l'esecuzione del codice dei driver e per uno scheduling equo ed efficiente dei processi quando essi sono bloccati e riavviati. I risultanti cambi di contesto sfruttano fino in fondo la CPU e le sue memorie cache. L'I/O, inoltre, rivela le eventuali inefficienze dei meccanismi del nucleo per la gestione delle interruzioni, e impegna il bus della memoria durante i trasferimenti di dati tra i controllori dei dispositivi e la memoria fisica, e ancora tra le aree di memoria per l'I/O del nucleo e lo spazio d'indirizzi delle applicazioni. Soddisfare in modo elegante tutte queste esigenze è una delle principali questioni che riguardano i progettisti di un calcolatore.

Sebbene i calcolatori moderni siano capaci di gestire migliaia di interruzioni al secondo, la gestione delle interruzioni è un processo relativamente oneroso: ciascuna di esse fa sì che il sistema cambi stato, esegua il gestore delle interruzioni, e infine ripristini lo stato originario. Se il numero di cicli impiegato nell'attesa attiva non è eccessivo, l'I/O programmato può essere più efficiente di quello basato sulle interruzioni. Il completamento di un'operazione di I/O implica tipicamente la riattivazione di un processo, comportando così lo svantaggio in termini di efficienza dovuto a un cambio di contesto.

Anche il traffico di una rete può portare a un alto numero di cambi di contesto; si consideri, ad esempio, una connessione a distanza fra due calcolatori. Ciascun carattere inserito in un calcolatore deve essere comunicato all'altro: quando si inserisce un carattere nel primo calcolatore, la tastiera produce un segnale d'interruzione; il carattere arriva tramite il gestore delle interruzioni al driver del dispositivo, al nucleo, e quindi al processo utente, il quale impartisce una chiamata del sistema per l'I/O di rete al fine di inviare il carattere al secondo calcolatore. All'interno del nucleo del primo calcolatore, il carattere attraversa gli strati di programmi e protocolli necessari per la costruzione di un pacchetto di rete e giunge al driver della rete, il quale trasferisce il pacchetto al controllore della rete. Quest'ultimo invia il carattere e genera un'interruzione che segnala al nucleo il completamento dell'operazione di I/O di rete.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

A questo punto, il dispositivo di rete del secondo calcolatore riceve il pacchetto, e genera un'interruzione. I protocolli di rete estraggono il carattere dal pacchetto e lo consegnano all'appropriato demone di rete, il quale individua la provenienza del carattere e lo passa al sottodemone che gestisce la sessione d'accesso del primo calcolatore. Durante questo processo avvengono cambi di contesto e di stato (Figura 13.12). Di solito il destinatario rispedisce al mittente, sotto forma di eco, una copia del carattere originario, e ciò raddoppia il lavoro necessario.

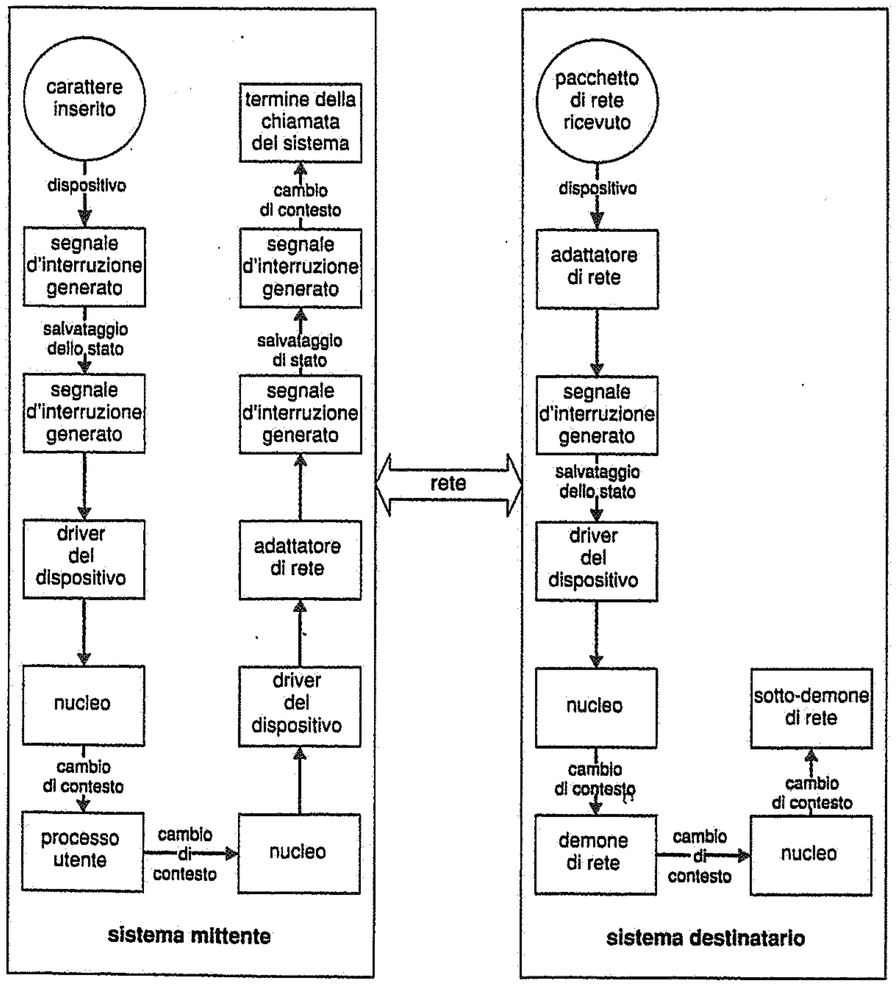


Figura 13.12 Comunicazione fra calcolatori e suoi costi.

I progettisti del sistema operativo Solaris hanno riscritto il demone `telnet` usando thread interni al nucleo per eliminare i cambi di contesto necessari per trasferire caratteri fra i demoni e il nucleo. Secondo stime della Sun, queste migliorie hanno portato il massimo numero di connessioni contemporanee sostenibili da un grande server da qualche centinaio a qualche migliaio.

Altri sistemi usano unità d'elaborazione specifiche per la gestione dell'I/O dei terminali, riducendo così il carico delle gestione delle interruzioni gravante sulla CPU. Ad esempio, i **concentratori di terminali** convogliano il traffico di informazioni proveniente da centinaia di terminali a un'unica porta di un grande calcolatore. I **canali di I/O** sono unità d'elaborazione specializzate presenti nei mainframe e altri sistemi di alto profilo, il loro compito è di sollevare la CPU da una parte del peso della gestione dell'I/O. L'idea di base è che i canali di I/O mantengano il flusso dei dati costante e uniforme, mentre la CPU rimane libera di elaborare i dati acquisiti. Come i controllori DMA e i dispositivi che si trovano nei calcolatori di minori dimensioni, un canale può eseguire programmi più raffinati e generali, e può quindi essere tarato per specifici carichi di lavoro.

Per migliorare l'efficienza dell'I/O si possono applicare diversi princìpi:

- Ridurre il numero dei cambi di contesto.
- Ridurre il numero di copiature dei dati nella memoria durante i trasferimenti fra dispositivi e applicazioni.
- Ridurre la frequenza delle interruzioni tramite il trasferimento di grandi quantità di dati in un'unica soluzione, usare controllori intelligenti e l'interrogazione ciclica (nel caso in cui i tempi d'attesa attiva si possano minimizzare).
- Aumentare il tasso di concorrenza usando controllori DMA intelligenti o canali di I/O per sollevare la CPU dalle semplici copiature di dati.
- Realizzare le primitive direttamente tramite i dispositivi fisici, così da permettere che la loro esecuzione sia simultanea alle operazioni di bus e di CPU.
- Equilibrare le prestazioni della CPU, del sottosistema per la gestione della memoria, del bus e dell'I/O, giacché il sovraccarico di uno qualunque di questi settori provoca l'inutilizzo degli altri.

La complessità dei dispositivi è assai variabile: un mouse ad esempio è piuttosto semplice; i suoi movimenti e la pressione dei pulsanti sono convertiti in valori numerici che sono passati attraverso il driver del mouse, all'applicazione. Per contro, i servizi forniti dal driver delle unità a disco del sistema operativo Windows NT sono assai complessi: non solo il driver gestisce singole unità a disco, ma costruisce anche batterie RAID (si veda il Paragrafo 14.5) convertendo le richieste di lettura o scrittura di un'applicazione in un insieme coordinato di operazioni di I/O sui dischi. Il driver, inoltre, esegue raffinati algoritmi di gestione degli errori e di recupero dei dati, e poiché hanno un'incidenza notevole sulle prestazioni complessive del sistema, svolge diverse funzioni di ottimizzazione delle prestazioni dell'unità a disco.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

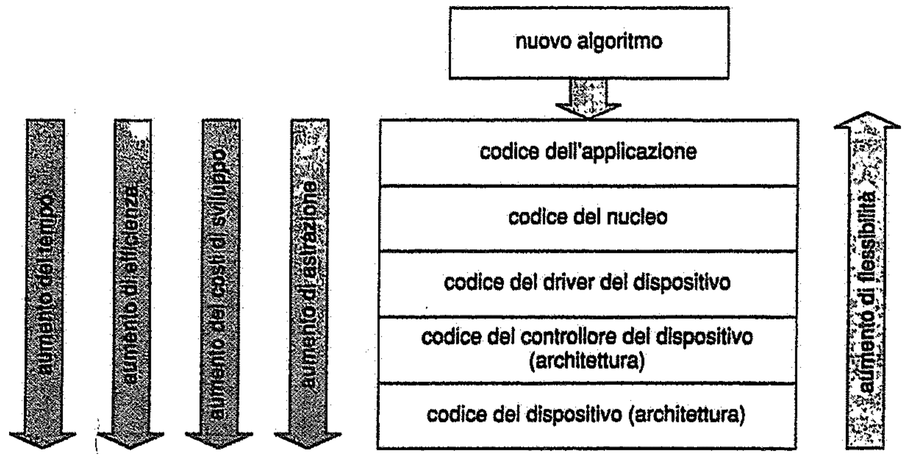


Figura 13.13 Successione delle realizzazioni dei servizi di I/O.

Ci si può chiedere se i servizi di I/O si debbano incorporare nei dispositivi, nei loro driver, o nelle applicazioni. Talvolta si può osservare (Figura 13.13) la seguente successione:

- Inizialmente, gli algoritmi sperimentali per l'I/O si codificano al livello delle applicazioni, ciò per ragioni di flessibilità, e poiché è difficile che errori di programmazione causino l'arresto completo del sistema. In questo modo si evita inoltre di dover riavviare o ricaricare i driver dei dispositivi ogni volta che si modifica il codice degli algoritmi. Tuttavia, questi algoritmi sono spesso inefficienti a causa del sovraccarico dovuto ai cambi di contesto necessari e dell'impossibilità di sfruttare le strutture di dati del nucleo e i suoi meccanismi interni (ad esempio, la gestione dei messaggi, l'uso dei thread, l'accesso bloccante alle risorse).
- Quando uno di questi algoritmi è stato messo a punto, è possibile codificarlo all'interno del nucleo. Ciò può portare a un miglioramento delle prestazioni, ma la stesura del codice è più impegnativa, perché il nucleo è un sistema vasto e complesso. È inoltre necessario verificare accuratamente la correttezza del codice al fine di evitare alterazioni dei dati e l'arresto del sistema.
- Le prestazioni migliori si ottengono con l'integrazione delle funzioni di tali algoritmi nei dispositivi o nei controllori. Gli svantaggi di questa tecnica comprendono la difficoltà e il costo di successive migliorie o dell'eliminazione di eventuali errori, il maggior tempo richiesto per portarne a termine la realizzazione (mesi invece di giorni), e la minore flessibilità. Ad esempio, un controllore RAID può essere privo di funzioni che permettono al nucleo di modificare la locazione o l'ordine di lettura o scrittura di singoli blocchi, anche se il nucleo potrebbe possedere informazioni particolari sul carico di lavoro che gli permetterebbero di migliorare le prestazioni dell'I/O.

13.8 Sommario

I principali elementi fisici di un calcolatore coinvolti nell'esecuzione dell'I/O sono i bus, i controllori dei dispositivi e i dispositivi stessi. Quando l'I/O è programmato, i trasferimenti dei dati tra i dispositivi e la memoria centrale sono controllati dalla CPU; altrimenti, sono demandati al controllore DMA. Un modulo del nucleo che controlla un dispositivo è detto driver. L'interfaccia fornita alle applicazioni, costituita dalle chiamate del sistema, è progettata per gestire diverse categorie fondamentali di elementi, come i dispositivi a blocchi e a caratteri, i temporizzatori programmabili, i file associati alla memoria, le socket di rete. Di solito le chiamate del sistema bloccano il processo che le ha invocate; il nucleo e le applicazioni che non devono attendere il completamento di un'operazione di I/O impiegano chiamate del sistema non bloccanti o asincrone.

Il sottosistema del nucleo per l'I/O fornisce numerosi servizi: fra gli altri, lo scheduling dell'I/O, la memorizzazione transitoria (*buffering*), la gestione asincrona con accodamento delle operazioni di I/O (*spooling*), la gestione degli errori e l'uso esclusivo dei dispositivi. Un altro servizio è l'interpretazione dei nomi, che permette di associare ai nomi simbolici dei file usati dalle applicazioni i dispositivi corrispondenti. Si tratta di un processo che passa attraverso diversi stadi intermedi: dalla sequenza originale di caratteri a un driver specifico e all'indirizzo di un dispositivo, e da qui all'indirizzo fisico delle porte di I/O o dei controllori. Tale interpretazione può avvenire nell'ambito dello spazio dei nomi del file system (come nello UNIX), o in uno specifico spazio per i nomi dei dispositivi (come nell'MS-DOS).

STREAMS è una realizzazione e un metodo per rendere i driver riutilizzabili e facili da usare. Tramite questo metodo i driver di possono comporre in modo modulare; il passaggio dei dati per l'elaborazione attraverso di essi è sequenziale e bidirezionale.

A causa dei molti strati di programmi presenti fra un dispositivo fisico e l'applicazione, le chiamate del sistema per l'I/O sono onerose in termini di tempo d'uso della CPU. Questa struttura a strati implica costi aggiuntivi per quel che riguarda l'uso delle risorse necessarie per realizzare i cambi di contesto, gestire le interruzioni e i segnali usati per la comunicazione con i dispositivi, e copiare dati fra le aree di memoria per l'I/O del nucleo e lo spazio d'indirizzi delle applicazioni.

13.9 Esercizi

13.1 Elencate tre vantaggi e tre svantaggi derivanti dal decentramento dei servizi dal nucleo ai controllori dei dispositivi.

13.2 Considerate le seguenti situazioni riguardanti l'I/O in un PC monoutente:

a) un mouse usato insieme con un'interfaccia grafica per l'utente;
b) un lettore di nastri in un sistema operativo multitasking (assumete l'impossibilità di preassegnare i dispositivi);

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

c) un'unità a disco contenente i file dell'utente;
d) una scheda grafica con connessione diretta tramite bus, accessibile per mezzo dell'I/O associato alla memoria.

Per ognuna di queste situazioni, dite se è opportuno progettare il sistema operativo in modo che possa impiegare la memorizzazione transitoria, la gestione asincrona con accodamento delle operazioni di I/O, memorie cache, o una loro combinazione. Dite inoltre se è opportuno usare le interrogazioni cicliche o le interruzioni. Argomentate le risposte.

13.3 L'esempio di negoziazione dato nel Paragrafo 13.2 impiega due bit: il primo detto busy, il secondo command-ready. Dite se è o non è possibile realizzare questa stessa negoziazione con un solo bit. Nel caso di risposta affermativa, descrivete il protocollo; altrimenti, spiegate perché un bit non è sufficiente.

13.4 Descrivete tre circostanze nelle quali sarebbe opportuno usare l'I/O bloccante, e altre tre nelle quali ciò non sarebbe invece opportuno. Riflettete sulla possibilità di realizzare semplicemente l'I/O non bloccante e lasciare che i processi interroghino ciclicamente i dispositivi richiesti finché essi sono pronti.

13.5 Spiegate perché un sistema potrebbe usare le interruzioni per gestire una singola porta seriale, ma le interrogazioni cicliche per un'unità d'elaborazione specifica per l'I/O, come un concentratore di terminali.

13.6 L'interrogazione ciclica nell'attesa del completamento di un'operazione di I/O può sprecare un gran numero di cicli di CPU se la CPU itera le interrogazioni molte volte prima che l'operazione sia conclusa. Ma se il dispositivo interessato è pronto a fornire il servizio, questa tecnica può essere molto più efficiente della gestione delle interruzioni. Descrivete una strategia ibrida che combini interrogazioni cicliche, attese passive e interruzioni per gestire l'I/O. Descrivete un sistema informatico nel quale ciascuna delle tre strategie considerate (interrogazione ciclica, interruzioni, strategia ibrida) è più efficiente delle due rimanenti.

13.7 Il sistema operativo UNIX coordina le attività dei componenti per l'I/O del nucleo manipolando strutture di dati condivise, mentre il sistema operativo Windows NT usa tecniche orientate agli oggetti di scambio di messaggi tra i componenti del nucleo. Discutete tre pro e tre contro di ciascun metodo.

13.8 Spiegate come la tecnica DMA aumenti il grado di concorrenza del sistema. Spiegate inoltre in che modo essa complica la progettazione dell'architettura di un calcolatore.

13.9 Scrivete in uno pseudocodice una procedura che realizzi un orologio virtuale che comprenda l'accodamento e la gestione delle richieste del temporizzatore per il nucleo e le applicazioni. Assumete che siano disponibili tre canali di temporizzazione.

13.10 Spiegate perché è importante aumentare la velocità dei dispositivi e del bus di sistema all'aumentare della velocità della CPU.

13.11 Spiegate la differenza fra un driver STREAMS e un modulo STREAMS.

13.10 Note bibliografiche

[Vahalia 1996] dà una buona visione d'insieme dell'I/O e della comunicazione di rete nel sistema operativo UNIX. [McKusick et al. 1996] descrive dettagliatamente le strutture e i metodi d'I/O usati nel BSD UNIX. [Milenkovic 1987] discute la complessità di metodi e tecniche riguardanti l'I/O. L'uso e la programmazione dei vari protocolli per la comunicazione di rete e fra processi sono trattati in [Stevens 1992]. [Brain 1996] documenta l'interfaccia per le applicazioni del sistema operativo Windows NT. La realizzazione dell'I/O nel sistema operativo didattico MINIX è descritta in [Tanenbaum e Woodhull 1997]. [Custer 1994] include informazioni dettagliate sulla realizzazione dell'I/O basata sui messaggi del sistema Windows NT.

Per i dettagli al livello fisico relativi alla gestione dell'I/O e dell'I/O associato alla memoria, i manuali delle CPU ([Motorola 1993], [Intel 1993]) sono fra le fonti migliori. [Hennessy e Patterson 1996] descrive i sistemi dotati di più unità d'elaborazione e le questioni relative alla coerenza della cache. [Tanenbaum 1990] descrive la progettazione dell'architettura per l'I/O fino al basso livello, e [Sargent e Shoemaker 1995] fornisce una guida per programmatori all'architettura dei PC. La tabella degli indirizzi dei dispositivi di I/O per PC IBM si trova in [IBM 1983]. [*IEEE Computer* marzo 1994] è dedicato ad architettura e programmazione per l'I/O. [Rago 1993] offre una buona trattazione del meccanismo STREAMS.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 14

Memoria secondaria e terziaria

Il file system, da un punto di vista logico, si può considerare composto da tre parti: nel Capitolo 11 è discussa l'interfaccia per il programmatore e per l'utente del file system; nel Capitolo 12 sono descritte le strutture di dati interne e gli algoritmi usati dal sistema operativo per realizzare quest'interfaccia; nel presente capitolo si discute il livello più basso del file system, e cioè la struttura dei mezzi di memorizzazione secondaria e terziaria. Si descrivono innanzi tutto gli algoritmi di scheduling delle unità a disco; essi ordinano la sequenza delle operazioni di I/O al fine di migliorare le prestazioni del sistema. Quindi si discutono la formattazione dei dischi e la gestione dei blocchi d'avviamento, dei blocchi danneggiati e dell'area d'avvicendamento dei processi. Si esaminano la struttura della memoria secondaria, trattando l'affidabilità dei dischi e la realizzazione della memoria stabile. Il capitolo termina con una breve descrizione dei dispositivi di memoria terziaria e dei problemi che si presentano quando in un sistema operativo si usa la memorizzazione terziaria.

14.1 Struttura dei dischi

I dischi sono il principale mezzo di memorizzazione secondaria nei moderni sistemi di calcolo. I nastri magnetici avevano in passato questo ruolo, ma il loro tempo d'accesso è molto maggiore di quello dei dischi, per questo motivo oggi si usano principalmente per conservare copie di riserva, per l'archiviazione d'informazioni usate raramente, come mezzo intermedio per il trasferimento d'informazioni da un sistema a un altro e per la memorizzazione di quantità di dati così ingenti da non poter essere convenientemente contenute in dischi magnetici. Maggiori dettagli sulle unità a nastro si trovano nel Paragrafo 14.8.

Dal punto di vista dell'indirizzamento, i moderni dischi si considerano come un grande vettore monodimensionale di **blocchi logici**, dove un blocco logico è la minima unità di trasferimento. La dimensione di un blocco logico è di solito di 512 byte, sebbene alcuni dischi si possano **formattare a basso livello** allo scopo di ottenere una diversa

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dimensione dei blocchi logici, ad esempio 1024 byte; per altre informazioni su quest'opzione, si veda il Paragrafo 14.3.1.

Il vettore monodimensionale di blocchi logici corrisponde in modo sequenziale ai settori del disco: il settore 0 è il primo settore della prima traccia sul cilindro più esterno; la corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via di cilindro in cilindro dall'esterno verso l'interno.

Sfruttando questa corrispondenza sarebbe possibile — almeno in teoria — trasformare il numero di un blocco logico in un indirizzo fisico di vecchio tipo relativo al disco, consistente in un numero di cilindro, un numero di traccia concernente quel cilindro, e un numero di settore relativo a quella traccia. In pratica, però, vi sono due motivi che rendono difficile quest'operazione: in primo luogo, la maggior parte dei dischi contiene settori difettosi, ma la corrispondenza nasconde questo fatto sostituendo ai settori malfunzionanti settori sparsi in altre parti del disco; in secondo luogo, il numero di settori per traccia in certe unità a disco non è costante. Nei mezzi che impiegano la **velocità lineare costante** (*constant linear velocity* — CLV) la densità di bit per traccia è uniforme. Più è lontana dal centro del disco, tanto maggiore è la lunghezza della traccia, tanto maggiore è il numero di settori che essa può contenere. Spostandosi da aree esterne verso aree più interne il numero di settori per traccia diminuisce. Le tracce nell'area più esterna contengono tipicamente il 40 per cento in più dei settori contenuti nelle tracce dell'area più interna. L'unità aumenta la sua velocità di rotazione a mano a mano che le testine si spostano dalle tracce esterne verso le tracce più interne per mantenere costante la quantità di dati che scorrono sotto le testine. Questo metodo si usa nelle unità per CD-ROM e DVD. In alternativa la velocità di rotazione dei dischi può rimanere costante, e la densità di bit decresce dalle tracce interne alle tracce più esterne per mantenere costante la quantità di dati che scorre sotto le testine. Questo metodo si usa nelle unità a disco magnetico ed è noto come **velocità angolare costante** (*constant angular velocity* — CAV).

Il numero di settori per traccia cresce al migliorare della tecnologia dei dischi, e l'area più esterna di un disco di solito contiene centinaia di settori per traccia. Anche il numero di cilindri è andato aumentando; le unità a disco contengono decine di migliaia di cilindri.

14.2 Scheduling del disco

Una delle responsabilità del sistema operativo è quella di fare un uso efficiente delle risorse fisiche: nel caso delle unità a disco, far fronte a questa responsabilità significa garantire tempi d'accesso contenuti e ampiezze di banda elevate. Il tempo d'accesso si può scindere in due componenti principali (si veda anche il Paragrafo 2.3.2): il **tempo di ricerca** (*seek time*), cioè il tempo necessario affinché il braccio dell'unità a disco sposti le testine fino al cilindro contenente il settore desiderato; e la **latenza di rotazione** (*rotational latency*), e cioè il tempo aggiuntivo necessario perché il disco ruoti finché il settore desiderato si trova sotto la testina. L'**ampiezza di banda** (*bandwidth*) è il numero totale di byte trasferiti diviso il tempo totale intercorso fra la prima richiesta e il completamento dell'ultimo trasferimento. Per mezzo dello scheduling delle richieste di I/O relative al disco si possono migliorare sia il tempo d'accesso sia l'ampiezza di banda.

Ogni volta che deve compiere operazioni di I/O con un'unità a disco, un processo impartisce una chiamata del sistema al sistema operativo. La richiesta contiene diverse informazioni:

- Se l'operazione sia di immissione o di emissione di dati.
- L'indirizzo nel disco rispetto al quale eseguire il trasferimento.
- L'indirizzo di memoria rispetto al quale eseguire il trasferimento.
- Il numero di byte da trasferire.

Se l'unità a disco desiderata e il controllore sono disponibili, la richiesta si può immediatamente soddisfare; altrimenti le nuove richieste si aggiungono alla coda di richieste inevase relativa a quell'unità. La coda relativa a un'unità a disco in un sistema con multiprogrammazione può spesso essere piuttosto lunga, sicché il sistema operativo sceglie quale fra le richieste inevase conviene servire prima.

14.2.1 Scheduling in ordine d'arrivo — FCFS

La forma più semplice di scheduling è, naturalmente, l'algoritmo di servizio secondo l'ordine d'arrivo (*first come, first served* — FCFS). Si tratta di un algoritmo intrinsecamente equo, ma che in generale non garantisce la massima velocità del servizio. Si consideri, ad esempio, una coda di richieste per l'unità a disco che dia una lista di cilindri sui quali individuare i blocchi richiesti, nell'ordine seguente:

98, 183, 37, 122, 14, 124, 65, 67.

Se si trova inizialmente al cilindro 53, la testina dell'unità a disco dovrà prima spostarsi al cilindro 98, poi al 183, 37, 122, 14, 124, 65 e infine al 67, per una distanza totale, misurata in numero di cilindri visitati, di 640 cilindri. La sequenza è rappresentata nella Figura 14.1.

Le deficienze di quest'algoritmo sono palesate dal gigantesco salto da 122 a 14 e poi di nuovo a 124: se le richieste per i cilindri 37 e 14 si potessero soddisfare in sequenza, la distanza totale percorsa diminuirebbe notevolmente e le prestazioni migliorerebbero di conseguenza.

14.2.2 Scheduling per brevità — SSTF

Sembra ragionevole servire tutte le richieste vicine alla posizione corrente della testina prima di spostarla in un'area lontana per soddisfarne altre: questa considerazione è alla base dell'**algoritmo di servizio secondo il più breve tempo di ricerca** (*shortest seek time first* — SSTF). Esso sceglie la richiesta che dà il minimo tempo di ricerca rispetto alla posizione corrente della testina; poiché questo tempo cresce al crescere della distanza dei cilindri dalla testina, l'algoritmo sceglie le richieste relative ai cilindri più vicini alla posizione della testina.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

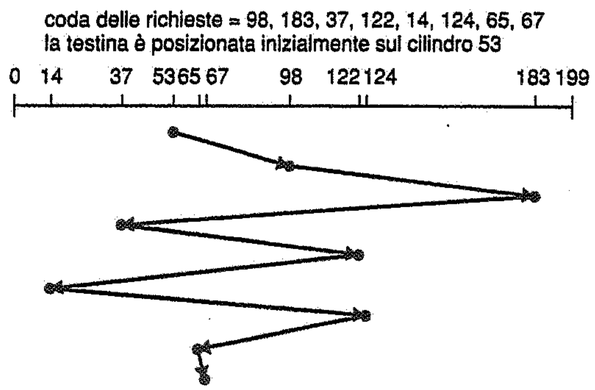


Figura 14.1 Scheduling FCFS.

Se si considera nuovamente la sequenza di richieste presa ad esempio sopra, il cilindro più vicino alla posizione iniziale della testina (cioè 53) è il 65, la successiva richiesta più vicina è quella relativa al cilindro 67; da questo cilindro, la richiesta relativa al cilindro 37 è più vicina di quella relativa al cilindro 98, ed è quindi servita per terza. Continuando allo stesso modo, sarà soddisfatta la richiesta relativa al cilindro 14, poi 98, 122, 124 e infine 183 (Figura 14.2). Questo metodo di scheduling implica una distanza totale percorsa di soli 236 cilindri, poco più di un terzo della distanza ottenuta con lo scheduling FCFS di questa coda di richieste: esso porta quindi sostanziali miglioramenti d'efficienza.

Lo scheduling SSTF è essenzialmente una forma di scheduling per brevità (*shortest job first* — SJF), e al pari di questo, può condurre a situazioni d'attesa indefinita (*starvation*) di alcune richieste. Si ricordi infatti che nuove richieste possono giungere in qualunque momento: si supponga di avere due richieste in coda, una per il cilindro 14 e l'altra per il 186, e che mentre si sta servendo la richiesta relativa al cilindro 14, arrivi una nuova richiesta per un cilindro vicino al 14. Questa sarà la prossima richiesta soddisfatta, mentre la richiesta per il cilindro 186 dovrà attendere. La stessa situazione potrebbe ripetersi, perché un'altra richiesta relativa a una posizione in prossimità del cilindro 14 potrà giungere mentre si sta servendo la precedente richiesta: in teoria, un flusso continuo di richieste riferite a posizioni le une vicine alle altre potrebbe causare l'attesa indefinita della richiesta relativa al cilindro 186. Quest'ipotesi diviene sempre più probabile man mano che la coda di richieste si allunga.

Sebbene l'algoritmo SSTF costituisca un miglioramento notevole rispetto al FCFS, esso non è ottimale: si può fare di meglio con uno spostamento dal cilindro 53 al 37, anche se questa non è la minima distanza possibile, e poi al 14, prima di invertire la marcia per servire i 65, 67, 98, 122, 124 e 183. L'adozione di questa strategia riduce la distanza totale percorsa a 208 cilindri.

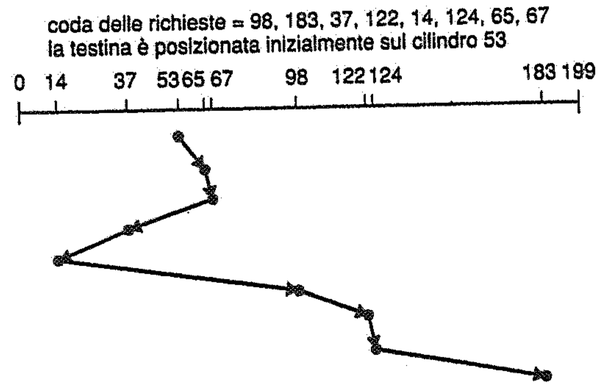


Figura 14.2 Scheduling SSTF.

14.2.3 Scheduling per scansione — SCAN

Secondo l'**algoritmo** SCAN il braccio dell'unità a disco parte da un estremo del disco e si sposta nella sola direzione possibile, servendo le richieste mentre attraversa i cilindri, fino a che non giunge all'altro estremo del disco: a questo punto, il braccio inverte la marcia, e la procedura continua. Le testine attraversano continuamente il disco nelle due direzioni.

Si consideri ancora l'esempio precedente, prima di poter applicare lo scheduling SCAN alle richieste per i cilindri 98, 183, 37, 122, 14, 124, 65, e 67, oltre la posizione corrente (53), occorre conoscere la direzione del movimento delle testine. Se lo spostamento è nella direzione del cilindro 0, l'unità a disco servirà prima la richiesta 37 e poi la 14; una volta giunto al cilindro 0, il braccio invertirà il movimento verso l'altro estremo del disco, servendo le richieste 65, 67, 98, 122, 124 e 183 (Figura 14.3). Se arriva una nuova richiesta riferita a uno dei cilindri posti davanti alla testina (relativamente alla sua direzione di moto), essa sarà quasi immediatamente soddisfatta; ma se la richiesta è riferita a uno dei cilindri appena sorpassati, essa dovrà attendere fino a che la testina non giunga alla fine del disco, inverta la direzione del moto, e torni indietro.
L'algoritmo SCAN è a volte chiamato **algoritmo dell'ascensore**, perché il braccio dell'unità a disco si comporta proprio come un ascensore che serva prima tutte le richieste in salita e poi tutte quelle in discesa.

Assumendo una distribuzione uniforme delle richieste per i cilindri da visitare, si consideri la densità di richieste quando il braccio giunge a un estremo e inverte la direzione del moto: in quel momento, relativamente poche richieste sono riferite a cilindri posti vicino alla testina e davanti a essa, perché i cilindri in questione sono stati recentemente visitati. La massima densità di richieste si riferisce all'altro estremo del disco, e queste richieste sono anche quelle che hanno atteso più a lungo: sembra ragionevole spostarsi lì come prossima mossa. Questa è l'idea dell'algoritmo seguente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

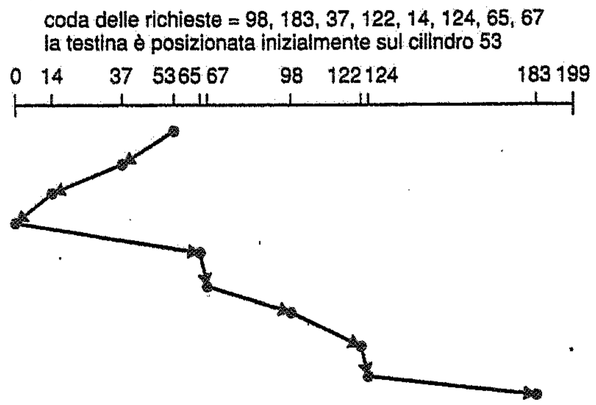


Figura 14.3 Scheduling SCAN.

14.2.4 Scheduling per scansione circolare — C-SCAN

L'algoritmo SCAN **circolare** (*circular SCAN* — C-SCAN) è una variante dello scheduling SCAN concepita per garantire un tempo d'attesa meno variabile. Anche l'algoritmo C-SCAN, come lo SCAN, sposta la testina da un estremo all'altro del disco, servendo le richieste lungo il percorso; tuttavia, quando la testina giunge all'altro estremo del disco, ritorna immediatamente all'inizio del disco stesso, senza servire richieste durante il viaggio di ritorno (Figura 14.4). L'algoritmo di scheduling C-SCAN, essenzialmente, tratta il disco come una lista circolare, cioè come se il primo e l'ultimo cilindro fossero adiacenti.

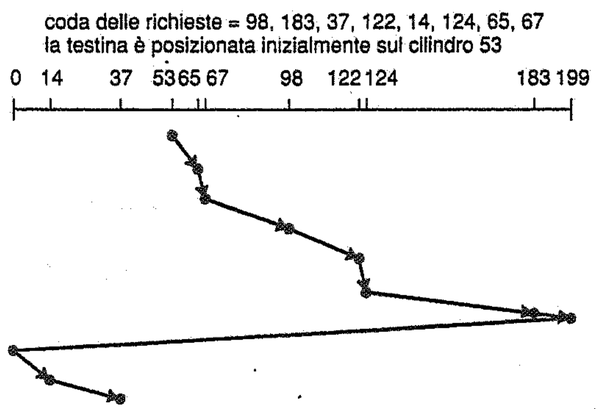


Figura 14.4 Scheduling C-SCAN.

14.2.5 Scheduling LOOK

Secondo la descrizione appena fatta, sia l'algoritmo SCAN sia il C-SCAN spostano il braccio dell'unità attraverso tutta l'ampiezza del disco; all'atto pratico, nessuno dei due algoritmi è codificato in questo modo: più comunemente, il braccio si sposta solo finché ci sono altre richieste da servire in quella direzione, dopo di che cambia immediatamente direzione, senza giungere all'estremo del disco. Queste versioni dello SCAN e del C-SCAN sono dette LOOK e C-LOOK, perché 'guardano' (in inglese, *look*) se ci sono altre richieste da soddisfare lungo la direzione attuale prima di continuare a spostare la testina in quella direzione (Figura 14.5).

14.2.6 Scelta di un algoritmo di scheduling

Giacché esistono tanti diversi algoritmi di scheduling, ci si può chiedere come si faccia a scegliere il migliore. Un algoritmo molto comune e naturalmente attraente è l'SSTF poiché aumenta le prestazioni rispetto all'FCFS; lo SCAN e il C-SCAN dànno migliori prestazioni in sistemi che sfruttano molto le unità a disco, perché conducono con minor probabilità a situazioni d'attesa indefinita. Per una data ma arbitraria lista di richieste si può definire un ordine ottimo di servizio, ma la computazione richiesta può non essere giustificata dal miglioramento in prestazioni rispetto agli algoritmi SSTF o SCAN.

Per qualunque algoritmo di scheduling, le prestazioni dipendono comunque in larga misura dal numero e dal tipo di richieste. Ad esempio, si supponga che la coda sia costituita in genere di una sola richiesta inevasa: tutti gli algoritmi dànno allora luogo allo stesso comportamento, perché hanno una sola scelta possibile relativamente al prossimo spostamento della testina. In questo caso, tutti gli algoritmi si comportano come l'FCFS.

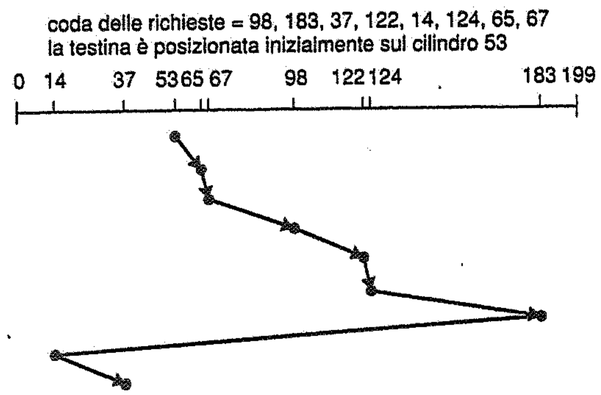


Figura 14.5 Scheduling C-LOOK.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le richieste di I/O per l'unità a disco possono essere notevolmente influenzate dal metodo adottato per l'assegnazione dei file. Un programma che legga un file assegnato in modo contiguo genererà molte richieste raggruppate, con un conseguente limitato spostamento della testina. Un file con assegnazione concatenata o indicizzata, d'altro canto, potrebbe includere blocchi sparsi per tutto il disco, e richiedere quindi un maggiore movimento della testina.

Anche la posizione delle directory e dei blocchi indice è importante: poiché ogni file deve essere aperto per essere usato, e visto che l'apertura di un file richiede una ricerca attraverso la struttura delle directory, vi saranno frequenti accessi alle directory. Si supponga che un elemento di directory risieda nel primo cilindro e che i dati del file relativo si trovino nell'ultimo cilindro; la testina dovrà allora percorrere l'intera ampiezza del disco nel caso di apertura del file in questione. Se l'elemento della directory che rappresenta logicamente il file fosse nel cilindro di mezzo, la testina dovrebbe spostarsi al più della metà dell'ampiezza. Anche l'uso della memoria centrale come cache delle directory e dei blocchi indice può contribuire a ridurre i movimenti del braccio dell'unità a disco, in particolare quando si tratta di operazioni di lettura.

A causa di queste complicazioni, l'algoritmo di scheduling del disco dovrebbe costituire un modulo a sé stante del sistema operativo, così da poter essere sostituito da un altro algoritmo qualora ciò fosse necessario; come algoritmo di partenza è ragionevole la scelta dell'SSTF o del LOOK.

Gli algoritmi di scheduling descritti tengono conto solamente del tempo di ricerca, mentre nelle moderne unità a disco la latenza di rotazione può essere lunga quasi quanto il tempo medio di ricerca. Tuttavia è difficile per il sistema operativo adottare una strategia di scheduling che porti a miglioramenti dei tempi di latenza di rotazione, perché le moderne unità a disco non rivelano la posizione fisica dei blocchi logici. I produttori di unità a disco hanno collaborato alla limitazione di questo problema incorporando algoritmi di scheduling all'interno dei controllori contenuti nelle unità a disco: se il sistema operativo invia un gruppo di richieste al controllore, esso può organizzarle in una coda e poi applicare algoritmi di scheduling che riducano sia i tempi di ricerca sia la latenza di rotazione. Se l'unico elemento di cui tener conto fossero le prestazioni dell'I/O, il sistema operativo scaricherebbe volentieri la responsabilità dello scheduling per il disco sull'apparato elettronico del dispositivo. In pratica, però, il sistema operativo può dover considerare altri vincoli relativi all'ordine in cui si devono servire le richieste: ad esempio, la richiesta di una pagina di memoria virtuale potrebbe avere maggiore priorità rispetto all'I/O delle applicazioni, e le scritture divengono più urgenti delle letture quando la cache sta per esaurire le pagine disponibili. Inoltre, può essere auspicabile mantenere l'ordine naturale delle richieste di scrittura al fine di rendere il file system robusto rispetto ai crolli del sistema: si consideri cosa accadrebbe se il sistema operativo assegnasse un blocco di disco a un file, e un'applicazione scrivesse dati in quel blocco; se il sistema crollasse a questo punto, il sistema operativo potrebbe non essere riuscito a copiare l'*inode* modificato e la nuova lista dei blocchi liberi sul disco. Per conciliare queste esigenze, il sistema operativo può scegliere di accollarsi la responsabilità dello scheduling del disco e, per alcuni tipi di I/O, fornire le richieste al controllore una alla volta.

14.3 Gestione dell'unità a disco

Il sistema operativo è anche responsabile di molti altri aspetti della gestione delle unità a disco. In questo paragrafo si discutono l'inizializzazione del disco, l'avviamento del sistema basato sull'unità a disco, e la gestione dei blocchi difettosi.

14.3.1 Formattazione del disco

Un disco magnetico nuovo è *tabula rasa*: un insieme di uno o più piatti sovrapposti ricoperti di materiale magnetico; prima che possa memorizzare dati, deve essere diviso in settori che possano essere letti o scritti dal controllore. Questo processo si chiama formattazione di basso livello, o **formattazione fisica**. La **formattazione di basso livello** riempie il disco con una speciale struttura di dati per ogni settore, tipicamente consistente di un'intestazione, un'area per i dati (di solito di 512 byte), e una coda. L'intestazione e la coda contengono informazioni usate dal controllore del disco, ad esempio il numero del settore e un **codice per la correzione degli errori** (*error-correcting code* — ECC). Quando il controllore scrive dati in un settore nel corso di un'ordinaria operazione di I/O, aggiorna il valore dell'ECC secondo il contenuto dell'area di dati del settore. Quando il controllore legge dati da quel settore, calcola anche l'ECC e lo confronta con il suo valore memorizzato: se risulta una discrepanza, l'area dei dati del settore non è integra, e il settore del disco potrebbe essere difettoso (si veda il Paragrafo 14.3.3). L'ECC è un codice per la *correzione* degli errori: se solo alcuni bit di dati sono stati alterati, esso contiene sufficienti informazioni affinché il controllore possa identificare i bit in questione e ricalcolare il loro corretto valore. Il controllore esegue automaticamente l'elaborazione descritta ogni volta che accede a un settore del disco.

La formattazione fisica dei dischi è eseguita nella maggior parte dei casi dal costruttore come parte del processo produttivo; ciò permette al costruttore di provare il disco, e di instaurare la corrispondenza fra blocchi logici e settori correttamente funzionanti del disco. In molte unità a disco, quando si richiede al controllore di formattare fisicamente il disco, si può anche specificare il numero di byte delle aree di dati comprese fra l'intestazione e la coda di un settore. La scelta è di solito ristretta a poche opzioni, come 256, 512 o 1024 byte. La formattazione in settori più grandi implica la presenza di meno settori su ogni traccia, ma anche meno intestazioni e code, e quindi maggior spazio per i dati veri e propri. Alcuni sistemi operativi gestiscono solo settori di 512 byte.

Per usare un disco come contenitore d'informazioni, il sistema operativo deve ancora registrare le proprie strutture di dati nel disco, cosa che fa in due passi. Il primo consiste nel suddividere il disco in uno o più gruppi di cilindri, detti **partizioni**. Il sistema operativo può trattare ogni gruppo come se fosse un'unità a disco a sé stante: ad esempio, una partizione può contenere una copia del codice eseguibile del sistema operativo, mentre un'altra contiene i file degli utenti. Il passo successivo alla suddivisione in partizioni è la **formattazione logica**, cioè la creazione di un file system: il sistema operativo registra nel disco le strutture di dati iniziali relative al file system. Le strutture di dati in questione possono includere descrizioni dello spazio libero e dello spazio assegnato (FAT o *inode*) e una directory iniziale vuota.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Alcuni sistemi operativi dànno l'opportunità a certi programmi speciali di impiegare una partizione del disco come un grande vettore sequenziale di blocchi logici, non contenente alcuna struttura di dati relativa al file system, detto disco di basso livello (*raw disk*); l'I/O relativo si chiama I/O di basso livello (*raw I/O*). Alcuni sistemi per la gestione di basi di dati, ad esempio, preferiscono questo tipo di I/O perché permette di controllare l'esatta posizione nel disco d'ogni informazione trattata. Esso scavalca tutti i servizi del file system: gestione delle *buffer cache*, prelievo anticipato, assegnazione dello spazio, nomi dei file, directory, e così via. È possibile rendere certe applicazioni più efficienti includendo in esse servizi di memorizzazione secondaria specializzati che usino una partizione di basso livello, ma la maggior parte delle applicazioni è in realtà migliore quando usufruisce degli ordinari servizi del file system.

14.3.2 Blocco d'avviamento

Affinché un calcolatore possa entrare in funzione, ad esempio quando viene acceso o riavviato, è necessario che esegua un programma iniziale; di solito, questo programma d'avviamento iniziale è piuttosto semplice. Esso inizializza il sistema in tutti i suoi aspetti, dai registri della CPU ai controllori dei dispositivi e al contenuto della memoria centrale, quindi avvia il sistema operativo. Per far ciò, il programma d'avviamento trova il nucleo del sistema operativo nei dischi, lo carica nella memoria, e salta a un indirizzo iniziale per avviare l'esecuzione del sistema operativo.

Per la maggior parte dei calcolatori, il programma d'avviamento è memorizzato in una **memoria a sola lettura** (*read-only memory* — ROM), il che è conveniente, perché la ROM non richiede inizializzazione, e ha un indirizzo iniziale fisso dal quale la CPU può cominciare l'esecuzione ogniqualvolta si accende o si riavvia la macchina. Inoltre, visto che la ROM è a sola lettura, non può essere contaminata da un virus informatico. Il problema, però, è che cambiare il programma d'avviamento richiede in questo caso la sostituzione dei circuiti integrati ROM. A causa di questo inconveniente, molti sistemi memorizzano nella ROM un piccolo caricatore d'avviamento (*bootstrap loader*) il cui solo compito è quello di caricare da un disco il programma d'avviamento completo. Quest'ultimo si può facilmente modificare: se ne scrive semplicemente una nuova versione nel disco. Il programma d'avviamento completo è registrato in una partizione del disco denominata blocchi d'avviamento, posta in una locazione fissata del disco; un disco contenente una tale partizione si chiama **disco d'avviamento** o **disco di sistema**.

Il codice contenuto nella ROM d'avviamento istruisce innanzi tutto il controllore dell'unità a disco affinché trasferisca il contenuto dei blocchi d'avviamento nella memoria (si noti che a questo fine non si carica alcun driver di dispositivo), quindi comincia a eseguire il codice. Il programma d'avviamento completo è più complesso del suo caricatore, ed è capace di trasferire nella memoria l'intero sistema operativo inizialmente residente in un disco in una locazione non definitivamente fissata, e di avviare il sistema operativo stesso. Il programma d'avviamento potrà comunque essere breve: quello dell'MS-DOS, ad esempio, non supera le dimensioni di un blocco di 512 byte (Figura 14.6).

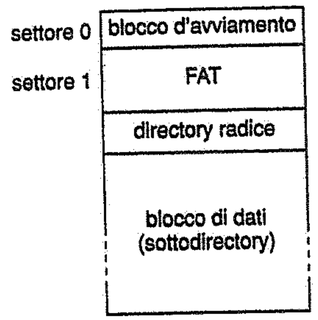


Figura 14.6 Configurazione del disco nell'MS-DOS.

14.3.3 Blocchi difettosi

Le unità a disco sono strutturalmente portate ai malfunzionamenti perché sono costituite da parti mobili a bassa tolleranza (si ricordi che una testina è sospesa appena sopra la superficie del disco). A volte si può verificare un guasto irreparabile, e l'unità a disco deve essere sostituita: le informazioni contenute nel disco dovranno essere recuperate da una copia di riserva mantenuta separatamente e trasferite nella nuova unità a disco. Più di frequente, uno o più settori divengono malfunzionanti; in effetti, la maggior parte dei dischi messi in commercio contiene già **blocchi difettosi**. Essi sono trattati in diversi modi secondo il controllore e l'unità a disco presenti nel sistema.

Nel caso di dischi semplici come quelli gestiti da un controllore IDE, i blocchi difettosi sono gestiti 'manualmente'. Ad esempio, il comando `format` dell'MS-DOS esegue una formattazione logica, e come parte del processo esamina il disco per rilevare la presenza di blocchi difettosi: se ne trova qualcuno, scrive un valore speciale nel corrispondente elemento della FAT al fine di segnalare alle procedure di assegnazione di non usare il blocco in questione. Se qualche blocco diviene malfunzionante nel corso dell'ordinario uso del sistema, un programma speciale (ad esempio `chkdsk`) deve essere eseguito a cura dell'utente per individuare i blocchi difettosi e isolarli come appena descritto. Di solito, i dati residenti nei blocchi difettosi vanno perduti.

Unità a disco più complesse come i dischi SCSI in uso nei PC di alto livello e nella maggior parte delle stazioni di lavoro e server, hanno strategie di recupero dei blocchi difettosi più raffinate. Il controllore mantiene una lista dei blocchi malfunzionanti dell'unità a disco che è inizializzata durante la formattazione fisica eseguita dal produttore, ed è aggiornata per tutto il periodo in cui l'unità a disco è operativa. La formattazione fisica mette anche da parte dei settori di riserva non visibili al sistema operativo: si può istruire il controllore affinché sostituisca da un punto di vista logico un settore difettoso con uno dei settori di riserva inutilizzati. Questa strategia è nota come **accantonamento di settori** (*sector sparing* o *sector forwarding*).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un tipico esempio di attuazione di questa strategia è il seguente:

- il sistema operativo tenta di leggere il blocco logico 87;
- il controllore calcola l'ECC e scopre che il settore è difettoso; quindi segnala questo malfunzionamento al sistema operativo;
- la volta successiva che il sistema viene riavviato, si esegue un comando speciale al fine di comunicare al controllore SCSI la necessità di sostituire il settore difettoso con uno di riserva;
- dopo di ciò, ogni volta che il sistema tenta di leggere il contenuto del blocco 87, il controllore traduce la richiesta nell'indirizzo del settore di rimpiazzo.

Un tale reindirizzamento da parte del controllore potrebbe inficiare ogni ottimizzazione fornita dall'algoritmo di scheduling del disco del sistema operativo. Per questa ragione la maggior parte dei dischi si formatta in modo tale da mantenere qualche settore di riserva in ogni cilindro, e anche un intero cilindro di riserva. Quando un numero di blocco logico è assegnato a dei settori di riserva, il controllore usa settori di riserva presenti nello stesso cilindro ogniqualvolta ciò sia possibile.

Un'alternativa all'accantonamento dei settori è data da quei controllori capaci di sostituire i settori difettosi tramite la tecnica della **traslazione dei settori** (*sector slipping*). Si supponga ad esempio che il blocco logico 17 divenga malfunzionante, e che il primo settore di riserva disponibile sia quello successivo al settore 202. La traslazione dei settori sposterebbe in avanti di un posto tutti i settori dal 17 al 202: quindi, il settore 202 sarebbe copiato sul settore di riserva, il settore 201 sul 202, il 200 sul 201, e così via, fino a che il settore 18 non sia stato copiato sul 19. Questa traslazione dei settori libera lo spazio del settore 18, e il settore 17 può essere fatto corrispondere a quest'ultimo.

La sostituzione di un blocco difettoso non è in genere un processo totalmente automatico, perché i dati contenuti nel blocco in questione vanno in generale perduti. Un file che usava quel blocco deve quindi essere riparato (ad esempio, ricopiandolo da un nastro contenente le copie di riserva), e ciò richiede un intervento manuale.

14.4 Gestione dell'area d'avvicendamento

La **gestione dell'area d'avvicendamento** è un altro compito di basso livello del sistema operativo. La memoria virtuale usa lo spazio dei dischi come estensione della memoria centrale: poiché l'accesso alle unità a disco è molto più lento dell'accesso alla memoria centrale, l'uso di un'area d'avvicendamento riduce notevolmente le prestazioni del sistema. L'obiettivo principale nella progettazione e realizzazione di un'area d'avvicendamento è di fornire la migliore produttività per il sistema della memoria virtuale. In questo paragrafo sono discussi l'uso, la collocazione nei dischi e la gestione dell'area d'avvicendamento.

14.4.1 Uso dell'area d'avvicendamento

L'area d'avvicendamento è usata in modi diversi da sistemi operativi diversi, in funzione degli algoritmi di gestione della memoria applicati. I sistemi che adottano l'avvicendamento dei processi nella memoria, ad esempio, possono usare l'area d'avvicendamento per mantenere l'intera immagine del processo, inclusi i segmenti dei dati e del codice; i sistemi a paginazione, invece, possono semplicemente memorizzarvi pagine non contenute nella memoria centrale. Lo spazio richiesto dall'area d'avvicendamento per un sistema può quindi variare secondo la quantità di memoria fisica, la quantità di memoria virtuale che esso deve sostenere, e il modo in cui quest'ultima è usata: tale spazio va da pochi megabyte a gigabyte di spazio su disco.

Alcuni sistemi operativi, fra i quali UNIX, permettono l'uso di aree d'avvicendamento multiple, poste di solito in unità a disco distinte per distribuire su più dispositivi il carico della paginazione e dell'avvicendamento dei processi gravante sul sistema per l'I/O.

Si noti che una stima per eccesso delle dimensioni dell'area d'avvicendamento è più prudente di una per difetto, perché un sistema che esaurisca l'area d'avvicendamento potrebbe essere costretto a terminare forzatamente i processi o ad arrestarsi completamente: una stima per eccesso spreca spazio dei dischi che si potrebbe usare per i file, ma non provoca altri danni.

14.4.2 Collocazione dell'area d'avvicendamento

Ci sono due possibili collocazioni per un'area d'avvicendamento: all'interno del normale file system, o in una partizione del disco a sé stante. Se l'area d'avvicendamento è semplicemente un grande file all'interno del file system, si possono usare le ordinarie funzioni del file system per crearla, assegnargli un nome, e assegnare spazio per essa. Questo criterio sebbene sia semplice da realizzare è inefficiente: l'attraversamento della struttura delle directory e l'uso delle strutture di dati per l'assegnazione dello spazio nei dischi richiede tempo, oltre che, almeno potenzialmente, accessi ai dischi aggiuntivi. La frammentazione esterna può aumentare molto i tempi d'avvicendamento causando ricerche multiple durante la scrittura o la lettura dell'immagine di un processo. Le prestazioni si possono migliorare impiegando la memoria fisica come cache per le informazioni relative alla posizione dei blocchi, e anche usando strumenti speciali per l'assegnazione in blocchi fisicamente contigui del file d'avvicendamento, ma il costo dovuto all'attraversamento del file system e delle sue strutture di dati permane.

In alternativa, l'area d'avvicendamento si può creare in una partizione del disco distinta: in essa non è presente alcuna struttura relativa al file system e alle directory, ma si usa uno speciale gestore dell'area d'avvicendamento per assegnare e rimuovere i blocchi. Esso adotta algoritmi ottimizzati rispetto alla velocità, e non rispetto allo spazio impiegato: la frammentazione interna può aumentare, ma questo prezzo da pagare è ragionevole perché i dati nell'area d'avvicendamento hanno una vita media molto più breve dei file ordinari, e gli accessi all'area d'avvicendamento sono in genere molto più frequenti.

http://www.adultpdf.com
Created by TIFF Tools Trial version. To remove this mark, please register this software.

Questo metodo assegna una dimensione fissa all'area d'avvicendamento al momento della creazione delle partizioni del disco, e l'aumento delle dimensioni dell'area d'avvicendamento deve quindi passare attraverso il ripartizionamento del disco (che implica lo spostamento o l'eliminazione e la sostituzione con copie di riserva delle altre partizioni del disco), o attraverso la creazione di un'altra area d'avvicendamento in qualche altra unità a disco del sistema.

Alcuni sistemi operativi non adottano una strategia rigida e possono costruire aree d'avvicendamento sia su partizioni specifiche sia all'interno del file system: il Solaris 2 ne è un esempio. I metodi di gestione e la loro concreta realizzazione sono diversi nei due casi, e la scelta fra le due soluzioni è lasciata all'amministratore del sistema: sul piatto della bilancia pesano da un lato la convenienza dell'assegnazione e della gestione dell'area d'avvicendamento all'interno del file system, e dall'altro le migliori prestazioni ottenibili grazie all'uso di una partizione specifica.

14.4.3 Gestione dell'area d'avvicendamento: un esempio

In questo paragrafo sono riassunti l'evoluzione dei meccanismi di avvicendamento dei processi nella memoria e della paginazione nello UNIX al fine di illustrare i metodi impiegati nella gestione dell'area d'avvicendamento. Il sistema operativo UNIX, come esaurientemente esposto nell'Appendice A, realizzava inizialmente l'avvicendamento dei processi spostando integralmente i processi fra aree contigue del disco e la memoria; le sue evoluzioni successive hanno portato a una combinazione di avvicendamento dei processi e paginazione man mano che divenivano disponibili i dispositivi per la paginazione.

Nella versione 4.3BSD si assegna l'area d'avvicendamento a un processo quando esso è avviato: si riserva spazio sufficiente per le **pagine di testo** o il **segmento di testo**, dove è contenuto il programma, e per il **segmento dei dati**. Questa forma di assegnazione preventiva in genere impedisce che il processo esaurisca il suo spazio d'avvicendamento durante l'esecuzione. Quando comincia l'esecuzione di un processo, si carica il suo testo nella memoria prelevandolo dal file system; se è necessario, si trasferiscono le sue pagine nell'area d'avvicendamento, per essere da qui rilette; in questo modo, si consulta il file system una sola volta per ciascuna pagina di testo. Le pagine del segmento dei dati si trasferiscono nella memoria dal file system oppure, nel caso esse non siano già state inizializzate, si creano *ex novo* e si scrivono nell'area d'avvicendamento in modo da poter essere rilette e caricate nella memoria quando se ne presenti la necessità. Una tecnica di ottimizzazione adottata è che processi con segmenti di testo identici condividono le relative pagine sia nella memoria sia nell'area d'avvicendamento.

Due **mappe d'avvicendamento** per ogni processo servono al nucleo per tenere traccia dell'area d'avvicendamento correntemente impiegata. Il segmento di testo ha una dimensione costante, perciò la sua area d'avvicendamento è assegnata per blocchi di 512 KB, eccezion fatta per l'ultimo blocco: esso contiene l'ultima porzione delle pagine e la sua dimensione è arrotondata a 1 KB (Figura 14.7).

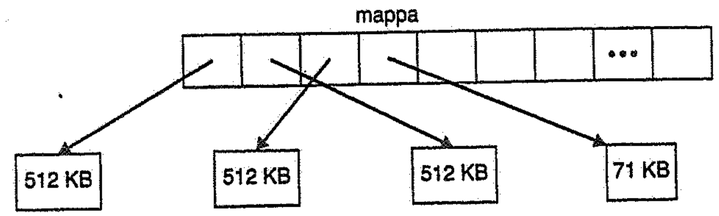


Figura 14.7 Mappa d'avvicendamento per il segmento di testo nello UNIX 4.3 BSD.

La mappa d'avvicendamento del segmento dei dati è più complicata, perché il segmento dei dati può crescere nel tempo. La dimensione della mappa è fissa, ma contiene indirizzi relativi a blocchi dell'area d'avvicendamento di dimensione variabile. Per ogni indice i il blocco puntato dall'elemento i-esimo della mappa è di $2^i \times 16$ KB, fino a un massimo di 2 MB. Nella Figura 14.8 è mostrata questa struttura di dati. (La minima e massima dimensione di un blocco sono variabili e si possono impostare al riavvio del sistema.) Se il segmento dei dati di un processo cresce oltre l'ultimo blocco assegnato alla sua area, il sistema operativo assegna un altro blocco grande il doppio del precedente. Questo schema fa sì che piccoli processi usino solamente blocchi piccoli, e minimizza la frammentazione. I blocchi dei processi più grandi sono facilmente individuati, e la mappa d'avvicendamento resta di piccole dimensioni.

I progettisti del sistema Solaris 1 (SunOS 4) hanno apportato alcuni cambiamenti agli ordinari metodi dello UNIX per migliorare l'efficienza e sfruttare meglio le nuove tecnologie. Quando si esegue un processo le pagine del suo segmento di testo sono trasferite nella memoria dal file system, lette tramite accessi alla memoria ed eliminate non appena debbano essere ritrasferite in un disco: è più efficiente rileggere una pagina dal file system che scriverla nell'area d'avvicendamento e rileggerla da lì.

Il sistema operativo Solaris 2 incorpora altri cambiamenti: il più importante consiste nel fatto che esso assegna spazio nell'area d'avvicendamento solo quando una pagina non può più risiedere nella memoria fisica, e non al momento della creazione della pagina di memoria virtuale. Questa variante migliora le prestazioni in virtù del fatto che un calcolatore moderno ha una maggiore disponibilità di memoria fisica e una minore necessità di paginare rispetto ai sistemi più antiquati.

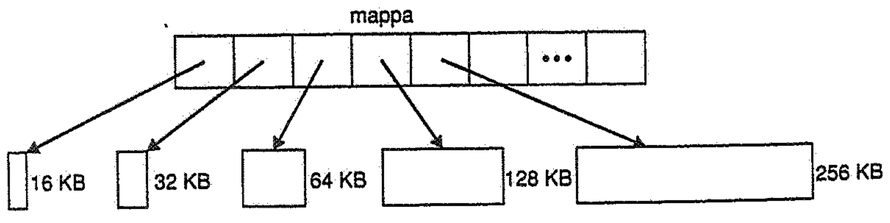


Figura 14.8 Mappa d'avvicendamento per il segmento dei dati nello UNIX 4.3 BSD.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

14.5 Strutture RAID

L'evoluzione tecnologica ha reso le unità a disco progressivamente più piccole e meno costose, tanto che oggi è possibile, senza eccessivi sforzi economici, equipaggiare un sistema di calcolo con molti dischi.

La presenza di più dischi, qualora si possano usare in parallelo, rende possibile l'aumento della frequenza alla quale i dati si possono leggere o scrivere. Inoltre, una configurazione di questo tipo permette di migliorare l'affidabilità della memoria secondaria, poiché diventa possibile memorizzare le informazioni in più dischi in modo ridondante. In questo caso, un guasto a uno dei dischi non comporta la perdita di dati. Ci sono varie tecniche per l'organizzazione dei dischi, note col nome comune di **batterie ridondanti di dischi** (*redundant array of independent* [*inexpensive*] *disks* — RAID), che hanno lo scopo di affrontare i problemi di prestazioni e affidabilità.

Nel passato, strutture RAID composte da piccoli dischi economici erano viste come un'alternativa economicamente vantaggiosa rispetto a costosi dischi di grande capacità; oggi, le strutture RAID s'impiegano per la loro maggiore affidabilità e velocità di trasferimento dei dati, piuttosto che per ragioni economiche. Quindi, la *I* in RAID attualmente andrebbe letta *independent* anziché *inexpensive* com'era interpretata originariamente.

14.5.1 Miglioramento dell'affidabilità tramite la ridondanza

Consideriamo in primo luogo l'affidabilità. La possibilità che uno dei dischi in un insieme di n dischi si guasti è molto più alta della possibilità che uno specifico disco isolato presenti un guasto. Si supponga che il **tempo medio di guasto** di un singolo disco sia 100.000 ore. In questo caso, il tempo medio di guasto per un qualsiasi disco in una batteria di 100 dischi sarebbe 100.000/100 = 1000 ore, o 41,66 giorni; cioè non molto tempo. Se si memorizzasse una sola copia dei dati, allora ogni guasto di un disco comporterebbe la perdita di una notevole quantità di dati; una frequenza di perdita di dati così alta sarebbe inaccettabile.

La soluzione al problema dell'affidabilità sta nell'introdurre una certa **ridondanza**, cioè nel memorizzare informazioni che non sono normalmente necessarie, ma che si possono usare nel caso di un guasto a un disco per ricostruire le informazioni perse.

Il metodo più semplice (ma anche il più costoso) di introduzione di ridondanza è quello della **copiatura speculare** (*mirroring* o *shadowing*); ogni disco logico consiste di due dischi fisici e ogni scrittura si effettua in entrambi i dischi. Se uno dei dischi si guasta, i dati si possono leggere dall'altro. I dati si perdono solo se il secondo disco si guasta prima della sostituzione del disco già guasto.

Il tempo medio di guasto di un disco con copiatura speculare, dove per *guasto* s'intende ora la perdita di dati, dipende da due fattori: il tempo medio di guasto di un singolo disco e il **tempo medio di riparazione**, cioè il tempo richiesto (in media) per sostituire un disco guasto e ripristinarvi i dati. Supponendo che i possibili guasti dei due dischi siano **indipendenti**, vale a dire che il guasto di un disco non sia mai legato a quello dell'altro, se il tempo medio di guasto di un singolo disco è 100.000 ore e il tempo medio di riparazione è di 10 ore, allora il tempo medio di perdita di dati di un sistema con copiatura speculare dei dischi è $100.000^2/(2 \times 10) = 500 \times 10^6$ ore, che corrispondono a 57.000 anni.

Occorre però notare che l'ipotesi di indipendenza tra i guasti dei dischi non è in realtà valida, poiché improvvisi cali di tensione e disastri naturali, quali terremoti, incendi e alluvioni, danneggerebbero con tutta probabilità entrambi i dischi. Inoltre, difetti di fabbricazione in una partita di dischi possono causare guasti simili e correlati. Con l'invecchiamento del disco, la probabilità di un guasto aumenta, accrescendo la probabilità che un secondo disco si guasti mentre il primo è in riparazione. Tuttavia, nonostante tutte queste considerazioni, i sistemi con copiatura speculare dei dischi offrono un'affidabilità assai più alta dei sistemi a disco singolo.

I casi di improvvisa mancanza di tensione elettrica costituiscono un problema particolarmente sentito, poiché avvengono con una frequenza molto più alta dei disastri naturali. Tuttavia, anche impiegando la copiatura speculare dei dischi, se si sta svolgendo un'operazione di scrittura nello stesso blocco in entrambi i dischi e si verifica una mancanza di tensione prima che sia completata la scrittura dell'intero blocco, i due blocchi possono ritrovarsi in uno stato incoerente. Una soluzione prevede la scrittura di una delle due copie e solo successivamente la scrittura della seconda, così che una delle due copie sia sempre coerente. Nel caso di un riavvio del sistema dopo una mancanza di tensione, si devono compiere azioni supplementari per ripristinare lo stato del sistema considerando possibili operazioni di scrittura ancora incomplete.

14.5.2 Miglioramento delle prestazioni tramite il parallelismo

L'accesso in parallelo a più dischi può portare vari vantaggi. Con la copiatura speculare dei dischi, la frequenza con la quale si possono gestire le richieste di lettura raddoppia, poiché ciascuna richiesta si può inviare indifferentemente a uno dei due dischi (sempre che entrambi i dischi siano funzionanti, condizione che è quasi sempre soddisfatta). La capacità di trasferimento di ciascuna lettura è la stessa di quella di un sistema a singolo disco, ma il numero di letture per unità di tempo raddoppia.

Attraverso l'uso di più dischi è possibile anche (o alternativamente) migliorare la capacità di trasferimento distribuendo i dati in sezioni su più dischi. Nella sua forma più semplice questa distribuzione, chiamata **sezionamento dei dati** (*data striping*), consiste nel distribuire i bit di ciascun byte su più dischi; in questo caso si parla di **sezionamento al livello dei bit**. Ad esempio, se il sistema impiega una batteria di otto dischi, si scriverà il bit i di ciascun byte nel disco i. La batteria di otto dischi si può trattare come un unico disco avente settori che hanno una dimensione otto volte superiore a quella normale e, soprattutto, che hanno una capacità di trasferimento otto volte superiore. In un'organizzazione di questo tipo, ogni disco è coinvolto in ogni accesso (lettura o scrittura che sia), così che il numero di accessi che si possono gestire nell'unità di tempo è circa lo stesso di quello per un sistema a disco singolo, ma ogni accesso permette di leggere una quantità di dati pari a otto volte quella che si può leggere con un singolo disco.

http://www.abuitpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il sezionamento al livello dei bit si può generalizzare a un numero di dischi multiplo di 8 o che divide 8. Ad esempio, se un sistema adopera una batteria di quattro dischi, i bit i e $i + 4$ di ciascun byte si memorizzano nel disco i. Inoltre, il sezionamento non si deve realizzare necessariamente al livello dei bit di un byte: nel **sezionamento al livello dei blocchi**, ad esempio, i blocchi di un file si distribuiscono su più dischi; con n dischi, il blocco i di un file si memorizza nel disco $(i \bmod n) + 1$. Sono possibili anche altri livelli di sezionamento, come quelli basati sui byte di un settore o sui settori di un blocco.

Riassumendo, gli obiettivi principali riguardo al parallelismo in un sistema di dischi sono due:

1. l'aumento, tramite il bilanciamento del carico, della produttività per accessi multipli a piccole porzioni di dati (cioè accessi a pagine);
2. la riduzione del tempo di risposta relativo ad accessi a grandi quantità di dati.

14.5.3 Livelli RAID

La tecnica di copiatura speculare offre un'alta affidabilità ma è costosa; la tecnica del sezionamento offre un'alta capacità di trasferimento dei dati, ma non migliora l'affidabilità. Sono stati proposti numerosi schemi per fornire ridondanza usando l'idea del sezionamento combinata con i bit di parità. Questi schemi realizzano diversi compromessi tra costi e prestazioni e sono stati classificati in livelli chiamati **livelli RAID**, che la Figura 14.9 mostra graficamente (nella figura, la lettera P indica i bit di correzione degli errori, la lettera C indica una seconda copia dei dati). In tutti i casi riportati nella figura, sono presenti quattro dischi di dati, mentre i dischi supplementari s'impiegano per memorizzare le informazioni ridondanti per il ripristino dai guasti.

- **RAID di livello 0.** Il livello 0 si riferisce a batterie di dischi con sezionamento al livello dei blocchi, ma senza ridondanza (come la copiatura speculare o i bit di parità). La Figura 14.9a mostra una batteria di dimensione 4.
- **RAID di livello 1.** Il livello 1 si riferisce alla tecnica della copiatura speculare. La Figura 14.9b mostra un'organizzazione basata sulla copiatura speculare che gestisce quattro dischi di dati.
- **RAID di livello 2.** Il livello 2 è anche noto come **organizzazione con codici per la correzione degli errori** (*error-correcting codes* — ECC). Da molto tempo i sistemi di memorizzazione impiegano tecniche di riconoscimento degli errori basate sui bit di parità. In un sistema di questo tipo, ogni byte di memoria ha associato un bit di parità che indica se i bit con valore 1 nel byte sono in numero pari (parità = 0) oppure dispari (parità = 1). Se si altera uno dei bit nel byte (un valore 1 diventa 0 o viceversa), la parità del byte cambia e quindi non concorda più con la parità memorizzata. Analogamente, se si altera il bit di parità, esso non concorda più con la parità calcolata. In questo modo s'identificano tutti gli errori di un singolo bit nel sistema di memoria. Gli schemi di correzione degli errori memorizzano due o

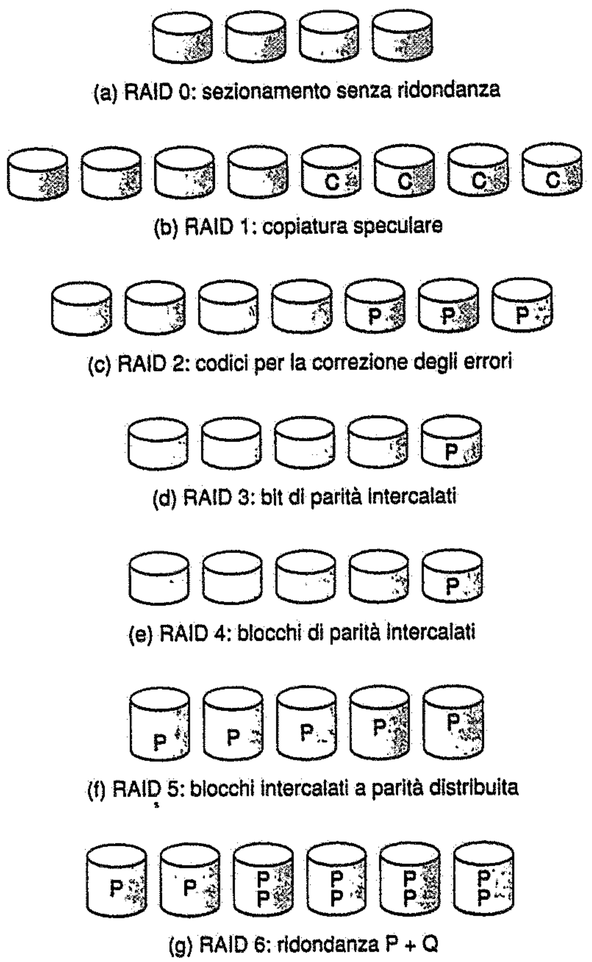


Figura 14.9 Livelli RAID.

più bit supplementari e possono ricostruire i dati nel caso di un singolo bit danneggiato. La stessa idea alla base dell'ECC si può usare immediatamente nelle batterie di dischi eseguendo il sezionamento dei byte presenti nei dischi. Ad esempio, il primo bit di ogni byte si potrebbe memorizzare nel disco 1, il secondo bit nel disco 2, e così via fino alla memorizzazione dell'ottavo bit nel disco 8 e alla memorizzazione dei bit di correzione degli errori in ulteriori dischi. Questo schema è rappresentato graficamente nella Figura 14.9, dove i dischi etichettati con la lettera P contengono i bit di correzione. Se uno dei dischi si guasta, i bit rimanenti del byte e i

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

bit di correzione a esso associati si possono leggere dagli altri dischi e usare per ricostruire i dati danneggiati. La Figura 14.9c mostra una batteria di dimensione 4; si noti che il RAID di livello 2 richiede tre soli dischi in più per quattro dischi di dati, a differenza del RAID di livello 1, che richiede quattro dischi in più.

- **RAID di livello 3.** Col livello 3, o **organizzazione con bit di parità intercalati**, si migliora l'organizzazione del livello 2 considerando che, a differenza dei sistemi di memoria centrale, i controllori dei dischi possono rilevare se un settore è stato letto correttamente, così che un unico bit di parità si può usare sia per individuare gli errori sia per correggerli. L'idea è la seguente: se uno dei settori è danneggiato, si conosce esattamente di quale settore si tratta e, per ogni bit nel settore, è possibile determinare se debba avere valore 1 o 0 calcolando la parità dei bit corrispondenti dai settori negli altri dischi. Se la parità dei rimanenti bit è uguale a quella memorizzata, il bit mancante è 0, altrimenti è 1. Il RAID di livello 3 è altrettanto valido del livello 2 ma richiede un solo disco supplementare, quindi il RAID di livello 2 non è utilizzato nella pratica. La Figura 14.9d illustra questo schema.

  Il livello 3 presenta due vantaggi rispetto al livello 1. Si usa un solo disco per la parità dei dati memorizzati in diversi dischi di dati, anziché un ulteriore disco per ciascun disco di dati come nel livello 1. Poiché le operazioni di lettura e scrittura di un byte sono distribuite su più dischi, con un sezionamento dei dati a n-vie, la capacità di trasferimento per la lettura o la scrittura di un singolo blocco è n volte più veloce di un'organizzazione di livello 1 che usa il sezionamento a n-vie. D'altra parte, il RAID di livello 3 consente un minor numero di operazioni di I/O al secondo, poiché ogni disco è coinvolto in ogni richiesta di I/O. Un altro problema di prestazioni riguardante il RAID di livello 3 (come per tutti i livelli RAID basati sui bit di parità) è il tempo richiesto dal calcolo e dalla scrittura della parità. Questo tempo aggiuntivo determina operazioni di scrittura significativamente più lente rispetto a batterie RAID senza parità. Per limitare questo calo di prestazioni, molte batterie RAID dispongono di un controllore capace di gestire il calcolo della parità. Questo sposta il carico dovuto al calcolo della parità dalla CPU alla batteria di dischi. La batteria ha anche una cache RAM **non volatile** (NVRAM) per memorizzare i blocchi mentre viene calcolata la parità e per memorizzare transitoriamente le scritture dal controllore ai dischi. Questa combinazione può rendere la tecnica RAID con parità altrettanto veloce di quella senza parità; infatti, una batteria RAID con cache e bit di parità può avere prestazioni migliori di un'organizzazione RAID senza cache e senza parità.

- **RAID di livello 4.** Nel livello 4, o **organizzazione con blocchi di parità intercalati**, s'impiega il sezionamento al livello dei blocchi, come nel RAID di livello 0 e inoltre si tiene un blocco di parità in un disco separato per i blocchi corrispondenti presenti in n dischi diversi da questo. Questo schema è illustrato nella Figura 14.9e. Se uno dei dischi si guasta, il blocco di parità si può usare insieme ai blocchi corrispondenti degli altri dischi per ripristinare i blocchi nel disco guasto.

La lettura di un blocco richiede l'accesso a un solo disco, permettendo la gestione di altre richieste da parte di altri dischi. Quindi, la capacità di trasferimento dei dati per ciascun accesso è minore, ma gli accessi per la lettura possono procedere in modo parallelo ottenendo una rapidità complessiva nell'I/O più alta. La capacità di trasferimento per la lettura di molti dati è alta, poiché si possono leggere in modo parallelo tutti i dischi e anche le operazioni di scrittura di grandi quantità di dati presentano un'alta capacità di trasferimento, poiché i dati e i bit di parità si possono scrivere in parallelo.

Le operazioni di scrittura di pochi dati indipendenti non si possono invece compiere in parallelo. La scrittura di un blocco richiede l'accesso al disco nel quale il blocco risiede e al disco in cui risiede l'informazione di parità, ciò poiché si deve aggiornare il blocco di parità. Inoltre, affinché si possa calcolare la nuova parità, si devono leggere sia il vecchio valore del blocco di parità sia il vecchio valore del blocco di dati. Si può concludere che una singola operazione di scrittura richiede quattro accessi ai dischi: due per leggere i due blocchi vecchi e due per scrivere i nuovi.

- **RAID di livello 5.** Il livello 5, o **organizzazione con blocchi intercalati a parità distribuita**, differisce dal livello 4 per il fatto che, invece di memorizzare i dati in n dischi e la parità in un disco separato, i dati e le informazioni di parità sono distribuite tra gli $n + 1$ dischi. Per ogni blocco, uno dei dischi memorizza la parità e gli altri i dati. Ad esempio, considerando una batteria di cinque dischi, la parità per il blocco m-esimo si memorizza nel disco (m mod 5) + 1, mentre i blocchi m-esimi degli altri quattro dischi contengono i dati effettivi per quel blocco. Questo schema è illustrato nella Figura 14.9f, dove i simboli P sono distribuiti su tutti i dischi. Un blocco di parità non può contenere informazioni di parità per blocchi che risiedono nello stesso disco, poiché un guasto al disco provocherebbe sia la perdita di dati sia la perdita dell'informazione di parità e quindi i dati non sarebbero ripristinabili. Con la distribuzione della parità sui diversi dischi, il RAID di livello 5 evita un uso intensivo del disco dove risiede la parità, che invece si ha con il RAID di livello 4.

- **RAID di livello 6.** Il livello 6, detto anche **schema di ridondanza P + Q**, è molto simile al RAID di livello 5, ma memorizza ulteriori informazioni ridondanti per poter gestire guasti contemporanei di più dischi. Invece di usare la parità, s'impiegano codici per la correzione degli errori come i **codici di Reed-Solomon**. Nello schema mostrato nella Figura 14.9g, sono memorizzati 2 bit di dati ridondanti ogni 4 bit di dati effettivi (a differenza di 1 bit di parità usato nel livello 5) e il sistema risultante può tollerare due guasti dei dischi.

- **RAID di livello 0 + 1.** Il livello 0 + 1 consiste in una combinazione dei livelli RAID 0 e 1. Il livello 0 fornisce le prestazioni, mentre il livello 1 l'affidabilità. Di solito, questo schema porta a prestazioni migliori rispetto al livello 5 e si usa prevalentemente negli ambienti in cui sono importanti sia le prestazioni sia l'affidabilità.

http://www.ashampdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Sfortunatamente, questo schema richiede, come il RAID di livello 1, un raddoppio del numero di dischi necessario per memorizzare i dati, quindi è anche più costoso del RAID di livello 5. Nel RAID di livello 0 + 1, si sezionano i dati presenti in un insieme di dischi e si duplica ogni sezione con la tecnica della copiatura speculare. Un altro metodo che sta diventando disponibile commercialmente è il RAID di livello 1 + 0, in cui si fa prima la copiatura speculare dei dischi a coppie, e poi il sezionamento su queste coppie. Questo schema RAID ha alcuni vantaggi teorici rispetto al RAID 0 + 1. Ad esempio, se si guasta un singolo disco nel RAID 0 + 1, l'intera sezione di dati diventa inaccessibile, lasciando disponibile solo l'altra sezione. Con un guasto nel RAID 1 + 0, il singolo disco diventa inaccessibile, ma il suo duplicato è ancora disponibile, come tutti gli altri dischi (Figura 14.10).

Infine, si deve considerare che sono state proposte numerose altre varianti agli schemi RAID di base illustrati sopra e questo ha portato anche una certa confusione nelle precise definizioni dei diversi livelli RAID.

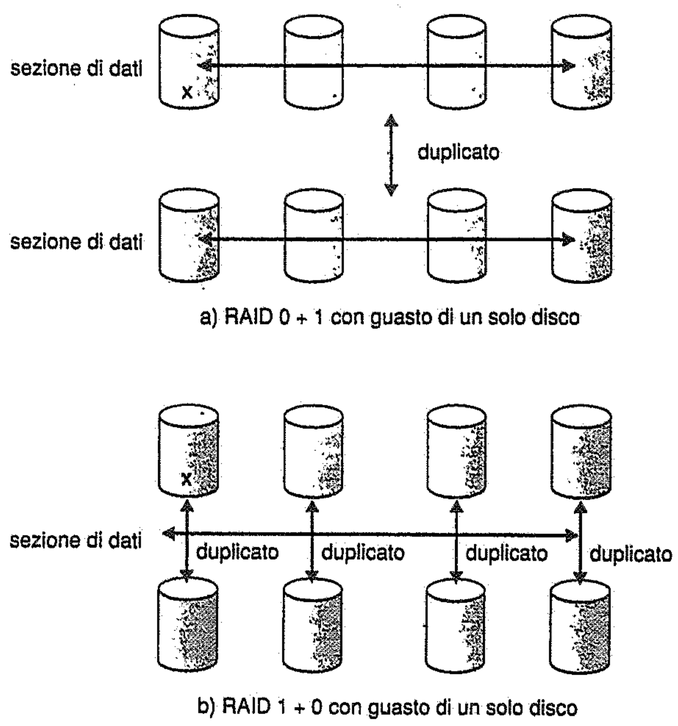


Figura 14.10 RAID 0 + 1 e 1 + 0.

14.5.4 Scelta di un livello RAID

Se un disco si guasta, il tempo per ricostruire i dati in esso memorizzati può essere rilevante e può variare secondo il livello RAID impiegato. La ricostruzione più semplice si ha con il livello 1, poiché i dati possono essere semplicemente copiati dal duplicato del disco; per gli altri livelli, è necessario accedere a tutti gli altri dischi della batteria. Le prestazioni di ricostruzione di un sistema RAID possono essere un fattore importante se è richiesta una fornitura continua dei dati, come nei sistemi ad alte prestazioni o nei sistemi interattivi di basi di dati. Inoltre, le prestazioni di ricostruzione influenzano il tempo medio di guasto.

Il RAID di livello 0 si usa nelle applicazioni ad alte prestazioni in cui le perdite di dati non sono critiche. Il RAID di livello 1 si usa comunemente nelle applicazioni che richiedono un'alta affidabilità e un rapido ripristino. I livelli RAID 0 + 1 e 1 + 0 si usano dove le prestazioni e l'affidabilità sono importanti, ad esempio per piccole basi di dati. A causa dell'elevata richiesta di spazio del RAID di livello 1, per la memorizzazione di grandi quantità di dati, spesso si preferisce impiegare il RAID di livello 5. Il livello 6, attualmente non disponibile in molti sistemi RAID, dovrebbe offrire una migliore affidabilità rispetto al livello 5.

I progettisti dei sistemi RAID devono prendere anche altre decisioni importanti. Ad esempio riguardo al numero ottimale di dischi in una batteria e al numero di bit che ciascun bit di parità deve proteggere. Maggiore è il numero di dischi in una batteria, maggiore sarà la capacità di trasferimento dei dati, ma il sistema sarà anche più costoso. Maggiore è il numero di bit protetti da un singolo bit di parità, minore sarà lo spazio richiesto dai bit di parità, ma sarà maggiore anche la probabilità che un secondo disco si guasti prima che si ripari un disco guasto e questo porterebbe alla perdita di dati.

Un'altra caratteristica della maggior parte dei sistemi RAID è l'uso di un disco di scorta. Un disco di scorta (*hot spare*) non è impiegato per i dati, ma è configurato in modo da poter essere usato per sostituire un altro disco nel caso di un guasto. Ad esempio, si può usare per ricostruire uno dei due dischi di una coppia configurata per la copiatura speculare nel caso si verifichi un guasto. In questo modo si può ristabilire automaticamente il livello RAID, senza attendere la sostituzione del disco guasto. Se si riservano più dischi di scorta, si può provvedere alla riparazione di più di un guasto senza l'intervento di un operatore.

14.5.5 Estensioni

I concetti relativi ai sistemi RAID sono stati generalizzati ad altri dispositivi di memorizzazione, comprese le batterie di nastri e anche alla diffusione dei dati tramite sistemi senza fili (*wireless*). Con strutture RAID applicate alle batterie di nastri si possono ripristinare i dati anche se uno dei nastri della batteria è danneggiato. Se applicate alla trasmissione dei dati, si divide ogni blocco di dati in unità più piccole che si trasmettono insieme a un'unità di parità; se per qualsiasi ragione una delle unità non viene ricevuta, può essere ricostruita dalle altre unità. Di solito, con l'uso di unità automatiche dotate di molte unità a nastro si esegue il sezionamento dei dati su tutte le unità per aumentare la produttività e diminuire il tempo di trasferimento dei dati.

http://www.adultpdf.com
Created by TIFF To PDF trial version. To remove this mark, please register this software.

14.6 Connessione dei dischi

I calcolatori accedono alla memoria secondaria in due modi: tramite le porte di I/O (**memoria secondaria connessa alla macchina**) è il modo più comune nei sistemi di piccole dimensioni; oppure vi accedono in modo remoto per mezzo di un file system distribuito (**memoria secondaria connessa alla rete**).

14.6.1 Memoria secondaria connessa alla macchina

Alla memoria secondaria connessa alla macchina si accede dalle porte locali di I/O. Queste porte sono disponibili in diverse tecnologie; i comuni PC impiegano un'architettura per il bus di I/O chiamata IDE o ATA. Quest'architettura consente di avere non più di due unità per ciascun bus di I/O. Le stazioni di lavoro di fascia alta e i server impiegano architetture più raffinate come SCSI e FC (*fibre channel*).

L'architettura SCSI è un'architettura a bus il cui mezzo fisico è di solito un cavo piatto con un gran numero di conduttori (di solito 50 o 68). Consente di avere sul bus fino a 16 dispositivi, comunemente divisi in una scheda con il controllore inserita nella macchina (*SCSI initiator*) e in 15 dispositivi di memorizzazione (*SCSI target*), tipicamente dischi SCSI. Il protocollo permette di accedere fino a 8 **unità logiche** per ciascun dispositivo di memorizzazione. Un uso tipico dell'accesso a unità logiche è l'invio di comandi ai componenti di una batteria RAID, o ai componenti di un archivio di mezzi rimovibili (come un *juke-box* di CD che invia comandi al meccanismo per la sostituzione dei CD o a una delle unità).

L'FC è un'architettura seriale ad alta velocità che può funzionare sia su fibra ottica sia su un cavo con 4 conduttori di rame e ha due varianti. La prima è una grande struttura a commutazione con uno spazio d'indirizzi a 24 bit. Per il futuro ci si aspetta che questo metodo prevalga, ed è la base per le **reti di memoria secondaria** (*storage-area networks* — SAN). Grazie al vasto spazio d'indirizzi e alla natura a commutazione della comunicazione, si possono connettere più macchine e dispositivi di memorizzazione alla struttura a commutazione, permettendo una notevole flessibilità nella comunicazione di I/O. La seconda variante si chiama FC-AL (*arbitrated loop*) e può accedere fino a 126 dispositivi (unità e controllori).

C'è un gran numero di dispositivi che si possono usare come memoria secondaria connessa alla macchina, tra questi le unità a disco, le batterie RAID, le unità a CD, DVD e a nastri magnetici.

I comandi di I/O che avviano trasferimenti di dati a un dispositivo di memoria connessa alla macchina sono letture e scritture di blocchi logici di dati, dirette a unità di memorizzazione specificamente identificate (ad esempio, tramite bus ID, SCSI ID e unità logica del dispositivo).

14.6.2 Memoria secondaria connessa alla rete

Un dispositivo di memoria secondaria connessa alla rete è un sistema di memoria speciale al quale si accede in modo remoto per mezzo di una rete di trasmissione di dati (Figura 14.11). I client accedono alla memoria connessa alla rete (*network-attached storage* — NAS) tramite un'interfaccia RPC, ad esempio l'NFS nel caso dei sistemi UNIX o CIFS per i sistemi Windows. Le chiamate di procedura remota (RPC) sono realizzate per mezzo dei protocolli TCP o UDP sopra una rete IP (di solito la stessa rete locale che porta tutto il traffico di dati ai client). La memoria secondaria connessa alla rete è normalmente realizzata come una batteria RAID con programmi di controllo che realizzano l'interfaccia per le RPC. Conviene pensare ai sistemi NAS semplicemente come a un altro protocollo per l'accesso alla memoria secondaria; ad esempio, anziché usare un driver per dispositivi SCSI e i relativi protocolli SCSI per accedere alla memoria secondaria, un sistema che usa sistemi NAS impiega le RPC sopra i protocolli TCP/IP.

La memoria secondaria connessa alla rete fornisce a tutti i calcolatori di una LAN un modo semplice per condividere spazio di memorizzazione, con la stessa facilità di gestione dei nomi e degli accessi caratteristica della memoria secondaria locale. Tuttavia, un sistema di questo genere tende a essere meno efficiente e ad avere prestazioni inferiori rispetto a sistemi che prevedono la connessione diretta alla memoria secondaria.

14.6.3 Reti di memoria secondaria

Uno svantaggio dei sistemi di memoria secondaria connessa alla rete è che le operazioni di I/O sulla memoria secondaria impegnano banda della rete e quindi aumentano la latenza della comunicazione nella rete. Questo problema può essere particolarmente grave per sistemi client-server di grandi dimensioni: l'ordinaria comunicazione tra i server e i client compete per la banda con la comunicazione tra i server e i dispositivi di memorizzazione.

Una rete di memoria secondaria (*storage-area network* — SAN) è una rete privata (che impiega protocolli specifici per la memorizzazione anziché protocolli di rete) tra i server e le unità di memoria secondaria, separata dalla LAN o WAN che collega i server ai

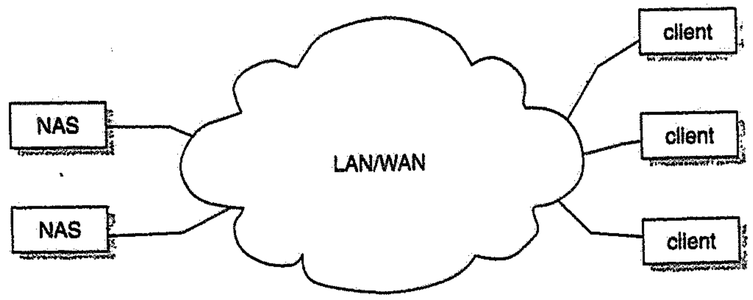


Figura 14.11 Memoria secondaria connessa alla rete.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

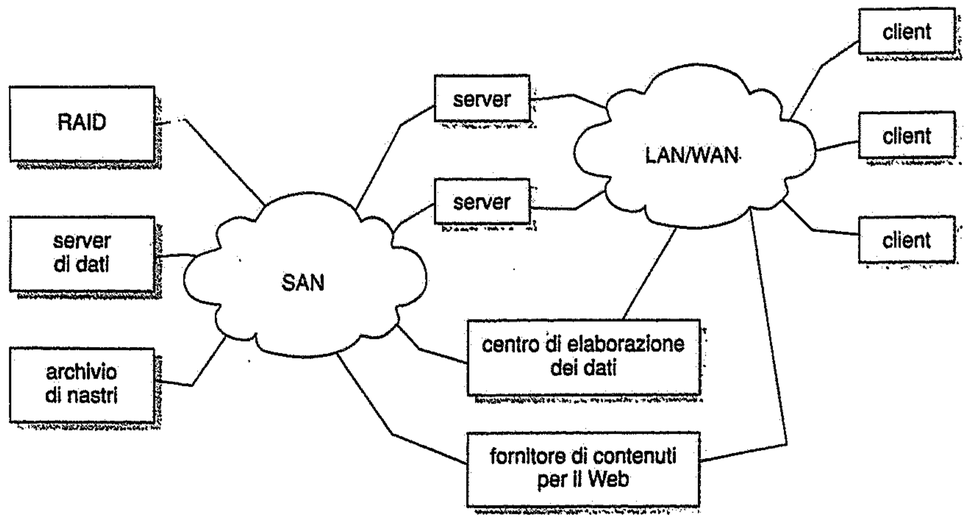


Figura 14.12 Rete di memoria secondaria.

client (Figura 14.12). La potenza di una SAN sta nella sua flessibilità: si possono connettere alla stessa SAN molte macchine e molte batterie di memoria, e la memoria può essere assegnata alle macchine dinamicamente. Ad esempio, se una macchina sta esaurendo il suo spazio nei dischi, la SAN si può configurare in modo da assegnargli più memoria secondaria. Attualmente sono disponibili in commercio molti sistemi SAN, ma non sono né ben standardizzati né capaci di integrarsi. La maggior parte dei sistemi SAN presenti sul mercato nel 2001 è basata su reti FC-AL o FC a commutazione. Un'alternativa all'interconnessione con l'FC che sta emergendo per le SAN è la memoria secondaria su infrastruttura di rete IP, come la Gigabit Ethernet. Un'altra potenziale alternativa è un'architettura specifica per SAN, chiamata Infiniband, per reti d'interconnessione ad alta velocità tra server e unità di memoria secondaria.

14.7 Realizzazione della memoria stabile

Nel Capitolo 7 si introduce la registrazione con scrittura anticipata, che richiede la disponibilità di una memoria stabile. Per definizione, le informazioni residenti in questo tipo di memoria non vanno *mai* perse. Per realizzare un tale tipo di memoria si devono replicare le informazioni necessarie in più dispositivi di memorizzazione (di solito dischi) con modi di malfunzionamento indipendenti. Inoltre è necessario coordinare l'aggiornamento delle informazioni in modo tale che un eventuale malfunzionamento durante l'aggiornamento non lasci tutte le copie in uno stato danneggiato, e che durante il ripristino delle informazioni a seguito di un guasto, sia possibile riportare ogni copia alla sua forma

corretta anche se si verifica un altro malfunzionamento proprio durante il ripristino. Nel resto del paragrafo si discutono i modi in cui si possono soddisfare queste esigenze.

Un'operazione di scrittura in un disco può avere uno dei seguenti esiti:

1. **Operazione riuscita.** I dati sono stati scritti correttamente nel disco.
2. **Insuccesso parziale.** È avvenuto un malfunzionamento durante il trasferimento, e solo alcuni tra i settori coinvolti sono stati correttamente aggiornati, mentre il settore interessato dalla scrittura al momento del malfunzionamento può essere ora danneggiato.
3. **Insuccesso totale.** Il malfunzionamento è avvenuto prima dell'avvio del processo di scrittura nel disco; i dati già residenti nel disco sono rimasti inalterati.

Si richiede che il sistema riconosca un malfunzionamento verificatosi durante il trasferimento e invochi una procedura di ripristino per riportare il blocco in uno stato coerente. A tale scopo il sistema deve mantenere due blocchi fisici per ciascun blocco logico, eseguendo ogni operazione nel modo seguente:

1. scrittura delle informazioni nel primo blocco fisico;
2. completata con successo la prima scrittura, scrittura delle stesse informazioni nel secondo blocco fisico;
3. l'operazione è considerata completa solamente dopo che la seconda scrittura è stata eseguita correttamente.

Durante il ripristino dovuto a un malfunzionamento, si esamina ogni coppia di blocchi fisici; se essi contengono gli stessi dati e non c'è traccia d'errori, non è necessario intraprendere alcuna azione ulteriore. Se un blocco contiene un errore riscontrabile, se ne sostituisce il contenuto con quello del secondo blocco. Se in nessuno dei due blocchi si riscontra un errore, ma ciò nonostante essi contengono informazioni differenti, si sostituisce il contenuto del primo blocco con quello del secondo. Questa procedura di ripristino assicura che gli unici due esiti possibili di un'operazione di scrittura nella memoria stabile siano o la completa riuscita dell'operazione oppure il suo insuccesso totale, in quest'ultimo caso i dati memorizzati non subiscono alcun cambiamento.

Questa procedura si può facilmente estendere all'uso di un numero arbitrariamente grande di copie per ciascun blocco di memoria stabile. Benché un gran numero di queste copie riduca la probabilità di malfunzionamenti, di solito è ragionevole simulare la memoria stabile con due sole copie. I dati di una memoria stabile non andranno mai persi purché un guasto non distrugga tutte le copie.

Poiché le attese per il completamento delle scritture (I/O sincrono) richiedono molto tempo, molte batterie di memorizzazione aggiungono NVRAM come cache. Poiché la memoria è non volatile (di solito è dotata di una pila per l'alimentazione elettrica di riserva), si può ritenere affidabile come un disco per la memorizzazione dei dati, e si può considerare parte della memoria stabile. Le scritture in essa sono molto più rapide di quelle nei dischi, quindi le prestazioni migliorano notevolmente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

14.8 Strutture per la memorizzazione terziaria

Nessuno comprerebbe un videoregistratore contenente un'unica cassetta non sostituibile o non estraibile, o un lettore di CD o di audiocassette capace di riprodurre un solo album incorporato: ci si attende di poter usare un videoregistratore o un lettore di CD con videocassette o dischi diversi e relativamente poco costosi. Anche nei calcolatori l'uso di mezzi di memorizzazione rimovibili economici e di un solo dispositivo di lettura o scrittura riduce i costi complessivi.

14.8.1 Dispositivi per la memorizzazione terziaria

La caratteristica peculiare della memoria terziaria è il suo basso costo: in pratica, quindi, essa consiste di **mezzi rimovibili**, dei quali i dischetti, i CD-ROM, i nastri magnetici sono gli esempi più comuni; sul mercato sono disponibili anche molti altri prodotti.

14.8.1.1 Dischi rimovibili

I dischi rimovibili sono un tipo di memoria terziaria. I dischetti sono un esempio di dischi magnetici rimovibili, e sono costituiti da un disco sottile e flessibile, ricoperto di materiale magnetico e racchiuso in un involucro protettivo di plastica. I comuni dischetti hanno una capacità di circa 1 MB, ma una tecnologia simile si usa per costruire dischi magnetici rimovibili della capacità di più di 1 GB. I dischi magnetici rimovibili possono funzionare a una velocità quasi pari a quella di un'unità a disco, anche se il rischio che la loro superficie sia danneggiata da graffi è maggiore.

I **dischi magneto-ottici** sono un altro tipo di dischi rimovibili: anche in questo caso i dati sono registrati su un disco ricoperto di materiale magnetico, ma la tecnologia impiegata per la registrazione è molto diversa da quella dei dischi magnetici. La testina di un disco magneto-ottico è sospesa a una distanza dalla superficie del disco molto maggiore rispetto alla testina di un disco magnetico, e il materiale magnetico è protetto da uno spesso strato di plastica o di vetro; di conseguenza, il disco è molto più resistente a eventuali collisioni della testina.

L'unità per i dischi magneto-ottici ha una bobina capace di produrre un campo magnetico, ma a temperature ordinarie il campo è troppo diffuso e debole per poter magnetizzare un bit sul disco; per rimediare a questo fatto la testina emette un raggio laser verso la superficie del disco, puntandolo sulla piccolissima area dove si vuole scrivere un bit. Il surriscaldamento dell'area provocato dal laser fa sì che essa divenga sensibile al campo magnetico, ed è in questo modo che un campo magnetico debole e diffuso riesce a registrare un minuscolo bit.

La testina di un disco magneto-ottico è troppo lontana dalla superficie del disco per poter leggere i dati rilevando i piccolissimi campi magnetici in modo analogo a quanto fa la testina di un disco magnetico. Perciò l'unità a disco legge i bit sfruttando una proprietà della luce laser detta **effetto Kerr**; quando un raggio laser è riflesso da un punto magnetizzato la sua polarizzazione è ruotata in senso orario o antiorario secondo l'orientazione del campo magnetico: per leggere i bit la testina rileva questa rotazione.

Un'altra categoria di dischi rimovibili è quella dei **dischi ottici**, i quali non sfruttano per niente il magnetismo, ma usano materiali speciali che la luce laser può alterare in modo da creare punti relativamente chiari o scuri. Un esempio di tecnologia per dischi ottici sono i dischi a cambio di fase.

Un **disco a cambio di fase** è ricoperto di un materiale che può solidificare passando a uno stato cristallino o a uno stato amorfo. Lo stato cristallino è più trasparente, perciò un raggio laser è più luminoso quando attraversa il materiale a cambio di fase ed è riflesso dall'apposito strato. Le unità per dischi a cambio di fase impiegano laser capaci di emettere raggi a tre differenti livelli di potenza: bassa, per leggere i dati; media, per cancellare il disco fondendo e facendo solidificare il mezzo di registrazione nello stato cristallino; alta, per scrivere nel disco fondendo e facendo solidificare il mezzo di registrazione nello stato amorfo. Gli esempi più comuni di questo tipo di tecnologia sono i dischi ottici riscrivibili CD-RW e DVD-RW.

I tipi di dischi fin qui descritti si possono riutilizzare: per questo sono detti **dischi a lettura e scrittura**. Per contro, i **dischi monoscrivibili** o (*write once, read many times* — WORM) costituiscono una categoria distinta. Un vecchio modo di costruire un disco WORM è di inserire una pellicola d'alluminio tra due piatti di plastica o di vetro. Per scrivere un bit l'unità usa un raggio laser per praticare un piccolo foro nell'alluminio: poiché questo processo non è reversibile, si può scrivere una sola volta su un qualunque settore del disco. Sebbene sia possibile distruggere l'informazione contenuta in un disco WORM, ad esempio praticando fori dappertutto, è praticamente impossibile alterare i dati in esso contenuti, perché l'unica azione possibile è quella di aggiungere fori, ed è assai probabile che il codice ECC associato a ogni settore rilevi le modifiche. I dischi WORM sono considerati durevoli e affidabili: lo strato di metallo è protetto dalla copertura di vetro o plastica, e i campi magnetici non possono danneggiare la registrazione. Una più recente tecnologia di monoscrittura compie la registrazione su un pigmento polimerico invece che su uno strato d'alluminio: il pigmento forma dei punti assorbendo la luce del laser. Questa tecnologia s'impiega nei dischi ottici scrivibili CD-R e DVD-R

I **dischi a sola lettura**, ad esempio i CD-ROM e i DVD, sono commercializzati con un contenuto preregistrato: fanno uso di una tecnologia simile a quella dei dischi WORM (sebbene in questo caso i fori siano stampati) e sono assai durevoli.

Generalmente i dischi rimovibili sono più lenti dei corrispondenti dischi fissi. Il processo di scrittura è più lento così come sono più lenti la rotazione e talvolta la ricerca.

14.8.1.2 Nastri

I nastri magnetici sono un altro tipo di mezzo rimovibile. In linea generale un nastro può contenere più dati di un disco ottico o magnetico rimovibile. Le unità a nastro e quelle a disco hanno velocità di trasferimento simili ma, poiché richiedono operazioni di avanzamento rapido o di riavvolgimento che possono durare decine di secondi o addirittura interi minuti, l'accesso diretto è molto più lento per un nastro che per un disco.

Anche se un'unità a nastro è più costosa di un'unità a disco, una cartuccia a nastro è più economica di un disco magnetico della stessa capacità: i nastri magnetici sono quindi un mezzo conveniente qualora non si richieda la possibilità di rapidi accessi di-

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

retti. I nastri si usano comunemente per contenere copie di riserva dei dati presenti nei dischi, ma si usano anche nei grandi centri di calcolo, dotati di supercalcolatori, per memorizzare le enormi quantità di dati che s'impiegano nella ricerca scientifica o per la gestione di grandi aziende.

Esistono nastri capaci di contenere molti più dati di un'unità a disco; in effetti, l'area superficiale di un nastro è molto più grande di quella di un disco. La capacità di memorizzazione di un nastro potrebbe aumentare ulteriormente, giacché a tutt'oggi la **densità superficiale** di dati nei nastri (bit al centimetro quadrato) è molto minore di quella nei dischi magnetici.

Grandi stazioni di registrazione a nastri usano tipicamente meccanismi automatici per spostare i nastri dalle unità ad appositi contenitori in un archivio di nastri; questi meccanismi offrono ai calcolatori l'accesso automatico a un elevato numero di nastri.

Un archivio automatico riduce i costi totali della registrazione dei dati. Un file non immediatamente necessario residente in un disco si può **archiviare** in un nastro a un costo per gigabyte che può essere inferiore; quando il file si renderà necessario, il calcolatore potrà installarlo nuovamente nel disco. Un archivio automatico realizza un tipo di memorizzazione talvolta detto **quasi in linea**, perché si situa fra le alte prestazioni della memorizzazione **in linea** nei dischi magnetici e il basso costo di una memorizzazione **non in linea** in nastri archiviati in qualche deposito.

14.8.1.3 Tecnologie future

È possibile che divengano importanti altre tecnologie di memorizzazione. Una promettente tecnologia di memorizzazione, la memorizzazione olografica, adopera la luce laser, per memorizzare ologrammi su mezzi speciali. Si pensi a un'immagine in bianco e nero come a una matrice bidimensionale di *pixel*: ogni pixel rappresenta un bit, nero per lo 0 e bianco per l'1. Una fotografia nitida può contenere milioni di bit di dati, e tutti i pixel di un ologramma si trasferiscono con un'unica emissione laser, cosicché la velocità di trasferimento è altissima. Grazie a un continuo sviluppo, questo tipo di memorizzazione potrebbe divenire commercialmente praticabile.

Un'altra tecnologia di memorizzazione oggetto di attive ricerche si fonda sui **sistemi meccanici microelettronici** (*microeletronic mechanical systems* — MEMS). L'idea consiste nell'applicare le tecnologie che s'impiegano nella fabbricazione dei circuiti integrati per costruire piccole macchine di memorizzazione. Una proposta prevede la fabbricazione di una matrice di 10.000 minuscole testine di disco, con un centimetro quadrato di materiale magnetico di registrazione sospeso sopra tale matrice. Quando si muove il materiale di registrazione lungo le testine, ognuna di esse accede alla propria traccia lineare di dati presente sul materiale magnetico. Il materiale di registrazione si può traslare leggermente in modo laterale per consentire a tutte le testine di accedere alla loro traccia successiva. Sebbene resti da vedere se questa tecnologia possa avere successo, essa può offrire sistemi di memorizzazione non volatile più rapidi dei dischi magnetici e più economici delle memorie a semiconduttori DRAM.

Indipendentemente dal fatto che il mezzo di memorizzazione in uso sia un disco magnetico rimovibile, un DVD, o un nastro magnetico, il sistema operativo deve offrire una serie di funzioni affinché i mezzi rimovibili siano utilizzabili per contenere dati: questo argomento è trattato nel Paragrafo 14.8.2.

14.8.2 Compiti del sistema operativo

Due tra gli obiettivi primari di un sistema operativo sono la gestione dei dispositivi fisici e la presentazione di una macchina virtuale alle applicazioni. Il sistema operativo realizza due astrazioni concernenti i dischi: una è il dispositivo a basso livello, un semplice vettore di blocchi di dati; l'altra è il file system. Se il file system è relativo a un disco magnetico il sistema operativo accoda e organizza le richieste provenienti da diverse applicazioni. Nel seguito si espone il comportamento del sistema operativo nel trattare i mezzi di memorizzazione rimovibili.

14.8.2.1 Interfaccia per le applicazioni

La maggior parte dei sistemi operativi gestisce i dischi rimovibili pressoché nella stessa maniera dei dischi fissi. Quando s'inserisce un nuovo disco nella relativa unità esso deve essere formattato, quindi si crea sul disco rimovibile un file system vuoto che si usa proprio come il file system di un'ordinaria unità a disco.

La gestione dei nastri, invece, è spesso differente; il sistema operativo di solito presenta un nastro come un mezzo di memorizzazione a basso livello. Un'applicazione non apre un file presente nel nastro: apre l'intera unità nastro come dispositivo a basso livello. In questo caso, di solito, l'unità a nastro si riserva per l'uso esclusivo da parte di tale applicazione fino a che essa termina o chiude il dispositivo. L'esclusività è ragionevole perché l'accesso diretto ai dati presenti in un nastro può richiedere decine di secondi o persino qualche minuto, sicché intercalare gli accessi diretti a un nastro determinerebbe probabilmente enormi tempi d'accesso per un ridottissimo lavoro utile.

Quando un'unità a nastro è presentata come dispositivo a basso livello, il sistema operativo non fornisce i servizi del file system: è l'applicazione che deve decidere come usare il vettore di blocchi. Un programma che crea una copia di riserva di un disco su un nastro potrebbe ad esempio scrivere un elenco dei nomi e delle dimensioni dei file all'inizio del nastro, e poi copiare i dati dei file sul nastro in quell'ordine.

È facile rendersi conto dei problemi che possono sorgere quando si usi l'unità a nastro in questo modo: visto che ogni applicazione stabilisce i propri criteri di organizzazione del nastro, un nastro contenente dati può essere generalmente usato solo dal programma che lo ha creato. Se anche si sapesse, ad esempio, che un nastro con copie di riserva di file contiene un elenco dei nomi e delle dimensioni dei file seguita dai dati dei file in quell'ordine, ci sarebbero comunque difficoltà nell'uso del nastro: non è nota l'esatta maniera in cui i nomi dei file sono registrati, né se le dimensioni siano espresse nel codice binario o ASCII, o ancora se i file siano scritti uno per blocco o invece concatenati assieme in una lunghissima sequenza di byte. Non è nota neanche la dimensione dei blocchi del nastro, perché questo è un parametro che si può fissare indipendentemente per ogni blocco scritto.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le operazioni fondamentali relative a un'unità a disco sono `read`, `write` e `seek`; per le unità a nastro s'impiega un insieme di operazioni fondamentali diverso. Anziché usare l'operazione `seek`, si usa l'operazione `locate`. Si tratta di un'operazione più precisa dell'operazione `seek` per il disco, perché posiziona il nastro in corrispondenza di uno specifico blocco logico, e non di un'intera traccia: localizzare il blocco 0 equivale a riavvolgere il nastro.

Nella maggior parte delle unità nastro è possibile localizzare qualunque blocco sia stato scritto in un nastro, ma se il nastro non è ancora completamente pieno non si può eseguire un'operazione `locate` nell'area vuota oltre l'area registrata; ciò poiché la maggior parte delle unità a nastro gestisce lo spazio fisico diversamente dalle unità a disco. I settori di un disco hanno una dimensione fissa, e si deve usare il processo di formattazione per assegnare la posizione definitiva ai settori vuoti prima che possano contenere qualunque informazione. La maggior parte delle unità a nastro ha una dimensione dei blocchi variabile, e la dimensione di ogni blocco si determina al momento della scrittura del blocco in questione. Se durante la scrittura s'incontra una regione difettosa del nastro, la si salta e si riscrive il blocco. Tutto ciò spiega perché non sia possibile compiere un'operazione `locate` nella regione di nastro vuota presente oltre l'area già registrata: le posizioni e la numerazione dei blocchi logici non sono ancora state determinate.

Nella maggior parte dei casi le unità a nastro dispongono di un'operazione `read position` che riporta il numero del blocco logico in corrispondenza del quale si trova la testina. Molte unità a nastro hanno anche un'operazione `space` per gli spostamenti relativi: l'operazione `space -2`, ad esempio, riavvolge il nastro di due blocchi logici.

In molti tipi di unità a nastro la scrittura di un blocco produce come effetto collaterale la cancellazione logica di tutto ciò che si trova oltre la posizione della scrittura. Ciò significa che nella maggior parte dei casi le unità a nastro sono dispositivi a solo accodamento di dati (*append-only devices*); in altre parole, l'aggiornamento di un blocco posto in mezzo al nastro comporta la cancellazione di tutto ciò che segue tale blocco. L'unità a nastro realizza l'accodamento scrivendo un simbolo di fine nastro (*end of tape* — EOT) dopo l'ultimo blocco registrato: l'unità rifiuta di compiere un'operazione `locate` oltre il simbolo EOT, ma può localizzare l'EOT stesso e poi cominciare a scrivere. Quest'azione ha l'effetto di sovrascrivere il simbolo EOT e di accodarne uno nuovo alla fine dei blocchi appena scritti.

In linea di principio si potrebbero realizzare file system per i nastri ma, poiché le unità a nastro sono dispositivi a solo accodamento, molte strutture di dati e algoritmi sarebbero diversi da quelli che si usano per i dischi.

14.8.2.2 Nomi dei file

Un'altra questione che il sistema operativo deve affrontare è l'assegnazione dei nomi ai file residenti nei mezzi rimovibili. Nel caso di un disco fisso ciò non è difficile: nei PC i nomi dei file consistono di una lettera rappresentante un'unità seguita da un nome di percorso; nel sistema operativo UNIX, i nomi dei file non contengono riferimenti alle unità, ma la tabella di montaggio permette al sistema operativo di identificare l'unità contenente ciascun file. Se però il disco è rimovibile, il fatto che un'unità abbia ospitato un

certo disco non facilita la ricerca del file. Se ogni disco rimovibile avesse un numero di serie diverso, un file residente in un'unità potrebbe presentare come prefisso il proprio numero di serie, ma in questo caso sarebbe necessario usare circa 12 cifre per evitare che due dischi possano avere lo stesso numero di serie: non è pensabile che gli utenti ricordino numeri di 12 cifre per identificare i loro file.

Le cose si complicano ulteriormente nel caso s'intenda scrivere dati su un mezzo rimovibile in un certo calcolatore e poi riutilizzare lo stesso mezzo in un altro calcolatore: se entrambi i calcolatori sono dello stesso tipo e hanno lo stesso tipo di unità, l'unico problema è quello di conoscere i contenuti e l'organizzazione dei dati contenuti nel mezzo in questione; ma se i calcolatori o le unità sono di diverso tipo possono sorgere molte altre difficoltà. Anche se le unità fossero compatibili, calcolatori diversi potrebbero memorizzare i dati secondo ordini diversi, o usare codifiche diverse per i numeri binari e persino per le lettere (ASCII nei PC, EBCDIC nei mainframe).

In genere gli attuali sistemi operativi lasciano irrisolto il problema dei nomi per i mezzi rimovibili, confidando nel fatto che le applicazioni o gli utenti forniranno una chiave di lettura e di interpretazione dei dati. Per fortuna alcuni tipi di mezzi rimovibili sono così ben standardizzati da essere usati allo stesso modo da tutti i calcolatori. Un esempio è dato dai CD: i CD musicali sono registrati in un formato universalmente noto e leggibile da ogni riproduttore di CD. I CD di dati sono disponibili in pochi formati diversi, ed è normale che sia l'unità di lettura sia il sistema operativo siano in grado di gestirli. Anche i formati dei DVD sono ben standardizzati.

14.8.2.3 Gestione gerarchica della memoria

Un **juke-box automatico** permette a un calcolatore di cambiare un nastro o un disco rimovibile senza l'intervento di un utente. Due tra le principali applicazioni di questa tecnica sono relative alla realizzazione di copie di riserva e ai sistemi di gestione gerarchica della memoria. L'uso dei juke-box per la creazione di copie di riserva è molto semplice: quando un nastro o un disco sono pieni, il calcolatore richiede al juke-box di passare al successivo. Alcuni juke-box contengono decine di unità e migliaia di nastri, con bracci automatici che spostano i nastri nelle unità.

Un sistema di gestione gerarchica della memoria estende la gerarchia di memorizzazione oltre la memoria centrale e secondaria (cioè, i dischi magnetici) comprendendo la memoria terziaria; quest'ultima è di solito costituita di un juke-box di nastri o di dischi rimovibili. Si tratta del livello di memoria meno costoso e più capiente, ma probabilmente anche più lento.

Sebbene il sistema della memoria virtuale si possa estendere senza difficoltà alla memoria terziaria, in pratica ciò avviene raramente, infatti recuperare dati per mezzo di un juke-box può richiedere decine di secondi o addirittura minuti: attese così lunghe sono inconciliabili con le tecniche di paginazione su richiesta e con altri modi d'uso della memoria virtuale.

La tecnica più comune per estendere la gerarchia di memorizzazione fino alla memoria terziaria consiste nell'ampliare il file system. I file piccoli e frequentemente usati

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

rimangono nei dischi magnetici, mentre i file vecchi, ingombranti e raramente necessari si archiviano nel juke-box. In alcuni sistemi per l'archiviazione dei file gli elementi di directory corrispondenti ai nomi dei file continuano a comparire nelle directory anche dopo l'archiviazione, ma i contenuti dei file non occupano più spazio nella memoria secondaria. Quando un'applicazione tenta di aprire un file archiviato, la chiamata del sistema open rimane sospesa fino a che i contenuti del file possono essere reinstallati dalla memoria terziaria; una volta nuovamente disponibili nei dischi magnetici, la open restituisce il controllo all'applicazione, e quest'ultima può ora accedere ai contenuti del file. La **gestione gerarchica della memoria** (*hierarchical storage management* — HSM) è stata realizzata in ordinari sistemi a partizione del tempo come il TOPS-20, un sistema operativo che negli ultimi anni Settanta s'impiegava in minicalcolatori della Digital Equipment Corporation. Al giorno d'oggi l'HSM si trova di solito in centri di calcolo basati su supercalcolatori e in altri grandi sistemi che posseggono enormi quantità di dati.

14.8.3 Prestazioni

Così come avviene per ogni componente del sistema operativo, i tre aspetti più importanti riguardanti le prestazioni della memorizzazione terziaria sono la velocità, l'affidabilità e i costi.

14.8.3.1 Velocità

La velocità della memoria terziaria è definita da due fattori: l'ampiezza di banda e la latenza. La prima si misura in byte al secondo; in particolare, l'**ampiezza di banda sostenuta** è la velocità media di trasferimento nel caso di una rilevante quantità di dati — in altre parole, il numero di byte diviso il tempo di trasferimento; l'**ampiezza di banda effettiva** è invece il numero di byte trasferiti rapportato al tempo di I/O totale, inclusi il tempo richiesto da una seek o una locate e l'attesa eventualmente dovuta a cambi di dischi o nastri eseguiti dal juke-box. Essenzialmente, l'ampiezza di banda sostenuta è la velocità di trasferimento nel momento in cui i dati stanno effettivamente fluendo, mentre l'ampiezza di banda effettiva è la velocità di trasferimento complessiva fornita dall'unità. Con l'espressione *ampiezza di banda di un'unità* s'intende generalmente l'ampiezza di banda sostenuta.

L'ampiezza di banda per le unità a dischi rimovibili varia da meno di 0,25 MB al secondo nei tipi più lenti, a svariati megabyte al secondo nei più veloci. Le unità a nastro hanno una variabilità ancora maggiore, da meno di 0,25 MB al secondo a più di 30 MB al secondo. Le unità a nastro più veloci hanno quindi un'ampiezza di banda significativamente maggiore delle unità a dischi rimovibili.

Il secondo fattore è la **latenza d'accesso**. Rispetto a questo parametro i dischi sono molto più veloci dei nastri: la memorizzazione nei dischi è essenzialmente bidimensionale — tutti i bit sono, per così dire, all'aperto; un accesso al disco si compie semplicemente spostando il braccio al cilindro selezionato e aspettando che il settore interessato ruoti sotto la testina, il che può avvenire in meno di 35 millisecondi. Per contro, la memorizzazione nei nastri è tridimensionale: a ogni dato istante solo una piccola parte del

nastro è accessibile alla testina, mentre il resto dei bit è sepolto sotto centinaia o migliaia di strati di nastro avvolto in una bobina. Un accesso diretto a un nastro richiede lo svolgimento o il riavvolgimento della bobina finché il blocco richiesto raggiunge la testina, cosa che può richiedere decine o centinaia di secondi. Si può quindi dire in linea generale che l'accesso diretto a un nastro è oltre mille volte più lento dell'accesso diretto a un disco.

Se in tutto ciò è coinvolto anche un juke-box la latenza d'accesso può crescere notevolmente: per cambiare un disco rimovibile l'unità deve fermare la rotazione del motore, il braccio automatico deve scambiare i dischi, e l'unità deve avviare la rotazione del nuovo disco. Questa operazione richiede parecchi secondi, un tempo pari a circa cento volte il tempo medio d'accesso diretto a un disco singolo: lo scambio di dischi in un juke-box comporta una penalizzazione relativamente alta delle prestazioni.

Il tempo impiegato da un braccio automatico nel caso dei nastri è circa lo stesso che nel caso dei dischi; in genere, però, un nastro deve essere riavvolto completamente prima di poter essere estratto dall'unità, e quest'operazione può richiedere anche 4 minuti. Inoltre, dopo che un nuovo nastro è stato caricato, l'unità può aver bisogno di molti secondi per calibrarsi rispetto al nuovo nastro e prepararsi all'I/O. Sebbene un lento juke-box per nastri possa richiedere 1 o 2 minuti per scambiare i nastri, questo tempo non è sproporzionatamente lungo rispetto al tempo necessario per l'accesso diretto a un singolo nastro.

Generalizzando, quindi, si può dire che l'accesso diretto a un disco in un juke-box ha una latenza dell'ordine delle decine di secondi, mentre nel caso dei nastri in un juke-box la latenza è dell'ordine delle centinaia di secondi; lo scambio dei dischi è oneroso, mentre non lo è quello dei nastri. Si tratta ovviamente di un discorso di carattere generale: alcuni costosi juke-box per nastri sono capaci di riavvolgere ed estrarre un nastro, caricarne uno nuovo e posizionarlo a uno specifico punto impiegando complessivamente meno di 30 secondi.

Se si considerano soltanto le prestazioni delle unità di lettura e scrittura di un juke-box, i tempi di latenza e l'ampiezza di banda appaiono ragionevoli; non appena si concentra l'attenzione sui mezzi rimovibili si riscontra però una tremenda strozzatura nelle prestazioni. Si consideri in primo luogo l'ampiezza di banda: rispetto a un'unità a disco fisso, in un archivio automatico il rapporto fra l'ampiezza di banda e la capacità di memorizzazione è molto meno favorevole: leggere tutti i dati contenuti in un disco di grandi dimensioni potrebbe richiedere circa un'ora, ma leggere tutti i dati memorizzati in un ingombrante archivio di nastri potrebbe richiedere anni. Per ciò che riguarda la latenza d'accesso la situazione è quasi altrettanto grama: se 100 richieste sono in coda per una unità a disco fisso, il tempo d'attesa medio sarà di circa 1 secondo; ma se 100 richieste sono in coda per un archivio di nastri, il tempo d'attesa medio potrebbe essere più di 1 ora. La convenienza economica della memoria terziaria è dovuta alla possibilità di usare molte cartucce (a disco o a nastro) a basso costo con poche costose unità di lettura e scrittura. Un archivio di mezzi rimovibili, però, e soprattutto adatto alla registrazione di dati usati raramente, perché il numero di richieste di I/O soddisfacibili per ogni ora d'uso di un tale archivio è relativamente basso.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

14.8.3.2 Affidabilità

Nonostante si tenda a identificare il concetto di *buone prestazioni* con quello di *alta velocità*, un altro importante aspetto delle prestazioni è l'*affidabilità*: se non si riuscisse a leggere dati a causa di un guasto dell'unità o del mezzo di memorizzazione, il tempo d'accesso sarebbe infinitamente lungo e l'ampiezza di banda infinitamente bassa. È quindi importante analizzare l'affidabilità dei mezzi rimovibili.

I dischi magnetici rimovibili sono meno affidabili dei dischi fissi, perché è più probabile che siano esposti a condizioni ambientali dannose come polvere, sbalzi di temperatura o umidità e forze meccaniche come urti o piegature. I dischi ottici sono considerati molto affidabili, perché lo strato che memorizza le informazioni è protetto da uno strato trasparente di plastica o vetro. L'affidabilità dei nastri magnetici è molto variabile poiché dipende dal tipo di unità di lettura e scrittura: alcune unità poco costose consumano un nastro dopo averlo usato poche decine di volte, mentre altre sono così delicate da permettere il reimpiego del nastro milioni di volte. Rispetto alla testina di un'unità a disco, la testina di un'unità a nastro è meno affidabile: la prima è sospesa sopra il disco, mentre la seconda è a stretto contatto col nastro, e l'attrito conseguente può rovinarla dopo un elevato numero di ore d'uso.

Riassumendo si può dire che le unità a disco fisso sono più affidabili delle unità a nastri o a dischi rimovibili, e che i dischi ottici sono probabilmente più affidabili dei dischi e dei nastri magnetici. Anche le unità a disco fisso hanno punti deboli: la collisione della testina col disco in genere distrugge i dati, mentre il guasto di un'unità a nastro o di un'unità a dischi ottici lascia spesso intatto il mezzo di memorizzazione in uso al momento del guasto.

14.8.3.3 Costi

Il costo della memoria è un altro fattore importante: l'esempio seguente mostra concretamente come i mezzi rimovibili possano ridurre i costi totali di memorizzazione. Si supponga che un'unità a disco di x GB costi 200 dollari; di questa cifra, 190 dollari sono dovuti al controllore, al motore e al contenitore esterno, e 10 dollari ai piatti magnetici. Il costo della memoria fornita da questa unità è quindi di $200/x$ dollari al gigabyte. Si supponga ora di poter incapsulare i piatti magnetici in un disco rimovibile: il costo complessivo di un'unità e di 10 dischi è allora di 190 + 100 dollari, e la capacità di memoria totale è di $10x$ GB, cosicché il costo per gigabyte scende a $29/x$ dollari al gigabyte. Sebbene il costo di produzione delle unità a dischi rimovibili sia un po' più alto, la maggiore spesa per un'unità è bilanciata dal basso costo di molte cartucce rimovibili, quindi il costo per gigabyte della memoria rimovibile può essere ben inferiore a quello delle unità a disco fisso.

Nelle Figure 14.13, 14.14 e 14.15 sono mostrate rispettivamente le tendenze dei prezzi al megabyte della memoria DRAM, delle unità a disco e delle unità a nastro. I prezzi riportati in questi grafici sono i più bassi riscontrati fra tutte le inserzioni pubblicitarie apparse nelle riviste *BYTE* e *PC Magazine* durante il corso di ogni anno. Essi riflettono le caratteristiche del mercato dei piccoli calcolatori, cioè quello cui si rivolgono i lettori della rivista; i prezzi sono più bassi rispetto al mercato dei minicalcolatori e dei mainframe.

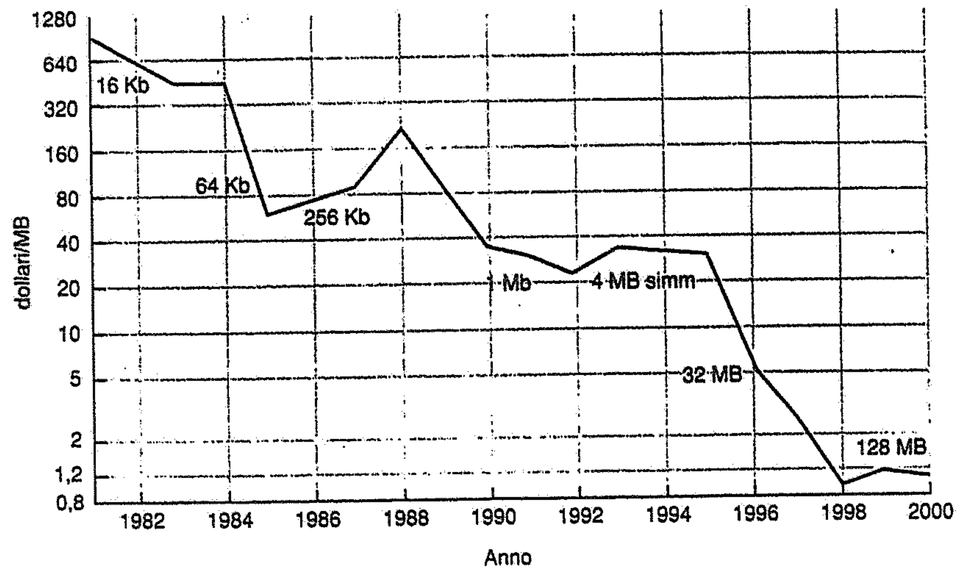


Figura 14.13 Prezzo al MB della memoria DRAM, dal 1981 al 2000.

Nel caso dei nastri, il prezzo si riferisce a un'unità dotata di un solo nastro; poiché il costo di un nastro è in genere una piccola frazione del costo di un'unità, il costo totale della memorizzazione nei nastri diviene progressivamente molto più basso in seguito all'acquisto di nuovi nastri da usare in una stessa unità. Infatti in un grande archivio automatico contenente migliaia di nastri, il costo di memorizzazione è dominato dal costo dei nastri. Nel corso del 2001 il costo al GB dei nastri era di circa 2 dollari.

Il costo della DRAM ha subito notevoli fluttuazioni: fra il 1981 e il 1996 si possono notare tre crolli del prezzo (attorno al 1981, al 1989 e al 1996) dovuti a una sovrapproduzione che causò la saturazione del mercato; si notano anche due periodi (attorno al 1987 e al 1993) durante i quali la scarsità d'offerta ha provocato un notevole aumento dei prezzi. Per quel che riguarda le unità a disco, invece, la diminuzione dei prezzi è stata molto più regolare, anche se dal 1992 la corsa al ribasso ha avuto un'accelerazione. I prezzi delle unità a nastro sono scesi con regolarità fino al 1997. In seguito il prezzo al GB delle unità a nastro economiche ha cessato di precipitare, sebbene i prezzi dei prodotti di tecnologie di medio livello, come i DAT/DDS, siano continuati a scendere, e attualmente siano vicini alle unità economiche. I prezzi delle unità a nastro sono indicati a partire dal 1984, perché la rivista *BYTE* si occupa del mercato dei piccoli calcolatori, dei quali le unità a nastro non erano un accessorio comune prima di questa data.

Confrontando questi grafici si nota che il prezzo della memorizzazione nei dischi è precipitato rispetto al prezzo delle memorie DRAM e dei nastri.

http://www.pdfdp.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

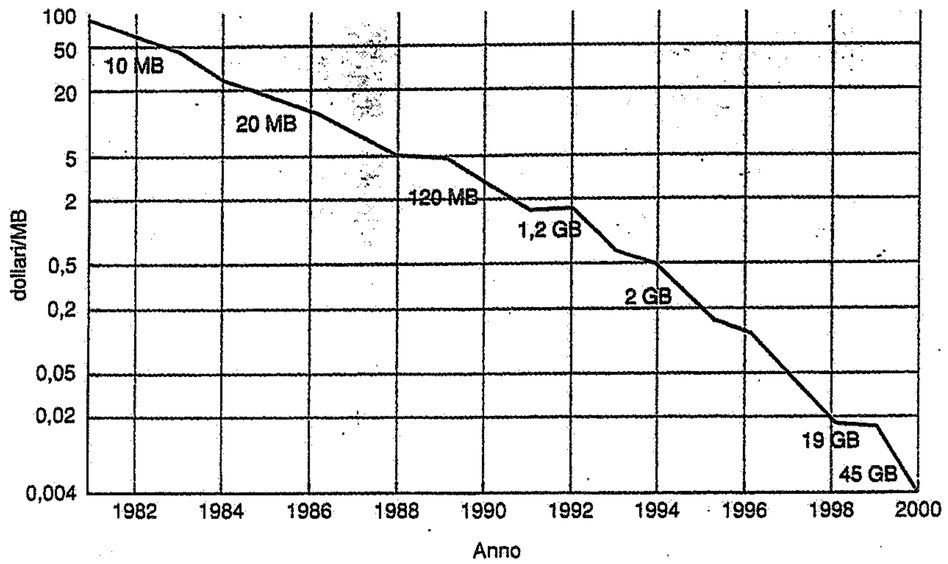


Figura 14.14 Prezzo al MB delle unità a disco magnetico, dal 1981 al 2000.

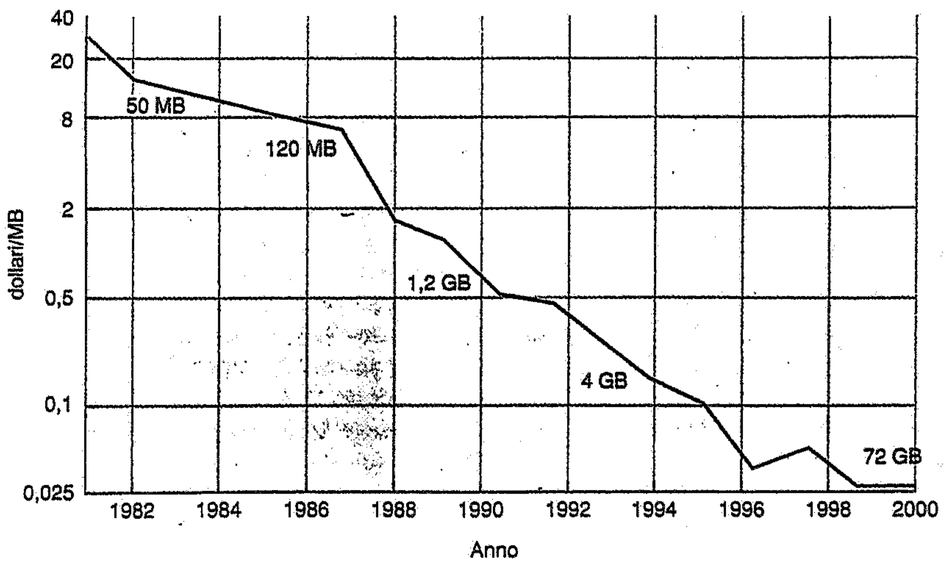


Figura 14.15 Prezzo al MB delle unità a nastro, dal 1984 al 2000.

Negli ultimi due decenni il prezzo al megabyte dei dischi magnetici si è ridotto di oltre quattro ordini di grandezza, mentre il prezzo corrispondente per gli elementi di memoria centrale è sceso di tre ordini di grandezza. La memoria centrale è attualmente 100 volte più costosa dei dischi.

Il prezzo al megabyte delle unità a dischi magnetici si avvicina al prezzo dei nastri magnetici, escludendo il costo delle unità a nastro. Ne segue che gli archivi di nastri medi e piccoli hanno un costo di memorizzazione dei dati maggiore di un sistema a dischi di corrispondente capacità. Il crollo dei prezzi dei dischi magnetici ha reso in gran parte obsoleta la memorizzazione terziaria: non si dispone più di alcuna tecnologia di memorizzazione terziaria che sia di più ordini di grandezza più economica dei dischi magnetici. Sembra che una ripresa della memorizzazione terziaria debba attendere un importante passo avanti in una tecnologia innovativa. Nel frattempo, l'uso dei nastri magnetici resterà per lo più limitato a scopi come la creazione di copie di riserva di dischi e di archivi composti di enormi quantità di nastri, che superano largamente l'effettiva capacità di memorizzazione delle grandi batterie di dischi magnetici.

14.9 Sommario

Per la maggior parte dei calcolatori le unità a dischi sono il principale dispositivo di I/O di memoria secondaria. Le richieste di I/O sui dischi sono generate sia dal file system sia dai sistemi di memoria virtuale, e ognuna di esse specifica l'indirizzo cui fare riferimento nel disco sotto forma di numero di un blocco logico.

Gli algoritmi di scheduling per i dischi possono aumentare l'ampiezza di banda, e ridurre il tempo di risposta medio e la variabilità del tempo di risposta. Algoritmi come l'SSTF, SCAN, C-SCAN, LOOK e C-LOOK sono progettati per realizzare questi miglioramenti tramite criteri di ordinamento della coda di richieste di operazioni sui dischi.

Le prestazioni possono essere influenzate negativamente dalla frammentazione esterna. Alcuni sistemi includono programmi rivolti a questo problema che esaminano l'intero file system alla ricerca di file frammentati, e spostano i blocchi tentando di ridurne la frammentazione. La deframmentazione di un file system può portare a un notevole incremento delle prestazioni, almeno quando esso si trova in avanzato stato di frammentazione; anche se, però, le prestazioni del sistema durante tale operazione si riducono. File system raffinati, come il Fast File System dello UNIX, adottano molte tecniche per controllare la frammentazione durante l'assegnazione dello spazio, rendendo superflua la riorganizzazione del disco.

Il sistema operativo gestisce i blocchi di un disco: innanzi tutto, si deve formattare fisicamente il disco per creare i settori direttamente sui suoi piatti — i dischi nuovi sono generalmente venduti già formattati; in seguito, il disco può essere diviso in partizioni, si può creare il file system, e si possono assegnare i blocchi d'avviamento che conterranno il programma d'avviamento del sistema; infine, quando un blocco diviene difettoso, il sistema deve essere in grado di isolarlo o di sostituirlo logicamente con un blocco di riserva.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Poiché un'area d'avvicendamento efficiente è essenziale per un sistema dalle buone prestazioni, molti sistemi usano un accesso di basso livello per l'I/O di paginazione. Certi sistemi riservano all'area d'avvicendamento una partizione di basso livello, altri impiegano un ordinario file all'interno del file system. Altri sistemi lasciano la scelta fra queste due possibilità all'utente o all'amministratore del sistema.

Il concetto di registrazione con scrittura anticipata richiede la disponibilità di una memoria stabile, e la realizzazione di quest'ultima passa attraverso la replicazione delle informazioni necessarie in più dispositivi di memorizzazione non volatile, di solito dischi, dotati di modi di malfunzionamento indipendenti. È inoltre necessario aggiornare le informazioni in maniera controllata allo scopo di assicurare la possibilità di recuperare i dati dopo un eventuale guasto, anche nel caso in cui quest'ultimo avvenga durante una procedura di ripristino dei dati stessi.

A causa della quantità di spazio per la registrazione dei dati richiesta nei grandi sistemi, i dischi sono spesso resi ridondanti tramite algoritmi RAID. Questi algoritmi permettono l'uso di più dischi per una data operazione, e consentono la prosecuzione del funzionamento del sistema e anche il ripristino automatico dei dati a fronte del guasto di un disco. Gli algoritmi RAID sono classificati in livelli che offrono diverse combinazioni di affidabilità ed elevate velocità di trasferimento.

I dischi si possono collegare a un sistema di calcolo in due modi: usando le porte locali di I/O oppure usando una connessione di rete, come una rete di memoria secondaria (SAN).

La memoria terziaria è realizzata da unità a nastro o a disco capaci di operare con mezzi rimovibili. Le tecnologie disponibili sono molte; alcune di esse sono i nastri magnetici, i dischi rimovibili magnetici e magneto-ottici e i dischi ottici.

Nel caso dei dischi rimovibili il sistema operativo fornisce in genere tutti i servizi di un file system, compresa la gestione dello spazio e lo scheduling delle richieste di I/O. Per molti sistemi operativi il nome di un file residente in un mezzo rimovibile è una combinazione del nome dell'unità e del nome di un file contenuto in quell'unità; questa convenzione è più semplice, ma anche potenzialmente più ambigua, dell'uso di un numero di serie per l'identificazione di una specifica cartuccia.

Nel caso dei nastri il sistema operativo in genere fornisce semplicemente un'interfaccia a basso livello. Molti sistemi operativi, inoltre, non incorporano metodi specifici per la gestione dei juke-box: questi ultimi devono essere controllati da un driver di dispositivo apposito o da un'applicazione che si occupa della creazione delle copie di riserva o della HSM.

Tre importanti fattori delle prestazioni sono l'ampiezza di banda, la latenza e l'affidabilità. Esistono unità a dischi e nastri con un'ampia gamma di diverse ampiezze di banda, ma la latenza d'accesso dei nastri è generalmente molto più elevata di quella dei dischi. Anche lo scambio di cartucce eseguito da un juke-box è relativamente lento; visto poi che per un juke-box il rapporto fra unità e cartucce è basso, la lettura di una gran parte dei dati archiviati può richiedere un tempo molto lungo. I dischi ottici sono in generale più affidabili di quelli magnetici, perché nei primi lo strato fisico contenente le informazioni è protetto da due strati di materiale trasparente, mentre nei secondi il materiale magnetico è maggiormente esposto a eventuali danni.

14.10 Esercizi

14.1 Eccetto l'FCFS, nessuno tra i criteri di scheduling del disco descritti è veramente *equo* (si può avere un blocco indefinito).

a) Spiegate perché questa affermazione è vera.
b) Descrivete una maniera di modificare gli algoritmi come lo SCAN cosicché risultino equi.
c) Spiegate perché l'equità è un obiettivo importante in un sistema a partizione del tempo.
d) Date tre o più esempi di circostanze in cui è importante che il sistema operativo sia iniquo nel servire le richieste di I/O.

14.2 Supponete che un'unità a disco abbia 5000 cilindri numerati da 0 a 4999. L'unità serve attualmente una richiesta relativa al cilindro 143, e la richiesta precedente era relativa al cilindro 125. La coda di richieste inevase, in ordine FIFO, è composta di richieste riguardanti i cilindri

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

Assumendo come punto di partenza la posizione attuale della testina, calcolate la distanza totale (in cilindri) che il braccio del disco percorre per soddisfare tutte le richieste inevase usando i seguenti algoritmi di scheduling:

a) FCFS
b) SSTF
c) SCAN
d) LOOK
e) C-SCAN
f) C-LOOK

14.3 È noto dalla fisica elementare che quando un oggetto è sottoposto a un'accelerazione costante a, la relazione fra la distanza d e il tempo t è data da $d = 1/2\ at^2$. Supponete che l'unità a disco dell'Esercizio 14.2, durante la ricerca di un cilindro, imprima al braccio del disco un'accelerazione costante per la prima metà del tragitto richiesto dalla ricerca, e imprima invece una decelerazione costante e della stessa intensità dell'accelerazione precedente per la seconda metà del tragitto. Ipotizzate che l'unità possa portare a termine la ricerca di un cilindro adiacente in 1 millisecondo, e una ricerca a tutto raggio lungo i 5000 cilindri in 18 millisecondi.

a) La distanza di una ricerca è il numero di cilindri attraverso i quali la testina deve passare. Spiegate perché il tempo di ricerca è proporzionale alla radice quadrata della distanza percorsa.
b) Scrivete il tempo di ricerca in funzione della distanza da percorrere. L'equazione dovrebbe avere la forma $t = x + y\sqrt{l}$, dove t è il tempo in millisecondi ed l è la distanza da percorrere in cilindri.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

c) Calcolate il tempo totale di ricerca per ogni algoritmo di scheduling dell'Esercizio 14.2 relativamente alla coda di richieste lì descritta. Determinate quale algoritmo è più veloce, cioè implica un tempo di ricerca totale minore.

d) L'*aumento percentuale di velocità* è il tempo risparmiato diviso il tempo originariamente necessario. Calcolate l'aumento percentuale di velocità dell'algoritmo più veloce rispetto all'FCFS.

14.4 Supponete che il disco dell'Esercizio 14.3 ruoti alla velocità di 7200 giri al minuto.

a) Calcolate la latenza di rotazione media di quest'unità a disco.

b) Dite quale distanza di ricerca è possibile coprire nel tempo calcolato al punto a).

14.5 La ricerca uniformemente accelerata descritta nell'Esercizio 14.3 è tipica delle unità a disco. Per contro, le unità a dischetti (e molte unità a disco costruite prima della metà degli anni Ottanta) di solito eseguono ricerche a velocità costante. Supponete che l'unità a disco dell'Esercizio 14.3 esegua le ricerche a velocità costante invece che ad accelerazione costante, in modo che il tempo di ricerca abbia la forma $t = x + yl$, dove t è il tempo in millisecondi ed l è la distanza da percorrere. Supponete che il tempo di ricerca relativo a un cilindro adiacente sia, come prima, di 1 millisecondo, e che il tempo ulteriore necessario per ogni cilindro aggiuntivo sia di 0,5 millisecondi.

a) Scrivete il tempo di ricerca come funzione della distanza di ricerca.

b) Sfruttando il risultato ottenuto al punto a) calcolate il tempo totale di ricerca per ogni algoritmo di scheduling dell'Esercizio 14.2, relativamente alla coda di richieste lì descritta. Confrontate la soluzione con i valori trovati al punto c) dell'Esercizio 14.3. Spiegate perché si ottengono tali risultati.

c) Calcolate anche in questo caso l'aumento percentuale di velocità dell'algoritmo più veloce rispetto all'FCFS.

14.6 Scrivete, in Java, un programma per lo scheduling delle richieste di I/O per unità a disco usando gli algoritmi SCAN e C-SCAN.

14.7 Confrontate le prestazioni di SCAN e C-SCAN assumendo una distribuzione uniforme delle richieste di I/O. Considerate il tempo di risposta medio (cioè il tempo che intercorre fra l'arrivo di una richiesta e il completamento dell'operazione a essa associata), la variazione del tempo di risposta, e l'effettiva ampiezza di banda. Analizzate come le prestazioni dipendono dalle dimensioni relative del tempo di ricerca e della latenza di rotazione.

14.8 Dite se in un sistema monoutente è utile adottare un algoritmo di scheduling diverso dall'FCFS. Spiegate la risposta.

14.9 Spiegate perché lo scheduling SSTF tende a favorire i cilindri centrali rispetto ai cilindri più interni e più esterni.

14.10 Le richieste non sono di solito uniformemente distribuite: ad esempio ci si può aspettare che un cilindro che contiene FAT o inode del file system sia visitato più spesso di un cilindro che contiene solo file. Supponete di sapere che il 50 per cento delle richieste è relativo a un piccolo numero fissato di cilindri.

a) Dite se uno fra gli algoritmi discussi in questo capitolo è particolarmente adatto a questa circostanza. Spiegate la risposta.

b) Proponete un algoritmo di scheduling che offra prestazioni anche migliori sfruttando questo 'punto caldo' del disco.

c) Di solito i file system localizzano i blocchi di dati usando una tabella indiretta, come la FAT nell'MS-DOS o gli inode nello UNIX. Descrivete uno o più modi di sfruttare queste tabelle per migliorare le prestazioni delle unità a disco.

14.11 Spiegate perché di solito s'ignora la latenza di rotazione nelle considerazioni riguardanti lo scheduling del disco. Dite come modifichereste SSTF, SCAN e C-SCAN per tentare di ottimizzare anche la latenza di rotazione.

14.12 Spiegate come la presenza di un disco RAM influenzerebbe la vostra scelta di un algoritmo di scheduling, elencando i fattori di cui bisogna tener conto. Posto che il file system memorizzi i blocchi usati di recente in una *buffer cache* della memoria centrale, discutete l'applicabilità di queste considerazioni allo scheduling di un disco.

14.13 Dite perché in un ambiente multitasking è importante cercare di bilanciare l'I/O del file system tra i dischi e i controllori del sistema.

14.14 Confrontate la rilettura delle pagine di codice dal file system e l'uso di un'area d'avvicendamento per memorizzarle. Discutete vantaggi e svantaggi delle due soluzioni.

14.15 Dite se esiste un modo per realizzare una memoria veramente stabile. Spiegate la risposta.

14.16 L'affidabilità di un'unità a disco generalmente si quantifica usando il **tempo medio fra due guasti** (*mean time between failures* — MTBF); sebbene sia chiamata 'tempo', questa quantità in effetti si misura in ore di funzionamento d'unità per guasto.

a) Dato un gruppo di 1000 unità a disco, ognuna delle quali ha un MTBF di 750.000 ore, dite con quale frequenza avverrà il guasto di un'unità del gruppo, scegliendo fra le seguenti possibilità quella che si adatta meglio alla situazione descritta: una volta ogni mille anni, una volta ogni cento anni, una volta ogni dieci anni, una volta al mese, una volta la settimana, una volta al giorno, una volta ogni ora, una volta al minuto, una volta al secondo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

b) Le statistiche di mortalità indicano che in media un individuo residente negli Stati Uniti d'America ha circa 1 probabilità su 1000 di morire fra i 20 e i 21 anni d'età. Deducete l'MTBF di un ventenne, e convertite il risultato da ore in anni. Spiegate cosa dice questo MTBF sulle aspettative di vita di un ventenne.

c) Un produttore asserisce che un certo modello di unità a disco ha un MTBF di 1 milione di ore. Dite cosa si può concludere circa il numero di anni per cui una di queste unità a disco è coperta dalla garanzia.

14.17 La locuzione *fast wide SCSI-II* denota un bus SCSI capace di trasferire pacchetti di byte fra un dispositivo e una macchina alla velocità di 20 MB di dati al secondo. Supponete che un'unità a disco con bus *fast wide SCSI-II* abbia una velocità di rotazione di 7200 giri al minuto, settori di 512 byte e contenga 160 settori per traccia.

a) Stimate la velocità media di trasferimento, in megabyte al secondo, di quest'unità, nel caso del trasferimento di una grande quantità di dati.

b) Supponete che il disco abbia 7000 cilindri, 20 tracce per cilindro, un tempo di attivazione della testina relativa alla facciata del piatto contenente il settore coinvolto nel trasferimento di 0,5 millisecondi, e un tempo di ricerca relativo a un cilindro adiacente di 2 millisecondi. Usate queste informazioni aggiuntive per fare una stima accurata della velocità media di trasferimento per una quantità di dati enorme.

c) Supponete che il tempo medio di ricerca per quest'unità sia di 8 millisecondi. Stimate il numero di operazioni di I/O al secondo e l'effettiva velocità di trasferimento per un carico di lavoro di accessi diretti che leggano singoli settori sparsi per tutto il disco.

d) Calcolate la velocità di trasferimento e il numero al secondo degli accessi diretti di I/O nel caso di richieste di trasferimenti di 4 KB, 8 KB e 64 KB rispettivamente.

e) Se ci sono molte richieste inevase in coda, un algoritmo come SCAN dovrebbe riuscire a ridurre la distanza media di ricerca. Supponete che un carico di lavoro di accessi diretti richieda la lettura di pagine di 8 KB, che la lunghezza media della coda sia 10, e che l'algoritmo di scheduling riduca il tempo medio di ricerca a 3 millisecondi. Calcolate il numero di I/O al secondo e l'effettiva velocità di trasferimento dell'unità.

14.18 È possibile collegare più di un'unità a disco a uno stesso bus SCSI; nel caso di un bus *fast wide SCSI-II* (Esercizio 14.17), in particolare, si possono collegare fino a 15 unità a disco. Si ricorda che questo bus ha un'ampiezza di banda di 20 MB al secondo. A ogni fissato istante solo un pacchetto può viaggiare sul bus fra la cache interna di un'unità e la macchina, ma le altre unità possono contemporaneamente spostare i loro bracci; inoltre, un'unità può trasferire dati dai suoi dischi magnetici alla sua cache interna mentre il bus SCSI-II è occupato da un'altra unità. Tenendo presenti le velocità di trasferimento che avete calcolato per i diversi carichi di lavoro nell'Esercizio 14.17, discutete il numero delle unità a disco che si possono efficacemente collegare a un unico bus *fast wide SCSI-II*.

14.19 La sostituzione dei settori difettosi tramite l'accantonamento o la traslazione dei settori può influenzare le prestazioni del sistema. Supponete che l'unità a disco dell'Esercizio 14.17 contenga 100 settori difettosi dislocati casualmente, e che ogni settore difettoso sia sostituito con un settore di riserva posto su una diversa traccia ma sullo stesso cilindro. Stimate il numero di I/O al secondo e l'effettiva velocità di trasferimento per un carico di lavoro di accessi diretti consistente in richieste di lettura di 8 KB ognuna, con la lunghezza della coda delle richieste inevase pari a 1 (cosicché la scelta dell'algoritmo di scheduling diviene irrilevante). Traete conclusioni riguardo all'impatto dei settori difettosi sulle prestazioni del sistema.

14.20 Discutete i vantaggi e gli svantaggi relativi all'accantonamento dei settori e alla traslazione dei settori.

14.21 In genere il sistema operativo tratta i dischi rimovibili come un file system condiviso, ma assegna l'unità a nastro a una sola applicazione alla volta. Elencate tre motivi che spieghino questa disparità di trattamento. Descrivete le caratteristiche aggiuntive che il sistema operativo dovrebbe avere per poter gestire un juke-box di nastri come un file system condiviso. Dite se anche le applicazioni che condividono il juke-box devono avere particolari caratteristiche, o se possono usare i file come se risiedessero in un disco; giustificate la risposta.

14.22 Dite quale sarebbe l'effetto di un numero di file aperti superiore al numero delle unità di lettura e scrittura in un juke-box di dischi.

14.23 Dite che effetto avrebbe sul costo e sulle prestazioni una densità superficiale dei dati nei nastri pari a quella dei dischi.

14.24 Nell'ipotesi che i dischi magnetici raggiungano lo stesso costo per gigabyte dei nastri, dite se questi ultimi diverranno obsoleti o saranno ancora necessari. Spiegate la risposta.

14.25 In questo esercizio userete alcune semplici stime per confrontare il costo e le prestazioni dei due seguenti sistemi: un sistema di memorizzazione con una capacità dell'ordine del terabyte interamente costituito di dischi; e un altro sistema che sfrutta la memoria terziaria. Supponete che ogni disco magnetico abbia una capacità di 10 GB, un costo di 1000 dollari, una velocità di trasferimento di 5 MB al secondo, e una latenza d'accesso media di 15 millisecondi. Supponete inoltre che un archivio di nastri costi 10 dollari per gigabyte, trasferisca 10 MB di dati al secondo e abbia una latenza d'accesso media di 20 secondi. Calcolate il costo totale, la massima velocità totale di trasferimento e il tempo d'attesa medio per il sistema a soli dischi. Se fate qualche ipotesi sul carico di lavoro, descrivetele e giustificatele. Supponete adesso che il 5 per cento dei dati sia usato di frequente e risieda quindi in dischi, ma che il restante 95 per cento sia memorizzato nell'archivio di nastri; supponete inoltre che il sistema dei dischi gestisca il 95 per cento delle richieste, e l'archivio di nastri il restante 5 per cento. Calcolate il costo totale, la massima velocità totale di trasferimento e il tempo d'attesa medio per questo sistema di gestione gerarchica della memoria.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

14.26 Si dice a volte che il nastro sia un mezzo ad accesso sequenziale, mentre il disco magnetico sia ad accesso diretto. In effetti, l'adattabilità di un dispositivo di memorizzazione all'accesso diretto dipende dalla dimensione dei trasferimenti. Il termine *velocità di trasferimento a regime* denota la velocità alla quale si esegue un trasferimento dopo che esso ha avuto inizio, non considerando la latenza d'accesso. Per contro, la *velocità di trasferimento effettiva* è il rapporto fra il numero totale di byte trasferiti e il numero totale di secondi impiegati, compresi i fattori di ritardo come la latenza d'accesso.

Supponete che in un certo calcolatore la cache di secondo livello abbia una latenza d'accesso di 8 nanosecondi e una velocità di trasferimento a regime di 800 MB al secondo; che la memoria centrale abbia una latenza d'accesso di 60 nanosecondi e una velocità di trasferimento a regime di 80 MB al secondo; che l'unità a disco abbia una latenza d'accesso di 15 millisecondi e una velocità di trasferimento a regime di 5 MB al secondo; e infine che l'unità a nastro abbia una latenza d'accesso di 60 secondi e una velocità di trasferimento a regime di 2 MB al secondo.

a) L'accesso diretto causa la diminuzione della velocità di trasferimento effettiva di un dispositivo, perché non c'è trasferimento di dati durante il tempo impiegato per l'accesso. Considerando l'unità a disco sopra descritta, dite qual è la velocità di trasferimento effettiva nel caso in cui un trasferimento continuo rispettivamente di 512 byte, 8 KB, 1 MB e 16 MB segua un accesso medio.
b) L'utilizzo di un dispositivo è il rapporto fra la velocità di trasferimento effettiva e la velocità di trasferimento a regime. Calcolate l'utilizzo dell'unità a disco nel caso di accessi diretti con i trasferimenti continui elencati al punto a).
c) Supponete che un utilizzo maggiore o uguale al 25 per cento sia considerato accettabile. Usate i dati forniti per calcolare la minima dimensione di trasferimento per l'unità a disco che dia un utilizzo accettabile.
d) Completate la frase seguente: Un'unità a disco è un dispositivo ad accesso diretto per trasferimenti di dimensioni maggiori di _____ byte, e un dispositivo ad accesso sequenziale per trasferimenti di dimensioni minori.
e) Calcolate le minime dimensioni di trasferimento che dànno un utilizzo accettabile per la cache, la memoria centrale e l'unità a nastro.
f) Dite quando un'unità a nastro si deve considerare un dispositivo ad accesso diretto e quando ad accesso sequenziale.

14.27 Immaginate che qualcuno inventi un dispositivo di memorizzazione olografica. Supponete che il suo costo sia di 10.000 dollari, il tempo medio d'accesso di 40 millisecondi e che impieghi cartucce da 100 dollari della dimensione di un CD. Ogni cartuccia può contenere 40.000 immagini, e ciascuna immagine è un quadrato in bianco e nero composto di 6000 × 6000 pixel (ogni pixel memorizza il valore di un bit). Supponete infine che il dispositivo possa leggere o scrivere un'immagine in 1 millisecondo. Discutete i seguenti punti:

a) i possibili usi di un tale dispositivo;
b) l'impatto di un tale dispositivo sulle prestazioni dell'I/O di un sistema;
c) l'eventuale possibilità che altri dispositivi di memorizzazione divengano obsoleti.

14.28 Supponete che un disco ottico di 5,25 pollici abbia una densità superficiale di dati di 1 gigabit per pollice quadrato (si noti che il disco ottico sfrutta un solo lato per la registrazione dei dati). Supponete inoltre che un nastro magnetico abbia una densità superficiale di dati pari a 20 megabit per pollice quadrato, sia largo 1/2 pollice e lungo 1800 piedi. Stimate la capacità di memorizzazione dei due mezzi. Ora immaginate l'esistenza di un nastro ottico delle stesse dimensioni fisiche del nastro magnetico descritto, ma con la stessa densità superficiale di dati del disco ottico. Calcolate la quantità di dati memorizzabile nel nastro ottico; posto che il nastro magnetico descritto costi 25 dollari, fissate un prezzo di mercato per il nastro ottico.

14.29 Supponete di convenire che 1 KB sia pari a 1024 byte, 1 MB a 1024^2 byte e 1 GB a 1024^3 byte; questa progressione continua per i terabyte, i petabyte e gli esabyte (1024^6 byte). Alcuni progetti scientifici proposti recentemente richiederanno capacità di registrazione di qualche esabyte di dati per le ricerche dei prossimi decenni. Per rispondere alle seguenti domande dovrete fare delle ipotesi ragionevoli e renderle esplicite:

a) dite quanti dischi sarebbero necessari per memorizzare 4 esabyte di dati;
b) dite quanti nastri sarebbero necessari per memorizzare 4 esabyte di dati;
c) dite quanti nastri ottici sarebbero necessari per memorizzare 4 esabyte di dati (si veda l'Esercizio 14.28);
d) dite quante cartucce a memorizzazione olografica sarebbero necessarie per memorizzare 4 esabyte di dati (si veda l'Esercizio 14.27);
e) calcolate il volume dello spazio d'archiviazione richiesto dalle scelte relative ai punti a)-d).

14.30 Discutete i modi in cui un sistema operativo potrebbe mantenere una lista dello spazio libero relativa a un file system residente in un nastro. Supponete che il dispositivo a nastro permetta il solo accodamento, e che usi il simbolo EOT e le operazioni `locate`, `space` e `read position` descritte nel Paragrafo 14.8.2.1.

14.11 Note bibliografiche

Le batterie ridondanti di dischi (RAID) sono discusse in [Patterson et al. 1988] e nella dettagliata rassegna di [Chen et al. 1994]. Le architetture delle unità a disco per elaborazioni ad alte prestazioni sono discusse da [Katz et al. 1989]. [Teorey e Pinkerton 1972] presentano una delle prime analisi comparate degli algoritmi di scheduling del disco; usano un modello di unità a disco con tempo di ricerca lineare rispetto al numero di

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

cilindri attraversati. Per questo tipo di disco l'algoritmo LOOK è una buona scelta per code di richieste inevase lunghe meno di 140 elementi, mentre l'algoritmo C-LOOK lo è per code che superano i 100 elementi. [King 1990] discute alcuni modi per ridurre il tempo di ricerca, basati su spostamenti del braccio fatti quando l'unità a disco sarebbe altrimenti inattiva. [Seltzer et al 1990] descrive algoritmi di scheduling che, oltre al tempo di ricerca, considerano la latenza di rotazione. [Worthington et al. 1994] discute le prestazioni dei dischi, e mostra l'effetto sulle prestazioni della gestione dei blocchi difettosi. Lo sfruttamento dell'esistenza di un 'punto caldo' al fine di ridurre i tempi di ricerca è stato considerato da [Ruemmler e Wilkes 1991] e [Akyurek e Salem 1993]. [Ruemmler e Wilkes 1994] descrive un accurato modello delle prestazioni di una moderna unità a disco. [Worthington et al. 1995] spiega come determinare caratteristiche del disco di basso livello quali la struttura delle aree all'interno dei cilindri; questo lavoro è ulteriormente sviluppato in [Schindler e Gregory 1999].

La dimensione dei trasferimenti richiesti e la casualità del carico di lavoro hanno un'influenza considerevole sulle prestazioni dell'unità a disco. [Ousterhout et al. 1985], e [Ruemmler e Wilkes 1993] riportano numerose interessanti caratteristiche dei carichi di lavoro, ad esempio che la maggior parte dei file sono brevi, la maggior parte dei file creati di recente viene eliminata assai presto, nella maggior parte dei casi i file aperti per lettura sono letti in modo sequenziale nella loro interezza, e che nella maggior parte dei casi le distanze di ricerca sono brevi. [McKusick et al. 1984] descrivono il Berkeley Fast File System (FFS), che adotta molte tecniche raffinate aventi lo scopo di ottenere buone prestazioni per parecchi tipi di carichi di lavoro. [McVoy e Kleiman 1991] discute ulteriori miglioramenti dell'FFS originario. [Quinlan 1991] discute la realizzazione di un file system per mezzi WORM con cache su dischi magnetici; [Richards 1990] discute un approccio alla memoria terziaria basato sul concetto di file system. [Maher et al. 1994] traccia una panoramica sull'integrazione di file system distribuiti e memoria terziaria.

Il concetto di gestione gerarchica della memoria è studiato da più di un quarto di secolo: l'articolo [Mattson et al. 1970], ad esempio, descrive un metodo matematico per prevedere le prestazioni di un sistema di gestione gerarchica della memoria. [Alt 1993] descrive l'integrazione della memoria rimovibile in un sistema operativo commerciale, e [Miller e Katz 1993] descrive le caratteristiche dell'accesso alla memoria terziaria in un ambiente basato su un supercalcolatore. [Benjamin 1990] fornisce una panoramica riguardante i problemi connessi alla necessità di un vastissimo spazio di archiviazione per il progetto EOSDIS della NASA.

La tecnologia della memorizzazione olografica è l'argomento dell'articolo [Psaltis e Mok 1995]; una raccolta di articoli su quest'argomento, a partire dal 1963, è stata curata da [Sincerbox 1994]. [Asthana e Finkelstein 1995] descrive diverse tecnologie di memorizzazione emergenti, compresa la memoria olografica, i nastri ottici e tecniche basate sul confinamento dell'elettrone. [Toigo 2000] offre un'approfondita descrizione delle moderne tecnologie dei dischi e di diverse future potenziali tecniche di registrazione dei dati.

Parte quinta

Sistemi distribuiti

Strutture dei sistemi distribuiti
File system distribuiti
Coordinazione distribuita

Un sistema distribuito è un insieme di unità d'elaborazione che non condividono la memoria o un clock. Ogni unità d'elaborazione ha invece la propria memoria locale e la comunicazione avviene per mezzo di diverse linee di comunicazione, come reti locali o geografiche. Le unità d'elaborazione di un sistema distribuito variano per dimensione e funzioni. I sistemi distribuiti possono comprendere infatti piccoli palmari o dispositivi d'elaborazione in tempo reale, PC, stazioni di lavoro, e grandi mainframe.

I vantaggi di un sistema distribuito comprendono l'accesso degli utenti alle risorse del sistema, e perciò l'accelerazione dei calcoli e una maggiore disponibilità e affidabilità dei dati. Un file system distribuito è un sistema di gestione dei file i cui utenti, server e dispositivi di memorizzazione sono dispersi tra i diversi siti di un sistema distribuito. Di conseguenza, l'attività di servizio si deve svolgere attraverso la rete; al posto di un unico magazzino centralizzato dei dati sono presenti molti dispositivi di memorizzazione indipendenti.

Un sistema distribuito deve inoltre offrire meccanismi per la sincronizzazione e la comunicazione tra processi, e per la gestione delle situazioni di stallo e dei guasti che non si verificano nei sistemi centralizzati.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 15

Strutture dei sistemi distribuiti

Un sistema distribuito è un insieme di unità d'elaborazione che non condividono la memoria o un clock, ma ogni unità d'elaborazione ha la propria memoria locale. La comunicazione avviene tramite diversi tipi di mezzi, come bus ad alta velocità o linee telefoniche. In questo capitolo si tratta la struttura generale dei sistemi distribuiti e delle reti che li interconnettono, e si evidenziano le principali differenze di progettazione fra tali sistemi e i sistemi centralizzati. I sistemi distribuiti sono trattati dettagliatamente nei Capitoli 16 e 17.

15.1 Introduzione

Un sistema distribuito è un insieme di unità d'elaborazione debolmente connesse tramite una **rete di comunicazione**. Ciascuna unità d'elaborazione di un sistema distribuito considera **remote** le altre unità d'elaborazione del sistema e le rispettive risorse, mentre considera locali le proprie risorse.

Le unità d'elaborazione presenti in un sistema distribuito possono variare per dimensioni e funzioni; possono comprendere piccole unità d'elaborazione, stazioni di lavoro, minicalcolatori e grandi calcolatori d'uso generale. Sono chiamate in molti modi: *siti*, *nodi*, *calcolatori*, *macchine*; secondo il contesto in cui sono citate. In questo testo si usa principalmente il termine *sito* per indicare una locazione di macchine, e *calcolatore* o *macchina* per riferirsi a uno specifico sistema in un sito. In genere un calcolatore in un sito, il *server*, dispone di una risorsa che un altro calcolatore in un altro sito, il *client* (o utente), vuole usare. Lo scopo del sistema distribuito è quello di fornire un ambiente efficiente e conveniente per tale condivisione delle risorse. Un sistema distribuito è illustrato nella Figura 15.1.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

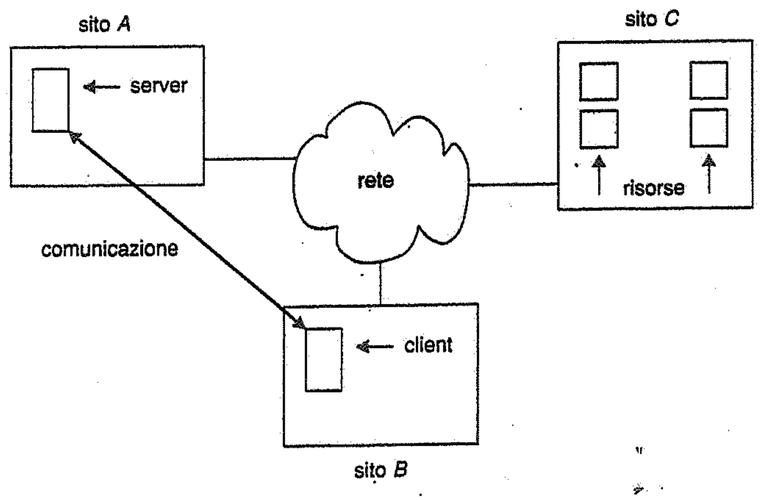


Figura 15.1 Sistema distribuito.

15.1.1 Vantaggi dei sistemi distribuiti

I quattro motivi principali per costruire sistemi distribuiti sono: la *condivisione delle risorse*, l'*accelerazione dei calcoli*, l'*affidabilità* e la *comunicazione*.

15.1.1.1 Condivisione delle risorse

Se siti diversi, con risorse diverse, sono collegati tra loro, allora l'utente di un sito può avere la possibilità di usare le risorse disponibili in un altro sito. Ad esempio, un utente del sito *A* può usare una stampante presente nel sito *B*. Nel frattempo, un utente del sito *B* può accedere a un file che risiede nel sito *A*. In generale, la **condivisione delle risorse** di un sistema distribuito offre meccanismi per la condivisione dei file in siti remoti, per l'elaborazione delle informazioni in una base di dati distribuita, per la stampa dei file in siti remoti, per l'uso di dispositivi specializzati remoti, come un elaboratore matriciale (*array processor*) ad alte prestazioni, e per eseguire altre operazioni.

15.1.1.2 Accelerazione dei calcoli

Se un calcolo particolare si può dividere in sottocalcoli eseguibili in modo concorrente, un sistema distribuito permette di ripartire i sottocalcoli tra i diversi siti; i sottocalcoli si possono eseguire in modo concorrente determinando un'**accelerazione dei calcoli**. Inoltre, se un particolare sito è attualmente sovraccarico di lavori, alcuni di essi si possono spostare in altri siti con carico inferiore. Lo spostamento dei lavori d'elaborazione si chiama **condivisione del carico**. Una condivisione del carico automatica, in cui il sistema operativo distribuito sposta i lavori autonomamente, non è ancora comune nei sistemi commerciali.

15.1.1.3 Affidabilità

Se un sito di un sistema distribuito si guasta, i siti restanti possono continuare a funzionare, offrendo al sistema una maggiore affidabilità. Se il sistema distribuito è formato da più grandi sistemi di calcolo autonomi, vale a dire calcolatori d'uso generale, il guasto di uno di essi non influisce sugli altri sistemi di calcolo. Se invece il sistema è formato da piccoli calcolatori, ciascuno dei quali è responsabile di una funzione cruciale per il sistema (come l'I/O di caratteri su un terminale oppure il file system) allora un singolo guasto può fermare il funzionamento di tutto il sistema. In generale, con un opportuno livello di ridondanza (sia dei dispositivi sia dei dati) il sistema può continuare a funzionare anche se qualcuno dei siti si guasta.

Il guasto di un sito deve essere rilevato dal sistema, e per superare tale situazione si devono intraprendere azioni adeguate. Il sistema non deve più servirsi dei servizi di quel sito; inoltre, se la funzione del sito guasto può essere rilevata da un altro sito, il sistema deve assicurare che il trasferimento della funzione avvenga correttamente. Infine, quando il sito guasto viene ripristinato o riparato devono essere disponibili meccanismi che lo reintegrino facilmente nel sistema. Come si vede nei Capitoli 16 e 17 queste operazioni presentano difficili problemi che hanno molte possibili soluzioni.

15.1.1.4 Comunicazione

Se più siti sono collegati per mezzo di una rete di comunicazione, gli utenti dei diversi siti hanno la possibilità di scambiarsi informazioni. A un livello basso, tra i sistemi si scambiano messaggi in modo simile a quello dei messaggi in un singolo calcolatore (Paragrafo 4.5). Disponendo dello scambio di messaggi, tutti i servizi di livello superiore disponibili nei sistemi centralizzati si possono estendere in modo da adattarsi al sistema distribuito. Tali funzioni comprendonò il trasferimento di file, le sessioni di lavoro remote, la posta elettronica e le chiamate di procedure remote (RPC).

Il vantaggio dato da un sistema distribuito consiste nel fatto che queste funzioni si possono eseguire su lunghe distanze. Due persone che lavorano in siti geograficamente diversi possono ad esempio collaborare a un progetto. Trasferendo i file del progetto, entrando nei rispettivi sistemi remoti per eseguire programmi e comunicando per coordinare il lavoro, gli utenti riducono al minimo i limiti impliciti in un lavoro a lunga distanza. Questo libro è stato scritto collaborando in tal modo.

I vantaggi dei sistemi distribuiti hanno determinato una tendenza, tra le industrie, verso la riduzione delle dimensioni dei calcolatori (*downsizing*): molte aziende sostituiscono i loro mainframe con reti di stazioni di lavoro o PC. Le aziende ottengono un miglior rapporto tra prezzo e prestazioni, una maggiore flessibilità nella dislocazione delle risorse, più servizi, migliori interfacce d'utente e una manutenzione più semplice.

Naturalmente un sistema operativo progettato come un insieme di processi comunicanti tramite un sistema a scambio di messaggi si può estendere a un sistema distribuito più facilmente di un sistema operativo privo dello scambio di messaggi. L'MS-DOS, ad esempio, non si integra facilmente in una rete poiché il suo nucleo è basato sulle interruzioni e non ha alcuna funzione d'ausilio allo scambio dei messaggi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

15.1.2 Tipi di sistemi operativi distribuiti

In questo paragrafo, si descrivono due categorie generali di sistemi operativi orientati alle reti: i sistemi operativi di rete e i sistemi operativi distribuiti. I sistemi operativi di rete sono più semplici da realizzare, ma spesso sono più difficili da usare dei sistemi operativi distribuiti, che offrono più servizi.

15.1.2.1 Sistemi operativi di rete

Un **sistema operativo di rete** offre un ambiente nel quale gli utenti, che sanno della presenza di più calcolatori, possono accedere alle risorse remote iniziando una sessione di lavoro sugli appropriati calcolatori remoti oppure trasferendo dati dai calcolatori remoti al proprio.

Sessioni di lavoro remote

Una funzione importante dei sistemi operativi di rete è consentire agli utenti di iniziare una sessione di lavoro a distanza in un altro calcolatore. A tale scopo la rete Internet fornisce la funzione `telnet`. Per illustrare questa funzione si supponga che un utente, alla Brown University, voglia compiere alcune elaborazioni servendosi di `cs.utexas.edu`, un calcolatore che si trova all'Università del Texas. Affinché ciò gli sia consentito l'utente deve risultare accreditato all'uso del calcolatore remoto. Per iniziare una sessione di lavoro remota l'utente usa il comando

```
telnet cs.utexas.edu
```

Questo comando fa sì che si realizzi una connessione tra il calcolatore locale nella Brown University e il calcolatore `cs.utexas.edu`. Dopo che tale connessione è stata stabilita, i programmi di gestione della rete creano un collegamento bidirezionale, trasparente, tale che tutti i caratteri inseriti dall'utente siano inviati a un processo in `cs.utexas.edu`, e tutti i dati emessi da questo processo siano rispediti all'utente. Il processo nel calcolatore remoto chiede all'utente il suo nome d'utente e la relativa parola d'ordine; ricevute le corrette informazioni, il processo agisce per conto dell'utente, che può compiere le proprie elaborazioni servendosi del calcolatore remoto come ogni utente locale.

Trasferimento di file remoti

Un'altra tra le funzioni principali di un sistema operativo di rete consiste nel fornire un meccanismo per il **trasferimento di file remoti** tra calcolatori. In un ambiente di questo tipo ogni calcolatore mantiene il proprio file system locale. Se un utente in un sito (ad esempio, `cs.brown.edu`) vuole accedere a un file che si trova in un altro calcolatore (ad esempio, `cs.utexas.edu`), si deve copiare esplicitamente il file dal calcolatore dell'Università del Texas al calcolatore della Brown University.

La rete Internet offre un meccanismo, il programma `ftp` (*file transfer protocol*), che permette di compiere tale trasferimento. Si supponga che un utente di `cs.brown.edu` voglia copiare il file `paper.tex`, che si trova in `cs.utexas.edu`, nel file locale `my-paper.tex`; l'utente deve prima invocare il programma `ftp`:

```
ftp cs.utexas.edu
```


Quindi il programma `ftp` chiede il nome d'utente e la relativa parola d'ordine; una volta che il programma ha ricevuto le corrette informazioni, l'utente deve connettersi alla directory in cui si trova il file `paper.tex` e copiare il file richiedendo l'esecuzione del comando

```
get paper.tex my-paper.tex
```

In questo schema la locazione del file non è trasparente per l'utente; gli utenti devono sapere esattamente dove si trova ogni file. Inoltre non esiste una reale condivisione di file, poiché un utente può solo *copiare* un file da un sito all'altro; possono quindi esistere più copie dello stesso file con conseguente spreco di spazio e, nel caso di differenti modifiche, di incoerenza tra le copie.

In questo esempio l'utente della Brown University deve avere il permesso d'accesso a `cs.utexas.edu`. Comunque l'`ftp` offre un'opzione che consente anche a un utente non accreditato nel calcolatore dell'Università del Texas di copiare file in modo remoto; questa possibilità si ottiene col metodo detto `FTP` **anonimo** che funziona come segue. Il file da copiare, cioè `paper.tex`, viene messo in una specifica directory, ad esempio `ftp`, con la protezione impostata in modo da consentire a chiunque la lettura del file. Un utente che vuole copiare il file usa il comando `ftp` come s'è appena descritto, fornendo il nome `anonymous` alla richiesta del nome d'utente e una parola d'ordine qualunque. Una volta che l'accesso anonimo è stato effettuato, si devono prendere precauzioni che assicurino che tale utente, parzialmente autorizzato, possa accedere soltanto ai file appropriati. Generalmente si consente l'accesso ai soli file presenti nell'albero di directory dell'utente `anonymous`; tutti i file in esso presenti sono accessibili a ogni utente anonimo, sottoposto all'usuale schema di protezione impiegato nel calcolatore. Gli utenti anonimi non possono comunque accedere ai file presenti fuori da tale albero di directory.

Il meccanismo `ftp` è realizzato in modo simile al `telnet`. Un demone nel sito remoto verifica le richieste di connessione alla porta per l'`ftp` del sistema; si convalida l'accesso e si consente all'utente di richiedere a distanza l'esecuzione dei comandi. Diversamente dal demone del `telnet`, che esegue per l'utente ogni comando, il demone dell'`ftp` risponde soltanto a un insieme predefinito di comandi riguardanti i file, tra i quali i seguenti:

- `get` trasferisce un file da un calcolatore remoto a uno locale;
- `put` trasferisce un file da un calcolatore locale a uno remoto;
- `ls` elenca i file presenti nella directory corrente nel calcolatore remoto.

Inoltre vari comandi consentono di cambiare i modi di trasferimento (per file binari o ASCII) e per determinare lo stato della connessione.

Sia il `telnet` sia l'`ftp` richiedono un cambio di paradigmi da parte dell'utente. Il programma `ftp` richiede che l'utente conosca un insieme di comandi completamente diverso dai normali comandi del sistema operativo. Il programma `telnet` richiede un mutamento minore: l'utente deve conoscere i comandi propri del sistema remoto; ad esempio, un utente di un calcolatore con sistema operativo UNIX che vuole collegarsi, tramite il `telnet`, a un calcolatore dotato del sistema operativo VMS deve passare ai relativi co-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

mandi per tutta la durata della sessione remota. I servizi che non richiedono l'uso di un diverso insieme di comandi sono più comodi per gli utenti. I sistemi operativi distribuiti sono progettati per migliorare questa situazione.

15.1.2.2 Sistemi operativi distribuiti

In un sistema operativo distribuito gli utenti accedono alle risorse remote nello stesso modo in cui accedono alle risorse locali. I trasferimenti di dati e processi tra siti sono sotto il controllo del sistema operativo distribuito.

Migrazione dei dati

Si supponga che un utente del sito *A* voglia accedere a dati (ad esempio, contenuti in un file) che risiedono nel sito *B*. Il sistema può trasferire i dati impiegando uno fra due metodi di base. Un orientamento alla **migrazione dei dati** consiste nel trasferire tutto il file al sito *A*. Da questo punto in poi l'intero accesso al file è di tipo locale. Quando l'utente non ha più necessità di accedere al file, se il file era stato modificato, si ritrasmette una sua copia al sito *B*. Si devono trasferire tutti i dati anche se il file ha subito solo una lieve modifica. Questo metodo, che si può considerare come un sistema FTP automatico, era usato nel file system Andrew (Paragrafo 16.6) ma è stato ritenuto troppo inefficiente.

L'altro orientamento consiste nel trasferire al sito *A* solo le parti del file immediatamente necessarie. Se in seguito è richiesta un'altra sua parte, si esegue un altro trasferimento. Quando l'utente non avrà più bisogno di accedere al file, si ritrasmetterà al sito *B* qualsiasi sua parte sia stata modificata (si noti l'analogia con la paginazione su richiesta). Questo metodo è adoperato dal protocollo NFS (*network file system*) della Sun Microsystems e dalle più recenti versioni di Andrew (Capitolo 16). Anche il protocollo SMB della Microsoft (eseguito sopra il TCP/IP o sopra il protocollo NetBEUI, della stessa Microsoft) consente la condivisione dei file in una rete (Paragrafo 21.6.1).

Per accedere a una piccola parte di un file di grandi dimensioni, ovviamente conviene servirsi del secondo metodo; quando è necessario accedere a parti considerevoli del file, invece, conviene copiare tutto il file.

In entrambi i metodi, la migrazione dei dati comprende assai più del mero trasferimento dei dati da un sito all'altro. Se i due siti coinvolti nel trasferimento non sono direttamente compatibili (ad esempio, se impiegano codici dei caratteri diversi o rappresentano i numeri interi con un diverso numero o ordine dei bit), il sistema deve eseguire anche diverse traduzioni dei dati.

Migrazione delle computazioni

In alcune circostanze si può voler trasferire le computazioni anziché i dati; questo metodo si chiama **migrazione delle computazioni**. Si consideri, ad esempio, un lavoro d'elaborazione che necessiti di accedere a diversi file di grandi dimensioni che risiedono in diversi siti per ottenere un sommario di tutti quei file. È più conveniente accedere ai file nei siti in cui risiedono e riportare i risultati al sito che ha iniziato il calcolo. In generale s'impiega il comando remoto se il tempo di trasferimento dei dati è maggiore del tempo d'esecuzione del comando remoto.

Tale calcolo si può eseguire in diversi modi. Si supponga che il processo *P* voglia accedere a un file presente nel sito *A*. L'accesso al file si esegue nel sito *A* e può essere avviato da una RPC. Una RPC usa un **protocollo per datagrammi** (UDP nella rete Internet) per far eseguire una procedura in un sistema remoto (Paragrafo 4.6.2). Il processo *P* invoca una procedura predefinita nel sito *A*, tale procedura esegue adeguatamente il proprio compito e riporta i risultati a *P*.

In alternativa il processo *P* può inviare un *messaggio* al sito *A*; il sistema operativo di *A* crea un nuovo processo *Q* la cui funzione è quella di eseguire il compito designato; quando il processo *Q* termina l'esecuzione, invia il risultato richiesto a *P* per mezzo del sistema di messaggi. In questo schema il processo *P* si può eseguire in modo concorrente al processo *Q* e, in effetti, più processi si possono eseguire in modo concorrente in diversi siti.

Entrambi i metodi si possono usare per accedere a più file residenti in vari siti. Una RPC può invocare un'altra RPC, o addirittura trasferire messaggi a un altro sito. Analogamente il processo *Q*, durante la propria esecuzione, può inviare un messaggio a un altro sito, che a sua volta crea un altro processo. Questo processo può inviare un messaggio a *Q*, oppure ripetere il ciclo.

Migrazione dei processi

Una logica estensione della migrazione delle computazioni è la **migrazione dei processi**. Quando si esegue un processo, non sempre l'esecuzione si svolge nel sito nel quale è stata avviata. L'intero processo, o parti di esso, si possono eseguire in siti diversi. Questo schema è preferibile per diversi motivi:

- **Bilanciamento del carico.** I processi (o sottoprocessi) si possono distribuire tramite la rete per equilibrare il carico di lavoro.
- **Accelerazione dei calcoli.** Se un singolo processo si può dividere in un certo numero di sottoprocessi che si possono eseguire in modo concorrente in siti diversi, è possibile ridurre il tempo di completamento totale.
- **Preferenza di sistemi e dispositivi.** Il processo può avere caratteristiche che lo rendono più adatto a un'esecuzione in un'unità d'elaborazione specializzata (ad esempio, un'inversione di matrice in un'unità d'elaborazione matriciale anziché in un'ordinaria CPU).
- **Preferenza di programmi.** Il processo può richiedere un programma disponibile solo in un particolare sito, ma il programma non può essere spostato, oppure è meno costoso spostare il processo.
- **Accesso ai dati.** Come nella migrazione delle computazioni, se i dati da usare nella computazione sono numerosi, l'esecuzione remota del processo può essere più efficiente del trasferimento dei dati.

Per spostare i processi in una rete di calcolatori si possono usare due metodi complementari. Col primo, il sistema può cercare di nascondere che il processo è migrato dal

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

client. Questo schema ha il vantaggio che l'utente non deve codificare esplicitamente il suo programma per compiere la migrazione; normalmente si usa per ottenere un bilanciamento del carico e un'accelerazione dei calcoli tra sistemi omogenei, poiché in questo modo essi non richiedono un'interazione con l'utente nell'esecuzione remota dei programmi.

L'altro metodo consiste nel permettere (o richiedere) all'utente di specificare esplicitamente come il processo debba migrare. Questo metodo si usa normalmente nelle situazioni in cui si deve trasferire un processo per soddisfare una preferenza di dispositivi o di programmi.

Il lettore si è probabilmente accorto del fatto che il World Wide Web ha molte caratteristiche di un ambiente d'elaborazione distribuito. Di certo offre la migrazione dei dati (tra un Web-server e un client) così come la migrazione delle computazioni: un client può ad esempio attivare un'operazione su una base di dati di un Web-server. Infine, con l'ambiente Java, fornisce una forma di migrazione dei processi: le cosiddette *Java applet* sono inviate dal server al client dove vengono eseguite. Un sistema operativo di rete offre la maggior parte di queste funzioni, ma un sistema operativo distribuito le integra e le rende più facilmente accessibili. Il risultato è uno strumento potente e facile da usare, una delle ragioni dell'enorme crescita del Web.

15.1.3 Elementi di un sistema distribuito

Dopo aver illustrato le motivazioni che spingono alla realizzazione dei sistemi distribuiti e i tipi di sistemi operativi distribuiti, si possono considerare gli elementi necessari alla realizzazione di questi sistemi. Nel resto di questo capitolo si analizza il livello più basso: la rete di calcolatori sulla quale si basa un sistema distribuito. Nel Capitolo 16 si studiano i file system distribuiti, che sono una realizzazione distribuita del classico modello a partizione del tempo di un file system (in cui più utenti condividono file e risorse di memoria secondaria), dove i file sono fisicamente localizzati nei vari siti di un sistema distribuito. Nel Capitolo 17 invece si descrivono i metodi necessari affinché i sistemi operativi distribuiti possano coordinare le loro azioni.

15.2 Topologie

I siti di un sistema si possono collegare fisicamente in diversi modi. Ogni configurazione ha i suoi pro e i suoi contro. In questo paragrafo si descrivono e si confrontano brevemente le configurazioni più diffuse, secondo i seguenti criteri:

- **Costo di installazione.** Occorre considerare il costo necessario per collegare i vari siti di un sistema.
- **Costo della comunicazione.** Un altro fattore importante è il tempo necessario per inviare un messaggio dal sito *A* al sito *B*.
- **Disponibilità.** Se un collegamento o un sito del sistema si guasta, occorre stabilire se gli altri siti possono continuare a comunicare tra loro.

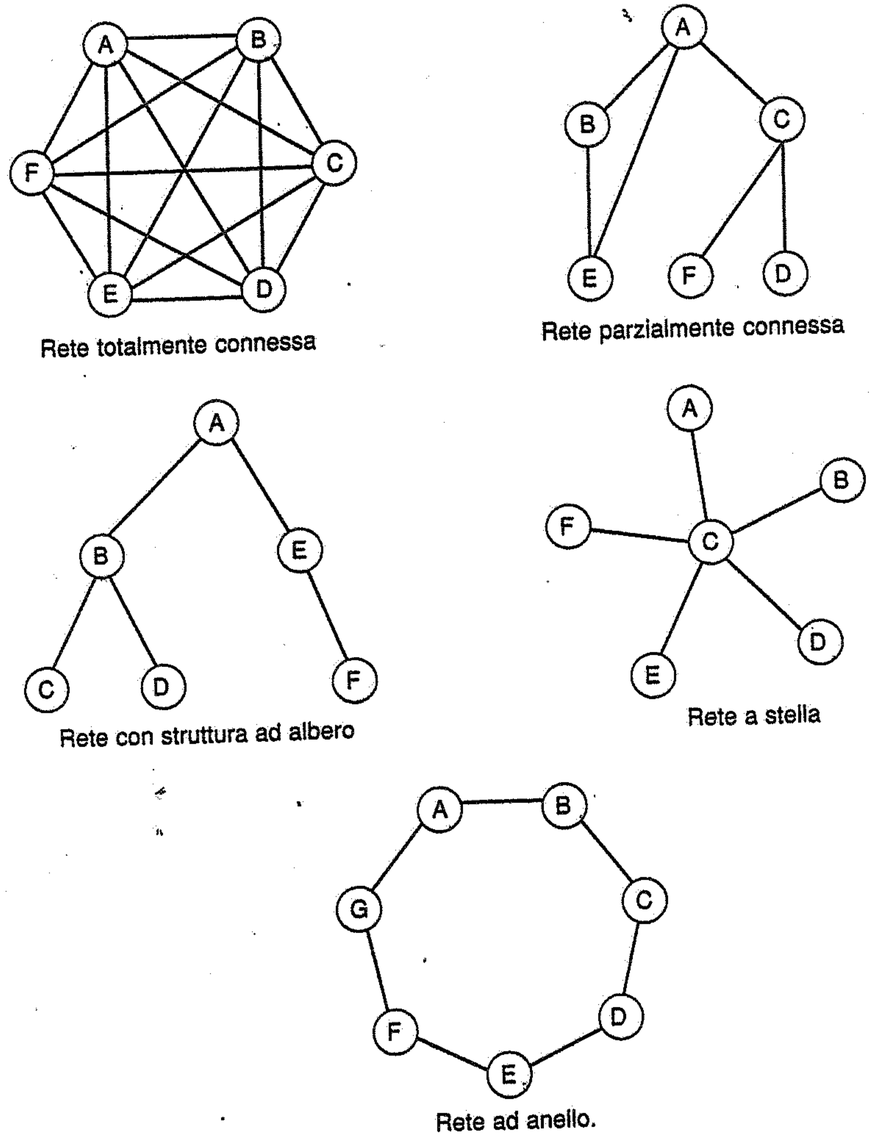


Figura 15.2 Topologie di reti.

Le diverse topologie sono rappresentate nella Figura 15.2 per mezzo di grafi, i cui nodi corrispondono ai siti. Un arco dal nodo *A* al nodo *B* corrisponde a un collegamento diretto tra i due siti.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

In una rete **totalmente connessa** ogni sito è collegato direttamente a tutti gli altri siti del sistema; tuttavia il numero di collegamenti aumenta con il quadrato del numero dei siti, portando a enormi costi d'installazione. Quindi, le reti totalmente connesse non sono praticamente utilizzabili per ogni sistema di grandi dimensioni.

In una rete **parzialmente connessa** esistono collegamenti diretti tra alcune coppie di siti (ma non tra tutte). Quindi il costo d'installazione di questa configurazione è inferiore rispetto a quello di una rete totalmente connessa. Tuttavia, per arrivare da un sito a un altro, un messaggio può essere costretto a passare attraverso parecchi siti intermedi, perciò il costo di comunicazione risulta più alto.

Se un collegamento diventa inutilizzabile, i messaggi che avrebbero dovuto passarvi devono essere reindirizzati. In alcuni casi è possibile trovare un percorso alternativo nella rete, in modo che i messaggi possano raggiungere la loro destinazione. In altri casi però, un guasto può portare all'assenza di connessioni tra determinate coppie di siti. Un sistema si dice partizionato se è stato diviso in due o più sottosistemi, chiamati partizioni, che non hanno connessioni tra loro. Secondo questa definizione, un sottosistema può anche essere composto da un solo nodo.

I diversi tipi di reti parzialmente connesse comprendono le reti strutturate ad albero, le reti ad anello e le reti a stella, riportate nella Figura 15.2. Queste reti hanno diverse caratteristiche rispetto a possibili guasti e diversi costi d'installazione e comunicazione. I costi d'installazione e comunicazione sono relativamente bassi per una rete strutturata ad albero; tuttavia, il guasto di un singolo collegamento in una struttura ad albero può comportare la partizione della rete. Nel caso delle reti ad anello, affinché si abbia la partizione di una rete, devono esservi almeno due collegamenti non funzionanti. Quindi, le reti ad anello hanno un più alto grado di disponibilità rispetto a quelle con struttura ad albero. Tuttavia, il costo delle comunicazioni è alto, poiché un messaggio per giungere a destinazione può dover attraversare molti collegamenti. In una rete a stella, il guasto di un singolo collegamento determina la partizione della rete, ma una delle partizioni contiene un singolo sito e il problema si può trattare analogamente al guasto di un singolo sito; tuttavia, il guasto del sito centrale porta alla irraggiungibilità di tutti i siti del sistema. La rete a stella ha anche un basso costo di comunicazione, poiché ciascun sito dista due collegamenti da ogni altro sito.

15.3 Tipi di reti

I due tipi di rete fondamentali sono le reti locali e le reti geografiche. La principale differenza tra le due consiste nella loro distribuzione geografica. Le **reti locali** (*local-area network* — LAN) comprendono unità d'elaborazione distribuite su piccole aree, come un unico edificio o alcuni edifici adiacenti. Le **reti geografiche** (*wide-area network* — WAN), invece, comprendono unità d'elaborazione autonome distribuite su vaste aree geografiche. Queste differenze implicano notevoli variazioni riguardo alla velocità e all'affidabilità delle reti di comunicazione, e si riflettono nella progettazione dei sistemi operativi distribuiti.

15.3.1 Reti locali

Le reti locali sono nate nei primi anni Settanta con lo scopo di sostituire i mainframe. Molte imprese trovano più economico avere tanti piccoli calcolatori, ciascuno con le proprie applicazioni interne, anziché un unico sistema di grandi dimensioni. Poiché è probabile che ogni piccolo calcolatore necessiti di una serie completa di dispositivi periferici, come dischi e stampanti, e poiché è probabile che esista qualche forma di condivisione di dati in un'unica società, è naturale collegare questi piccoli sistemi in una rete.

Generalmente le LAN sono progettate per coprire piccole aree, come un solo edificio o alcuni edifici adiacenti, e di solito sono usate in ambienti di uffici. In questi sistemi tutti i siti sono vicini fra loro, i collegamenti tendono ad avere una maggiore velocità e una minor frequenza di errori rispetto alle reti geografiche. Per ottenere tale velocità e affidabilità sono necessari (costosi) cavi di elevata qualità. I cavi si possono usare esclusivamente per il traffico dei dati della rete. Per lunghe distanze il costo dovuto all'impiego di cavi di elevata qualità è enorme, e il loro uso esclusivo tende a diventare proibitivo.

Nelle reti locali i collegamenti sono in genere realizzati con cavi a doppino intrecciato e cablaggi con fibre ottiche. Le configurazioni più diffuse sono le reti con bus multiaccesso, ad anello e a stella. La velocità di comunicazione varia dall'ordine del **megabit** al secondo per reti come l'Appletalk e a raggi infrarossi fino a 1 **gigabit** al secondo per le reti Gigabit Ethernet; la velocità più comune è di 10 megabit al secondo ed è la velocità dell'*Ethernet 10 BaseT*; l'*Ethernet 100 BaseT*, che sta diventando comune, richiede un cablaggio di alta qualità ma funziona alla velocità di 100 megabit al secondo. Le interconnessioni FDDI basate su fibre ottiche hanno incrementato le loro quote di mercato; tali reti sono basate sul passaggio di un contrassegno (*token*) e funzionano a 100 megabit al secondo.

Una LAN tipica è composta da diversi calcolatori, dai mainframe ai portatili e PDA, vari dispositivi periferici condivisi (come stampanti o unità a nastri magnetici), e uno o più gateway (unità d'elaborazione specializzate) che dànno accesso ad altre reti (Figura 15.3). Per costruire le LAN comunemente si usa uno schema Ethernet. In una rete Ethernet non c'è un controllore centrale poiché è una rete con bus multiaccesso, quindi si possono aggiungere facilmente nuovi calcolatori alla rete.

15.3.2 Reti geografiche

Le reti geografiche sono nate alla fine degli anni Sessanta come progetto di ricerca accademica per fornire un sistema di comunicazione efficiente tra siti, che permettesse a un'ampia comunità di utenti di condividere in modo conveniente ed economico le risorse di calcolo. La prima rete progettata e sviluppata è stata l'**Arpanet**, il lavoro su di essa è cominciato nel 1968. Nata come rete sperimentale a quattro siti, è cresciuta fino a diventare una rete mondiale di reti, l'Internet, che comprende milioni di sistemi di calcolo. Varie reti private commerciali internazionali mettono in comunicazione punti specifici, come le filiali e gli uffici delle aziende. Queste reti offrono ai loro clienti la possibilità di accedere a un'ampia gamma di risorse di calcolo.

http://www.edutpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

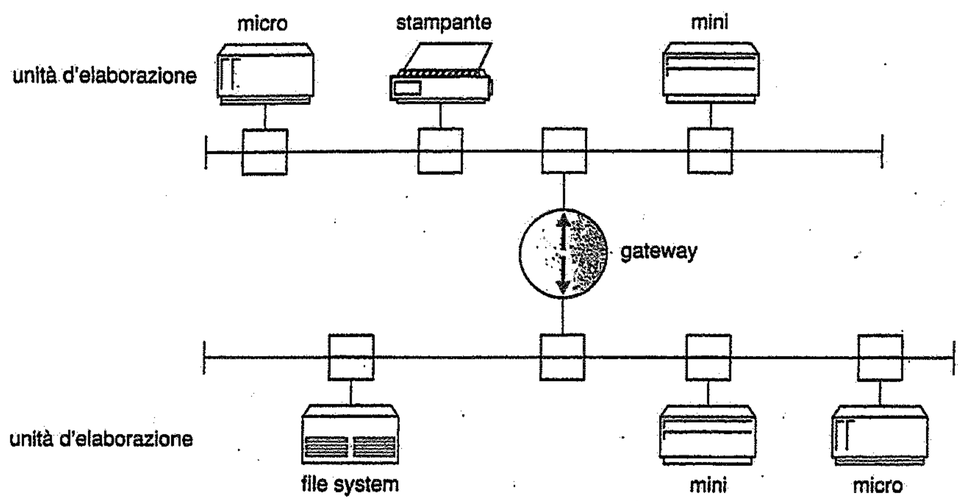


Figura 15.3 Rete locale.

Poiché i siti di una WAN sono fisicamente distribuiti su una vasta zona geografica, i collegamenti per le comunicazioni sono relativamente lenti e inaffidabili. Di solito si tratta di linee telefoniche, collegamenti a microonde e canali via satellite. Questi collegamenti sono controllati da speciali **unità d'elaborazione di comunicazione** (Figura 15.4) responsabili della definizione dell'interfaccia attraverso la quale i siti comunicano nella rete e del trasferimento delle informazioni tra i diversi siti.

Si consideri, ad esempio, la WAN Internet, che offre ai calcolatori in siti geograficamente separati la possibilità di comunicare tra loro. Generalmente, i calcolatori differiscono per tipo, velocità, lunghezza delle parole, sistema operativo e così via. I calcolatori vengono inseriti in LAN, che a loro volta sono collegate alla rete Internet per mezzo di reti regionali. Le reti regionali, come la NSFnet degli Stati Uniti del Nord-Est, sono interconnesse tramite instradatori (*router*), descritti nel Paragrafo 15.4.2, per formare la rete mondiale. I collegamenti tra le reti impiegano spesso un servizio telefonico, chiamato T1, che fornisce una velocità di trasferimento di 1,544 megabit al secondo su una linea noleggiata. Per siti che richiedono un accesso più veloce, i T1 sono raccolti in unità di più T1 che lavorano in parallelo per fornire una maggiore produttività. Ad esempio, un T3 è composto da 28 connessioni T1 e ha una velocità di trasferimento di 45 megabit al secondo. Gli instradatori controllano i percorsi seguiti da ogni messaggio attraverso la rete; tale instradamento può essere dinamico, per incrementare l'efficienza delle comunicazioni oppure statico, per ridurre i rischi per la sicurezza o per permettere di calcolare le spese di comunicazione.

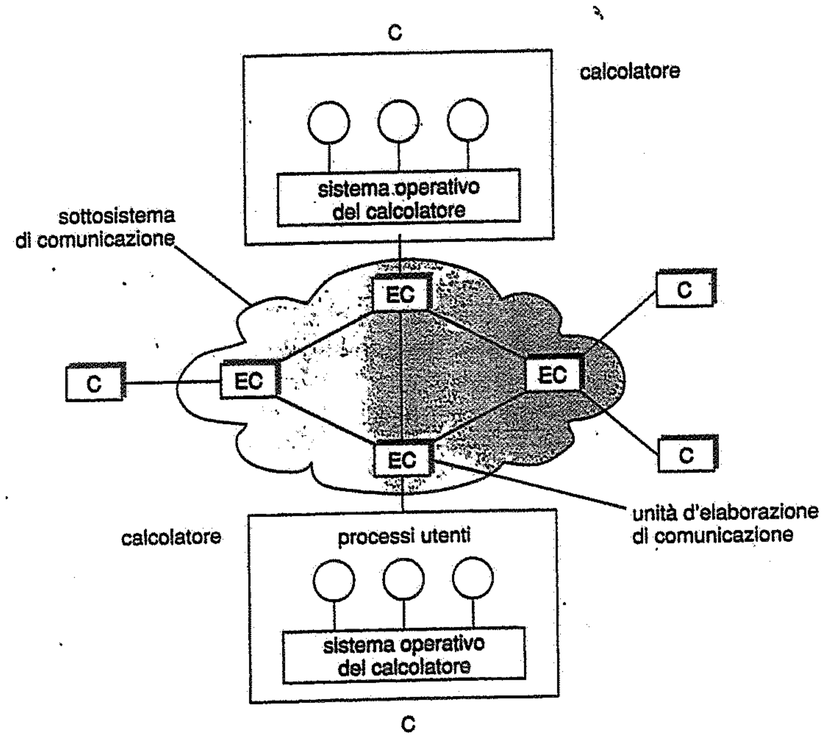


Figura 15.4 Unità d'elaborazione di comunicazione in una rete geografica.

Altre WAN si servono di ordinarie linee telefoniche come mezzo di comunicazione principale. I **modem** sono dispositivi che ricevono dati digitali da un calcolatore e li convertono nei segnali analogici del sistema telefonico; un modem nel sito di destinazione riconverte il segnale analogico in forma digitale e il destinatario riceve i dati. La rete di scambio di messaggi di posta elettronica dell'ambiente UNIX (UUCP) permette ai sistemi di comunicare tra loro via modem a ore prefissate per scambiarsi messaggi. I messaggi sono quindi instradati ad altri sistemi vicini e in questo modo si propagano a tutti i calcolatori della rete (messaggi pubblici), oppure vengono trasferiti alla loro destinazione (messaggi privati). Le WAN sono generalmente più lente delle LAN; le loro velocità di trasmissione variano da 1200 bit al secondo a oltre 1 megabit al secondo.

L'UUCP sta per essere rapidamente soppiantato dal protocollo PPP (*point-to-point protocol*); si tratta di una versione del protocollo IP che funziona con connessioni realizzate tramite modem e consente il collegamento dei calcolatori domestici alla rete Internet.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

15.4 Comunicazione

Dopo aver trattato gli aspetti fisici della comunicazione tramite reti si può considerarne il funzionamento interno. Il progettista di una rete di comunicazione deve considerare quattro problemi fondamentali:

- **Nominazione e risoluzione dei nomi.** Trattano di come due processi si individuano l'un l'altro allo scopo di comunicare.
- **Strategie d'instradamento.** Trattano di come s'inviano i messaggi attraverso la rete.
- **Strategie riguardanti i pacchetti.** Trattano l'invio individuale o in sequenza dei pacchetti.
- **Strategie di connessione.** Trattano dello scambio di sequenze di messaggi fra due processi.
- **Contesa.** Riguarda i problemi di conflittualità legati all'uso della rete come risorsa condivisa.

Questi argomenti sono sviluppati nei Paragrafi dal 15.4.1 al 15.4.5.

15.4.1 Nominazione e risoluzione dei nomi

Il primo elemento nelle comunicazioni tramite rete è la nominazione (*naming*) dei sistemi nella rete. Affinché due processi in esecuzione in due siti differenti *A* e *B* possano scambiarsi informazioni devono poter fare riferimento l'uno all'altro. All'interno di un sistema di calcolo ciascun processo ha un identificatore di processo, quindi i messaggi si possono indirizzare con l'identificatore del processo. Poiché i sistemi presenti nella rete non condividono memoria, inizialmente non hanno informazioni riguardanti il calcolatore in cui è in esecuzione il processo destinatario, né sanno se il processo in questione esiste.

Per risolvere questo problema i processi di un sistema remoto sono generalmente identificati dalla coppia <*nome del calcolatore*, *identificatore*>, dove *nome del calcolatore* è di solito un nome unico all'interno della rete, e *identificatore* può essere un identificatore di processo o un altro numero unico all'interno del calcolatore.

Un *nome del calcolatore* è di solito un identificatore alfanumerico, anziché un numero, per renderne più semplice l'utilizzo da parte degli utenti. Ad esempio, il sito *A* potrebbe contenere calcolatori con i nomi *emanuela*, *carlo*, *anna* e *livia*; è sicuramente più facile ricordare il nome *emanuela* che il numero 12814831100.

Sebbene i nomi siano comodi per gli esseri umani, nei calcolatori è preferibile usare i numeri, sia per motivi di velocità sia di semplicità. Per questa ragione ci deve essere un meccanismo per la **risoluzione** del nome del calcolatore nel corrispondente identificatore, che descrive il sistema di destinazione ai dispositivi della rete. Questo meccanismo di risoluzione è simile all'associazione (*binding*) tra nomi e indirizzi definita durante la compilazione, il collegamento, il caricamento e l'esecuzione di un programma

(Capitolo 9). Nel caso dei nomi di calcolatori esistono due possibilità; nella prima ciascun calcolatore può avere un file contenente i nomi e gli indirizzi di tutti i calcolatori raggiungibili tramite la rete (situazione analoga all'associazione nella fase di compilazione). Il problema di questo modello è che l'aggiunta o la rimozione di un calcolatore dalla rete richiede l'aggiornamento in tutti i calcolatori dei suddetti file. La seconda possibilità consiste nel distribuire le informazioni tra i sistemi connessi alla rete, la quale deve quindi adoperare un protocollo per distribuire e recuperare queste informazioni. Questo schema corrisponde all'associazione tra nomi e indirizzi nella fase d'esecuzione. Il primo metodo era originariamente adottato nella rete Internet che ora, a causa della sua inarrestabile crescita, adotta il secondo metodo chiamato DNS (*domain name-system*).

Il DNS specifica la struttura di nominazione dei calcolatori e la risoluzione dei nomi in indirizzi. I calcolatori della rete Internet si indirizzano logicamente con un nome composto, le cui parti progrediscono dalla parte più specifica a quella più generale dell'indirizzo, con i diversi campi separati da un punto: `bob.cs.brown.edu` si riferisce al calcolatore `bob` del *Department of Computer Science* della Brown University. In generale, il sistema risolve l'indirizzo esaminando gli elementi del nome del calcolatore nell'ordine inverso. Per ciascun elemento c'è un **server dei nomi** — semplicemente un processo in un sistema — che accetta un nome e riporta l'indirizzo del server dei nomi responsabile per la risoluzione di quel nome. Come passo finale si interpella il server dei nomi associato al calcolatore in questione, che provvederà a riportare l'identificatore del calcolatore. Per quel che riguarda il sistema dell'esempio precedente, `bob.cs.brown.edu`, una richiesta di comunicare con `bob.cs.brown.edu` inoltrata da un processo in esecuzione in un sistema *A* comporterebbe l'esecuzione dei seguenti passi:

1. Il nucleo del sistema *A* inoltra una richiesta al server dei nomi del dominio `edu`, richiedendo l'indirizzo del server dei nomi di `brown.edu`. Per poter essere interrogato, il server dei nomi di `edu` deve avere un indirizzo noto. (Tra gli altri domini al livello più alto si trovano `com` per le società commerciali, `org` per le organizzazioni, e uno per ogni nazione connessa alla rete; quest'ultimo dominio è usato da tutti i sistemi suddivisi per nazione invece che per tipo di organizzazione.)
2. Il server dei nomi di `edu` riporta l'indirizzo del calcolatore nel quale risiede il server dei nomi di `brown.edu`.
3. Il nucleo del sistema *A* interroga il server dei nomi situato a questo indirizzo riguardo al dominio `cs.brown.edu`.
4. S'invia una richiesta all'indirizzo riportato per `bob.cs.brown.edu` e si ottiene finalmente come risposta l'identificatore del calcolatore sotto forma di indirizzo d'Internet di quel sistema (ad esempio, 128.148.31.100).

Questo protocollo può sembrare inefficiente, ma ciascun server dei nomi mantiene di solito cache locali per accelerare il processo. Ad esempio, il server dei nomi di `edu` potrebbe avere `brown.edu` memorizzato nella propria cache, e potrebbe informare il sistema *A* della possibilità di risolvere due parti del nome, riportando il puntatore al server dei nomi di `cs.brown.edu`. Naturalmente il contenuto di queste cache deve essere ag-

http://www.pdftpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

giornato col passare del tempo, poiché sia il server dei nomi sia il suo indirizzo potrebbero essere cambiati. Nella pratica questo servizio è così importante da indurre l'introduzione di numerose ottimizzazioni e molte protezioni nel protocollo. Si consideri cosa accadrebbe nel caso di un crollo del server dei nomi primario edu: si potrebbe perdere la capacità di risolvere gli indirizzi dei calcolatori del dominio edu, che diventerebbero irraggiungibili. La soluzione consiste nell'impiego di server dei nomi secondari, di riserva, nei quali duplicare il contenuto dei server dei nomi primari.

Prima dell'introduzione dei server dei nomi, tutti i calcolatori connessi alla rete Internet avevano bisogno di copie di un file contenente i nomi e gli indirizzi di tutti i calcolatori della rete. Tutte le modifiche a questo file dovevano essere registrate presso un particolare sito, il sistema SRI-NIC, e periodicamente tutti i calcolatori dovevano prelevarvi la versione aggiornata di questo file per poter contattare nuovi sistemi o trovare i calcolatori che avevano cambiato indirizzo. Col DNS, il server dei nomi di ciascun sito è responsabile dell'aggiornamento delle informazioni riguardanti il proprio dominio. Ad esempio, il server dei nomi di brown.edu è responsabile di tutte le modifiche riguardanti i calcolatori della Brown University, e tali modifiche non devono essere notificate a nessun altro. Le ricerche del DNS recupereranno automaticamente le informazioni aggiornate, poiché contatteranno direttamente brown.edu. All'interno del dominio possono esistere sottodomìni autonomi allo scopo di distribuire ulteriormente la responsabilità circa le variazioni dei nomi e degli identificatori dei calcolatori.

In generale, il sistema operativo è responsabile dell'accettazione dai propri processi di un messaggio destinato a *<nome del calcolatore, identificatore>* e del trasferimento del messaggio al calcolatore appropriato. Il nucleo del calcolatore di destinazione è quindi responsabile del trasferimento del messaggio al processo specificato dall'identificatore. Questo scambio di informazioni è tutt'altro che elementare, ed è descritto nel Paragrafo 15.4.4.

15.4.2 Strategie d'instradamento

In questo paragrafo si descrive il modo in cui avviene la trasmissione di un messaggio inviato da un processo del sito *A* che vuole comunicare con un processo del sito *B*. Se tra *A* e *B* esiste un solo percorso fisico, come in una rete a stella o ad albero, il messaggio deve seguire necessariamente quel percorso, ma se i percorsi fisici da *A* a *B* sono più di uno esistono diverse possibilità d'instradamento. Ogni sito ha una **tabella d'instradamento** che indica i percorsi che si possono seguire nel recapitare un messaggio ad altri siti.

La tabella può contenere informazioni sulla velocità e sul costo dei diversi percorsi di comunicazione e, se è necessario, può anche essere aggiornata manualmente oppure tramite programmi che scambiano informazioni d'instradamento. I tre schemi d'instradamento più diffusi sono l'**instradamento fisso**, l'**instradamento virtuale** e l'**instradamento dinamico**:

- **Instradamento fisso.** Un percorso da *A* a *B* viene determinato in anticipo e non cambia, se non per un guasto fisico che interrompe il percorso stesso. Generalmente si sceglie il percorso più breve, in modo da ridurre al minimo i costi della comunicazione.

- **Instradamento virtuale.** Si fissa un percorso da *A* a *B* per la durata di una sessione. Sessioni diverse con messaggi che vanno da *A* a *B* possono avere percorsi diversi. Una sessione può essere breve, come il trasferimento di un file, o lunga, come la durata di un sessione di lavoro remota.
- **Instradamento dinamico.** Il percorso da impiegare per inviare un messaggio dal sito *A* al sito *B* si sceglie al momento dell'invio del messaggio. Poiché la decisione viene presa dinamicamente, a messaggi distinti si possono assegnare percorsi diversi. Il sito *A* decide di inviare il messaggio al sito *C*, che a sua volta decide di inviare il messaggio al sito *D* e così via. Alla fine, un sito invia il messaggio a *B*. Generalmente, un sito invia un messaggio al sito connesso tramite il collegamento che in quel momento è meno usato.

Questi tre schemi hanno diversi pro e contro. L'instradamento fisso non si adatta a malfunzionamenti nelle linee di comunicazione o a cambiamenti del carico; in altre parole, se è stato fissato un percorso tra *A* e *B*, si devono inviare i messaggi lungo questo percorso anche se è interrotto o è più fortemente usato di altri possibili percorsi. Questo problema si può risolvere parzialmente impiegando l'instradamento virtuale, e si può evitare impiegando l'instradamento dinamico. L'uso dell'instradamento fisso o dell'instradamento virtuale assicura che i messaggi da *A* a *B* arrivino nell'ordine in cui sono stati trasmessi. Con l'instradamento dinamico i messaggi possono arrivare disordinati; questo problema si può risolvere assegnando a ogni messaggio un numero di sequenza.

L'instradamento dinamico è molto più complicato da impostare ed eseguire, ma è il modo migliore per gestire l'instradamento in ambienti complessi. Il sistema operativo UNIX offre sia l'instradamento fisso, da usare in calcolatori all'interno di reti semplici, sia l'instradamento dinamico per ambienti di rete complessi; è possibile anche una combinazione dei due metodi. Nell'ambito di un sito i calcolatori devono soltanto sapere come raggiungere il sistema che connette la rete locale ad altre reti (come reti aziendali o l'Internet); tale nodo è noto come *gateway*. Questi singoli calcolatori potrebbero avere un instradamento statico verso il gateway, che potrebbe impiegare l'instradamento dinamico per raggiungere ogni calcolatore nel resto della rete.

L'instradatore è l'entità interna alla rete di calcolatori responsabile dell'instradamento dei messaggi. Può essere un calcolatore dotato di programmi d'instradamento o un dispositivo specifico. In entrambi i casi deve avere almeno due connessioni di rete, altrimenti non avrebbe nessun posto dove instradare i messaggi. Un instradatore decide se ogni dato messaggio deve essere passato dalla rete da cui viene ricevuto a un'altra rete a esso connessa. Tale decisione si prende esaminando l'indirizzo d'Internet di destinazione del messaggio: l'instradatore esamina le proprie tabelle per determinare la locazione del calcolatore di destinazione, o almeno della rete cui inviare il messaggio verso il calcolatore di destinazione. Nel caso dell'instradamento statico questa tabella viene modificata solo da aggiornamenti eseguiti manualmente (si carica un nuovo file nell'instradatore). Con l'instradamento dinamico, si usa un **protocollo d'instradamento** tra gli instradatori per informare gli stessi delle modifiche alla rete e consentire l'aggiornamento automatico delle rispettive tabelle d'instradamento. I gateway e gli instradatori sono di solito dispositivi specifici appositamente programmati, anziché calcolatori dotati di un sistema operativo d'uso generale che eseguono applicazioni di rete.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

15.4.3 Strategie riguardanti i pacchetti

I messaggi sono generalmente di lunghezza variabile. Per semplificare la progettazione del sistema, di solito si realizza la comunicazione usando messaggi di lunghezza costante detti **pacchetti**, **frame** o **datagrammi**. Le informazioni codificate in un singolo pacchetto si possono inviare al destinatario tramite una comunicazione **priva di connessione**. Essa può essere **inaffidabile**, in tal caso il mittente non ha alcuna garanzia del fatto che il pacchetto giunga a destinazione e non dispone di alcun modo per verificare se un pacchetto spedito sia andato perduto. In alternativa, il pacchetto può essere **affidabile**, in tal caso s'invia un altro pacchetto dal destinatario al mittente come conferma dell'arrivo del primo pacchetto. (Naturalmente il pacchetto di conferma potrebbe andar perso e non giungere al mittente.) Se un messaggio è troppo lungo per essere codificato in un solo pacchetto, o se si devono ripetutamente scambiare pacchetti fra due interlocutori, si stabilisce una connessione al fine di permettere una comunicazione affidabile.

15.4.4 Strategie di connessione

Una volta che i messaggi sono in grado di raggiungere le loro destinazioni, i processi possono instaurare **sessioni di comunicazione** per scambiarsi informazioni. Le coppie di processi che intendono comunicare attraverso la rete si possono connettere in diversi modi. I tre schemi più diffusi sono la **commutazione di circuito**, la **commutazione di messaggio** e la **commutazione di pacchetto**:

- **Commutazione di circuito.** Se due processi vogliono comunicare, si stabilisce tra essi un collegamento fisico permanente. Questo collegamento resta assegnato per tutta la durata della comunicazione e nessun altro processo può usarlo per tutto questo periodo, anche se esistono intervalli di tempo nei quali i due processi non comunicano attivamente. Questo schema è simile a quello adoperato nel sistema telefonico. Una volta aperta una linea di comunicazione tra due parti, vale a dire che la parte *A* chiama la parte *B*, nessun altro può usare questo circuito finché non si termina esplicitamente la comunicazione (ad esempio, quando una delle due parti riappende).
- **Commutazione di messaggio.** Se due processi vogliono comunicare, si stabilisce un collegamento temporaneo per la durata del trasferimento di un messaggio. I collegamenti fisici si assegnano dinamicamente secondo le necessità e tale assegnazione ha breve durata. Ogni messaggio è composto di un blocco di dati e di informazioni di sistema, come l'origine, la destinazione e codici per la correzione degli errori, che permettono alla rete di comunicazione di recapitare messaggi senza errori. Questo schema è simile a quello postale: ogni lettera è considerata come un messaggio che contiene sia l'indirizzo di destinazione sia quello d'origine. Sullo stesso collegamento si possono inviare i messaggi di utenti diversi.
- **Commutazione di pacchetto.** Un messaggio logico si può dividere in un dato numero di pacchetti; ciascuno dei quali si può inviare alla sua destinazione separatamente, perciò deve contenere, oltre ai dati, un indirizzo di provenienza e uno di destinazione; ogni pacchetto può seguire un percorso diverso attraverso la rete. Quando arrivano a destinazione, i pacchetti devono essere ricomposti nei messaggi originari.

Anche questi schemi hanno diversi pro e contro. La commutazione di circuito richiede un sostanziale tempo d'impostazione e può sprecare banda della rete, ma richiede un minor carico per la spedizione di ogni messaggio. La commutazione di messaggi e pacchetti, invece, richiede un tempo di impostazione minore, ma un carico maggiore per ogni messaggio. Inoltre, la commutazione di pacchetto implica la divisione di ogni messaggio in pacchetti e la successiva ricomposizione. La commutazione di pacchetto è il metodo più comunemente usato nelle reti di comunicazione dei dati poiché fa il miglior uso dell'ampiezza di banda della rete e poiché per i dati non è dannoso essere suddivisi in pacchetti, eventualmente instradati separatamente, e ricomposti a destinazione. Lo spezzettamento di un segnale audio (ad esempio, una comunicazione telefonica) d'altra parte potrebbe, se non eseguita accuratamente, causare una gran confusione.

15.4.5 Contesa

Secondo la topologia della rete, un collegamento può connettere più di due siti nella rete di calcolatori, quindi parecchi siti possono trasmettere contemporaneamente informazioni nella stessa linea (questa situazione si presenta soprattutto nelle reti ad anello o a bus multiaccesso), con la conseguenza che le informazioni trasmesse possono risultare confuse e si devono scartare. In questo caso i siti devono essere informati del problema per poter ritrasmettere tali informazioni. Se non si prendono provvedimenti specifici, questa situazione può ripetersi e le prestazioni del sistema degradarsi. Per evitare collisioni ripetute sono state sviluppate parecchie tecniche, tra cui il **rilevamento delle collisioni**, il **passaggio di contrassegno** e gli **intervalli di messaggi**:

- CSMA/CD. Prima di trasmettere un messaggio in una linea di comunicazione, un sito deve verificare se su quella linea non sia già in corso la trasmissione di un altro messaggio. Questa tecnica è chiamata **ascolto della portante con accessi multipli** (*carrier sense with multiple access* — CSMA). Se la linea è libera il sito può iniziare la trasmissione, altrimenti deve attendere e continuare ad ascoltare finché la linea non si libera. Se due o più siti iniziano a trasmettere nello stesso istante, si ha un **rilevamento di collisione** (*collision detection* — CD) e interrompono la trasmissione. Ciascun sito ritenta la trasmissione dopo un intervallo di tempo casuale. Quando il sito *A* inizia la trasmissione in una linea deve continuare ad ascoltare per rilevare eventuali collisioni con messaggi di altri siti. Il problema maggiore di questo metodo consiste nel fatto che, quando il sistema è molto carico, si possono verificare molte collisioni, con un conseguente calo delle prestazioni. Ciononostante, il metodo CSMA/CD è impiegato con successo nel sistema Ethernet, il sistema di rete più diffuso. (Il protocollo Ethernet è definito dallo standard IEEE 802.3.) Per limitare il numero di collisioni in una rete Ethernet occorre limitare il numero dei calcolatori. L'inserimento di altri calcolatori in una rete congestionata determinerebbe una scarsa produttività della rete. A mano a mano che diventano più veloci, i sistemi possono spedire più pacchetti per unità di tempo; da ciò segue che, per mantenere ragionevoli le prestazioni della rete, il numero di sistemi per segmento di rete Ethernet è in diminuzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- **Passaggio di contrassegno.** Nel sistema, di solito con una struttura ad anello, circola continuamente un messaggio di tipo unico avente la funzione di contrassegno (*token*). Un sito che intenda trasmettere informazioni deve attendere finché non arriva il contrassegno, quindi lo rimuove dall'anello e inizia a trasmettere i propri messaggi; completata la trasmissione dei messaggi ritrasmette il contrassegno: quest'operazione permette a un altro sito di ricevere e rimuovere il contrassegno e iniziare la trasmissione dei propri messaggi. Se il contrassegno viene perso, i sistemi devono individuare la perdita e generare un nuovo contrassegno. Normalmente ciò si fa dichiarando un'**elezione**, per eleggere, appunto, un unico sito che deve generare un nuovo contrassegno; nel Paragrafo 17.6 si descrive un algoritmo di elezione. Uno schema con passaggio di contrassegno è stato adottato dai sistemi IBM e HP/Apollo. Il vantaggio delle reti con passaggio di contrassegno è che le prestazioni sono costanti; l'inserimento di nuovi sistemi nella rete può allungare il tempo d'attesa del contrassegno, ma non determina un gran decremento delle prestazioni come può accadere con l'Ethernet. In reti con basso carico, comunque, l'Ethernet è più efficiente poiché i sistemi possono inviare messaggi in tutti i momenti.

- **Intervalli di messaggi.** Nel sistema, normalmente con una struttura ad anello, circola continuamente un certo numero di 'intervalli' di messaggi (*message slot*) di lunghezza fissa. Ogni intervallo può contenere un messaggio di dimensione fissa e informazioni di controllo, come l'origine e la destinazione e un'informazione che indichi se l'intervallo è pieno o vuoto. Un sito pronto per la trasmissione deve attendere fino all'arrivo di un intervallo vuoto, quindi inserisce il suo messaggio nell'intervallo, impostando l'opportuna informazione di controllo. L'intervallo continua a circolare nella rete con il nuovo messaggio. Quando arriva a un sito, quest'ultimo ispeziona l'informazione di controllo per stabilire se l'intervallo contiene un messaggio per il sito stesso; in tal caso, il sito preleva il messaggio, impostando l'informazione di controllo a indicare che ora l'intervallo è vuoto. Il sito può quindi impiegare l'intervallo per inviare un suo messaggio oppure rilasciarlo vuoto. Se invece il messaggio non è diretto a tale sito, quest'ultimo rimette in circolazione intervallo e messaggio. Poiché un intervallo può contenere solo messaggi di dimensione fissa, un unico messaggio logico si può dividere in pacchetti più piccoli, ciascuno dei quali viene inviato in un intervallo diverso. Questo schema è stato adottato nel sistema di comunicazione sperimentale Cambridge Digital Communication Ring.

15.5 Protocolli di comunicazione

Nel progettare una rete di comunicazione, è necessario considerare la complessità del coordinamento di operazioni asincrone che comunicano in un ambiente potenzialmente lento e soggetto a errori. Inoltre i sistemi nella rete devono accordarsi su un protocollo o su un insieme di protocolli per determinare i nomi dei calcolatori, individuare i calcolatori nella rete, stabilire le connessioni, e così via. Il problema della progettazione e

della relativa realizzazione si può semplificare mediante una suddivisione in strati. Ciascuno strato in un sistema comunica con lo strato corrispondente negli altri sistemi. Ogni strato può avere i suoi protocolli o può essere il risultato di una suddivisione logica. I protocolli si possono realizzare in forma di dispositivi o possono essere programmi. La Figura 15.5 mostra, ad esempio, uno schema delle comunicazioni logiche tra due calcolatori, con i tre strati di livello più basso realizzati come dispositivi. Secondo il modello di riferimento OSI (*open systems interconnection*) dell'ISO (*international standards organization*), tali strati sono definiti come segue:

1. **Strato fisico.** È responsabile della gestione dei particolari meccanici ed elettrici della trasmissione fisica di un flusso di bit. Nello strato fisico il sistema di comunicazione deve accordarsi sulla rappresentazione elettrica delle cifre binarie 0 e 1, in modo che quando s'inviano i dati come flusso di segnali elettrici il ricevitore possa interpretare i dati correttamente come dati binari. Questo strato si realizza nei dispositivi di rete.

2. **Strato di collegamento dati.** È responsabile delle funzioni di gestione dei *frame*, o di parti di pacchetti di lunghezza fissa, comprese l'individuazione e la correzione degli errori che si sono verificati nello strato fisico.

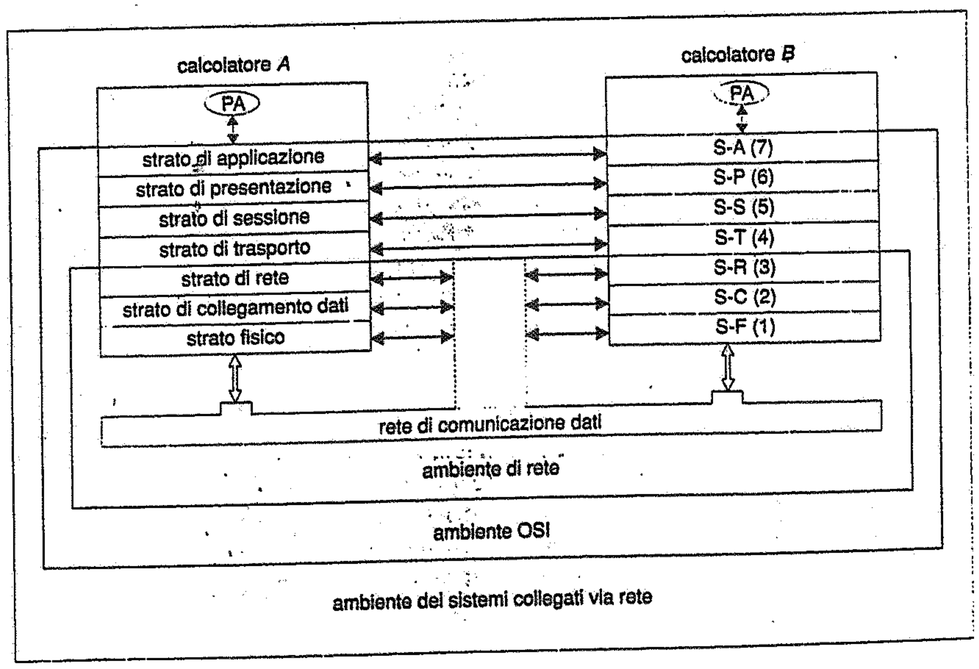


Figura 15.5 Ambiente definito da due calcolatori comunicanti attraverso il modello di rete OSI.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

3. **Strato di rete.** È responsabile della fornitura dei collegamenti e dell'instradamento dei pacchetti nella rete di comunicazione, comprese la gestione degli indirizzi dei pacchetti in uscita, la decodifica degli indirizzi dei pacchetti in arrivo e la gestione delle informazioni d'instradamento per rispondere adeguatamente ai cambiamenti dei livelli del carico. Gli instradatori operano in questo strato.
4. **Strato di trasporto.** È responsabile delle funzioni d'accesso a basso livello alla rete e del trasferimento dei messaggi tra i client, comprese la suddivisione dei messaggi in pacchetti, l'ordinamento dei pacchetti, il controllo del flusso e la generazione degli indirizzi fisici.
5. **Strato di sessione.** È responsabile della realizzazione di sessioni, o di protocolli di comunicazione da processo a processo. Generalmente questi protocolli consentono le effettive comunicazioni per le sessioni di lavoro remote, i trasferimenti di file e la posta elettronica.
6. **Strato di presentazione.** È responsabile della risoluzione delle differenze di formato che si possono presentare tra diversi siti della rete, fra cui la conversione di caratteri e i modi *half duplex-full duplex* (eco dei caratteri).
7. **Strato di applicazione.** È responsabile dell'interazione diretta con gli utenti. Questo strato tratta il trasferimento di file, i protocolli per la gestione delle sessioni di lavoro remote, la posta elettronica, e gli schemi per le basi di dati distribuite.

La Figura 15.6 riassume la **pila di protocolli** OSI — un insieme di protocolli cooperanti — e illustra il flusso fisico dei dati. Dal punto di vista logico ciascuno strato di una pila di protocolli comunica con lo strato corrispondente negli altri sistemi. Fisicamente, però, un messaggio parte dallo strato di applicazione, o da sopra di esso, e passa attraverso ciascun livello più basso. Ciascuno strato può modificare il messaggio e includere dati d'intestazione del messaggio per lo strato corrispondente del ricevente. Infine il messaggio raggiunge lo strato della rete di comunicazione e viene trasferito sotto forma di uno o più pacchetti (Figura 15.7). Lo strato di collegamento dati del sistema di destinazione riceve questi dati e il messaggio risale attraverso la pila di protocolli; viene analizzato, modificato e privato delle intestazioni; infine raggiunge lo strato di applicazione e può essere usato dal processo ricevente.

Il modello OSI formalizza alcuni precedenti lavori svolti sui protocolli di rete, è stato sviluppato alla fine degli anni Settanta ma non è ancora molto diffuso; una parte delle sue basi è costituita della più logora e largamente usata pila di protocolli sviluppata nell'ambiente UNIX per essere impiegata nell'Arpanet (divenuta la rete Internet).

La maggior parte dei siti della rete Internet comunica ancora mediante il protocollo IP (*internet protocol*). I servizi sono realizzati sopra l'IP tramite il protocollo senza connessione UDP (*user datagram protocol*) e attraverso il protocollo orientato alla connessione TCP (*transmission control protocol*). La pila di protocolli TCP/IP ha meno strati di quelli stabiliti dal modello OSI, e poiché combina diverse funzioni in ciascuno strato è, teoricamente, più difficile da realizzare ma più efficiente delle reti OSI. La pila del TCP/IP (in corrispondenza col modello OSI) è mostrata nella Figura 15.8. Il protocollo IP è responsabile della trasmissione dei datagrammi IP, l'unità di base d'informazione, attraverso un'interconnessione di reti basata sui protocolli TCP/IP. Il TCP si serve dell'IP per trasportare in modo affidabile un flusso d'informazioni tra due processi. L'altro usuale protocollo di trasmissione tra reti interconnesse è UDP/IP. L'UDP è un protocollo di trasporto non affidabile e senza connessione, usa l'IP per trasferire i pacchetti, ma offre anche la correzione degli errori e un protocollo di **indirizzo di porta** per specificare il processo del sistema remoto cui è destinato il pacchetto.

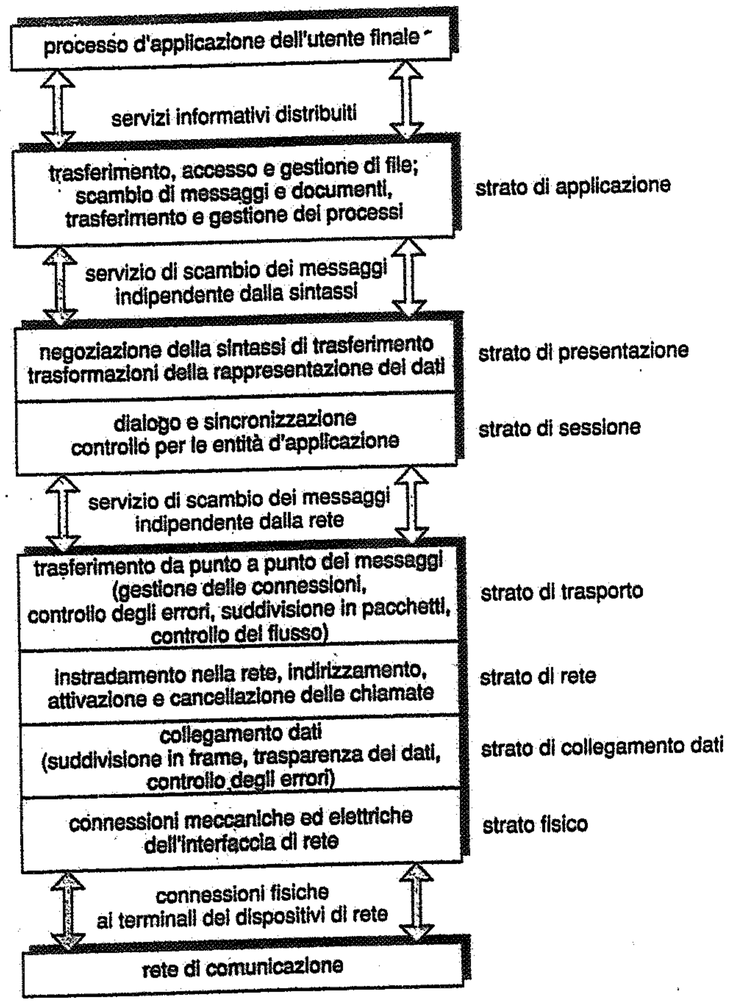


Figura 15.6 Strati del protocollo OSI.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

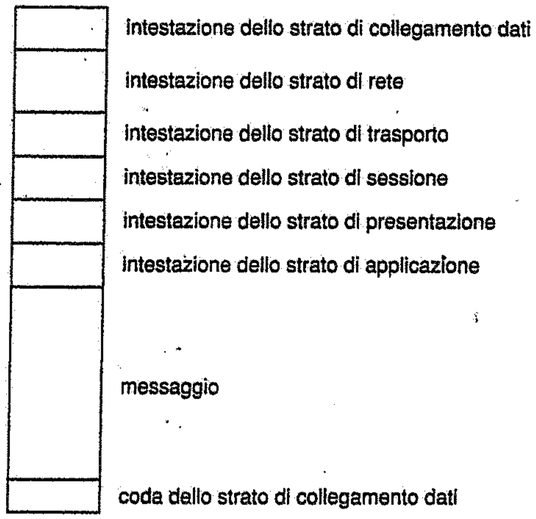


Figura 15.7 Messaggio di rete OSI.

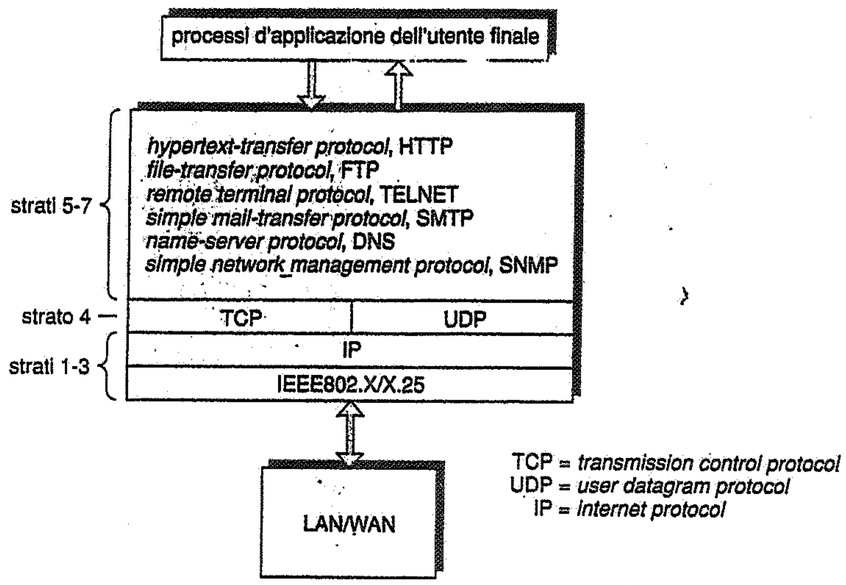


Figura 15.8 Strati del protocollo TCP/IP.

15.6 Robustezza

Un sistema distribuito può soffrire di guasti fisici di diverso tipo. Il guasto di un collegamento, il guasto di un sito e la perdita di un messaggio sono quelli più frequenti. Per assicurare la robustezza del sistema occorre **individuare** tutti questi guasti, **riconfigurare** il sistema per poter continuare la computazione ed effettuare il **ripristino** una volta riparato il collegamento o il sito.

15.6.1 Rilevamento dei guasti

In un ambiente senza memoria condivisa non si è generalmente capaci di distinguere tra il guasto di un collegamento, il guasto di un sito e la perdita di un messaggio. Normalmente si può stabilire solo che si è verificato uno di questi guasti, ma non è possibile identificarne il tipo di guasto. Una volta rilevato un guasto è necessario intraprendere azioni appropriate, che dipendono dalla particolare applicazione.

Per rilevare un guasto in un collegamento o in un sito si usa una procedura di **negoziazione** (*handshaking*). Si supponga che i siti *A* e *B* abbiano tra loro un collegamento fisico diretto. A intervalli fissi entrambi i siti s'inviano il messaggio *I-am-up*. Se il sito *A* non riceve questo messaggio entro un dato periodo di tempo, può supporre che il sito *B* sia guasto, che sia guasto il collegamento tra *A* e *B*, oppure che sia andato perduto il messaggio proveniente da *B*. A questo punto il sito *A* ha due possibilità: può attendere un altro periodo di tempo per ricevere un messaggio *I-am-up* da *B*, oppure può inviare il messaggio *Are-you-up?* a *B*.

Se il sito *A* non riceve un messaggio *I-am-up* oppure una risposta alla sua domanda, si può ripetere la procedura. L'unica conclusione che il sito *A* può trarre con sicurezza è che si è verificato un guasto.

Il sito *A* tenta di distinguere tra la possibilità di un guasto al collegamento e quella di un guasto al sito inviando a *B* il messaggio *Are-you-up?* per un altro percorso, se questo esiste. Se e quando riceve questo messaggio, *B* risponde subito positivamente. Questa risposta positiva indica ad *A* che *B* è pronto, e che il guasto si trova nel collegamento diretto. Poiché non si può sapere a priori quanto tempo impieghi il messaggio per andare da *A* a *B* e tornare indietro, è necessario usare uno schema d'attesa (*time-out*). Quando *A* invia il messaggio *Are-you-up?*, specifica un intervallo di tempo entro il quale attende la risposta di *B*. Se *A* riceve il messaggio di risposta entro quell'intervallo di tempo, può concludere con sicurezza che *B* è pronto; altrimenti *A* può concludere che si è verificata una (o più) delle seguenti situazioni:

- il sito *B* è fuori servizio;
- il collegamento diretto, se esiste, tra *A* e *B* è fuori servizio;
- il percorso alternativo da *A* a *B* è fuori servizio;
- il messaggio è andato perduto.

Il sito *A* non può in ogni caso determinare quale di questi eventi si sia verificato.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

15.6.2 Riconfigurazione

Si supponga che il sito *A* abbia scoperto, per mezzo del meccanismo descritto nel paragrafo precedente, che si è verificato un guasto. Deve allora cominciare una procedura che permetta al sistema di riconfigurarsi e continuare a funzionare normalmente:

- Se si è guastato un collegamento diretto tra *A* e *B*, questa informazione deve essere diffusa a ogni sito del sistema, in modo da poter aggiornare adeguatamente le tabelle d'instradamento.
- Se il guasto si imputa al sito, poiché tale sito non è più raggiungibile, si deve informare della situazione ciascun sito del sistema in modo che non tenti più di usare i servizi del sito guasto. Il guasto di un sito usato come coordinatore centrale di qualche attività, come il rilevamento delle situazioni di stallo, implica l'elezione di un nuovo coordinatore. Analogamente, se il sito guasto fa parte di un anello logico, è necessario costruire un nuovo anello logico. Occorre notare che, se il sito non si è guastato (vale a dire che è funzionante ma non raggiungibile), è possibile incorrere nella situazione spiacevole in cui due siti agiscono da coordinatore. Se la rete viene partizionata, i due coordinatori, uno per ogni partizione, possono condurre ad azioni conflittuali. Se i coordinatori sono responsabili, ad esempio, della realizzazione della mutua esclusione, si può avere una situazione in cui due processi sono in esecuzione contemporaneamente nelle rispettive sezioni critiche.

15.6.3 Ripristino dopo un guasto

Quando un collegamento o un sito guasto è stato riparato, deve essere reintegrato nel sistema in modo semplice e lineare:

- Si supponga che si sia guastato un collegamento tra *A* e *B*. Una volta che il guasto è stato riparato, è necessario informare della riparazione sia *A* sia *B*. Quest'informazione può essere data ripetendo continuamente la procedura di negoziazione descritta nel Paragrafo 15.6.1.
- Si supponga che si sia guastato il sito *B*. Questo, una volta reinserito, deve informare tutti gli altri siti che è di nuovo pronto. Il sito *B* può quindi ricevere informazioni da altri siti per aggiornare le sue tabelle locali; ad esempio, possono servirgli le informazioni contenute nella tabella d'instradamento, un elenco di siti fuori servizio, oppure messaggi e posta non consegnati. Se il sito non era guasto, ma era semplicemente irraggiungibile, queste informazioni devono comunque essergli inviate.

15.7 Problemi di progettazione

Rendere trasparente agli utenti la molteplicità delle unità d'elaborazione e dei dispositivi di memorizzazione è stato lo scopo di molti progettisti. Teoricamente un sistema distribuito dovrebbe apparire ai suoi utenti come un sistema centralizzato convenzionale. L'interfaccia d'utente di un sistema distribuito trasparente non deve fare distinzioni tra risorse locali e risorse remote. Ciò significa che gli utenti dovrebbero poter accedere a sistemi distribuiti remoti come se fossero locali, e spetterebbe al sistema distribuito localizzare le risorse e predisporne l'interazione.

Un altro aspetto della trasparenza riguarda la mobilità degli utenti; è conveniente permettere agli utenti di aprire sessioni in qualsiasi calcolatore del sistema anziché obbligarli a usare un calcolatore specifico. Un sistema distribuito trasparente facilita la mobilità degli utenti trasportando l'ambiente dell'utente, ad esempio la directory iniziale, ovunque l'utente apra sessioni. Sia il file system Andrew della CMU sia il Project Athena del MIT offrono questa funzione su vasta scala. L'NFS può fornire tale trasparenza su scala minore.

Il termine *tolleranza ai guasti* si usa in senso lato; gli errori di comunicazione, i guasti di macchina (del tipo *fail-stop*), le rotture dei dispositivi di memoria e il deterioramento dei mezzi di memorizzazione si considerano in una certa misura tollerabili. Un sistema tollerante i guasti (*fault-tolerant system*) deve continuare a funzionare, sia pure in modo ridotto, anche quando si presentano tali guasti. La degradazione può riguardare le prestazioni, il funzionamento oppure entrambi; tuttavia questa dovrebbe essere proporzionale agli errori che l'hanno causata. Un sistema che si ferma quando si guastano solo alcuni dei suoi componenti non si può certo considerare tollerante i guasti. Sfortunatamente la tolleranza ai guasti è difficile da realizzare. Ad esempio, il *DEC VAX cluster* permette a più calcolatori di condividere un gruppo di dischi. Se un sistema si guasta, gli utenti possono continuare ad accedere alle informazioni usando un altro sistema. Naturalmente, se un disco si guasta, tutti i sistemi perdono la possibilità di accedervi. In questo caso per assicurare la possibilità d'accesso si potrebbero impiegare dispositivi RAID (Paragrafo 14.5).

La capacità di un sistema di adattarsi a un maggior carico di servizio è la cosiddetta 'scalabilità'. I sistemi hanno risorse limitate e, alla presenza di un grande carico, possono giungere alla completa saturazione. Ad esempio, per quel che riguarda un file system, la saturazione avviene quando la CPU di un server funziona a un alto tasso d'utilizzo, oppure quando i dischi sono quasi pieni. La scalabilità è una caratteristica relativa, ma che si può misurare con precisione. Un sistema scalabile reagisce più favorevolmente a un carico maggiore di quanto non lo faccia un sistema non scalabile. Innanzi tutto le sue prestazioni degradano più moderatamente; in secondo luogo, le sue risorse raggiungono la saturazione più tardi. Anche un sistema perfettamente progettato non può sopportare un carico sempre crescente. L'aggiunta di nuove risorse può risolvere il problema, ma può anche generare un ulteriore carico indiretto su altre risorse (ad esempio, l'aggiunta di cal-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

colatori a un sistema distribuito potrebbe produrre un intasamento della rete e aumentare i carichi di servizio). O peggio, l'espansione del sistema potrebbe richiedere costose modifiche strutturali. Un sistema scalabile deve avere la potenzialità di crescere evitando questi problemi. In un sistema distribuito una conveniente scalabilità ha un'importanza particolare, poiché è una pratica diffusa espandere la rete aggiungendo calcolatori o connettendo due reti. In breve, un progetto scalabile deve resistere a un alto carico di servizio, far fronte alla crescita della comunità di utenti e permettere una facile integrazione di ulteriori risorse.

La tolleranza ai guasti e la scalabilità sono interdipendenti. Un componente estremamente carico può paralizzarsi e comportarsi come un componente guasto, quindi lo spostamento del carico da un componente guasto a quello di riserva può saturare anche quest'ultimo. È essenziale, ai fini dell'affidabilità, disporre di risorse di ricambio, anche per poter gestire in modo conveniente i picchi di carico. Un sistema distribuito ha il vantaggio intrinseco della potenziale tolleranza ai guasti e la potenziale scalabilità date dalla molteplicità delle risorse. Tuttavia una progettazione non appropriata può occultare queste potenzialità. Le considerazioni di tolleranza ai guasti e di scalabilità richiedono una progettazione caratterizzata dalla distribuzione del controllo e dei dati.

I sistemi distribuiti su scala molto vasta sono ancora teorici. Non esistono direttive 'magiche' che assicurino la scalabilità di un sistema; è più facile stabilire perché i progetti attuali *non* sono scalabili. Nel seguito si analizzano alcuni progetti che pongono problemi, per i quali si propongono possibili soluzioni tutte nell'ambito della scalabilità.

Un principio su cui si fonda la progettazione di sistemi su scala molto vasta è quello per cui il carico di ogni componente del sistema dovrebbe essere limitato da una costante indipendente dal numero dei nodi del sistema. Qualsiasi meccanismo di servizio la cui richiesta di carico è proporzionale alla dimensione del sistema è destinato a intasarsi quando il sistema supera una certa misura. Il problema non si può risolvere con l'aggiunta di nuove risorse. La capacità di questo meccanismo limita semplicemente la crescita del sistema.

Gli schemi con controllo centrale e le risorse centrali non si devono usare per costruire sistemi scalabili e tolleranti i guasti. Esempi di entità centralizzate sono i server di convalida centrali, i server di nominazione centrali e i file server centrali. La centralizzazione è una forma di asimmetria funzionale tra le macchine che costituiscono il sistema. La soluzione ideale è una configurazione funzionalmente simmetrica: tutte le macchine componenti hanno lo stesso ruolo nel funzionamento del sistema; quindi ogni macchina ha lo stesso grado di autonomia. Nella pratica è virtualmente impossibile soddisfare tale principio. L'inserimento di calcolatori privi di dischi, ad esempio, viola la simmetria funzionale poiché tali calcolatori dipendono da un disco centrale; tuttavia l'autonomia e la simmetria sono obiettivi importanti ai quali si dovrebbe tendere.

L'approssimazione pratica di una configurazione simmetrica e autonoma è rappresentata dalla strutturazione del sistema in batterie di calcolatori (*clustering*); si suddivide il sistema in un insieme di batterie semiautonome di calcolatori, ciascuna delle quali è a sua volta composta da un insieme di calcolatori e di un server. Ciò in modo che i riferimenti alle risorse esterne a ciascuna batteria di calcolatori siano relativamente infrequenti: le richieste dei calcolatori di una batteria dovrebbero essere soddisfatte per la maggior parte delle volte dal relativo server. Naturalmente questo schema dipende dalla possibilità di localizzare i riferimenti alle risorse e di collocare adeguatamente le unità componenti. Se una batteria di calcolatori è ben bilanciata, cioè se il relativo server riesce a soddisfare tutte le sue richieste, può essere impiegata come un blocco di costruzione modulare per accrescere progressivamente il sistema.

Un problema importante nella progettazione di qualsiasi servizio riguarda la struttura dei processi del server. S'ipotizza che i server operino in modo efficiente nei periodi di picco, quando centinaia di client attivi richiedono contemporaneamente di essere serviti. I server a processi singoli non rappresentano sicuramente una buona scelta, poiché ogni volta che una richiesta ha bisogno di compiere operazioni di I/O su un disco l'intero servizio si blocca. L'assegnazione di un processo per ogni client rappresenta una scelta migliore, sebbene si debbano considerare i costi dei frequenti cambi di contesto tra i processi. Un problema correlato si verifica poiché tutti i processi server devono condividere informazioni.

Una delle soluzioni migliori per l'architettura dei server è quella di impiegare processi leggeri o thread, descritti nel Capitolo 5. L'astrazione rappresentata da un gruppo di processi leggeri è quella di thread di controllo multipli associati a certe risorse condivise. Di solito un processo leggero non è legato a un client particolare, ma serve singole richieste di client diversi. Lo scheduling dei thread può essere con diritto di prelazione o senza diritto di prelazione. Se i thread si eseguono sempre sino alla fine (scheduling senza diritto di prelazione) non è necessario proteggere esplicitamente i dati condivisi; altrimenti è necessario ricorrere a qualche meccanismo esplicito di bloccaggio. È chiaro che affinché i server siano scalabili è essenziale qualche tipo di schema basato su processi leggeri.

15.8 Un esempio di comunicazione in rete

A questo punto è utile tornare al problema della risoluzione dei nomi affrontato nel Paragrafo 15.4.1 per esaminarne il comportamento rispetto alla pila di protocolli TCP/IP nella rete Internet. Si possono considerare le elaborazioni necessarie al trasferimento di un pacchetto tra due calcolatori appartenenti a due diverse reti Ethernet.

In una rete TCP/IP, ciascun calcolatore possiede un nome e un numero di Internet a 32 bit a esso associato, che ha il ruolo di identificatore del calcolatore (*host-id*). Entrambi i valori devono essere unici e sono suddivisi per facilitare la gestione dello spazio dei nomi. Il nome è gerarchico (come si spiega nel Paragrafo 15.4.1) e specifica sia il nome del calcolatore sia le organizzazioni alle quali il calcolatore è associato. L'identificatore del calcolatore è suddiviso in un numero di rete e un numero di macchina. Le proporzioni della scomposizione variano secondo la dimensione della rete. Una volta che gli amministratori di Internet hanno assegnato il numero di rete, nel sito con quel numero si è liberi di assegnare gli identificatori dei calcolatori come meglio si crede.

Il sistema mittente ricerca all'interno delle proprie tabelle d'instradamento un instradatore cui spedire il pacchetto. Gli instradatori sfruttano la parte di rete dell'identifi-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

catore del calcolatore per trasferire il pacchetto dalla rete di partenza a quella di destinazione. Il sistema destinatario riceve quindi il pacchetto, che può essere il messaggio completo o semplicemente una sua parte. In quest'ultimo caso, prima di ricostruire il messaggio e passarlo allo strato TCP/UDP per la trasmissione al processo destinatario, è necessario attendere l'arrivo degli altri pacchetti.

A questo punto si sa come un pacchetto passa dalla rete di partenza alla sua destinazione, ma ci si può chiedere come fa un pacchetto, all'interno di una rete, a passare dal mittente (calcolatore o instradatore) al ricevente. Ogni dispositivo Ethernet ha un numero unico che lo individua detto **indirizzo** MAC (*medium access control address*). Due dispositivi di una rete locale comunicano tra loro servendosi unicamente di questo numero. Se un sistema necessita di inviare dati a un altro sistema, il nucleo genera un pacchetto ARP (*address resolution protocol*) contenente l'indirizzo IP del sistema di destinazione. Questo pacchetto viene **diffuso** (*broadcast*) a tutti gli altri sistemi della rete Ethernet. Una diffusione adopera uno speciale indirizzo di rete (di solito l'indirizzo massimo) per segnalare a tutti i calcolatori connessi di ricevere ed elaborare il pacchetto. I pacchetti trasmessi per diffusione non sono ritrasmessi dai gateway, perciò sono ricevuti solamente dai sistemi connessi alla rete locale. Solo il sistema il cui indirizzo IP corrisponde all'indirizzo IP dell'ARP risponde e invia il proprio indirizzo MAC al sistema che ha fatto l'interrogazione. Per motivi di efficienza, il calcolatore copia in una tabella cache interna la coppia <*indirizzo IP*, *indirizzo MAC*>. Gli elementi della cache sono fatti **invecchiare** fino a essere rimossi qualora in un certo tempo non sia richiesto un accesso al sistema. In questo modo i calcolatori rimossi dalla rete vengono *dimenticati*. Per aumentare le prestazioni, si possono cablare nella cache degli ARP gli elementi relativi agli ARP dei calcolatori più usati.

Dopo che un dispositivo Ethernet ha annunciato il suo identificatore del calcolatore e il suo indirizzo, la comunicazione può cominciare. Un processo può specificare il nome del calcolatore con cui intende comunicare, il nucleo prende questo nome e determina l'indirizzo di Internet della destinazione attraverso un'interrogazione al DNS. Il messaggio passa dallo strato di applicazione allo strato fisico, attraversando tutti gli strati di programmi e protocolli. Nello strato fisico, il pacchetto (o i pacchetti) contiene l'indirizzo Ethernet al suo inizio e una coda, che indica la fine del pacchetto e contiene una **somma di verifica** che serve a rilevare eventuali errori nel pacchetto stesso (Figura 15.9). Il pacchetto viene posto nella rete dal dispositivo Ethernet. La sezione dei dati del pacchetto può contenere tutto il messaggio o solamente una parte dei suoi dati, così come può contenere alcune delle intestazioni di livello superiore che compongono il messaggio. In altre parole, tutti gli elementi del messaggio originale si devono inviare dalla sorgente alla destinazione e tutte le intestazioni poste sopra lo strato 802.3 (lo strato di collegamento dati) sono incluse nel pacchetto Ethernet sotto forma di dati. Se la destinazione si trova nella stessa rete locale della sorgente, il sistema può analizzare la propria cache di ARP, trovare l'indirizzo Ethernet del calcolatore e immettere il pacchetto nella linea. Il dispositivo Ethernet della destinazione riconosce il proprio indirizzo e legge il pacchetto, passandolo alla pila di protocolli superiore.

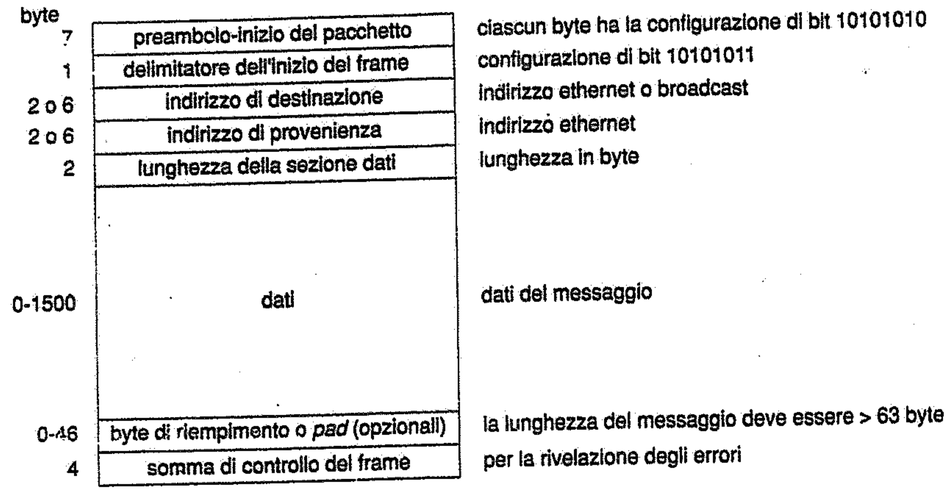


Figura 15.9 Pacchetto Ethernet.

Se il sistema di destinazione si trova in una rete diversa da quella della sorgente, il sistema di partenza cerca l'instradatore appropriato per quella rete e gli spedisce il pacchetto. Gli instradatori trasmettono quindi il pacchetto lungo la WAN finché esso raggiunge la sua rete di destinazione. L'instradatore che connette la rete di destinazione esamina la propria cache di ARP, cerca il numero Ethernet della destinazione e spedisce il pacchetto al calcolatore. Attraverso tutti questi trasferimenti, l'intestazione dello strato di collegamento dati può cambiare a mano a mano che viene usato l'indirizzo Ethernet del successivo instradatore della catena, mentre le altre intestazioni del pacchetto restano invariate fino a quando il pacchetto non viene ricevuto ed elaborato dalla pila di protocolli e quindi passato dal nucleo al processo ricevente.

15.9 Sommario

Un sistema distribuito è un insieme di unità d'elaborazione che non condividono memoria o un clock. Al contrario, ciascuna unità d'elaborazione ha la propria memoria locale e la comunicazione avviene attraverso varie linee di comunicazione, come bus ad alta velocità o linee telefoniche. Le dimensioni e le funzioni delle unità d'elaborazione di un sistema distribuito sono diverse. Il sistema può comprendere piccole unità d'elaborazione, stazioni di lavoro, minicalcolatori e grandi calcolatori d'uso generale.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le unità d'elaborazione del sistema sono collegate attraverso una rete di comunicazione che può essere configurata in diversi modi. La rete può essere parzialmente o totalmente connessa; può essere un albero, una stella, un anello o un bus multiaccesso. La progettazione di una rete di comunicazione deve comprendere le strategie d'instradamento e di collegamento, e deve risolvere i problemi di contesa e di sicurezza.

Un sistema distribuito fornisce all'utente l'accesso alle diverse risorse offerte dal sistema stesso. L'accesso a una risorsa condivisa può essere fornito tramite la migrazione dei dati, la migrazione delle computazioni o la migrazione dei processi.

In linea di principio esistono due tipi di sistema distribuito: le reti locali (LAN) e le reti geografiche (WAN). La principale differenza tra le due è legata alla loro distribuzione geografica. Le LAN sono composte da unità d'elaborazione distribuite su piccole aree geografiche, come un unico edificio o alcuni edifici adiacenti, mentre le WAN sono composte da unità d'elaborazione autonome distribuite su una vasta area geografica.

Le pile di protocolli, com'è specificato dai modelli di stratificazione delle reti, manipolano i messaggi aggiungendovi informazioni per assicurare che essi raggiungano la loro destinazione. Per tradurre il nome di un calcolatore nel corrispondente indirizzo di rete si deve usare un sistema di nominazione come il DNS; un altro protocollo, come l'ARP, può servire a tradurre il numero di rete in un indirizzo di dispositivo di rete, ad esempio un indirizzo Ethernet. Se i sistemi risiedono su reti diverse sono necessari degli instradatori per trasferire i pacchetti dalla rete sorgente a quella di destinazione.

Un sistema distribuito può essere soggetto a vari tipi di malfunzionamenti dei dispositivi fisici che lo compongono. Affinché il sistema sia tollerante i guasti deve essere capace d'individuarli e di riconfigurare il sistema; una volta che il guasto è stato riparato il sistema deve essere nuovamente riconfigurato.

15.10 Esercizi

15.1 Confrontate le varie topologie di rete rispetto all'affidabilità.

15.2 Dite perché la maggior parte delle WAN impiega solo una topologia parzialmente connessa.

15.3 Dite quali sono le principali differenze tra una WAN e una LAN.

15.4 Dite quale configurazione di rete è più adatta ai seguenti ambienti:
- a) un piano di un dormitorio;
- b) un campus universitario;
- c) una regione;
- d) una nazione.

15.5 Anche se il modello di rete OSI specifica sette strati di funzioni, nella maggior parte dei sistemi di calcolo si usano meno strati per realizzare una rete; spiegate perché e dite quali problemi può causare l'uso di un minor numero di strati.

15.6 Spiegate perché il raddoppio della velocità di un sistema in un segmento Ethernet può causare una diminuzione delle prestazioni della rete. Dite quali modifiche potrebbero migliorare la situazione.

15.7 Dite in quali circostanze una rete a passaggio di contrassegno (*token*) è più efficiente di una rete Ethernet.

15.8 Dite perché il trasferimento tra reti, da parte dei gateway, di pacchetti di diffusione potrebbe avere effetti negativi. Dite quali potrebbero essere i vantaggi di tale metodo.

15.9 Dite quali vantaggi porta l'uso di dispositivi specifici come instradatori e gateway. Individuate gli svantaggi rispetto all'impiego di calcolatori d'uso generale

15.10 Dite in quali situazioni l'uso di un server dei nomi è più vantaggioso dell'uso di tabelle statiche nel calcolatore. Dite quali sono i problemi e le complicazioni connessi all'uso dei server dei nomi. Dite quali metodi si possono usare per ridurre il traffico generato dai server dei nomi per soddisfare le richieste di traduzione.

15.11 L'originario protocollo HTTP impiegava il TCP/IP come protocollo soggiacente: per ciascuna pagina, grafico o applet si costruiva, usava, quindi annullava una distinta sessione TCP. Con questo metodo, a causa del sovraccarico dovuto alla costruzione ed eliminazione delle connessioni TCP/IP, si verificavano problemi di prestazioni. Dite se l'uso dell'UDP al posto del TCP costituirebbe una buona alternativa e se è possibile fare altre modifiche che migliorino le prestazioni dell'HTTP.

15.12 Dite a cosa serve un protocollo di risoluzione degli indirizzi, e perché è meglio usare tale protocollo anziché far leggere ogni pacchetto a ciascun calcolatore per determinarne la destinazione. Dite se una rete a passaggio di contrassegno necessita di un tale protocollo. Spiegate la risposta.

15.13 Elencate i vantaggi e gli svantaggi della trasparenza di una rete di calcolatori per l'utente.

15.14 Elencate due difficilissimi problemi che i progettisti devono risolvere per realizzare un sistema di rete trasparente.

15.15 Date le differenze fra le architetture e i sistemi operativi, la migrazione dei processi all'interno di una rete eterogenea è generalmente impossibile. Descrivete un metodo per la migrazione dei processi fra diverse architetture nelle seguenti situazioni:
- a) se eseguono lo stesso sistema operativo;
- b) se eseguono sistemi operativi diversi.

15.16 Per costruire un sistema operativo robusto è necessario sapere quali tipi di malfunzionamento possono verificarsi.
- a) Elencate tre possibili tipi di malfunzionamento in un sistema distribuito.
- b) Individuate tra i malfunzionamenti elencati quelli che si possono verificare anche in un sistema centralizzato.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

15.17 Dite se è sempre cruciale sapere che il messaggio inviato è felicemente arrivato alla sua destinazione. Se la risposta è *sì*, spiegate perché; se la risposta è *no*, fornite esempi opportuni.

15.18 Presentate un algoritmo per la ricostruzione di un anello logico dopo il malfunzionamento di un processo dell'anello.

15.19 Considerate un sistema distribuito con due siti, *A* e *B*; giudicate se il sito *A* può distinguere tra i seguenti eventi:

a) *B* è fuori servizio;
b) la connessione tra *A* e *B* è fuori servizio;
c) *B* è estremamente sovraccarico e il tempo di risposta è 100 volte più lungo del normale.

Dite quali sono le implicazioni della risposta data in relazione al problema del ripristino nei sistemi distribuiti.

15.11 Note bibliografiche

[Tanenbaum 1996], [Stallings 2000a], [Kurose e Ross 2001] offrono una trattazione generale delle reti di calcolatori. [Williams 2001] tratta l'interconnessione dei calcolatori dal punto di vista dell'architettura dei calcolatori.

La rete Internet e parecchie altre reti sono discusse in [Quarterman e Hoskins 1986]. L'Internet e i suoi protocolli sono descritti in [Comer 1999] e [Comer 2000]. Una trattazione dei protocolli TCP/IP si trova in [Stevens 1994] e [Stevens 1995]. La programmazione nell'ambiente di rete UNIX è descritta esaurientemente in [Stevens 1997] e [Stevens 1998].

Discussioni concernenti le strutture dei sistemi operativi distribuiti si trovano in [Popek e Walker 1985] (il sistema Locus) e [Cheriton e Zwaenepoel 1983] (il *nucleo V*), [Ousterhout et al. 1988] (il sistema operativo di rete Sprite), [Balkovich et al. 1985] (Project Athena), e [Tanenbaum et al. 1990] e [Mullender et al. 1990] (il sistema operativo distribuito Amoeba). Un confronto tra Amoeba e Sprite è offerto da [Douglis et al. 1991]. [Mullender 1993] fornisce un'esauriente trattazione di molti argomenti riguardanti l'elaborazione distribuita.

Discussioni riguardanti il bilanciamento e la condivisione del carico sono presentate da [Harchol-Balter e Downey 1997] e [Vee e Hsu 2000]. [Harish e Owens 1999] discute il bilanciamento del carico dei server DNS. La migrazione dei processi è trattata da [Han e Ghosh 1998] e [Milo et al. 2000]. [Artsy 1989] è un'edizione speciale sulla migrazione dei processi.

[Nichols 1987], [Mutka e Livny 1987] e [Litzkow et al. 1988] presentano schemi di condivisione delle stazioni di lavoro inattive in un ambiente di calcolo distribuito condiviso. [Birman e Joseph 1987] trattano l'affidabilità della comunicazione alla presenza di guasti. L'integrazione della sicurezza nei grandi sistemi distribuiti è trattata in [Satyanarayanan 1989].

Capitolo 16

File system distribuiti

Nel Capitolo 15 sono trattati la costruzione di una rete e i protocolli di basso livello necessari per il trasferimento di messaggi tra i sistemi. In questo capitolo si affronta l'uso di questa infrastruttura. Un **file system distribuito** (*distributed file system* — DFS) è una realizzazione distribuita del modello classico di un file system in un sistema a partizione del tempo, dove più utenti condividono file e risorse di memorizzazione (Capitolo 12). Un DFS consente di estendere lo stesso tipo di condivisione al caso in cui i file sono fisicamente dispersi tra i siti di un sistema distribuito.

In questo capitolo si presentano diversi metodi per progettare e realizzare un DFS. Innanzi tutto si considerano i concetti comuni sui quali si fondano i DFS, per poi illustrarli attraverso l'esame di un influente DFS — l'*Andrew File System* (AFS).

16.1 Introduzione

Un sistema distribuito è un insieme di macchine debolmente connesse tramite una rete di comunicazione. Il termine *macchina* si usa per indicare sia un mainframe sia una stazione di lavoro. Una determinata macchina di un sistema distribuito considera *remote* le restanti macchine e le rispettive risorse, mentre considera *locali* le proprie risorse.

In questo capitolo si usa il termine DFS per intendere i file system distribuiti in generale, non il prodotto commerciale DFS della Transarc. A quest'ultimo si fa riferimento col termine *Transarc DFS*.

Per descrivere la struttura di un DFS occorre definire innanzi tutto i termini *servizio*, *server* e *client*. Il **servizio** è un programma in esecuzione in una o più macchine e fornisce un tipo particolare di funzione a client sconosciuti a priori; il **server** è un programma di servizio in esecuzione in una singola macchina; il **client** è un processo che può richiedere un servizio servendosi di una serie di operazioni che costituiscono la sua interfaccia, detta **interfaccia del client**. Talvolta si definisce un'interfaccia di livello inferiore per l'effettiva interazione tra le macchine; l'**interfaccia intermacchina**.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Secondo questa terminologia, si può affermare che un file system fornisce servizi ai suoi client. Un'interfaccia del client per un servizio relativo ai file è composta da un insieme di operazioni primitive su file, come la creazione di un file, la cancellazione, la lettura e la scrittura. Il principale elemento fisico controllato da un file server è composto da un insieme di dispositivi locali di memoria secondaria, generalmente dischi magnetici, nei quali si memorizzano i file e dai quali si recuperano gli stessi secondo le richieste dei client.

Un DFS è un file system i cui client, server e dispositivi di memorizzazione sono sparsi tra le macchine di un sistema distribuito. Di conseguenza l'attività di servizio si deve eseguire attraverso la rete e, anziché un unico magazzino di dati centralizzato, il sistema ha più dispositivi di memorizzazione indipendenti. La configurazione e la realizzazione concrete di un DFS possono essere di vario tipo. In alcune configurazioni i server sono eseguiti su macchine specifiche, in altre una macchina può essere sia un server sia un client. Un DFS si può realizzare come parte di un sistema operativo distribuito o, in alternativa, con uno strato di programmi il cui compito consiste nella gestione della comunicazione tra sistemi operativi convenzionali e file system. Le caratteristiche che contraddistinguono un DFS sono la molteplicità e l'autonomia dei client e dei server del sistema.

In teoria un DFS deve apparire ai client come un file system centralizzato convenzionale. La molteplicità e la dispersione dei suoi server e dispositivi di memorizzazione devono essere rese trasparenti. Ciò significa che l'interfaccia del client di un DFS non deve distinguere tra file locali e file remoti. Spetta al DFS localizzare i file e predisporre il trasporto dei dati. Un DFS **trasparente** facilita la mobilità dell'utente portando il suo ambiente, cioè la directory iniziale, ovunque egli apra una sessione.

La misura delle prestazioni più importante per un DFS è rappresentata dal tempo necessario a soddisfare le richieste di servizio. Nei sistemi convenzionali questo tempo è dato dal tempo d'accesso al disco e da una piccola quantità di tempo di CPU. In un DFS, invece, un accesso remoto risente anche di un ulteriore carico dovuto alla struttura distribuita. In questo carico rientrano il tempo necessario per inviare la richiesta a un server e il tempo necessario per ricevere la risposta. Pertanto per ogni direzione occorre considerare, oltre al trasferimento dell'informazione, anche il carico della CPU dovuto all'esecuzione del protocollo di comunicazione. Le prestazioni di un DFS si possono considerare come un'altra misura della sua trasparenza. Ciò significa che le prestazioni di un DFS ideale devono essere paragonabili a quelle di un file system convenzionale.

Il fatto che un DFS gestisca un insieme di dispositivi di memorizzazione sparsi è la caratteristica che contraddistingue il DFS stesso. Lo spazio di memorizzazione totale gestito da un DFS è composto da diversi spazi di memorizzazione più piccoli, localizzati in posizioni remote. Generalmente questi spazi di memorizzazione costituenti corrispondono a insiemi di file. Un'**unità componente** è il più piccolo insieme di file che si può memorizzare in una singola macchina, indipendentemente da altre unità. Tutti i file che appartengono alla stessa unità componente devono risiedere nella stessa locazione.

16.2 Nominazione e trasparenza

La **nominazione** è una funzione di associazione tra oggetti logici e oggetti fisici. Ad esempio, gli utenti trattano oggetti di dati logici, rappresentati dai nomi dei file, mentre il sistema manipola blocchi fisici di dati, memorizzati sulle tracce di un disco. Generalmente l'utente fa riferimento a un file attraverso un nome testuale. A quest'ultimo si fa corrispondere un identificatore numerico di livello inferiore che a sua volta si fa corrispondere ai blocchi dei dischi. Quest'associazione a più livelli fornisce agli utenti un'astrazione del file che nasconde i particolari concernenti la sua memorizzazione.

In un DFS trasparente, all'astrazione si aggiunge un nuovo aspetto: nascondere la posizione all'interno della rete. In un file system convenzionale un valore della funzione di nominazione è un indirizzo in un disco. In un DFS si estende l'insieme dei valori in modo da comprendere anche la specifica macchina nel cui disco è contenuto il file. Astraendo ulteriormente il concetto di file, è possibile giungere alla **replicazione dei file**. Dato il nome di un file, il sistema di associazione riporta una serie di locazioni delle repliche di questo file. In quest'astrazione sono nascoste sia l'esistenza di copie multiple sia la loro locazione.

16.2.1 Strutture di nominazione

Occorre distinguere due nozioni correlate riguardo al sistema di associazione dei nomi di un DFS:

1. **Trasparenza di locazione.** Il nome di un file non rivela alcun indizio sulla sua locazione fisica.
2. **Indipendenza dalla locazione.** Non si deve modificare il nome di un file se cambia la sua locazione di memoria fisica.

Entrambe le definizioni si riferiscono al livello di nominazione discusso precedentemente, poiché i file hanno nomi diversi a livelli diversi: al livello d'utente si usano nomi, mentre al livello del sistema si usano gli identificatori numerici. L'associazione dinamica è uno schema di nominazione indipendente dalla locazione, poiché può far corrispondere locazioni diverse in due momenti diversi allo stesso nome di file. Perciò l'indipendenza dalla locazione è una caratteristica più forte di quanto non lo sia la trasparenza di locazione.

In pratica la maggior parte degli attuali DFS offre un'associazione statica con trasparenza di locazione per i nomi al livello d'utente. Tuttavia questi schemi non gestiscono la **migrazione dei file**. Ciò significa che è impossibile cambiare automaticamente la locazione di un file, e quindi, per questi schemi, la nozione di indipendenza dalla locazione è irrilevante. I file si associano in modo permanente a uno specifico gruppo di blocchi di dischi. I file e i dischi si possono spostare manualmente da una macchina all'altra, ma la migrazione dei file implica un'azione automatica da parte del sistema operativo. Soltanto l'AFS e alcuni file system sperimentali gestiscono l'indipendenza dalla locazione e la

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

mobilità dei file. L'AFS gestisce la mobilità dei file soprattutto per scopi amministrativi; un protocollo fornisce la migrazione delle unità componenti dell'AFS per soddisfare le richieste ad alto livello da parte degli utenti, senza modificare i nomi al livello d'utente o i nomi a basso livello dei corrispondenti file.

L'indipendenza dalla locazione e la trasparenza di locazione statica si differenziano ancora per altri aspetti:

- La separazione dei dati dalla locazione, come nel caso dell'indipendenza dalla locazione, offre una migliore astrazione per i file. Il nome di un file deve indicare gli attributi più significativi del file stesso, come il suo contenuto, piuttosto che la sua locazione. I file indipendenti dalla locazione si possono considerare come contenitori di dati logici che non si assegnano a una specifica locazione di memoria secondaria. Se è prevista soltanto la trasparenza di locazione statica, il nome del file continua a indicare un gruppo specifico, anche se nascosto, di blocchi fisici di dischi.
- La trasparenza di locazione statica fornisce agli utenti un modo per condividere convenientemente i dati. Gli utenti possono condividere file remoti impiegando la trasparenza di locazione per compiere la nominazione dei file, esattamente come se questi fossero locali. Ciononostante la condivisione dello spazio di memorizzazione è scomoda poiché i nomi logici sono vincolati in modo statico ai dispositivi fisici di memorizzazione. L'indipendenza dalla locazione consente la condivisione dello spazio di memorizzazione stesso e degli oggetti di dati. Quando i file si possono rendere mobili, lo spazio di memorizzazione dell'intero sistema assume l'aspetto di una singola risorsa virtuale. Un vantaggio dato da questa visione consiste nella possibilità di bilanciare l'utilizzo dei dischi all'interno del sistema.
- L'indipendenza dalla locazione separa la gerarchia di nominazione dalla gerarchia dei dispositivi di memorizzazione e dalla struttura esistente tra i calcolatori. Al contrario, usando la trasparenza di locazione statica, si può evidenziare la corrispondenza tra unità componenti e macchine anche se i nomi sono trasparenti. Le macchine sono configurate impiegando un modello simile alla struttura di nominazione, questa configurazione può forzare l'architettura del sistema a rispettare vincoli non necessari, ed è anche in conflitto con altre considerazioni. Un server che gestisca una directory radice è un esempio di struttura dettata dalla gerarchia di nominazione e contraddice le direttive di decentramento.

Una volta completata la separazione tra nome e locazione, i client possono accedere a file che risiedono in sistemi server remoti. Infatti questi client possono essere privi di dischi e adoperare un server per l'uso di tutti i file, compreso il nucleo del sistema operativo. Per la sequenza d'avviamento sono però necessari protocolli speciali; si consideri il problema di trasferire il nucleo a una stazione di lavoro senza dischi. Poiché tale stazione non ha il nucleo, non può usare il codice del DFS per recuperarlo. Invece s'impiega uno speciale protocollo d'avviamento, contenuto nella memoria di sola lettura (ROM) del client, che abilita la connessione alla rete e recupera solo un file speciale, il nucleo o il codice d'avviamento da una locazione fissata. Una volta che il nucleo è stato copiato attraverso la rete e caricato, il suo DFS rende disponibili tutti gli altri file del sistema operativo. I vantaggi dati da client senza dischi sono molti, tra i quali il minor costo (le macchine sono appunto prive di dischi) e una maggiore convenienza (quando si aggiorna un sistema operativo, occorre modificare solo il server anziché tutti i client). Lo svantaggio invece è dato dalla maggiore complessità dei protocolli d'avviamento e dal calo delle prestazioni dovuto all'uso di una rete anziché di un disco locale.

L'attuale tendenza è verso client con dischi locali. La capacità delle unità a disco è in rapida crescita, mentre il loro prezzo è in diminuzione e nuove generazioni appaiono ogni anno, o quasi. La stessa cosa non si può dire per le reti, che si evolvono ogni cinque o dieci anni. Complessivamente i sistemi evolvono molto più rapidamente delle reti, quindi è necessario un ulteriore lavoro per limitare gli accessi alla rete in modo da migliorare la produttività del sistema.

16.2.2 Schemi di nominazione

Sono disponibili tre metodi principali per gli schemi di nominazione in un DFS. Il metodo più semplice prevede di nominare i file per mezzo di una combinazione del loro nome di macchina e nome locale; ciò garantisce un unico nome su tutto il sistema. Nel sistema Ibis, ad esempio, un file è identificato unicamente dal nome *macchina:nome-locale*, dove *nome-locale* indica un percorso di tipo UNIX. Questo schema di nominazione non gode della trasparenza di locazione e tanto meno dell'indipendenza dalla locazione. Ciononostante si possono usare le stesse operazioni sia per i file locali sia per i file remoti. La struttura del DFS è rappresentata da un insieme di unità componenti isolate costituite da interi file system convenzionali. In questo primo metodo le unità componenti rimangono isolate, anche se sono disponibili mezzi per fare riferimento ai file remoti. Tale schema non si prende più in considerazione in questo testo.

Il secondo metodo si è diffuso col Network File System della Sun (NFS). L'NFS è il file system dell'ONC+, un sistema di rete compreso in molte versioni del sistema operativo UNIX. L'NFS offre i mezzi per unire le directory remote alle directory locali, dando all'utente l'impressione di un albero di directory coerente. Nelle prime versioni era possibile accedere in modo trasparente solo a directory remote montate in precedenza. Con l'avvento della funzione di **automontaggio** i montaggi si eseguono su richiesta secondo una tabella di punti di montaggio e nomi di strutture di file. I componenti sono integrati per gestire la condivisione trasparente, sebbene tale integrazione sia limitata e non uniforme poiché ogni macchina può aggiungere diverse directory remote al proprio albero. La struttura risultante è incostante; normalmente si ottiene una foresta di alberi UNIX con sottoalberi condivisi.

L'integrazione totale dei componenti del file system si ottiene per mezzo del terzo metodo. Una sola struttura globale di nomi si estende a tutti i file del sistema. In teoria, la struttura composta del file system è *isomorfa* alla struttura di un file system convenzionale. In pratica, comunque, esistono molti file speciali (ad esempio i file che rappresentano i dispositivi e le directory dei file eseguibili della specifica macchina) che rendono difficile il conseguimento di questo scopo.

http://www.abulpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per valutare le strutture di nominazione se ne considera la **complessità amministrativa**. La struttura più complessa e più difficile da mantenere è la struttura dell'NFS. Poiché ogni directory remota può essere montata in qualsiasi punto dell'albero di una directory locale, la gerarchia risultante può essere quasi priva di struttura. Se un server diventa non disponibile, diventa non disponibile anche un insieme arbitrario di directory di diverse macchine. Inoltre, un meccanismo separato di accreditamento controlla quale macchina può aggiungere una determinata directory al suo albero. Così un utente può ottenere l'accesso a un albero di directory remoto in un client, ma vederselo negare in un altro client.

16.2.3 Realizzazione

La realizzazione della nominazione trasparente richiede un sistema di associazione per far corrispondere le locazioni ai nomi dei file. Per mantenere gestibile tale corrispondenza occorre aggregare insiemi di file in unità componenti e fornire una corrispondenza secondo l'unità componente anziché secondo il singolo file. Detta aggregazione è utile anche per scopi amministrativi. I sistemi del tipo UNIX impiegano l'albero gerarchico delle directory per fornire la corrispondenza tra nomi e locazioni e per aggregare i file nelle directory in modo ricorsivo.

Per aumentare la disponibilità delle informazioni fondamentali su tali associazioni si possono usare metodi come la replicazione, l'uso di cache locali, o entrambi. L'indipendenza dalla locazione implica che l'associazione cambi col tempo; quindi, replicando l'associazione, diventa impossibile ottenere un aggiornamento semplice e coerente di queste informazioni. Per superare quest'ostacolo si può adoperare una tecnica che introduce **identificatori di file indipendenti dalla locazione** di basso livello. Ai nomi dei file si fanno corrispondere identificatori di file di basso livello, che indicano a quale unità componente appartiene il file; tali identificatori sono comunque indipendenti dalla locazione. Si possono replicare liberamente e copiare in una cache, senza che siano invalidati dalla migrazione delle unità componenti. L'inevitabile prezzo di tale tecnica è un secondo livello di associazione che metta in corrispondenza le unità componenti con le locazioni e richiede un meccanismo di aggiornamento semplice e coerente. La realizzazione degli alberi di directory di tipo UNIX, che impiega questi identificatori di basso livello indipendenti dalla locazione, rende invariante l'intera gerarchia nel caso della migrazione di unità componenti. L'unica variazione riguarda l'associazione delle locazioni delle unità componenti.

Un metodo diffuso per realizzare questi identificatori di basso livello consiste nell'uso di nomi strutturati. Questi nomi sono composti da sequenze di bit formate normalmente da due parti: la prima identifica l'unità componente alla quale appartiene il file; la seconda identifica il file all'interno dell'unità. Sono possibili varianti con più parti. L'aspetto invariante dei nomi strutturati, comunque, è che le parti individuali del nome sono uniche in ogni momento soltanto nel contesto delle parti restanti. L'unicità in ogni momento si può ottenere avendo cura di non riutilizzare un nome già usato, oppure aggiungendo ulteriori bit (metodo utilizzato nell'AFS) o impiegando una marca temporale come se fosse una parte del nome (come accade nell'Apollo Domain). Un altro metodo

per svolgere questo processo consiste nel prendere un sistema con trasparenza di locazione, come l'Ibis, e aggiungervi un altro livello di astrazione per produrre uno schema di nominazione con indipendenza dalla locazione.

L'uso delle tecniche di aggregazione di file in unità componenti e di identificatori di file di basso livello indipendenti dalla locazione è esemplificato nell'AFS.

16.3 Accesso ai file remoti

Si consideri un utente che richieda l'accesso a un file remoto. Supponendo che il server che contiene il file sia stato localizzato dallo schema di nominazione, per soddisfare la richiesta d'accesso dell'utente remoto è necessario compiere l'effettivo trasferimento dei dati.

Tale trasferimento si può ottenere col **meccanismo del servizio remoto**, nel quale le richieste d'accesso sono inviate al server che esegue gli accessi e ritrasmette i risultati all'utente. Uno dei modi più diffusi per realizzare il servizio remoto è il paradigma della chiamata di procedura remota (RPC), descritto nel Paragrafo 4.6.2.

Esiste un'analogia diretta tra il metodo d'accesso al disco nei file system convenzionali e il metodo del servizio remoto in un DFS: l'uso del servizio remoto è analogo al compiere un accesso al disco per ciascuna richiesta d'accesso.

Per assicurare prestazioni ragionevoli del meccanismo di servizio remoto si usano delle cache. Nei file system convenzionali lo scopo delle cache consiste nel ridurre gli accessi ai dischi, aumentando così le prestazioni; nei DFS lo scopo è quello di ridurre il traffico nella rete e gli accessi ai dischi. Nel seguito si discute la gestione delle cache in un DFS, che si confronta col paradigma del servizio remoto di base.

16.3.1 Uso delle cache

Il concetto alla base dell'uso delle cache è semplice; se i dati necessari a soddisfare la richiesta d'accesso non sono già stati copiati nella cache, si trasferisce una copia di quei dati dal server al sistema client. Gli accessi richiesti si eseguono alla memoria cache. L'idea è di tenere nella cache blocchi di disco ai quali si è fatto riferimento di recente, gestendo localmente gli accessi ripetuti alle stesse informazioni, e riducendo il traffico della rete. Un criterio di sostituzione (ad esempio LRU) limita la dimensione della cache. Non esiste una corrispondenza diretta fra gli accessi e il traffico verso il server: i file sono ancora identificati con una copia principale che risiede nella macchina server, ma altre copie del file o di parti di esso sono sparse in diverse cache. Se si modifica una copia contenuta in una cache, le modifiche devono essere riportate anche sulla copia principale in modo da conservare la coerenza dei dati. Il problema di mantenere le copie coerenti con il file principale si chiama **problema della coerenza della cache** (Paragrafo 16.3.4). L'impiego delle cache in un DFS si può chiamare semplicemente **memoria virtuale di rete**: infatti funziona in modo analogo alla memoria virtuale con paginazione su richiesta, con la differenza che in questo caso la memoria ausiliaria non è un disco locale, ma un server remoto.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Nel caso di un DFS, la dimensione dei dati da copiare nelle cache può variare dai blocchi di file ai file interi. In genere si copiano più dati di quanti non siano necessari a soddisfare un singolo accesso, in modo da aumentare le probabilità di soddisfare più accessi a tali dati. Questo procedimento è molto simile alla lettura anticipata dei dati dai dischi (Paragrafo 12.6.2). L'AFS copia i file in grandi sezioni (64 KB); gli altri sistemi citati in questo capitolo copiano blocchi singoli in modo controllato dalle richieste dei client. Aumentando la quantità unitaria di dati copiati, aumenta il tasso di successi, ma aumenta anche il costo di ogni insuccesso nella ricerca nella cache, poiché ogni insuccesso richiede il trasferimento di una maggiore quantità di dati, e aumentano le possibilità dei problemi di coerenza. Nella scelta della quantità di dati unitaria da copiare nelle cache occorre considerare parametri come l'unità di trasferimento della rete e l'unità di servizio del protocollo delle RPC, nel caso in cui questo sia usato. L'unità di trasferimento della rete (nell'Ethernet è un pacchetto) è di circa 1,5 KB, perciò le unità di dati più grandi devono essere scomposte prima di essere inviate e quindi ricomposte al momento della ricezione.

Naturalmente la dimensione dei blocchi e la dimensione totale della memoria cache sono importanti per gli schemi di gestione delle cache a blocchi. Nei sistemi di tipo UNIX le dimensioni ordinarie dei blocchi sono di 4 KB o 8 KB. Per cache di grandi dimensioni (oltre 1 MB) si usano blocchi più grandi (oltre 8 KB); per cache più piccole, i blocchi di grandi dimensioni sono meno utili poiché implicano un numero inferiore di blocchi contenuto nelle cache, e un tasso di successi inferiore.

16.3.2 Locazione delle cache

Ci si può chiedere che tipo di cache si debba usare. Le cache su dischi hanno un evidente vantaggio rispetto alle cache nella memoria centrale: sono affidabili. Se la cache è nella memoria volatile, le modifiche ai dati in essa contenuti si perdono nel caso di un crollo del sistema. Inoltre, se la cache è mantenuta nella memoria secondaria, i dati continuano a trovarvisi anche durante il ripristino, e non è necessario prelevarli di nuovo. D'altra parte le cache nella memoria centrale hanno parecchi vantaggi specifici:

- Le cache nella memoria centrale permettono l'uso di stazioni di lavoro prive di dischi.
- È possibile accedere più rapidamente ai dati in una cache nella memoria centrale, che ai dati in una cache su disco.
- Attualmente si tende a produrre memorie più grandi e meno costose. Si prevede che l'incremento delle prestazioni supererà in importanza i vantaggi offerti dalle cache su dischi.
- Le cache dei server, impiegate per accelerare l'I/O dei dischi, si trovano nella memoria centrale a prescindere dalla locazione delle cache degli utenti; se si usano le cache nella memoria centrale anche per le macchine degli utenti, si può realizzare un unico meccanismo di gestione utilizzabile sia dai server sia dagli utenti.

Molti sistemi d'accesso remoto si possono considerare come un ibrido fra l'uso di cache e servizio remoto. Nell'NFS, ad esempio, il servizio remoto, è affiancato dall'uso di cache nella memoria sia da parte del client sia da parte del server per incrementare le prestazioni. D'altra parte, il sistema Sprite si fonda sull'uso delle cache, ma in alcuni casi adotta il metodo del servizio remoto. Così nel valutare i due metodi si considera fino a che punto è enfatizzato ciascun metodo.

Il protocollo NFS e la maggior parte delle sue realizzazioni non offrono la gestione di cache su dischi. Recenti versioni dell'NFS per il sistema operativo Solaris (Solaris 2.6 e successive) comprendono la possibilità di gestione di cache su dischi da parte dei client (il file system *cachefs*). Una volta che legge blocchi di file dal server, il client dell'NFS li copia nella propria memoria e in un proprio disco. Se si cancella la copia nella memoria o se si riavvia il sistema, si fa riferimento alla copia nel disco. Se il blocco richiesto non è né nella memoria né nella cache nel disco, s'invia una RPC al server per recuperare il blocco, che viene scritto nella cache del disco e nella cache della memoria a uso del client.

16.3.3 Criteri di aggiornamento delle cache

Il criterio che si usa per riscrivere blocchi di dati modificati nella copia principale del server ha un effetto critico sulle prestazioni e sull'affidabilità del sistema. Il criterio più semplice consiste nello scrivere direttamente i dati nei dischi non appena questi vengono modificati in una cache qualsiasi. Il vantaggio della **scrittura diretta** è l'affidabilità; se un sistema client si guasta, si perdono solo poche informazioni. Tuttavia questo criterio richiede che ogni accesso per scrittura attenda l'invio delle informazioni al server, il che implica scarse prestazioni delle operazioni di scrittura. L'uso di cache con scrittura diretta corrisponde all'uso del servizio remoto per le scritture e allo sfruttamento delle cache per le sole letture.

Un'alternativa è il criterio di **scrittura differita**, che consiste nel differire gli aggiornamenti della copia principale. Le modifiche si scrivono nella cache e successivamente nel server. Questo criterio ha due vantaggi rispetto alla scrittura diretta: innanzi tutto, poiché le scritture sono dirette alla cache, gli accessi per scrittura sono molto più rapidi; in secondo luogo, si possono sovrascrivere i dati prima che siano riportati al server, in tal caso è necessario riportare solo l'ultimo aggiornamento. Sfortunatamente gli schemi di scrittura differita causano problemi di affidabilità, poiché se una macchina utente subisce un guasto si perdono i dati non scritti.

Le varianti del criterio della scrittura differita si differenziano secondo il momento in cui i blocchi di dati modificati sono inviati al server. Un'altra possibilità è quella che prevede l'invio di un blocco quando questo sta per essere espulso dalla cache del client. Ciò può avere esiti positivi sulle prestazioni, ma può accadere che alcuni blocchi risiedano per un lungo periodo nella cache del client prima di essere riscritti nel server. Un compromesso tra questo metodo e la scrittura diretta consiste nell'esaminare la cache a intervalli regolari e inviare al server solo i blocchi modificati dopo l'ultima verifica, esattamente come nello UNIX si scandisce la cache locale. Nel sistema Sprite s'impiega que-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sto criterio con un intervallo di verifica di 30 secondi. L'NFS adopera questo criterio anche per i file di dati, ma una volta che una scrittura è stata inviata al server durante uno svuotamento della cache, per essere considerata completa, deve giungere al disco del server. L'NFS tratta i metadati (i dati sulle directory e i dati sugli attributi dei file) in modo differente: tutti i cambiamenti dei metadati s'inviano in modo sincrono ai relativi server. In questo modo si evitano le perdite di file e le alterazioni delle strutture di directory in seguito a un crollo di un client o di un server.

Per l'NFS con la *cachefs*, le scritture nel server si riportano anche nella cache locale su disco per mantenere la coerenza di tutte le copie. Così l'NFS con la *cachefs* migliora le prestazioni dell'NFS standard per le richieste di lettura con successo della *cachefs*, ma le riduce per le richieste di lettura o scrittura con fallimento della cache. Come con tutte le cache, per ottenere buone prestazioni, è essenziale avere un alto tasso di successi.

Un'altra variante della scrittura differita consiste nello scrivere i dati nel server quando si chiude un file. Questo criterio, detto di **scrittura su chiusura**, è adottato dal sistema AFS. Nel caso in cui si aprano i file per periodi brevi oppure si modifichino solo di rado, questo criterio non riduce significativamente il traffico presente nella rete. Inoltre il criterio di scrittura su chiusura richiede che il processo in chiusura sia ritardato per permettere la scrittura diretta del file, ciò ne riduce i vantaggi. Le migliori prestazioni di questo criterio, rispetto a quelli della scrittura differita con un più frequente invio al server, sono più evidenti quando i file rimangono aperti per lunghi periodi e vengono modificati spesso.

16.3.4 Coerenza

Una macchina client deve affrontare il problema di decidere se una copia in una cache sia o no coerente con la copia principale, e quindi se possa essere usata. Se la macchina client stabilisce che i suoi dati nella cache non sono aggiornati, occorre ottenere una copia aggiornata. Per compiere tale verifica si possono seguire due metodi:

1. **Metodo iniziato dal client.** Il client inizia un controllo di validità mettendosi in contatto col server per controllare se i dati locali sono coerenti con la copia principale. La frequenza del controllo di validità è il fulcro di questo metodo e determina la conseguente semantica della coerenza. Si può fare un controllo prima di ogni accesso, fino a scendere a un solo controllo durante il primo accesso a un file (normalmente quando il file viene aperto). Ogni accesso unito a un controllo di validità è ritardato rispetto a un accesso servito immediatamente dalla memoria cache. In alternativa si può iniziare un controllo a intervalli di tempo prefissati. Secondo la frequenza d'esecuzione il controllo di validità può caricare sia la rete sia il server.
2. **Metodo iniziato dal server.** Il server registra, per ogni client, i file (o le parti di file) che copia nella cache e interviene se individua una potenziale incoerenza. Una situazione di questo tipo si verifica quando due diversi client copiano un file in una cache in modi conflittuali. Nel caso della semantica UNIX (Paragrafo 11.5.4), si possono risolvere le potenziali incoerenze con il server che svolge un ruolo attivo.

Il server deve essere informato ogni volta che un file viene aperto, e per ogni apertura deve essere indicato il modo (lettura o scrittura). Se dispone di queste informazioni, il server può intervenire, quando individua un file che viene aperto contemporaneamente in modalità conflittuali, disabilitando la possibilità di copiare quel file nelle cache. Tale disabilitazione significa un passaggio a un modo di funzionamento di servizio remoto.

16.3.5 Confronto tra uso delle cache e servizio remoto

La scelta tra l'uso di cache e del servizio remoto può condurre a un incremento delle prestazioni contro una diminuzione della semplicità. I due metodi hanno diversi vantaggi e svantaggi:

- L'uso di cache locali permette di gestire in modo efficiente un consistente numero di accessi remoti. Traendo vantaggio dal carattere di località delle sequenze d'accesso ai file, l'uso delle cache diventa ancora più interessante; quindi la maggior parte degli accessi remoti viene servita velocemente quanto gli accessi locali. Di conseguenza si riducono il carico dei server e il traffico della rete, e si aumenta il potenziale di scalabilità. Per contro, quando si usa il servizio remoto, ogni accesso remoto è gestito attraverso la rete. Con le ovvie conseguenze sul traffico della rete, il carico dei server e le prestazioni.
- Il carico totale della rete legato alla trasmissione di grosse porzioni di dati, come nel caso dell'uso di cache, è inferiore a quello che si ha quando si trasmettono serie di risposte a richieste specifiche, come con il metodo del servizio remoto. Inoltre le procedure d'accesso ai dischi nei server si possono ottimizzare meglio sapendo che le richieste sono sempre relative a segmenti di dati grandi e contigui, anziché a blocchi di disco casuali.
- Il problema della coerenza della cache è l'inconveniente principale. Se le scritture non sono frequenti, l'uso delle cache è certamente conveniente. Altrimenti i meccanismi impiegati per superare il problema della coerenza implicano un impegno considerevole per quel che riguarda le prestazioni, il traffico di rete e il carico dei server.
- Per trarre vantaggio dall'uso delle cache, si devono usare macchine con dischi locali, oppure grandi memorie centrali. Per l'accesso remoto con macchine prive di dischi e con ridotta capacità di memoria è più conveniente il metodo del servizio remoto.
- Poiché con l'uso delle cache i dati si trasferiscono in massa tra server e client anziché in risposta alle richieste specifiche di un'operazione su un file, l'interfaccia intermacchina inferiore è diversa dall'interfaccia d'utente superiore. Il paradigma del servizio remoto, d'altra parte, è solo un'estensione dell'interfaccia del file system locale attraverso la rete; quindi l'interfaccia tra le macchine rispecchia l'interfaccia locale tra l'utente e il file system.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

16.4 Servizio con informazioni di stato e servizio senza informazioni di stato

Esistono due metodi relativi alle informazioni del server: o il server tiene traccia di tutti i file cui ha accesso ogni client, oppure si limita a fornire i blocchi secondo quel che richiede il client, senza sapere che uso se ne farà.

Di seguito si descrive l'usuale scenario di un **servizio di file con informazioni di stato**. Prima di accedere a un file, un client deve eseguire su quel file un'operazione open. Il server preleva dai suoi dischi informazioni sul file, le registra nella propria memoria e fornisce al client un identificatore di connessione, unico per quel client e quel file aperto. Nei termini del sistema UNIX, il server preleva l'*inode* e fornisce al client un descrittore di file, che si usa come indice per una tabella di *inode* presente nella memoria centrale (*in-core*). Quest'identificatore si usa per gli accessi successivi, fino al termine della sessione. Un servizio con informazioni di stato è caratterizzato da una connessione tra il client e il server durante una sessione. Alla chiusura del file, oppure tramite un meccanismo di 'ripulitura' (*garbage collection*), il server può reclamare lo spazio di memoria centrale usato per client che non sono più attivi. Per quel che riguarda la tolleranza ai guasti, il punto chiave del servizio con informazioni di stato consiste nel fatto che il server conserva nella memoria centrale informazioni sui suoi client. L'AFS è un servizio di file con informazioni di stato.

Un **server senza informazioni di stato** evita l'uso di informazioni di stato rendendo ogni richiesta autocontenuta. Ciò significa che ogni richiesta identifica completamente il file e la posizione nel file, per le operazioni di lettura e di scrittura. Il server non deve tenere nella memoria centrale una tabella di file aperti, anche se in realtà ciò avviene per motivi d'efficienza. Inoltre non è necessario stabilire e terminare una connessione per mezzo delle operazioni open e close; sono del tutto superflue poiché ogni operazione su file è indipendente e non si considera parte di una sessione. L'apertura di un file da parte di processo client non causa la trasmissione di un messaggio remoto. Le letture e le scritture avvengono come messaggi remoti, o come ricerche nella cache. La chiusura finale da parte del client è di nuovo soltanto un'operazione locale. L'NFS è un servizio di file senza informazioni di stato.

Il vantaggio di un servizio con informazioni di stato rispetto a un servizio senza informazioni di stato è dato dalle migliori prestazioni. Le informazioni sul file si copiano in una cache nella memoria centrale ed è possibile accedervi tramite l'identificatore di connessione, risparmiando così accessi ai dischi. Inoltre un server con informazioni di stato può sapere se un file è stato aperto per l'accesso sequenziale e può quindi leggere in anticipo i blocchi successivi (i server senza informazioni di stato non possono poiché ignorano lo scopo delle richieste del client).

La differenza tra il servizio con informazioni di stato e il servizio senza informazioni di stato diviene evidente se si considerano gli effetti di un crollo del sistema durante un'attività di servizio. Un server con informazioni di stato perde completamente il proprio stato volatile, che occorre recuperare per assicurare un ripristino conveniente. Tale recupero generalmente si compie per mezzo di un protocollo di ripristino basato su un dialogo con i client. Un recupero meno elegante richiede che si annullino le operazioni in corso al momento del crollo. Un problema diverso è causato dai guasti ai client: il server deve essere informato di tali guasti, per poter reclamare lo spazio assegnato e registrare lo stato dei processi client guasti. Talvolta questo fenomeno si chiama **rilevamento ed eliminazione degli orfani**.

Un server senza informazioni di stato non incorre in questi problemi, poiché una volta riparato può rispondere senza difficoltà a una richiesta autocontenuta. Perciò gli effetti dei guasti ai server e il relativo ripristino passano pressoché inosservati. Dal punto di vista del client non ci sono differenze tra un server lento e un server in via di ripristino: se non riceve una risposta, il client continua a ritrasmettere la propria richiesta.

L'uso del robusto servizio senza informazioni di stato ha però uno svantaggio: i messaggi di richiesta sono più lunghi e l'elaborazione delle richieste è più lenta, poiché non si dispone d'informazioni caricate nella memoria centrale capaci di accelerare l'elaborazione. Inoltre il servizio senza informazioni di stato impone ulteriori vincoli nella progettazione del DFS. Innanzi tutto, poiché ogni richiesta identifica il file di destinazione, è necessario l'uso di uno schema di nominazione uniforme a basso livello su tutto il sistema. La traduzione dei nomi remoti in nomi locali per ogni richiesta rallenta anche l'elaborazione delle richieste. In secondo luogo, poiché i client ritrasmettono richieste per operazioni su file, queste operazioni devono essere idempotenti; ciò significa che ogni operazione, se si esegue più volte consecutivamente, deve avere sempre lo stesso effetto e dare sempre lo stesso risultato. Gli accessi per lettura e scrittura autocontenuti sono idempotenti finché adoperano un contatore assoluto che contiene la posizione d'accesso all'interno del file e non fanno affidamento su uno scostamento incrementale, come avviene con le chiamate del sistema read e write dello UNIX. Tuttavia occorre avere molta cura nella codificazione delle operazioni distruttive, come la cancellazione di un file, per rendere anch'esse idempotenti.

In alcuni ambienti un servizio con informazioni di stato diventa una vera necessità. Se il server usa il metodo iniziato dal server per la validazione della cache, non può fornire un servizio senza informazioni di stato, poiché registra quali file sono stati copiati nelle cache di quali client.

Il modo in cui lo UNIX si serve dei descrittori di file e degli scostamenti impliciti è intrinsecamente con informazioni di stato. I server devono conservare le tabelle che mettono in corrispondenza i descrittori di file con gli *inode*, e memorizzare lo scostamento corrente all'interno dei file. Questo requisito è richiesto perché l'NFS, che impiega un servizio senza informazioni di stato, non usa descrittori di file e include uno scostamento esplicito per ogni accesso.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

16.5 Replicazione dei file

La replicazione di file in macchine diverse rappresenta una ridondanza utile per migliorare la disponibilità; inoltre può influire positivamente anche sulle prestazioni: la scelta di una replica vicina per servire una richiesta d'accesso riduce il tempo di servizio.

Il requisito fondamentale di uno schema di replicazione consiste nel fatto che, nel caso di un guasto, repliche diverse dello stesso file risiedano in macchine indipendenti. Ciò significa che la disponibilità di una replica non dipende dalla disponibilità delle altre repliche. Quest'ovvio requisito implica che la gestione della replicazione si deve considerare un'attività intrinsecamente priva di trasparenza di locazione. Devono essere disponibili i mezzi per collocare una replica in una macchina particolare.

Conviene tenere nascosti agli utenti i particolari dell'operazione di replicazione. Spetta allo schema di nominazione associare un nome di file replicato a una replica particolare. Le repliche devono essere invisibili ai livelli superiori, ma devono essere differenziate le une dalle altre ai livelli più bassi per mezzo di nomi di livello inferiore diversi. Un'altra questione che riguarda la trasparenza è costituita dall'offerta del controllo della replicazione a livelli superiori. Il controllo della replicazione comprende la determinazione del grado e della disposizione delle repliche. In alcuni casi si può voler esporre questi particolari anche agli utenti. Il sistema Locus, ad esempio, offre agli utenti e agli amministratori del sistema opportuni meccanismi per controllare lo schema di replicazione.

Il problema principale è quello dell'aggiornamento delle repliche. Dal punto di vista dell'utente le repliche di un file indicano la stessa entità logica, e quindi l'aggiornamento di una replica deve riflettersi anche su tutte le altre repliche. Più precisamente, la relativa semantica della coerenza deve essere conservata quando gli accessi alle repliche sono considerati accessi virtuali al file logico originario. Se non è un fattore d'importanza centrale, la coerenza può essere sacrificata a vantaggio della disponibilità e delle prestazioni. In questo fondamentale compromesso nell'ambito della tolleranza ai guasti, si deve fare una scelta tra la conservazione della coerenza a ogni costo, che implica la creazione di un potenziale blocco indefinito, e il sacrificio della coerenza nei casi (si spera rari) di guasti catastrofici, a vantaggio di un procedere garantito. Il sistema Locus ad esempio impiega ampiamente la replicazione e sacrifica la coerenza nel caso in cui la rete sia partizionata, a vantaggio della disponibilità dei file per gli accessi per lettura e scrittura.

Il sistema Ibis usa una variante del metodo della copia primaria. Il dominio della funzione di associazione dei nomi è rappresentato da una coppia *<identificatore della replica primaria, identificatore della replica locale>*. Se non c'è alcuna replica locale si usa un valore speciale. In questo modo l'associazione è relativa a una macchina.

Se la replica locale è quella primaria, la coppia contiene due identificatori identici. L'Ibis gestisce la replicazione su richiesta; si tratta di un criterio per il controllo automatico della replicazione analogo alla copiatura nelle cache degli interi file. Con la replicazione su richiesta, la lettura di una replica non locale causa la sua copiatura nella cache locale, perciò si genera una nuova replica non primaria. Gli aggiornamenti si eseguono solo sulla copia primaria e causano l'invalidazione di tutte le altre repliche tramite l'invio di opportuni messaggi. Non è garantita l'invalidazione atomica e serializzata di tutte le repliche non primarie. Quindi una replica vecchia si può considerare valida. Per soddisfare gli accessi per scrittura remoti, si trasferisce la copia primaria alla macchina richiedente.

16.6 Un esempio: AFS

Il sistema Andrew era un ambiente di calcolo distribuito progettato e sviluppato dal 1983 presso la Carnegie Mellon University (CMU). Il file system del sistema Andrew (AFS) è la base per la condivisione delle informazioni tra i client di tale ambiente. In seguito, la Transarc Corporation, che ne aveva continuato lo sviluppo, è stata acquistata dall'IBM; la quale ne ha prodotto diverse versioni commerciali. L'AFS è anche stato scelto come il DFS di riferimento da un gruppo di aziende; tale intesa ha prodotto il *Transarc DFS*, che è parte dell'ambiente di calcolo distribuito DCE dell'OSF.

Nel 2000, il Transarc Lab dell'IBM ha annunciato che l'AFS sarebbe stato distribuito come prodotto *open-source* con l'*IBM Public Licence*. Il Transarc DFS è invece commercializzato normalmente. Molti rivenditori del sistema operativo UNIX e la Microsoft, hanno annunciato l'adozione del sistema DCE. Il lavoro di sviluppo continua con l'obiettivo di renderlo un DFS indipendente dalla piattaforma e universalmente accettato. L'AFS e il Transarc DFS sono molto simili, quindi in questo paragrafo si descrive l'AFS, sempre che non sia esplicitamente nominato il Transarc DFS.

L'AFS cerca di risolvere molti tra i problemi presenti in DFS più semplici, come l'NFS, ed è indubbiamente il DFS non sperimentale più ricco di funzioni. Offre uno spazio di nomi uniforme, la condivisione dei file indipendente dalla locazione, l'uso di cache da parte dei client con meccanismi di coerenza della cache e autenticazione sicura per mezzo del protocollo Kerberos. Prevede anche l'uso di cache da parte dei server nella forma di repliche, con alta disponibilità garantita dal reindirizzamento automatico verso una replica nel caso in cui un server sorgente diventi inutilizzabile. Uno degli attributi più importanti del sistema Andrew è la scalabilità: è progettato per comprendere oltre 5000 stazioni di lavoro. Considerando sia l'AFS sia il Transarc DFS esistono centinaia di versioni di questi sistemi.

16.6.1 Generalità

L'AFS distingue le *macchine client* (talvolta indicate col nome di *stazioni di lavoro*) dalle specifiche *macchine server*. I server e i client impiegavano originariamente solo il sistema operativo UNIX 4.2 BSD, ma l'AFS è stato adattato a molti sistemi operativi. I client e i server sono interconnessi tramite una rete di LAN e WAN.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I client sono presentati con uno spazio di nomi di file partizionato: uno **spazio di nomi locali** e uno **spazio di nomi condivisi**. Server specifici, chiamati nel loro insieme *Vice*, dal nome del programma che eseguono, presentano ai client lo spazio dei nomi condivisi nella forma di una gerarchia di file; tale gerarchia è omogenea, e ha caratteristiche di trasparenza di locazione. Lo spazio dei nomi locali è il file system radice di una stazione di lavoro, dal quale discende lo spazio dei nomi condivisi. Le stazioni di lavoro eseguono il protocollo *Virtue* per comunicare con il Vice e devono possedere dischi locali in cui memorizzare il loro spazio di nomi locali. L'insieme dei server è responsabile della memorizzazione e della gestione dello spazio dei nomi condivisi. Lo spazio dei nomi locali è piccolo; ne esiste uno per ogni stazione di lavoro e ciascuno contiene programmi di sistema fondamentali per un funzionamento autonomo e prestazioni migliori. Sono locali anche i file temporanei e i file che il proprietario della stazione di lavoro, per motivi privati, dichiara esplicitamente di voler memorizzare localmente.

Attraverso un esame più ravvicinato, è possibile notare che client e server sono strutturati in batterie (*cluster*) interconnesse tramite una WAN. Ciascuna batteria è composta da un insieme di stazioni di lavoro su una LAN e un rappresentante di Vice (il cosiddetto *cluster server*) che è collegato alla WAN tramite un instradatore. La decomposizione in batterie di macchine si fa soprattutto per affrontare il problema di scala. Per ottenere prestazioni ottimali, le stazioni di lavoro dovrebbero usare il server della batteria cui appartengono per la maggior parte del tempo, rendendo così relativamente poco frequenti i riferimenti ai file presenti in macchine appartenenti ad altre batterie.

L'architettura del file system è basata anche su considerazioni di scala. L'euristica di base consiste nello scaricare lavoro dei server sui client, tenendo conto del fatto che la velocità della CPU del server è la strozzatura del sistema. Seguendo questa euristica, si è scelto (come meccanismo chiave per le operazioni remote su file) di copiare i file nelle cache in grosse porzioni (64 KB). Tale caratteristica riduce la latenza di apertura del file e permette che le letture e le scritture siano dirette alla copia nella cache, senza chiamare in causa frequentemente i server.

Nel seguito si accennano brevemente alcuni altri aspetti dell'AFS:

- **Mobilità dei client.** I client possono accedere da qualsiasi stazione di lavoro a qualsiasi file che si trovi nello spazio dei nomi condivisi. Possono accusare qualche iniziale riduzione delle prestazioni dovuta alla copiatura dei file nelle cache quando accedono a file residenti in stazioni di lavoro diverse da quelle usuali.
- **Sicurezza.** Poiché le macchine Vice non eseguono programmi client, l'interfaccia rappresentata dal Vice è considerata il limite di fidatezza. Le funzioni di convalida e trasmissione sicura sono fornite come parte di un pacchetto di comunicazione basato sulla connessione, che si fonda sul paradigma RPC. Dopo la mutua convalida, un server Vice e un client comunicano tramite messaggi cifrati. La cifratura è eseguita da dispositivi o (meno rapidamente) da programmi. Le informazioni sui client e sui gruppi sono memorizzate in una base di dati di protezione, della quale esiste una copia in ogni server.

- **Protezione.** L'AFS fornisce liste d'accesso per la protezione delle directory e i normali bit dello UNIX per la protezione dei file. La lista d'accesso può contenere informazioni sugli utenti cui è consentito l'accesso a una directory, così come sugli utenti cui *non* è consentito l'accesso a una directory. Con questo schema è facile specificare che chiunque, eccetto, ad esempio, Gianni, può accedere a una directory. Il sistema Andrew prevede i seguenti tipi d'accesso: lettura, scrittura, ricerca, inserimento, amministrazione, blocco e cancellazione.
- **Eterogeneità.** La definizione di una chiara interfaccia col Vice rappresenta la chiave per l'integrazione delle architetture e dei sistemi operativi delle diverse stazioni di lavoro. Per facilitare l'eterogeneità, alcuni file della directory locale `/bin` sono collegamenti simbolici a file eseguibili di macchine specifiche residenti nel Vice.

16.6.2 Spazio dei nomi condivisi

Lo spazio dei nomi condivisi dell'AFS è composto da unità chiamate **volumi**; si tratta di unità insolitamente piccole, generalmente associate ai file di un singolo client. All'interno di una singola partizione del disco risiedono pochi volumi, che possono crescere (fino a una certa quota) e restringersi di dimensione. Dal punto di vista concettuale, i volumi vengono uniti tra loro per mezzo di un meccanismo simile al meccanismo di montaggio dello UNIX, ma la differenza di granularità è significativa, poiché nello UNIX è possibile montare soltanto un'intera partizione di disco (contenente un file system). I volumi rappresentano un'unità amministrativa fondamentale e hanno un ruolo essenziale nell'identificazione e nella localizzazione di un singolo file.

Un file o una directory Vice sono identificati da un identificatore di basso livello chiamato *fid*. Ogni elemento di directory dell'Andrew fa corrispondere un *fid* a un nome di percorso. Un *fid* è lungo 96 bit ed è composto da tre elementi di uguale lunghezza: un *numero di volume*, un *numero di vnode* e un *unicizzatore*. Il **numero di vnode** serve come indice per una matrice contenente gli *inode* dei file di un singolo volume. L'**unicizzatore** permette di riutilizzare numeri di *vnode*, mantenendo così compatte alcune strutture di dati. I *fid* sono trasparenti alla locazione, quindi gli spostamenti di file da server a server non invalidano il contenuto della directory che si trova nella cache.

Le informazioni sulla locazione sono conservate in una **base di dati locazione-volume**, della quale esiste una replica in ogni server. Un client può identificare la locazione di ogni volume del sistema interrogando questa base di dati, le cui dimensioni si mantengono gestibili grazie all'aggregazione dei file in volumi.

Per bilanciare lo spazio disponibile nei dischi e l'utilizzo dei server, si devono trasferire i volumi tra partizioni dei dischi e server. Quando s'invia un volume alla sua nuova locazione, nel server d'origine rimangono temporaneamente le informazioni riguardanti la spedizione, quindi non è necessario aggiornare in modo sincrono anche la base di dati delle locazioni. Durante il trasferimento del volume, il server originale può continuare a gestire aggiornamenti, che in seguito vengono spediti al nuovo server. A un certo punto, si disabilita il volume per il tempo necessario a elaborare le ultime modifiche appor-

http://www.ultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

tate; quindi il nuovo volume è nuovamente disponibile nel nuovo sito. L'operazione di spostamento dei volumi è atomica; se uno dei server si guasta, si annulla l'operazione.

La replicazione per la sola lettura, di un intero volume, per i file eseguibili e per i file con aggiornamenti poco frequenti, è gestita dai livelli superiori dello spazio dei nomi del Vice. La base di dati locazione-volume specifica il server contenente solo la copia per la lettura e scrittura di un volume e una lista dei siti delle repliche per la sola lettura.

16.6.3 Operazioni sui file e semantica della coerenza

Il principio fondamentale su cui si fonda l'architettura dell'AFS è la copiatura nelle cache di file interi eseguita dai server. Di conseguenza, una stazione di lavoro client interagisce con i server Vice soltanto durante l'apertura e la chiusura dei file; inoltre tale interazione non è sempre necessaria. La lettura e la scrittura di file, contrariamente a quanto accade con il metodo del servizio remoto, non causa alcuna interazione remota. Questa distinzione fondamentale influenza notevolmente le prestazioni e la semantica delle operazioni sui file.

Il sistema operativo di ogni stazione di lavoro intercetta le chiamate del sistema relative ai file e le invia a un processo al livello di client in quella stessa stazione di lavoro. Questo processo, chiamato *Venus*, copia nella cache i file provenienti dal Vice nel momento della loro apertura, e al momento della chiusura riporta le copie modificate dei file nei server di provenienza. Il Venus può contattare il Vice solo quando si apre o si chiude un file; la lettura e la scrittura di singoli byte in un file si eseguono direttamente sulla copia presente nella cache e aggirano il Venus. Il risultato di quest'operazione è che le scritture effettuate in un sito non sono immediatamente visibili negli altri siti.

I file nelle cache si sfruttano ulteriormente per le aperture successive. Se non è specificato diversamente, il processo Venus presuppone che gli elementi nella cache (file o directory) siano validi; perciò, all'apertura di un file, non deve far intervenire il Vice per validare la copia nella cache. Il meccanismo che gestisce questo criterio riduce drasticamente il numero delle richieste di validazione delle cache ricevute dai server. Tale meccanismo funziona come segue: quando un client copia nella propria cache un file o una directory, il server aggiorna le sue informazioni di stato registrando quest'operazione; a questo punto il client riceve un contrassegno di validità per quel file. Prima di permettere che un altro client modifichi il file, il server informa il client. In questo caso, si dice che il server revoca il contrassegno di validità al file per il primo client. Un client può aprire un file nella cache solo se il file è contrassegnato come valido. Se un client chiude un file dopo averlo modificato, tutti gli altri client che hanno copiato questo file nella propria cache perdono i loro contrassegni di validità. Perciò, quando questi client riapriranno il file dovranno ricevere dal server la sua nuova versione.

La lettura e la scrittura di byte di un file sono eseguite direttamente dal nucleo, senza l'intervento del Venus, sulla copia nella cache. Il processo Venus riacquista il controllo quando il file viene chiuso e, se il file è stato modificato localmente, aggiorna il file nel

server appropriato. Così, le uniche occasioni in cui il Venus contatta i server Vice sono le aperture dei file, che non sono nella cache o i cui contrassegni di validità sono stati revocati, e le chiusure dei file modificati localmente.

Sostanzialmente, l'AFS realizza la semantica delle sessioni. Le uniche eccezioni sono le operazioni sui file diverse dalle primitive di lettura e scrittura (come le modifiche di protezione al livello delle directory), che sono visibili ovunque nella rete subito dopo essere state completate.

Nonostante il meccanismo dei contrassegni di validità, è sempre presente un certo traffico per la validazione della cache, che normalmente si usa per sostituire i contrassegni di validità perduti a causa di guasti della macchina o della rete. Quando si riavvia una stazione di lavoro, il processo Venus considera sospetti tutti i file e tutte le directory copiate nelle cache e genera una richiesta di validazione della cache per il primo uso di ciascuno di questi elementi.

Il meccanismo dei contrassegni di validità impone a ogni server di mantenere le relative informazioni e a ogni client di mantenere le informazioni di validità. Se la quantità di tali informazioni conservata da un server è eccessiva, il server può interrompere l'uso dei contrassegni di validità e richiedere memoria, informando unilateralmente i client e revocando la validità dei loro file nelle rispettive cache. Se lo stato dei contrassegni di validità conservato dal Venus non è più sincronizzato con il corrispondente stato conservato dai server, si ha una potenziale incoerenza.

Ai fini della traduzione dei nomi di percorso, il processo Venus copia nelle cache anche il contenuto delle directory e i collegamenti simbolici. Si preleva ogni componente presente nel nome di percorso e si stabilisce per esso un contrassegno di validità se non è stato ancora copiato in una cache, oppure se il client non ha un contrassegno di validità per esso. Il Venus esegue localmente, per mezzo dei *fid*, le ricerche sulle directory prelevate. Non ci sono invii di richieste da un server all'altro. Alla fine dell'attraversamento del nome di percorso, tutte le directory intermedie e il file d'arrivo si trovano nella cache con i propri contrassegni di validità. Le successive operazioni di apertura per questo file non implicano alcuna comunicazione di rete, a meno che non sia stato revocato un contrassegno di validità per un componente del nome di percorso.

Le uniche eccezioni al metodo della copiatura nelle cache sono le modifiche alle directory apportate direttamente nel server responsabile di quelle directory per motivi d'integrità. Per tali scopi, l'interfaccia del Vice ha operazioni ben definite. Il Venus riporta le modifiche alla propria copia nella cache per evitare nuovi prelievi della directory.

16.6.4 Realizzazione

I processi client sono interfacciati a un nucleo UNIX con il solito insieme di chiamate del sistema. Il nucleo è leggermente modificato per individuare riferimenti ai file del Vice nelle relative operazioni e per inviare le richieste al processo Venus del livello client nella stazione di lavoro.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il processo Venus esegue la traduzione dei nomi di percorso componente per componente. Per evitare che si debba interrogare il server per le locazioni di volumi già note, ha una cache che associa volumi a locazioni di server. Se in questa cache non è presente un volume, il Venus si mette in contatto con i server con i quali ha già stabilito una connessione e richiede le informazioni sulla locazione che poi inserisce nella suddetta cache. Se il processo Venus non ha già stabilito una connessione con il server, ne stabilisce una nuova e la usa per prelevare il file o la directory. La connessione è necessaria ai fini della convalida e della sicurezza. Quando un file di arrivo viene trovato e copiato in una cache, se ne crea una copia nel disco locale. Il Venus quindi restituisce il controllo al nucleo, che apre la copia e rimanda la sua maniglia (*handle*) al processo client.

Il file system del sistema UNIX si usa come sistema di memorizzazione a basso livello sia per i server sia per i client. La cache del client è una directory locale situata in un disco della stazione di lavoro. All'interno di questa directory ci sono file i cui nomi hanno il ruolo di segnaposto per gli elementi della cache. Sia il Venus sia i processi server accedono ai file dello UNIX direttamente dagli *inode* di quest'ultimo, per evitare la costosa procedura di traduzione dei nomi di percorso in *inode* (*namei*). Poiché l'interfaccia interna degli *inode* non è visibile ai processi al livello di client (il processo Venus e i processi server sono processi al livello di client) è stato aggiunto un opportuno insieme di chiamate del sistema. L'AFS impiega il suo proprio file system annotato per migliorare le prestazioni e l'affidabilità rispetto all'UFS.

Il Venus gestisce due cache diverse: una per lo stato e l'altra per i dati. Per limitare le dimensioni di queste cache, impiega un semplice algoritmo LRU. Quando si rimuove un file dalla cache, il processo Venus informa il server appropriato affinché revochi il contrassegno di validità riguardante questo file. La cache di stato viene conservata nella memoria virtuale per permettere un rapido servizio delle chiamate del sistema `stat` (comunicazione dello stato del file). La cache dei dati risiede in un disco locale, ma il meccanismo di gestione delle aree di memoria per l'I/O dello UNIX impiega tali aree come cache per i blocchi di dischi in modo trasparente al processo Venus.

Un singolo processo al livello di client in ogni file server serve tutte le richieste di file provenienti dai client. Questo processo impiega una serie di processi leggeri con scheduling senza diritto di prelazione per servire in modo concorrente molte richieste dei client. Il sistema di RPC è integrato con quello dei processi leggeri, consentendo in tal modo al file server un'esecuzione concorrente, cioè il servizio di ciascuna RPC da parte di un processo leggero. Il sistema di RPC si trova al vertice di un'astrazione basata su datagrammi a basso livello. L'intero trasferimento di file è realizzato come effetto collaterale di tali RPC. Per ogni client esiste una connessione RPC, ma non esiste un legame a priori tra i processi leggeri e queste connessioni. Un gruppo di processi leggeri, invece, serve le richieste dei client per tutte le connessioni. L'uso di un solo processo server multithread permette la copiatura in una cache delle strutture di dati necessarie a soddisfare le richieste di servizio. D'altra parte, il malfunzionamento di un singolo processo server ha il disastroso effetto di paralizzare quel particolare server.

16.7 Sommario

Un DFS è un sistema di servizio dei file i cui client, server e dispositivi di memorizzazione sono sparsi tra i siti di un sistema distribuito. Di conseguenza, l'attività di servizio si deve eseguire per mezzo della rete; invece di un unico magazzino di dati centralizzato ci sono più dispositivi di memorizzazione indipendenti.

In teoria, un DFS dovrebbe apparire ai propri client come un file system centralizzato convenzionale. La molteplicità e la dispersione dei suoi server e dispositivi di memorizzazione dovrebbe essere resa trasparente: l'interfaccia per i client di un DFS non dovrebbe fare distinzioni tra file locali e file remoti. Spetta al DFS localizzare i file e disporre per il trasferimento dei dati. Un DFS trasparente facilita la mobilità del client trasportando l'ambiente di quest'ultimo al sito nel quale un client apre le proprie sessioni.

Esistono parecchi metodi per gli schemi di nominazione in un DFS. In quello più semplice i file si nominano attraverso una combinazione del loro nome di macchina e del loro nome locale, il che garantisce un nome unico per tutto il sistema. Un altro metodo, reso comune dall'NFS, fornisce mezzi per unire directory remote a directory locali, dando così l'impressione di un albero di directory coerente.

Le richieste d'accesso a un file remoto sono generalmente gestite con due metodi complementari. Con il servizio remoto, le richieste d'accesso sono consegnate al server; la macchina server esegue gli accessi e i loro risultati sono riportati al client. Con l'uso di cache, se i dati necessari per soddisfare la richiesta d'accesso non sono ancora stati copiati nella cache, una copia di quei dati viene portata dal server al client. Gli accessi si eseguono sulla copia presente nella cache. L'idea è quella di trattenere nella cache i blocchi del disco cui si è avuto accesso di recente, in questo modo si possono gestire localmente gli accessi ripetuti alle stesse informazioni, evitando un ulteriore traffico di rete. Per mantenere limitate le dimensioni delle cache si ricorre a un criterio di sostituzione. Il problema di mantenere le copie nelle cache coerenti con il file principale costituisce il problema della coerenza della cache.

Esistono due metodi relativi alle informazioni del server: il server tiene traccia dei file ai quali ciascun client vuole accedere, oppure fornisce semplicemente i blocchi richiesti dal client senza conoscere l'uso che se ne farà. Si tratta del paradigma del servizio con informazioni di stato rispetto al servizio senza informazioni di stato.

La replicazione di file in macchine diverse rappresenta una ridondanza utile per migliorare la disponibilità. La replicazione in più macchine può migliorare anche le prestazioni, poiché selezionando una replica vicina per servire una richiesta d'accesso il tempo di servizio è inferiore.

L'AFS, un predecessore *open source* del Transarc DFS, è un DFS ricco di funzioni che mette in evidenza l'indipendenza dalla locazione e la trasparenza. Inoltre impone una significativa semantica della coerenza e impiega le cache e la replicazione per migliorare le prestazioni.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

16.8 Esercizi

16.1 Elencate i vantaggi di un DFS rispetto al file system di un sistema centralizzato.

16.2 Dite quale tra i DFS presentati in questo capitolo gestirebbe nel modo più efficiente una grande applicazione di base di dati con più client. Spiegate la risposta.

16.3 Dite in quali circostanze un client preferisce un DFS con trasparenza di locazione e in quali circostanze preferisce un DFS con indipendenza di locazione. Discutete i motivi di queste preferenze.

16.4 Dite quali aspetti di un sistema distribuito sceglieresteper un sistema in esecuzione su una rete totalmente affidabile.

16.5 Confrontate le tecniche con copiatura in una cache locale dei blocchi dei dischi in un sistema client con le tecniche con copiatura in una cache remota in un server.

16.6 Elencate i vantaggi e gli svantaggi ottenuti dall'associazione di oggetti alla memoria virtuale (come fa l'Apollo Domain).

16.7 Descrivete alcune differenze fondamentali tra l'AFS e l'NFS (si veda il Paragrafo 12.2).

16.9 Note bibliografiche

Discussioni concernenti il controllo della coerenza e il ripristino per file replicati si trovano in [Davcev e Burkhard 1985]. La gestione dei file replicati in un ambiente UNIX è trattata in [Brereton 1986] e [Purdin et al. 1987]. [Wah 1984] tratta il problema della collocazione dei file in un sistema di calcolo distribuito. [Svobodova 1984] offre una rassegna dettagliata dei principali file server centralizzati.

Il Network File System della Sun (NFS) è presentato in [Callaghan 2000]. Il sistema AFS è discusso in [Morris et al. 1986], [Howard et al. 1988] e [Satyanarayanan 1990]. Nel sito `http://www.transarc.ibm.com/DFS` sono disponibili informazioni sul Transarc AFS e i DFS.

Questo testo non tratta nei particolari molti interessanti DFS, tra i quali UNIX United, Sprite, Locus. Il sistema UNIX United è descritto in [Brownbridge et al. 1982]; il Locus in [Popek e Walker 1985]; lo Sprite in [Ousterhout et al. 1988] e [Nelson et al. 1988].

Capitolo 17

Coordinazione distribuita

Nel Capitolo 7 sono descritti vari meccanismi che permettono ai processi di sincronizzare le loro azioni; inoltre sono discussi diversi schemi atti ad assicurare la proprietà di atomicità di una transazione eseguita isolatamente o in modo concorrente con altre transazioni. Nel Capitolo 8 sono descritti vari metodi che un sistema operativo può usare per affrontare il problema delle situazioni di stallo. In questo capitolo si esamina il modo in cui i meccanismi di sincronizzazione centralizzati si possono estendere a un ambiente distribuito, e si discutono alcuni metodi per la gestione delle situazioni di stallo in un sistema distribuito.

17.1 Ordinamento degli eventi

Poiché un sistema centralizzato ha un'unica memoria comune e un unico clock, anch'esso comune, si può sempre stabilire l'ordine in cui due eventi si sono verificati. Molte applicazioni possono richiedere di stabilire tale ordine. Ad esempio, in uno schema di assegnazione di risorse si specifica che una risorsa si può usare solo *dopo* essere stata assegnata. Un sistema distribuito, però, non ha memoria e clock comuni, quindi talvolta è impossibile stabilire l'ordine degli eventi. La relazione *verificato-prima* stabilisce soltanto un ordinamento parziale degli eventi nei sistemi distribuiti. Poiché la possibilità di definire un ordinamento totale è fondamentale per molte applicazioni, in questo paragrafo si presenta un algoritmo distribuito che estende la relazione *verificato-prima* in modo da ottenere un ordinamento totale coerente di tutti gli eventi del sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

17.1.1 Relazione verificato-prima

Poiché si considerano solo processi sequenziali, tutti gli eventi eseguiti in un unico processo sono totalmente ordinati. Quindi, per la legge di causalità, un messaggio si può ricevere solo dopo che è stato inviato. Perciò la relazione *verificato-prima* (indicata dal simbolo $\rightarrow$) su un insieme di eventi, supponendo che l'invio o la ricezione di un messaggio costituisca un evento, si può definire come segue:

1. Se A e B sono eventi dello stesso processo e A è stato eseguito prima di B, allora $A \rightarrow B$.
2. Se A è l'evento corrispondente all'invio di un messaggio da parte di un processo, e B è l'evento corrispondente alla ricezione di quel messaggio da parte di un altro processo, allora $A \rightarrow B$.
3. Se $A \rightarrow B$ e $B \rightarrow C$, allora $A \rightarrow C$.

Poiché un evento non può accadere prima di se stesso, la relazione $\rightarrow$ rappresenta un ordinamento parziale non riflessivo.

Se due eventi, A e B, non sono correlati dalla relazione $\rightarrow$ (cioè A non si verifica prima di B e B non si verifica prima di A) questi due eventi sono stati eseguiti in modo **concorrente**. In questo caso nessuno dei due eventi può influire in modo causale sull'altro. Se, però, $A \rightarrow B$, allora è possibile che l'evento A influisca in modo causale sull'evento B.

Uno schema spazio-temporale come quello illustrato nella Figura 17.1 può ben illustrare le definizioni di concorrenza e *verificato-prima*. La direzione orizzontale rappresenta lo spazio, cioè processi diversi, mentre quella verticale rappresenta il tempo. Le linee verticali etichettate indicano i processi, i punti etichettati indicano gli eventi. Una linea ondulata indica un messaggio inviato da un processo a un altro processo. Da questo diagramma risulta che gli eventi A e B sono concorrenti se e solo se non esiste alcun cammino da A a B o da B ad A.

Si consideri, ad esempio, la Figura 17.1. I seguenti eventi sono collegati dalla relazione *verificato-prima*:

$$p_1 \rightarrow q_2,$$
$$r_0 \rightarrow q_4,$$
$$q_3 \rightarrow r_1,$$
$$p_1 \rightarrow q_4 \text{ (poiché } p_1 \rightarrow q_2 \text{ e } q_2 \rightarrow q_4).$$

I seguenti eventi sono concorrenti:

$$q_0 \text{ e } p_2,$$
$$r_0 \text{ e } q_3,$$
$$r_0 \text{ e } p_3,$$
$$q_3 \text{ e } p_3.$$

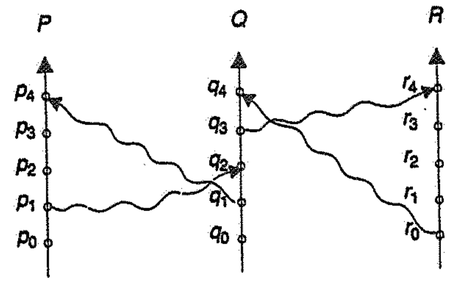


Figura 17.1 Tempo relativo per tre processi concorrenti.

Non è possibile sapere quale fra due eventi concorrenti, come q_0 e p_2, si è verificato per primo. Tuttavia, poiché nessuno dei due eventi può influire sull'altro (nessuno di essi può sapere se l'altro si è già verificato), non è importante stabilire quale si sia verificato per primo. È importante solo che processi interessati all'ordine di due eventi concorrenti si trovino d'accordo su qualche ordine.

17.1.2 Realizzazione

Per stabilire che un evento A si è verificato prima di un evento B occorre avere un clock comune, oppure un insieme di clock perfettamente sincronizzati. Poiché in un sistema distribuito nessuna di queste due condizioni è possibile, occorre definire la relazione *verificato-prima* senza usare clock fisici.

A ogni evento del sistema si associa una **marca temporale** (*timestamp*), tramite la quale si può definire il requisito di **ordinamento totale**: per ogni coppia di eventi A e B, se $A \rightarrow B$, allora la marca temporale di A è minore della marca temporale di B (nel seguito si vede che il contrario non deve necessariamente essere vero).

A questo punto occorre stabilire un metodo per ottenere un ordinamento totale in un ambiente distribuito. All'interno di *ciascun* processo P_i si definisce un **clock logico** LC_i. Il clock logico si può realizzare come un semplice contatore che si incrementa tra ogni coppia di eventi successivi verificati all'interno di un processo. Poiché ha un valore **monotonamente** crescente, il clock logico assegna un unico valore a ogni evento e, se nel processo P_i l'evento A si verifica prima dell'evento B, allora $LC_i(A) < LC_i(B)$. La marca temporale di un evento è il valore del clock logico di quell'evento. Questo schema assicura che il requisito d'ordinamento totale è soddisfatto per ogni coppia di eventi dello stesso processo.

Sfortunatamente questo schema non assicura che il requisito d'ordinamento totale sia soddisfatto da processi diversi. Per illustrare questo problema si considerano i due processi comunicanti P_1 e P_2. Si supponga che P_1 invii a P_2 (evento A) un messaggio con $LC_1(A) = 200$ e P_2 riceva il messaggio (evento B) con $LC_2(B) = 195$, perché l'unità d'elaborazione di P_2 è più lenta di quella di P_1 e quindi il suo clock logico avanza più lentamente. Questa situazione viola il requisito d'ordinamento totale, poiché $A \rightarrow B$, ma la marca temporale di A è maggiore della marca temporale di B.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per superare questa difficoltà occorre che un processo faccia avanzare il proprio clock logico quando riceve un messaggio la cui marca temporale è maggiore dell'attuale valore del suo clock logico. In particolare, se il processo P_i riceve un messaggio (evento B) con marca temporale t e $LC_i(B) \leq t$, allora deve far avanzare il suo clock in modo che $LC_i(B) = t + 1$. Così, nell'esempio illustrato, quando P_2 riceve il messaggio da P_1, fa avanzare il suo clock logico in modo che $LC_2(B) = 201$.

Infine, per realizzare un ordinamento totale occorre soltanto osservare che, applicato lo schema d'ordinamento con marca temporale, se le marche temporali di due eventi A e B sono uguali, allora gli eventi sono concorrenti. In questo caso, per evitare ogni ambiguità e creare un ordinamento totale, si usano i numeri d'identità dei processi. L'uso delle marche temporali è trattato nel Paragrafo 17.4.2.

17.2 Mutua esclusione

In questo paragrafo sono presentati alcuni algoritmi per la realizzazione della mutua esclusione in un ambiente distribuito. S'ipotizza che il sistema sia composto di n processi, ciascuno dei quali risiede in una diversa unità d'elaborazione. Per semplificare la discussione, si suppone che i processi siano numerati in modo unico da 1 a n e che esista una corrispondenza da uno a uno tra processi e unità d'elaborazione, vale a dire che ogni processo ha la propria unità d'elaborazione.

17.2.1 Metodo centralizzato

Per realizzare la mutua esclusione in modo centralizzato si sceglie uno dei processi del sistema per coordinare l'accesso alle sezioni critiche. Ogni processo che necessiti della mutua esclusione invia un messaggio di *richiesta* al coordinatore. Quando il processo riceve un messaggio di *risposta* dal coordinatore, può procedere ed entrare nella propria sezione critica. Una volta uscito dalla sezione critica, il processo invia un messaggio di *rilascio* al coordinatore e procede con la sua esecuzione.

Quando il coordinatore riceve un messaggio di *richiesta*, controlla se qualche altro processo si trova nella propria sezione critica. Se nessuno dei processi è nella propria sezione critica, il coordinatore invia immediatamente un messaggio di *risposta*, altrimenti la richiesta viene accodata. Quando il coordinatore riceve un messaggio di *rilascio*, preleva dalla coda uno dei messaggi di *richiesta*, secondo qualche algoritmo di scheduling, e invia un messaggio di *risposta* al processo richiedente.

Dovrebbe essere evidente che quest'algoritmo assicura la mutua esclusione. Inoltre, se il criterio di scheduling del coordinatore è equo, come nel caso dello scheduling in ordine d'arrivo (FCFS), non c'è alcuna possibilità di situazioni d'attesa indefinita. Questo schema richiede tre messaggi per ogni ingresso nella sezione critica: una *richiesta*, una *risposta* e un *rilascio*.

Se il processo coordinatore si guasta, allora un nuovo processo deve sostituirlo. Nel Paragrafo 17.6 si descrivono alcuni algoritmi per l'elezione di un nuovo coordinatore. Una volta eletto, il nuovo coordinatore deve interrogare ciclicamente tutti i processi del sistema per ricostruire la coda di *richieste*. Dopodiché, la computazione può riprendere.

17.2.2 Metodo totalmente distribuito

Il problema si complica quando è necessario distribuire il processo di decisione su tutto il sistema. In questo paragrafo si presenta un algoritmo basato sullo schema di ordinamento degli eventi descritto nel Paragrafo 17.1.

Quando un processo P_i vuole entrare nella propria sezione critica, genera una nuova marca temporale TS, e invia il messaggio `request`(P_i, TS) a tutti gli altri processi del sistema, compreso se stesso. Quando riceve un tale messaggio di richiesta, un processo può rispondere immediatamente (cioè inviare un messaggio di risposta a P_i), oppure può differire l'invio della risposta, ad esempio perché si trova già nella propria sezione critica.

Un processo che abbia ricevuto un messaggio di risposta da tutti gli altri processi del sistema può entrare nella propria sezione critica, accodando e differendo le richieste in arrivo. Dopo essere uscito dalla sezione critica, il processo invia messaggi di risposta a tutte le richieste che gli sono pervenute e che aveva differito.

La decisione di rispondere immediatamente al messaggio `request`(P_j, TS) oppure differire tale risposta dipende dai tre fattori elencati di seguito:

1. Se si trova nella propria sezione critica, il processo P_i differisce la risposta a P_j.
2. Se *non* vuole entrare nella propria sezione critica, il processo P_i invia immediatamente una risposta a P_j.
3. Se il processo P_i vuole entrare nella propria sezione critica ma non vi è ancora entrato, confronta la marca temporale della propria richiesta con la marca temporale TS della richiesta fatta dal processo P_j. Se la marca temporale della sua richiesta è maggiore di TS, allora invia immediatamente una risposta a P_j, perché P_j ha fatto per primo la richiesta, altrimenti differisce la risposta.

Quest'algoritmo mostra il seguente comportamento:

- Ottiene la mutua esclusione.
- Assicura l'assenza di situazioni di stallo.
- Assicura l'assenza di situazioni d'attesa indefinita: l'entrata nella sezione critica è programmata secondo l'ordinamento delle marche temporali, che assicura che i processi siano serviti nell'ordine FCFS.
- Il numero di messaggi per ogni entrata in una sezione critica è $2 \times (n - 1)$. Si tratta del numero minimo di messaggi richiesti per ogni entrata nella sezione critica quando i processi agiscono in modo indipendente e concorrente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Per comprendere il funzionamento di quest'algoritmo, si consideri un sistema composto dei processi P_1, P_2 e P_3. Si supponga che i processi P_1 e P_3 vogliano entrare nelle rispettive sezioni critiche. Il processo P_1 invia il messaggio `request(`P_1, TS = 10`)` ai processi P_2 e P_3, mentre il processo P_3 invia il messaggio `request(`P_3, TS = 4`)` ai processi P_1 e P_2. Le marche temporali 4 e 10 sono state ottenute dai clock logici descritti nel Paragrafo 17.1.2. Quando il processo P_2 riceve questi messaggi risponde immediatamente. Quando il processo P_1 riceve la richiesta dal processo P_3 risponde immediatamente, poiché la marca temporale (10) del proprio messaggio di richiesta è maggiore della marca temporale (4) per il processo P_3. Quando il processo P_3 riceve la richiesta dal processo P_1, differisce la risposta, poiché la marca temporale (4) del suo messaggio di richiesta è inferiore alla marca temporale (10) del messaggio del processo P_1. Il processo P_3 riceve le risposte dai due processi P_1 e P_2, e può entrare nella propria sezione critica. Dopo esserne uscito, il processo P_3 invia una risposta al processo P_1 il quale può entrare a sua volta nella propria sezione critica.

Poiché richiede la partecipazione di tutti i processi del sistema, questo schema ha comunque tre conseguenze sgradite mostrate di seguito:

1. I processi devono conoscere l'identità di tutti gli altri processi del sistema. Quando un nuovo processo si aggiunge al gruppo dei processi che partecipano all'algoritmo di mutua esclusione, si devono intraprendere le seguenti operazioni:
   a) il processo deve ricevere i nomi di tutti gli altri processi del gruppo;
   b) il nome del nuovo processo deve essere distribuito a tutti gli altri processi del gruppo.

   Questo compito non è banale come può sembrare poiché, quando il nuovo processo si aggiunge al gruppo, nel sistema possono essere in circolazione alcuni messaggi di richiesta e risposta. Ulteriori dettagli su quest'argomento sono contenuti nelle Note bibliografiche.
2. Se uno dei processi s'interrompe a causa di un errore, crolla tutto lo schema. Questa difficoltà si può risolvere tenendo continuamente sotto controllo lo stato di tutti i processi del sistema. Se un processo s'interrompe a causa di un errore, s'informano gli altri dell'accaduto, che perciò non inviano più messaggi di richiesta al processo guasto. Quando il processo viene ripristinato, deve cominciare la procedura che gli permette di ricongiungersi al gruppo.
3. I processi che non sono entrati nelle rispettive sezioni critiche devono interrompersi spesso per dichiarare agli altri processi che intendono entrare nella propria sezione critica. Questo protocollo è quindi adatto a insiemi piccoli e stabili di processi cooperanti.

17.2.3 Metodo con passaggio di contrassegno

Un altro metodo per fornire la mutua esclusione consiste nel far circolare tra i processi del sistema un contrassegno. Un **contrassegno** (*token*) è uno speciale tipo di messaggio che si fa circolare nel sistema e il cui possesso autorizza il proprietario a entrare nella sezione critica. Poiché nel sistema esiste un solo contrassegno, solo un processo alla volta può essere autorizzato a entrare nella propria sezione critica.

Si supponga che i processi del sistema siano organizzati *logicamente* in una **struttura ad anello**. La rete fisica di comunicazione non deve essere necessariamente un anello: finché i processi rimangono collegati fra loro, si può realizzare un anello logico. Per realizzare la mutua esclusione, si fa passare il contrassegno lungo l'anello. Quando un processo riceve il contrassegno, può trattenerlo per entrare nella propria sezione critica; quando il processo esce dalla sezione critica, il contrassegno viene rimesso in circolazione. Se il processo che riceve il contrassegno non vuole entrare nella sezione critica, passa il contrassegno al suo vicino. Questo schema è simile all'Algoritmo 1 descritto nel Capitolo 7, ma, in questo caso, il contrassegno sostituisce una variabile condivisa.

Se l'anello è unidirezionale, è assicurata anche l'assenza di situazioni d'attesa indefinità. Il numero dei messaggi richiesti per realizzare la mutua esclusione può variare da un messaggio per ciascuna entrata, nel caso di alta contesa (vale a dire che ogni processo vuole entrare nella propria sezione critica), a un numero infinito di messaggi, nel caso di bassa contesa (vale a dire che nessun processo vuole entrare nella propria sezione critica).

È necessario considerare due tipi di guasti: se il contrassegno va perduto, occorre indire un'elezione per generarne uno nuovo; se un processo s'interrompe a causa di un errore, occorre definire un nuovo anello logico. Nel Paragrafo 17.6 è presentato un algoritmo di elezione; sono possibili altri algoritmi. Lo sviluppo di un algoritmo per la ricostruzione dell'anello è lasciato al lettore nell'Esercizio 17.6.

17.3 Atomicità

Nel Paragrafo 7.9 s'introduce il concetto di transazione atomica, definita come un'unità di programma che si deve eseguire in modo **atomico**. In altre parole, o tutte le operazioni a essa associate sono portate a compimento oppure nessuna di esse deve essere eseguita. Quando si è alle prese con un sistema distribuito, assicurare l'atomicità di una transazione diventa assai più complicato di quanto non lo sia in un sistema centralizzato. Questa difficoltà è dovuta al fatto che più siti possono partecipare all'esecuzione di una singola transazione. Il verificarsi di un malfunzionamento in uno di questi siti o nella linea di comunicazione che li connette può causare una computazione errata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il **coordinatore delle transazioni** di un sistema distribuito deve assicurare che l'esecuzione delle diverse transazioni nel sistema rispetti la proprietà di atomicità. Ciascun sito ha il proprio coordinatore delle transazioni locale, che ha il compito di coordinare l'esecuzione di tutte le transazioni avviate in quel sito. Per ogni transazione il coordinatore è responsabile delle seguenti attività:

- avviare l'esecuzione della transazione;
- scomporre la transazione in sottotransazioni, e distribuire queste ultime nei siti appropriati per l'esecuzione;
- coordinare la terminazione della transazione, che potrebbe essere completata con successo o fallita in tutti i siti.

Si suppone che ogni sito mantenga un giornale delle modifiche per le attività di ripristino.

17.3.1 Protocollo di conferma a due fasi

Per assicurare l'atomicità, tutti i siti in cui si esegue la transazione T devono essere d'accordo sul risultato finale dell'esecuzione: T deve essere completata con successo (*commited*) in tutti i siti o deve essere fallita (*aborted*) in tutti i siti. Per assicurare questa proprietà il coordinatore delle transazioni di T deve eseguire un **protocollo di conferma**. Tra i protocolli più semplici e più largamente adottati c'è il **protocollo di conferma a due fasi** (*two-phase commit protocol* — 2PC) discusso in questo paragrafo.

Sia T una transazione avviata nel sito S_i e sia C_i il coordinatore in S_i. Quando T completa la propria esecuzione (quando cioè tutti i siti nei quali T è in esecuzione informano C_i che T è terminata) C_i attiva il protocollo 2PC.

- Fase 1. C_i aggiunge l'elemento <T `prepare`> al giornale delle modifiche e lo registra nella memoria stabile. Quindi, spedisce il messaggio `prepare(`T`)` a tutti i siti nei quali T è in esecuzione. Ricevuto il messaggio, il gestore delle transazioni in ciascuno di tali siti determina se il sito è disposto a terminare con successo (*commit*) la sua parte di T. Se la risposta è *no*, il gestore delle transazioni aggiunge un elemento <T no > al giornale delle modifiche e risponde inviando il messaggio `abort(`T`)` a C_i. Se la risposta è *sì*, aggiunge l'elemento <T `ready`> al giornale delle modifiche e registra nella memoria stabile tutti gli elementi del giornale delle modifiche corrispondenti a T. Quindi, il gestore delle transazioni risponde a C_i con il messaggio `ready(`T`)`.
- Fase 2. Una volta ricevuta la risposta al messaggio `prepare(`T`)` a esso inviata da tutti i siti, o dopo che è trascorso un predeterminato intervallo di tempo dall'invio del messaggio, C_i può determinare se la transazione T può essere chiusa con successo o è fallita. T può essere chiusa con successo se C_i ha ricevuto il messaggio `ready(`T`)` da tutti i siti partecipanti, altrimenti T è fallita. Secondo il verdetto, uno tra gli elementi <T `commit`> o <T `abort`> viene aggiunto al giornale delle modifiche e registrato nella memoria stabile. A questo punto il destino della transazione è stato deciso irrevocabilmente. Di conseguenza il coordinatore invia ai siti partecipanti uno tra i messaggi `commit(`T`)` o `abort(`T`)`, i quali lo registrano nel proprio giornale delle modifiche.

Un sito nel quale T è in esecuzione può interrompere incondizionatamente l'esecuzione della transazione solo prima di avere spedito il messaggio `ready(`T`)` al coordinatore. In effetti questo messaggio è una promessa di seguire l'ordine del coordinatore di terminare con successo T o far fallire T. L'unico caso in cui un sito può fare questa promessa è quando le informazioni richieste sono state registrate nella memoria stabile. Altrimenti, se il sito crolla dopo l'invio del messaggio `ready(`T`)`, potrebbe non essere più in grado di rispettare la promessa fatta.

Il successo di una transazione richiede l'unanimità, perciò il destino di T è deciso non appena almeno un sito risponde con un messaggio `abort(`T`)`. Essendo anch'esso coinvolto nell'esecuzione, il sito di coordinamento S_i può decidere in modo unilaterale d'interrompere l'esecuzione di T. Il verdetto finale su T è determinato nel momento in cui il coordinatore scrive la decisione (`commit` o `abort`) nel giornale delle modifiche e nella memoria stabile. In alcune versioni del protocollo 2PC, un sito invia un messaggio `acknowledge(`T`)` al coordinatore al termine della seconda fase del protocollo. Quando il coordinatore riceve questo messaggio da tutti i siti coinvolti, aggiunge l'elemento <T `complete`> al giornale delle modifiche.

17.3.2 Gestione dei guasti nel 2PC

La discussione procede con un esame dettagliato di come il protocollo 2PC risponde a diversi tipi di malfunzionamenti. Uno dei principali svantaggi del protocollo consiste nel fatto che il verificarsi di un guasto nel coordinatore può causare il blocco dell'intero sistema, poiché potrebbe essere necessario rinviare la decisione di chiusura con successo o fallimento di T fino al momento in cui C_i riprenderà il normale funzionamento.

17.3.2.1 Guasto di un sito partecipante

Quando un sito partecipante S_k si riprende da un guasto deve innanzi tutto esaminare il proprio giornale delle modifiche per determinare la sorte delle transazioni che si trovavano nel mezzo di un'esecuzione quando si è verificato il guasto. Sia T una di queste transazioni, si deve considerare ciascuno dei casi possibili:

- Se il giornale delle modifiche contiene un elemento <T `commit`>, il sito esegue `redo(`T`)`.
- Se il giornale delle modifiche contiene un elemento <T `abort`>, il sito esegue `undo(`T`)`.
- Se il giornale delle modifiche contiene un elemento <T `ready`>, il sito deve consultare C_i al fine di stabilire la sorte di T. Se C_i è attivo, notifica a S_k il successo o il fallimento di T. Nel primo caso, esegue `redo(`T`)`, mentre nel secondo esegue `undo(`T`)`. Se C_i non è attivo, S_k deve cercare di risalire alla sorte di T dagli altri siti, inviando a ciascuno di essi il messaggio `query-status(`T`)`. Quando riceve questo messaggio, un sito esamina il proprio giornale delle modifiche per determinare se ha eseguito T e, nel caso affermativo, per stabilire com'è terminata l'esecuzione, con successo o fallita; quindi notifica il risultato della ricerca a S_k. Se nessun sito contie-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ne le informazioni necessarie a stabilire se T ha terminato la propria esecuzione con successo o no, S_k non è in grado di chiudere T né con successo né con fallimento. La decisione viene rimandata al momento in cui S_k sarà in grado di ottenere le informazioni richieste. Quindi S_k deve inviare periodicamente il messaggio `query-status(T)` agli altri siti, fino al ripristino di uno dei siti che contiene le informazioni richieste. Il sito nel quale risiede C_i contiene sempre queste informazioni.

- Se il giornale delle modifiche non contiene nessuno dei precedenti elementi di controllo (`abort`, `commit`, `ready`) riguardanti T, L'assenza di questi elementi implica che S_k si è guastato prima di rispondere al messaggio `prepare(T)` proveniente da C_i. Poiché il guasto di S_k preclude l'invio di questa risposta, l'algoritmo stabilisce che C_i deve interrompere T e, quindi, che S_k deve eseguire l'operazione `undo(T)`.

17.3.2.2 Guasto del coordinatore

Se durante l'esecuzione del protocollo di conferma per la transazione T si verifica un malfunzionamento del coordinatore, la sorte di T deve essere decisa dai siti partecipanti all'esecuzione. In certi casi i siti partecipanti non sono in grado di decidere come terminare (successo o fallimento) l'esecuzione di T, perciò devono attendere il ripristino del coordinatore guasto.

- Se un sito attivo contiene l'elemento <T `commit`> nel proprio giornale delle modifiche, T è terminata con successo.
- Se un sito attivo contiene l'elemento <T `abort`> nel proprio giornale delle modifiche, T è fallita.
- Se alcuni tra i siti attivi *non* contengono l'elemento <T `ready`> nel proprio giornale delle modifiche, il coordinatore guasto C_i non può aver deciso il successo di T. Si può trarre questa conclusione dal fatto che se un sito non contiene nel proprio giornale delle modifiche l'elemento <T `ready`> non può avere spedito il messaggio `ready(T)` a C_i. D'altra parte il coordinatore potrebbe avere deciso il fallimento di T. Piuttosto che attendere il ripristino di C_i è preferibile l'interruzione dell'esecuzione di T.
- Se non si presenta nessuno dei casi precedenti, tutti i siti attivi devono contenere nel proprio giornale delle modifiche l'elemento <T `ready`>, senza però altri elementi di controllo (come <T `abort`> o <T `commit`>). Poiché il coordinatore si è guastato, è impossibile stabilire prima del suo ripristino se questi avesse preso una decisione o quale essa fosse; quindi i siti attivi devono attendere il ripristino di C_i. Poiché la sorte di T è dubbia, può succedere che T continui a detenere risorse del sistema. Ad esempio, T può continuare a detenere un diritto d'accesso bloccante sui dati contenuti nei siti attivi. Questa è chiaramente una situazione indesiderabile poiché possono trascorrere ore o giorni prima che C_i torni in attività. Durante questo periodo altre transazioni potrebbero essere costrette ad attendere il completamento di T. Ne segue che i dati non saranno disponibili non solo nel sito guasto (C_i) ma anche nei siti attivi. Il numero di dati non disponibili aumenta col crescere del periodo d'inattività di C_i. Questa situazione prende il nome di problema del *bloccaggio*, poiché T è bloccata nell'attesa del ripristino del sito C_i.

17.3.2.3 Guasto della rete

Quando il guasto riguarda un collegamento di una rete, tutti i messaggi instradati in esso non arrivano intatti alla destinazione. Dal punto di vista dei siti dovunque connessi a tale collegamento, sembra che a fallire siano stati gli altri siti. Quindi, anche in questo caso sono applicabili gli schemi appena proposti.

Quando a guastarsi sono più connessioni di una rete, si può verificare un partizionamento della rete stessa. In questo caso ci sono due possibilità: il coordinatore e tutti i suoi partecipanti restano nella stessa partizione, e in questo caso il guasto non ha alcun effetto sul protocollo di conferma; oppure può accadere che coordinatore e partecipanti si ritrovino in partizioni diverse; in questo caso si ha la perdita dei messaggi scambiati tra il coordinatore e i partecipanti, che si riduce al caso del guasto di un collegamento.

17.4 Controllo della concorrenza

Nel presente paragrafo s'illustra come sia possibile modificare alcuni schemi per il controllo della concorrenza proposti nel Capitolo 7 affinché si possano impiegare in un ambiente distribuito.

Il gestore delle transazioni di una base di dati distribuita gestisce l'esecuzione delle transazioni (o sottotransazioni) che accedono ai dati memorizzati in un sito locale. Ciascuna di queste transazioni può essere sia una transazione locale (in esecuzione solamente in quel sito) sia parte di una transazione globale (che coinvolge cioè più siti). Ciascun gestore delle transazioni è responsabile del mantenimento di un giornale delle modifiche per le attività di ripristino e della partecipazione, nell'appropriato schema di controllo della concorrenza, al coordinamento dell'esecuzione concorrente delle transazioni in esecuzione in quel sito. Gli schemi di controllo della concorrenza introdotti nel Capitolo 7 devono essere modificati in modo da adattarli alla distribuzione delle transazioni.

17.4.1 Protocolli d'accesso bloccante

Il protocollo d'accesso bloccante a due fasi descritto nel Paragarfo 7.9.4.2 si può adoperare in un ambiente distribuito. L'unica modifica richiesta riguarda il modo in cui è realizzato il gestore dei diritti d'accesso bloccante. In questo paragrafo sono presentati diversi schemi possibili: il primo riguarda il caso in cui non sia consentita la replicazione dei dati; gli altri si applicano al caso più generale in cui i dati si possono replicare in siti differenti. Come nel Paragarfo 7.9.4.2, si suppone l'esistenza dei **modi d'accesso bloccante condiviso** ed **esclusivo**.

http://www.edutpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

17.4.1.1 Schema senza replicazione

Se i dati non sono replicati nel sistema, gli schemi d'accesso bloccante descritti nel Paragrafo 7.9.4.2 si possono applicare nel modo seguente: ciascun sito mantiene un gestore locale dei diritti d'accesso bloccante che amministri le richieste d'accesso e di rilascio dirette ai dati memorizzati in quel sito. Se una transazione desidera accedere in modo bloccante a un elemento Q nel sito S_i, invia un messaggio al gestore dei diritti d'accesso bloccante di S_i col quale richiede un accesso bloccante (nel modo voluto). Se l'elemento Q è già bloccato in modo incompatibile con la richiesta, questa viene ritardata fino al momento in cui potrà essere soddisfatta. Una volta stabilito che la richiesta può essere soddisfatta, il gestore risponde al richiedente comunicandogli la concessione del diritto d'accesso bloccante.

Questo schema ha il vantaggio di essere facilmente realizzabile, poiché richiede il trasferimento di due messaggi per la gestione della richiesta d'accesso bloccante e di uno per la richiesta di rilascio. D'altra parte, la gestione delle situazioni di stallo è più complicata. Poiché le richieste d'accesso bloccante e di rilascio non vengono più inoltrate a uno stesso sito, gli algoritmi di gestione delle situazioni di stallo trattati nel Capitolo 8 devono essere modificati; tali modifiche sono trattate nel Paragrafo 17.5.

17.4.1.2 Metodo con coordinatore singolo

Se s'impiega un solo coordinatore il sistema mantiene un *singolo* gestore dei diritti d'accesso bloccante che risiede in un *singolo* sito S_i opportunamente scelto. Tutte le richieste d'accesso bloccante e di rilascio s'inoltrano al sito S_i. Quando una transazione deve accedere in modo bloccante a un elemento, spedisce la relativa richiesta a S_i. Il gestore dei diritti d'accesso bloccante stabilisce se tale richiesta si può soddisfare immediatamente. Nel caso affermativo spedisce un messaggio in tal senso al sito da cui è stata inoltrata la richiesta; altrimenti la richiesta viene differita fino al momento in cui potrà essere soddisfatta. Dopodiché si spedirà un messaggio al sito da cui è stata inoltrata la richiesta. La transazione può leggere l'elemento da *qualsiasi* sito in cui risiede una replica del dato. Nel caso di un'operazione `write` tutti i siti in cui risiedono le repliche devono essere coinvolti nell'operazione. Questo schema ha i seguenti vantaggi:

- **Semplice realizzazione.** Richiede due messaggi per la gestione di una richiesta d'accesso bloccante e uno per una richiesta di rilascio.
- **Semplice gestione delle situazioni di stallo.** Poiché tutte le richieste d'accesso bloccante e di rilascio s'inoltrano a un unico sito, si possono applicare a questo ambiente gli algoritmi di gestione delle situazioni di stallo discussi nel Capitolo 8.

Tra gli svantaggi sono inclusi i seguenti:

- **Strozzatura.** Il sito S_i diventa una strozzatura, poiché deve elaborare tutte le richieste.
- **Vulnerabilità.** Se il sito S_i si guasta, manca interamente il controllore della concorrenza. In tal caso si devono interrompere le elaborazioni o si deve impiegare uno schema di ripristino.

Un compromesso tra vantaggi e svantaggi si può ottenere impiegando un metodo **con coordinatori multipli**, nel quale le funzioni di gestione dei diritti d'accesso bloccante sono distribuite tra più siti.

Ogni gestore amministra le richieste d'accesso bloccante e di rilascio di un sottoinsieme di dati e risiede in un sito diverso. Questa distribuzione riduce l'effetto di strozzatura del coordinatore, ma complica la gestione delle situazioni di stallo, poiché le richieste d'accesso bloccante e di rilascio non s'inoltrano più a un unico sito.

17.4.1.3 Protocollo di maggioranza

Il protocollo di maggioranza è una modifica dello schema dei dati senza replicazione proposto in precedenza. Il sistema mantiene un gestore dei diritti d'accesso bloccante in ciascun sito, che si occupa di tutti i dati o repliche di dati memorizzati nel relativo sito. Quando una transazione desidera accedere in modo bloccante a un elemento Q, replicato in n siti differenti, deve inviare una richiesta a più della metà degli n siti nei quali Q risiede. Ogni gestore determina se la richiesta può essere immediatamente soddisfatta (per quel che lo riguarda). Come in precedenza, si rimanda la risposta fino quando si potrà soddisfare la richiesta. La transazione non opererà su Q fino a quando non avrà ottenuto il diritto d'accesso bloccante sulla maggioranza delle repliche di Q.

Questo schema gestisce i dati replicati in modo decentralizzato, eliminando perciò gli inconvenienti propri dei controlli centralizzati. Tuttavia risente dei seguenti svantaggi:

- **Realizzazione.** La realizzazione del protocollo di maggioranza è più complessa rispetto agli schemi precedenti. Richiede $2(n/2 + 1)$ messaggi per la gestione delle richieste d'accesso bloccante e $(n/2 + 1)$ messaggi per la gestione delle richieste di rilascio.
- **Gestione delle situazioni di stallo.** Poiché le richieste d'accesso bloccante e di rilascio non vengono inoltrate a uno stesso sito, gli algoritmi di gestione delle situazioni di stallo devono essere modificati (Paragrafo 17.5). Inoltre può verificarsi uno stallo anche se si blocca un solo elemento di dati. Per illustrare questo fatto si consideri un sistema di quattro siti con replicazione totale dei dati. Si supponga che due transazioni T_1 e T_2 desiderino bloccare un dato Q in modo esclusivo. La transazione T_1 potrebbe ottenere il diritto d'accesso bloccante a Q nei siti S_1 e S_2, mentre la transazione T_2 potrebbe avere successo nell'ottenere il diritto d'accesso bloccante nei siti S_3 e S_4. A questo punto entrambe attendono l'acquisizione del terzo diritto d'accesso bloccante, determinando uno stallo.

17.4.1.4 Protocollo sbilanciato

Il protocollo sbilanciato si basa su un modello simile a quello del protocollo di maggioranza. La differenza risiede nel fatto che le richieste d'accesso bloccante condivise sono gestite in modo più vantaggioso rispetto a quelle d'accesso bloccante esclusive. Il sistema mantiene un gestore dei diritti d'accesso bloccante in ciascun sito, che si occupa delle ri-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

chieste riguardanti tutti i dati memorizzati nel relativo sito. Le richieste d'accesso bloccante condivise sono gestite in maniera diversa da quelle d'accesso bloccante esclusive:

- **Accesso bloccante condiviso.** Quando richiede l'accesso bloccante condiviso a un elemento Q, una transazione si limita a inoltrare la richiesta al gestore dei diritti d'accesso bloccante di un sito contenente una replica di Q.
- **Accesso bloccante esclusivo.** Quando richiede l'accesso bloccante esclusivo a un elemento Q, una transazione inoltra la richiesta ai gestori dei diritti d'accesso bloccante di tutti i siti contenenti una replica di Q.

Come prima, si rimanda la risposta fino a quando si potrà soddisfare la richiesta.

Questo schema ha il vantaggio di imporre un minore carico rispetto al protocollo di maggioranza sulle operazioni di lettura. E tale vantaggio è particolarmente significativo nelle situazioni comuni, in cui la frequenza delle letture è notevolmente più alta di quella delle scritture. D'altra parte, lo svantaggio si manifesta in un ulteriore carico nella gestione delle scritture. Inoltre, il protocollo sbilanciato condivide lo svantaggio del protocollo di maggioranza legato alla complessità nella gestione delle situazioni di stallo.

17.4.1.5 Copia primaria

Nei casi di replicazione dei dati si può scegliere una replica come copia primaria. Quindi, per ogni elemento Q, la copia primaria di Q deve risiedere esattamente in un sito, chiamato *sito primario di Q*.

Quando deve accedere in modo bloccante a Q, una transazione invia la richiesta al sito primario di Q. Anche in questo caso, si rimanda la risposta fino quando si potrà soddisfare la richiesta. La presenza di una copia primaria consente quindi di gestire il controllo della concorrenza alla presenza di dati replicati in modo simile a quando i dati non sono replicati. Questo metodo di gestione permette una semplice realizzazione. Però se il sito primario di Q si guasta, l'elemento Q resta inaccessibile anche se altri siti che ne contengono una replica sono accessibili.

17.4.2 Uso delle marche temporali

L'idea principale alla base dello schema con uso di marche temporali discusso nel Paragarfo 7.9.4.3 è quella di associare a ogni transazione una marca temporale *unica* da adoperare per decidere l'ordine di serializzazione. Il primo compito della generalizzazione dello schema centralizzato a uno schema distribuito è lo sviluppo di un metodo per la generazione di marche temporali uniche. Una volta sviluppato questo metodo, i protocolli precedenti si possono applicare direttamente agli ambienti privi di replicazione.

17.4.2.1 Generazione di marche temporali uniche

Esistono principalmente due metodi di generazione di marche temporali uniche, uno centralizzato e uno distribuito. Nello schema centralizzato per la distribuzione delle marche temporali si sceglie un singolo sito, che può adoperare allo scopo sia un contatore logico sia il proprio clock locale.

Nel caso del metodo distribuito ogni sito genera una marca temporale unica locale usando o un contatore logico o il clock locale. La marca temporale unica globale si ottiene concatenando la marca temporale unica locale all'identificatore del sito, che deve a sua volta essere unico (Figura 17.2). L'ordine di concatenazione è importante: si dispone l'identificatore del sito nella parte meno significativa della marca temporale per assicurare che le marche temporali globali generate in un sito non siano sempre maggiori di quelle generate in un altro sito. Si confronti questa tecnica per la generazione di marche temporali uniche con quella per la generazione di nomi unici presentata nel Paragrafo 17.1.2.

Un ulteriore problema può essere costituito da un sito che genera marche temporali locali a una frequenza maggiore rispetto ad altri siti. In questo caso il contatore logico del sito più veloce tenderà a essere sempre maggiore di quello degli altri siti, perciò le marche temporali generate dal sito più rapido saranno maggiori di quelle generate dagli altri siti. Quindi è necessario un meccanismo che assicuri una generazione equa delle marche temporali locali in tutto il sistema. Per ottenere questa situazione, si definisce in ogni sito S_i un clock logico (LC_i) che genera le marche temporali uniche locali (si veda il Paragrafo 17.1.2). Per garantire la sincronizzazione dei diversi clock logici, si richiede che il sito S_i incrementi il proprio clock logico ogniqualvolta una transazione T_i con marca temporale $<y, x>$, con x maggiore del valore corrente di LC_i, visita quel sito. In questo caso, il sito S_i avanza il proprio clock logico al valore $x + 1$.

Se s'intende usare il clock di sistema per la generazione delle marche temporali, è possibile garantire una ripartizione uniforme delle marche temporali solo se non esistono siti con clock più veloci o più lenti degli altri. Poiché i clock possono non essere perfettamente precisi, è necessario adoperare una tecnica simile a quella adottata per i clock logici per assicurare che nessun clock si discosti troppo dagli altri.

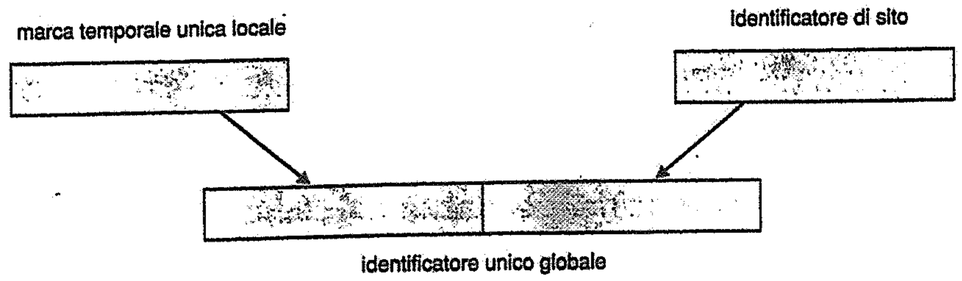


Figura 17.2 Generazione di marche temporali uniche.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

17.4.2.2 Schema a ordinamento di marche temporali

Lo schema di base con marche temporali introdotto nel Paragrafo 7.9.4.3 si può estendere in modo immediato ai sistemi distribuiti. Come nel caso centralizzato, senza alcun meccanismo che impedisca a una transazione la lettura del valore di un dato su cui sta operando un'altra transazione, si possono verificare annullamenti in cascata. Per eliminare gli annullamenti in cascata si può combinare lo schema con marche temporali del Paragrafo 7.9.4.3 con il protocollo 2PC del Paragrafo 17.3.1, al fine di ottenere un protocollo che assicuri la serializzabilità senza annullamenti in cascata. La realizzazione di quest'algoritmo è lasciata al lettore.

Lo schema di base con marche temporali appena descritto soffre dell'indesiderabile proprietà che i conflitti fra le transazioni si risolvono mediante annullamenti anziché mediante semplici attese. Per attenuare il problema si possono memorizzare transitoriamente le diverse operazioni `read` e `write` (in altri termini, *differirle*) fino a quando si è certi che potranno essere eseguite senza causare fallimenti. Un'operazione `read(x)` di T_i si deve differire se esiste una transazione T_j che eseguirà un'operazione `write(x)` ma non lo ha ancora fatto, e $TS(T_j) < TS(T_i)$. Analogamente, un'operazione `write(x)` di T_i si deve differire se esiste una transazione T_j che eseguirà un'operazione `read(x)` o una operazione `write(x)`, e $TS(T_j) < TS(T_i)$. Sono disponibili diversi metodi per garantire il soddisfacimento di questa proprietà. Uno di questi, chiamato **schema conservativo a ordinamento di marche temporali**, richiede che ogni sito mantenga una coda di letture e scritture contenente tutte le richieste di operazioni `read` e `write` che, rispettivamente, si devono eseguire in quel sito e che si devono differire per preservare la precedente proprietà. Anche questo schema non viene presentato, lasciando al lettore la realizzazione dell'algoritmo.

17.5 Gestione delle situazioni di stallo

Gli algoritmi per prevenire, evitare e rilevare le situazioni di stallo presentati nel Capitolo 8 si possono estendere in modo da permetterne l'uso in un sistema distribuito. Nel seguito sono descritti alcuni di questi algoritmi distribuiti.

17.5.1 Prevenzione delle situazioni di stallo

Gli algoritmi per prevenire ed evitare le situazioni di stallo discussi nel Capitolo 8, se opportunamente modificati, si possono impiegare anche in un sistema distribuito. Ad esempio, la tecnica di prevenzione basata sull'ordinamento delle risorse si può impiegare semplicemente definendo un ordinamento *totale* tra le risorse del sistema. Ciò significa che alle risorse di tutto il sistema si assegnano numeri unici, e un processo può richiedere una risorsa con numero unico i, su qualsiasi unità d'elaborazione, solo se non possiede una risorsa con un numero unico maggiore di i. Analogamente, si può usare l'algoritmo del banchiere in un sistema distribuito scegliendo uno dei processi del sistema (il *banchiere*) per mantenere le informazioni necessarie per eseguire tale algoritmo. Ogni richiesta di risorsa deve passare attraverso il banchiere.

Questi due schemi si possono adoperare per gestire il problema delle situazioni di stallo in un ambiente distribuito. Lo schema di prevenzione delle situazioni di stallo a ordinamento totale delle risorse è facile da realizzare ed è poco oneroso dal punto di vista dell'elaborazione. Anche l'algoritmo del banchiere si realizza facilmente, ma può richiedere un carico eccessivo. Il banchiere può trasformarsi in una strozzatura poiché il numero di messaggi a esso diretti e da esso provenienti può essere elevato. Sicché l'uso dello schema del banchiere in un sistema distribuito appare poco pratico.

In questo paragrafo si presenta un nuovo schema di prevenzione delle situazioni di stallo, basato su un metodo a ordinamento delle marche temporali con diritto di prelazione delle risorse. Sebbene tale metodo possa gestire ogni situazione di stallo che si può presentare in un sistema distribuito, per semplicità si considera il solo caso di una singola istanza per ciascun tipo di risorsa.

Per controllare il diritto di prelazione si assegna a ogni processo un unico numero di priorità. Questi numeri si usano per decidere se un processo P_i debba attendere un processo P_j. Ad esempio, si può lasciare che P_i attenda P_j se P_i ha una priorità più alta di P_j; altrimenti si annullano gli effetti dell'azione di P_i. Questo schema previene le situazioni di stallo perché, per ogni arco $P_i \rightarrow P_j$ del grafo d'attesa, P_i ha una priorità più alta di P_j, e quindi non può esserci un ciclo.

Purtroppo questo schema è soggetto a situazioni d'attesa indefinita: i processi con priorità molto bassa possono sempre subire l'annullamento della loro azione. Questa difficoltà si può evitare impiegando le marche temporali. Al momento della sua creazione, ogni processo del sistema riceve una marca temporale unica. Sono stati proposti due metodi complementari di prevenzione delle situazioni di stallo che si servono delle marche temporali:

1. **Schema attesa-morte.** Questo metodo è basato su una tecnica senza diritto di prelazione. Quando un processo P_i richiede una risorsa correntemente posseduta da P_j, si permette a P_i di attendere solo se ha una marca temporale inferiore a quello di P_j (cioè P_i è più vecchio di P_j); altrimenti l'azione di P_i viene annullata (P_i *muore*). Si supponga, ad esempio, che i processi P_1, P_2 e P_3 abbiano rispettivamente marche temporali 5, 10 e 15. Se P_1 richiede una risorsa posseduta da P_2, P_1 attende. Se P_3 richiede una risorsa posseduta da P_2, P_3 viene annullato.

2. **Schema ferita-attesa.** Questo metodo è basato su una tecnica con diritto di prelazione ed è il complementare del sistema attesa-morte. Quando un processo P_i richiede una risorsa correntemente posseduta da P_j, si permette a P_i di attendere solo se ha una marca temporale maggiore di quella di P_j (cioè P_i è più giovane di P_j). Altrimenti si sottrae la risorsa a P_j, la cui azione viene annullata, e la si assegna a P_i (P_j è stato *ferito* da P_i). Ritornando all'esempio precedente con i processi P_1, P_2 e P_3, se P_1 richiede una risorsa posseduta da P_2, la risorsa viene sottratta a P_2 e l'azione di P_2 viene annullata. Se P_3 richiede una risorsa posseduta da P_2, P_3 deve attendere.

Entrambi gli schemi possono evitare le situazioni d'attesa indefinita, purché, quando un processo subisce l'annullamento della sua azione, *non* riceva una nuova marca temporale. Poiché le marche temporali aumentano costantemente, un processo la cui azione è stata annullata ha probabilmente la marca temporale minore, e quindi non sarà annullato

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ancora una volta. Tuttavia esistono notevoli differenze tra i modi in cui i due schemi lavorano:

- Nello schema attesa-morte un processo più vecchio deve attendere che un processo più giovane rilasci la sua risorsa. Quindi più il processo invecchia, più rischia di dover attendere. Viceversa, nello schema ferita-attesa un processo più vecchio non attende mai un processo più giovane.
- Nello schema attesa-morte, se un processo P_i muore e si annullano gli effetti della sua azione perché richiede una risorsa posseduta dal processo P_j, allora P_i, dopo essere stato riavviato, può eseguire di nuovo la stessa sequenza di richieste. Se la risorsa è sempre posseduta da P_j, allora P_i muore di nuovo. Quindi, P_i può morire più volte prima di riuscire ad acquisire la risorsa richiesta. Questa sequenza d'eventi contrasta con quel che accade nello schema ferita-attesa. Il processo P_i viene ferito e la sua azione viene annullata perché P_j ha richiesto una sua risorsa. Quando P_i viene riavviato e richiede la risorsa che ora è in possesso di P_j, P_i attende. Quindi, con lo schema ferita-attesa si hanno meno annullamenti.

Il problema principale con entrambi gli schemi è costituito dal fatto che si possono verificare annullamenti non necessari.

17.5.2 Rilevamento delle situazioni di stallo

L'algoritmo di prevenzione delle situazioni di stallo può esercitare il diritto di prelazione sottraendo risorse anche se non si è verificato alcuno stallo. Per prevenire inutili sottrazioni di risorse si può usare un algoritmo di rilevamento delle situazioni di stallo. Si costruisce un grafo d'attesa che descrive lo stato di assegnazione delle risorse, e poiché si considera un'unica risorsa per ogni tipo, la presenza di un ciclo nel grafo d'attesa indica la presenza di uno stallo.

Il problema principale in un sistema distribuito è quello di decidere come mantenere il grafo d'attesa. Nel seguito sono descritti alcuni metodi per affrontare questo problema. Questi metodi richiedono che ogni sito abbia un grafo d'attesa *locale*. I nodi del grafo corrispondono a tutti i processi (locali e non) correntemente in possesso di qualche risorsa locale di quel sito, o che la stanno richiedendo. Nella Figura 17.3, ad esempio,

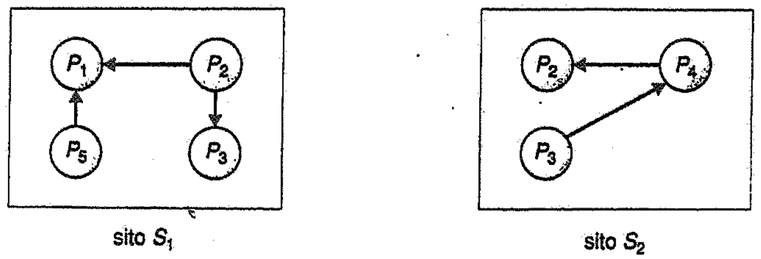


Figura 17.3 Due grafi d'attesa locali.

è presentato un sistema composto di due siti, ciascuno con il proprio grafo d'attesa locale. Occorre notare che i processi P_2 e P_3 compaiono in entrambi i grafi, ciò indica che i processi hanno richiesto risorse in entrambi i siti.

Questi grafi d'attesa locali sono costruiti nel solito modo per processi e risorse locali. Quando un processo P_i del sito S_1 necessita di una risorsa posseduta dal processo P_j del sito S_2, P_i invia un messaggio di richiesta al sito S_2. A questo punto s'inserisce l'arco $P_i \rightarrow P_j$ nel grafo locale d'attesa del sito S_2.

Chiaramente, se in uno dei grafi d'attesa locali c'è un ciclo, significa che si è verificato uno stallo. D'altra parte, il fatto che non vi siano cicli in alcuno dei grafi d'attesa locali non significa che non vi siano situazioni di stallo. Per comprendere questo problema si consideri il sistema illustrato nella Figura 17.3. I grafi d'attesa sono tutti aciclici; ciononostante, nel sistema c'è uno stallo. Per provare che non si è verificato alcuno stallo, occorre mostrare che l'**unione** di tutti i grafi locali è aciclica. Il grafo illustrato nella Figura 17.4 è ottenuto dall'unione dei due grafi d'attesa della Figura 17.3, ma contiene un ciclo, quindi il sistema è in stallo.

Esistono diversi metodi per organizzare il grafo d'attesa relativo a un sistema distribuito. Nel seguito sono descritti alcuni tra gli schemi più comuni.

17.5.2.1 Metodo centralizzato

Il metodo centralizzato prevede che si costruisca il grafo d'attesa globale unendo tutti i grafi d'attesa locali e sia mantenuto in un *singolo* processo: il **coordinatore per il rilevamento delle situazioni di stallo**. Poiché nel sistema c'è un ritardo di comunicazione, occorre distinguere tra due tipi di grafo d'attesa: il grafo *reale* descrive lo stato reale, ma sconosciuto, del sistema in qualsiasi istante, così come sarebbe visto da un osservatore onnisciente; il grafo *costruito* è un'approssimazione generata dal coordinatore durante l'esecuzione del suo algoritmo. Il grafo costruito si deve generare in modo che, ogni volta che s'invoca l'algoritmo di rilevamento, i risultati indicati siano corretti. Per *correttezza* s'intende che valgano le due seguenti proprietà:

- se esiste uno stallo, la sua esistenza viene correttamente riportata;
- se si indica uno stallo, il sistema è effettivamente in stallo.

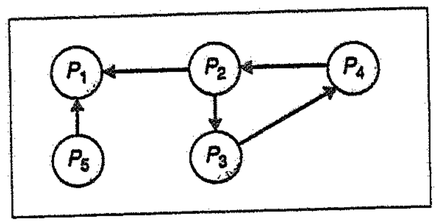


Figura 17.4 Grafo d'attesa globale per la Figura 17.3.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La costruzione di simili algoritmi non è facile.

Il grafo d'attesa si può costruire in tre momenti diversi:

1. ogni volta che s'inserisce o si rimuove un nuovo arco in uno dei grafi d'attesa locali;
2. periodicamente, quando si verifica un certo numero di cambiamenti nel grafo d'attesa;
3. ogni volta che il coordinatore per il rilevamento delle situazioni di stallo deve invocare l'algoritmo di rilevamento dei cicli.

Si consideri il punto 1. Ogni volta che s'inserisce o si toglie un arco da un grafo locale, il sito locale deve inviare un messaggio al coordinatore per informarlo di questa modifica. Alla ricezione di tale messaggio, il coordinatore aggiorna il suo grafo globale. In alternativa (punto 2), un sito può indicare periodicamente un certo numero di cambiamenti in un unico messaggio. Ritornando all'esempio precedente, il processo coordinatore mantiene il grafo d'attesa così come appare nella Figura 17.4. Quando il sito S_2 inserisce l'arco $P_3 \rightarrow P_4$ nel proprio grafo d'attesa locale, invia anche un messaggio al coordinatore. Analogamente, quando il sito S_1 cancella l'arco $P_5 \rightarrow P_1$, perché P_1 ha rilasciato una risorsa richiesta da P_5, s'invia un opportuno messaggio al coordinatore.

Quando s'invoca l'algoritmo per il rilevamento delle situazioni di stallo, il coordinatore esamina il proprio grafo globale. Se individua un ciclo, si sceglie una *vittima* da annullare. Il coordinatore deve informare tutti i siti che un dato processo è stato scelto come vittima, in modo che, a loro volta, ne annullino l'azione.

Occorre notare che con questo schema (punto 1) si possono compiere annullamenti non necessari. Si possono, infatti, presentare le seguenti situazioni:

1. Nel grafo d'attesa globale possono esistere **cicli falsi**. Per illustrare questo punto si considera un'istantanea del sistema come quella illustrata nella Figura 17.5. Si supponga che P_2 rilasci la risorsa di cui è in possesso nel sito S_1, causando la cancellazione dell'arco $P_1 \rightarrow P_2$ in S_1. Quindi il processo P_2 richiede una risorsa posseduta da P_3 nel sito S_2, causando l'aggiunta dell'arco $P_2 \rightarrow P_3$ in S_2. Se il messaggio d'*inserimento* $P_2 \rightarrow P_3$ inviato da S_2 arriva prima del messaggio di *cancellazione* $P_1 \rightarrow P_2$ inviato da S_1, il coordinatore può rilevare il falso ciclo $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1$ dopo l'*inserimento*, ma prima della *cancellazione*. In tale situazione si potrebbe avviare il ripristino dallo stallo, anche se questo non si è verificato.
2. Si possono compiere annullamenti non necessari anche quando si è effettivamente verificato uno stallo ed è stata scelta una vittima, ma *contemporaneamente* l'esecuzione di uno dei processi è stata terminata, per motivi indipendenti dallo stallo stesso, ad esempio il fatto che il processo abbia superato il tempo d'elaborazione assegnatogli. Si supponga, ad esempio, che il sito S_1 della Figura 17.3 decida di interrompere l'esecuzione di P_2. Nello stesso tempo, il coordinatore ha rilevato un ciclo e ha scelto P_3 come vittima. Vengono annullati sia P_2 sia P_3, anche se tale operazione sarebbe stata necessaria solo per P_2.

Gli stessi problemi sorgono anche ricorrendo a soluzioni corrispondenti ai punti 2 e 3.

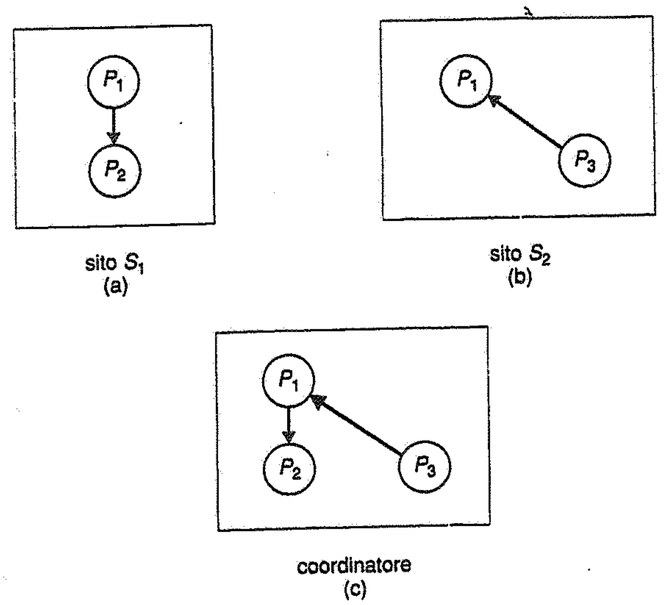


Figura 17.5 Grafi d'attesa locali e globale.

A questo punto si presenta un algoritmo centralizzato per il rilevamento delle situazioni di stallo, secondo il punto 3, che rileva tutte le situazioni di stallo che si verificano effettivamente, mentre non rileva quelle false. Per evitare di indicare false situazioni di stallo, è necessario che le richieste provenienti da siti diversi siano contraddistinte da identificatori (o marche temporali). Quando il processo P_i del sito S_1 richiede una risorsa a P_j del sito S_2, s'invia un messaggio di richiesta con marca temporale *TS*. L'arco $P_i \rightarrow P_j$ con etichetta *TS* viene inserito nel grafo d'attesa locale di S_1. Quest'arco viene inserito nel grafo d'attesa locale del sito S_2 solo se il sito S_2 ha ricevuto il messaggio di richiesta e non può concedere immediatamente la risorsa richiesta. Una richiesta da P_i diretta a P_j nello stesso sito si gestisce nel solito modo; nessuna marca temporale viene associata all'arco $P_i \rightarrow P_j$. L'algoritmo di rilevamento è dunque il seguente:

1. Il controllore invia un messaggio d'avvio a ogni sito del sistema.
2. Ricevendo questo messaggio, un sito invia al coordinatore il proprio grafo d'attesa locale. Ciascuno di questi grafi d'attesa contiene tutte le informazioni locali di cui il sito dispone sullo stato del grafo reale. Il grafo riflette uno stato istantaneo del sito, ma non è sincronizzato rispetto ad alcun altro sito.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

3. Una volta ricevuta una risposta da ogni sito, il controllore costruisce un grafo nel modo illustrato di seguito:
   a) il grafo costruito contiene un vertice per ogni processo del sistema;
   b) il grafo ha un arco $P_i \rightarrow P_j$ *se e solo se* esiste un arco $P_i \rightarrow P_j$ in uno dei grafi d'attesa, *oppure* un arco $P_i \rightarrow P_j$ con un'etichetta *TS* compare in più di un grafo d'attesa.

A questo punto si può affermare che, se nel grafo costruito esiste un ciclo, il sistema è in stallo. Se il grafo costruito non contiene alcun ciclo, il sistema non era in stallo nel momento in cui era stato invocato l'algoritmo di rilevamento in seguito ai messaggi d'avvio inviati dal coordinatore (nel passo 1).

17.5.2.2 Metodo totalmente distribuito

Nell'**algoritmo totalmente distribuito per il rilevamento delle situazioni di stallo**, tutti i controllori condividono in eguale misura la responsabilità del rilevamento delle situazioni di stallo. In questo schema ogni sito costruisce un grafo d'attesa che rappresenta una parte del grafo totale, dipendente dal comportamento dinamico del sistema. L'idea consiste nel fatto che, se c'è uno stallo, almeno uno dei grafi parziali contiene un ciclo. Nel seguito si presenta un algoritmo di questo tipo, che prevede la costruzione di grafi parziali in ogni sito.

Ogni sito mantiene il proprio grafo d'attesa locale. In questo schema il grafo d'attesa locale è diverso da quello descritto precedentemente perché, in questo caso, si aggiunge un ulteriore nodo P_{ex} al grafo. Nel grafo esiste un arco $P_i \rightarrow P_{ex}$ se P_i attende una risorsa di un altro nodo posseduta da qualche processo. Analogamente, nel grafo esiste un arco $P_{ex} \rightarrow P_j$ se un processo in un altro sito attende di acquisire una risorsa attualmente posseduta da P_j in questo sito locale.

Per illustrare questa situazione si considerino i due grafi d'attesa locali della Figura 17.3. L'aggiunta del nodo P_{ex} nei due grafi determina la formazione dei grafi d'attesa locali della Figura 17.6.

Se un grafo d'attesa locale contiene un ciclo che non comprende il nodo P_{ex}, il sistema è in stallo. Se, però, esiste un ciclo che comprende P_{ex}, ciò implica la *possibilità* di uno stallo. Per accertarsi dell'esistenza di uno stallo occorre impiegare un algoritmo distribuito per il rilevamento delle situazioni di stallo.

Si supponga che nel sito S_i il grafo d'attesa locale contenga un ciclo che comprende il nodo P_{ex}. Questo ciclo deve avere la forma

$$P_{ex} \rightarrow P_{k_1} \rightarrow P_{k_2} \rightarrow \ldots \rightarrow P_{k_n} \rightarrow P_{ex},$$

che indica che la transazione P_{k_n} nel sito S_i attende l'acquisizione di una risorsa presente in qualche altro sito, ad esempio S_j. Quando rileva questo ciclo, il sito S_i invia al sito S_j un messaggio di rilevamento di uno stallo contenente le informazioni sul ciclo.

Quando il sito S_j riceve questo messaggio, aggiorna il proprio grafo d'attesa locale con le nuove informazioni, quindi cerca nel grafo d'attesa appena costruito un ciclo che non comprenda P_{ex}. Se tale ciclo esiste, è stato trovato uno stallo e si attiva un opportuno schema di ripristino. Se si scopre un ciclo che comprende P_{ex}, S_j trasmette un mes-

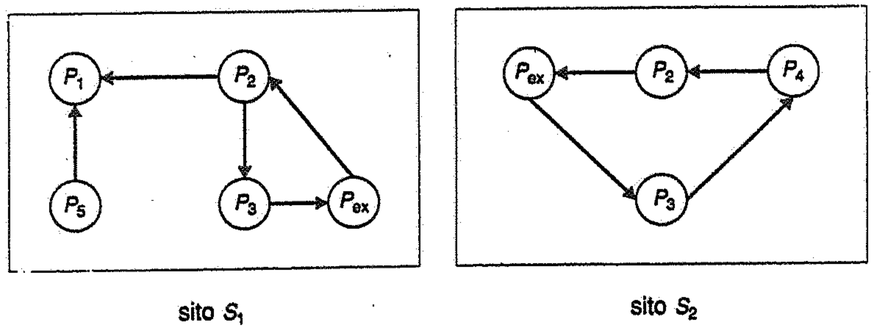


Figura 17.6 Grafi d'attesa locali della Figura 17.3 con l'aggiunta del nodo P_{ex}.

saggio di rilevamento di uno stallo al sito interessato, ad esempio S_k, che a sua volta ripete la procedura. Quindi, dopo un numero finito di volte, o si scopre uno stallo oppure la procedura di rilevamento delle situazioni di stallo si arresta.

Per illustrare questa procedura si considerino i grafi d'attesa locali illustrati nella Figura 17.6. Si supponga che il sito S_1 scopra il ciclo

$$P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}.$$

Poiché P_3 attende di acquisire una risorsa presente nel sito S_2, il sito S_1 trasmette a S_2 un messaggio di rilevamento dello stallo che descrive il ciclo. Quando riceve questo messaggio, il sito S_2 aggiorna il proprio grafo d'attesa locale, ottenendo il grafo d'attesa illustrato nella Figura 17.7. Questo grafo contiene il ciclo

$$P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2,$$

che non contiene P_{ex}. Quindi il sistema è in stallo e si deve attivare un opportuno schema di ripristino.

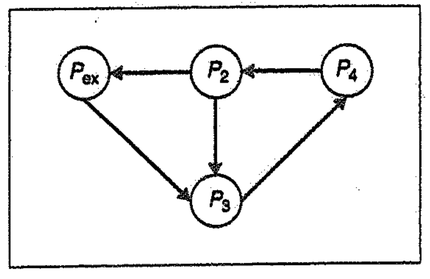


Figura 17.7 Grafo d'attesa locale aggiornato del sito S_2 della Figura 17.6.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Si noti che l'esito sarebbe stato lo stesso anche se il sito S_2 avesse scoperto prima il ciclo nel proprio grafo d'attesa locale e avesse inviato il messaggio di rilevamento dello stallo al sito S_1. Nel peggiore dei casi entrambi i siti scoprono il ciclo pressoché nello stesso istante, e così s'inviano due messaggi di rilevamento dello stallo: uno da S_1 a S_2 e un altro da S_2 a S_1. Questa situazione causa un trasferimento di messaggi non necessario, e il carico dovuto all'aggiornamento dei due grafi d'attesa locali e alla ricerca dei cicli in entrambi i grafi.

Per ridurre il traffico di messaggi, si assegna un identificatore unico, denotato da $ID(P_i)$, a ogni processo P_i. Quando il sito S_k scopre che il proprio grafo d'attesa locale contiene un ciclo che comprende il nodo P_{ex}, la cui forma è del tipo

$$P_{ex} \rightarrow P_{k_1} \rightarrow P_{k_2} \rightarrow \ldots \rightarrow P_{k_n} \rightarrow P_{ex},$$

invia un messaggio di rilevamento dello stallo a un altro sito solo se

$$ID(P_{k_n}) < ID(P_{k_1}),$$

altrimenti il sito S_k continua la sua normale esecuzione, lasciando il compito di avviare l'algoritmo per il rilevamento delle situazioni di stallo a qualche altro sito.

Per comprendere questo schema, si considerino nuovamente i grafi d'attesa dei siti S_1 e S_2 rappresentati nella Figura 17.6. Si supponga che

$$ID(P_1) < ID(P_2) < ID(P_3) < ID(P_4)$$

e che entrambi i siti scoprano questi cicli locali pressoché nello stesso istante. Il ciclo del sito S_1 è del tipo

$$P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}.$$

Poiché $ID(P_3) > ID(P_2)$, il sito S_1 non invia a S_2 alcun messaggio di rilevamento di stallo.

Il ciclo del sito S_2 è del tipo

$$P_{ex} \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_{ex}.$$

Poiché $ID(P_2) < ID(P_3)$, il sito S_2 invia un messaggio di rilevamento dello stallo al sito S_1, il quale, ricevendo questo messaggio, aggiorna il proprio grafo d'attesa locale, cerca un ciclo nel grafo e scopre che il sistema è in stallo.

17.6 Algoritmi di elezione

Come si è evidenziato nel Paragrafo 17.3, molti algoritmi distribuiti impiegano un processo coordinatore che esegue funzioni richieste da altri processi del sistema. Tra queste funzioni sono presenti l'imposizione della mutua esclusione, il mantenimento di un grafo d'attesa globale per il rilevamento delle situazioni di stallo, la sostituzione di un contrassegno perduto, il controllo di un dispositivo di I/O del sistema. Se il processo coordinatore si guasta a causa di un difetto del sito in cui risiede, il sistema può continuare l'esecuzione solo avviando una nuova copia del coordinatore in un altro sito. Gli algoritmi che determinano dove si debba avviare una nuova copia del coordinatore si chiamano **algoritmi di elezione**.

Gli algoritmi di elezione si fondano sul presupposto che a ogni processo attivo del sistema sia associato un numero di priorità unico. Per semplificare la notazione conviene supporre che il numero di priorità del processo P_i sia i. Un'ulteriore semplificazione si ottiene ipotizzando una corrispondenza da uno a uno tra processi e siti, che consente di considerare soltanto i processi. Il coordinatore è sempre il processo col numero di priorità maggiore, perciò, quando un coordinatore si guasta, l'algoritmo deve eleggere il processo attivo con il numero di priorità maggiore. Questo numero deve essere inviato a ogni processo attivo del sistema. Inoltre, l'algoritmo deve fornire un meccanismo per consentire ai processi ripristinati dopo un guasto di identificare l'attuale coordinatore.

In questo paragrafo sono presentati alcuni esempi di algoritmi di elezione per due diverse configurazioni di sistemi distribuiti. Il primo algoritmo si applica a sistemi in cui ogni processo può inviare un messaggio a ogni altro processo del sistema. Il secondo algoritmo si applica a sistemi organizzati, fisicamente o logicamente, in forma d'anello. Entrambi gli algoritmi richiedono n^2 messaggi per un'elezione, dove n è il numero dei processi presenti nel sistema. Si suppone che un processo che si è guastato riconosca, al momento del ripristino, di essersi guastato e quindi intraprenda le opportune azioni per ricongiungersi all'insieme dei processi attivi.

17.6.1 Algoritmo dello spaccone

Si supponga che il processo P_i invii una richiesta che non ottiene risposta da parte del coordinatore entro un intervallo t. In questo caso si suppone che il coordinatore sia guasto, e P_i tenta di eleggersi nuovo coordinatore. Questo compito si svolge tramite il seguente algoritmo.

Il processo P_i invia un messaggio di elezione a ogni processo con numero di priorità maggiore. Il processo P_i attende poi per un intervallo t che arrivi una risposta da uno di questi processi.

Se entro l'intervallo t non giunge alcuna risposta, P_i suppone che tutti i processi con numeri maggiori di i siano guasti e si elegge nuovo coordinatore. Il processo P_i avvia una nuova copia del coordinatore e invia un messaggio per informare tutti i processi attivi con numeri di priorità minori di i che P_i è il nuovo coordinatore.

Se invece arriva una risposta, P_i comincia ad attendere, per un intervallo t', un messaggio che lo informi dell'elezione di uno tra i processi con numero di priorità maggiore (qualche altro processo si sta eleggendo coordinatore e deve indicare il risultato entro l'intervallo t'). Se entro tale intervallo non viene inviato alcun messaggio di questo tipo, si suppone che il processo con il numero maggiore sia guasto e il processo P_i deve riavviare l'algoritmo.

Se P_i non è il coordinatore, allora, in un momento qualsiasi dell'esecuzione, può ricevere dal processo P_j uno tra i seguenti messaggi:

1. P_j è il nuovo coordinatore ($j > i$), e il processo P_i, a sua volta, registra quest'informazione;
2. P_j ha avviato un'elezione ($j < i$), il processo P_i invia una risposta a P_j e inizia il proprio algoritmo di elezione, purché P_i non abbia già iniziato tale elezione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il processo che completa il suo algoritmo ha il numero maggiore e viene eletto coordinatore; ha inviato il proprio numero a tutti i processi attivi con numero minore. Dopo che un processo guasto è stato ripristinato, comincia immediatamente l'esecuzione dello stesso algoritmo. Se non esistono processi attivi con numeri maggiori, il processo ripristinato s'impone come coordinatore su tutti i processi con numeri minori, anche se esiste un coordinatore attualmente attivo con un numero minore; per questo motivo l'algoritmo si chiama **algoritmo dello spaccone**.

A questo punto s'illustra il funzionamento dell'algoritmo con un semplice esempio relativo a un sistema formato dai processi da P_1 a P_4. Le operazioni sono le seguenti:

1. Tutti i processi sono attivi; P_4 è il processo coordinatore.
2. P_1 e P_4 si guastano. P_2 stabilisce che si è guastato P_4 inviando una richiesta cui non viene data risposta entro l'intervallo t. P_2 inizia allora il proprio algoritmo di elezione inviando una richiesta a P_3.
3. P_3 riceve la richiesta, risponde a P_2 e inizia il proprio algoritmo inviando una richiesta di elezione a P_4.
4. P_2 riceve la risposta di P_3 e inizia l'attesa per un intervallo t'.
5. P_4 non risponde entro l'intervallo t, sicché P_3 si elegge nuovo coordinatore e invia il numero 3 a P_2 e P_1, ma P_1 non lo riceve poiché è guasto.
6. Più tardi, quando P_1 viene ripristinato, invia una richiesta di elezione a P_2, P_3 e P_4.
7. P_2 e P_3 rispondono a P_1 e iniziano i rispettivi algoritmi di elezione. P_3 viene nuovamente eletto poiché si verificano gli stessi eventi di prima.
8. Infine, P_4 viene ripristinato e informa P_1, P_2 e P_3 di essere l'attuale coordinatore. P_4 non invia richieste di elezione, poiché è il processo col numero maggiore in tutto il sistema.

17.6.2 Algoritmo ad anello

L'**algoritmo ad anello** è basato sul fatto che i collegamenti sono unidirezionali e che i processi inviano i messaggi ai loro vicini di destra. La principale struttura di dati impiegata dall'algoritmo è la **lista attiva**, la quale contiene i numeri di priorità di tutti i processi attivi nel sistema al termine dell'algoritmo. Ogni processo conserva la propria lista attiva. L'algoritmo funziona nel modo seguente:

1. Se rileva un guasto nel coordinatore, il processo P_i crea una nuova lista attiva che è inizialmente vuota. Quindi invia un messaggio `elect(i)` al vicino alla sua destra e aggiunge il numero i alla propria lista attiva.
2. Se riceve un messaggio `elect(j)` dal processo alla sua sinistra, P_i deve rispondere in uno dei tre modi seguenti:

   a) Se è il primo messaggio `elect` che ha ricevuto o inviato, P_i crea una nuova lista attiva con i numeri i e j. Quindi invia il messaggio `elect(i)`, seguito dal messaggio `elect(j)`.
   b) Se $i \neq j$, cioè il messaggio ricevuto non contiene il numero di P_i, allora P_i aggiunge j alla propria lista attiva e inoltra il messaggio al vicino alla sua destra.
   c) Se $i = j$, cioè P_i riceve il messaggio `elect(i)`, allora la lista attiva di P_i contiene i numeri di tutti i processi attivi del sistema. Il processo P_i può ora determinare il numero maggiore nella lista attiva per identificare il nuovo processo coordinatore.

Quest'algoritmo non specifica come un processo nella fase di ripristino possa determinare il numero del processo che è attualmente coordinatore. Una soluzione è quella di richiedere al processo nella fase di ripristino di inviare un messaggio d'interrogazione. Questo messaggio viene ritrasmesso nell'anello fino al coordinatore attuale, il quale invia una risposta contenente il proprio numero.

17.7 Raggiungimento di un accordo

Affinché un sistema sia affidabile deve esistere un meccanismo che permetta a un insieme di processi di accordarsi su un *valore* comune. Tale accordo può non esserci per diversi motivi: innanzi tutto il mezzo di comunicazione può essere difettoso, quindi si verificano perdite o alterazioni dei messaggi; in secondo luogo, gli stessi processi possono essere difettosi, quindi il loro comportamento diviene imprevedibile; in questo caso bisogna sperare che i processi arrestino la propria esecuzione senza deviare dal loro comportamento normale. Nel caso peggiore, i processi possono inviare messaggi confusi o non corretti ad altri processi, o addirittura collaborare con altri processi difettosi nel tentativo di distruggere l'integrità del sistema.

Questo problema è rappresentato dal **problema dei generali bizantini**. Diverse divisioni dell'esercito bizantino, ciascuna comandata dal proprio generale, circondano un campo nemico. I generali bizantini devono raggiungere un accordo sul fatto di attaccare o no il nemico all'alba. È fondamentale che tutti i generali siano d'accordo, altrimenti un attacco portato solo da alcune divisioni può risolversi in una sconfitta. Le diverse divisioni sono geograficamente sparse e i generali possono comunicare tra loro solo tramite messaggeri che corrono da un campo all'altro. I generali possono non essere in grado di raggiungere un accordo per almeno due ragioni principali:

1. Il nemico potrebbe catturare i messaggeri, che in questo caso non potrebbero consegnare i loro messaggi. Questa situazione corrisponde alla comunicazione inaffidabile di un calcolatore, discussa nel Paragrafo 17.7.1.
2. Alcuni generali possono essere *traditori* che tentano d'impedire che i generali *leali* raggiungano un accordo. Questa situazione corrisponde ai processi difettosi di un sistema di calcolo, argomento trattato nel Paragrafo 17.7.2.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

17.7.1 Comunicazione inaffidabile

Si supponga che, se dei processi si guastano, lo facciano in modo chiaro e che il mezzo di comunicazione sia inaffidabile. Si supponga che il processo P_i del sito S_1, che ha inviato un messaggio al processo P_j del sito S_2, debba sapere se P_j ha ricevuto il messaggio per poter decidere come procedere con l'elaborazione. P_i può decidere, ad esempio, di calcolare una funzione f se P_j ha ricevuto il messaggio, oppure di calcolare una funzione g se P_j non ha ricevuto il messaggio (a causa di un guasto fisico).

Per rilevare i guasti si può impiegare uno schema d'attesa simile a quello descritto nel Paragrafo 15.6.1. Quando P_i invia un messaggio, specifica anche un intervallo di tempo entro il quale è disposto ad attendere un messaggio di conferma da parte di P_j. Quando riceve il messaggio, P_j invia immediatamente un messaggio di conferma a P_i; se P_i riceve la conferma entro l'intervallo specificato può concludere con certezza che P_j ha ricevuto il suo messaggio; altrimenti, P_i deve ritrasmettere il suo messaggio e attendere una conferma. Questa procedura continua finché P_i non ottiene il messaggio di risposta, o il sistema gli comunica che il sito S_2 è fuori servizio. Nel primo caso, P_i calcola f; nel secondo caso, calcola g. Occorre notare che, se queste sono le uniche possibilità, P_i deve attendere finché non riceve la comunicazione che si è verificata una delle due situazioni.

Si supponga ora che anche P_j abbia bisogno di sapere se P_i ha ricevuto il messaggio di conferma, per decidere come procedere con la propria elaborazione. Ad esempio, P_j può calcolare f solo se gli si assicura che P_i ha ricevuto la sua conferma. In altre parole, P_i e P_j calcolano f se e solo se entrambi hanno raggiunto un accordo. Risulta che, alla presenza di un guasto, non è possibile svolgere questo compito. Più precisamente, non è possibile che in un ambiente distribuito i processi P_i e P_j concordino completamente sui rispettivi stati.

Per dimostrare quest'asserzione si supponga l'esistenza di una sequenza minima di trasferimenti di messaggi tale che, una volta consegnati i messaggi, entrambi i processi concordino sul calcolo di f. Sia m' l'ultimo messaggio inviato da P_i a P_j. Poiché P_i non sa se il suo messaggio arriverà a P_j, poiché il messaggio può andare perduto per un guasto, P_i esegue f a prescindere dall'esito della consegna del messaggio. Quindi, si può rimuovere m' dalla sequenza senza influire sulla procedura di decisione. Di conseguenza, la sequenza originale non è minima, contraddicendo l'ipotesi e dimostrando che non esiste detta sequenza. I processi non possono mai avere la certezza che entrambi eseguiranno il calcolo di f.

17.7.2 Processi difettosi

Si supponga che il mezzo di comunicazione sia affidabile, ma che i processi possano guastarsi in modi imprendibili. Si consideri un sistema di n processi, dei quali non più di m siano difettosi. Si supponga che ogni processo P_i abbia un valore privato di V_i. Si vuole ideare un algoritmo che permetta a ogni processo non difettoso P_i di costruire un vettore $X_i = (A_{i,1}, A_{i,2}, \ldots, A_{i,n})$ tale che valgano le seguenti condizioni:

1. se P_j è un processo non difettoso, allora $A_{i,j} = V_j$;
2. se P_i e P_j sono entrambi processi non difettosi, allora $X_i = X_j$.

Esistono molte soluzioni a questo problema; esse hanno in comune le seguenti proprietà:

1. si può ideare un algoritmo corretto solo se $n \geq 3 \times m + 1$;
2. il ritardo maggiore nel raggiungimento dell'accordo è proporzionale al ritardo dovuto a $m + 1$ scambi di messaggi;
3. il numero dei messaggi richiesti per raggiungere un accordo è elevato; nessun singolo processo è affidabile, quindi tutti i processi devono raccogliere tutte le informazioni e prendere le proprie decisioni.

Anziché presentare una soluzione generale, che sarebbe complicata, si presenta un algoritmo per il semplice caso in cui $m = 1$ e $n = 4$. L'algoritmo richiede due giri di scambio di informazioni:

1. ogni processo invia il proprio valore privato agli altri tre processi;
2. ogni processo invia le informazioni ottenute nel primo giro a tutti gli altri processi.

Un processo difettoso può naturalmente rifiutare di inviare messaggi; in questo caso, un processo non difettoso può scegliere un valore arbitrario e assumere che quel valore sia stato inviato dal processo difettoso.

Una volta completati questi due giri, un processo P_i non difettoso può costruire il proprio vettore $X_i = (A_{i,1}, A_{i,2}, A_{i,3}, A_{i,4})$ come segue:

1. $A_{i,i} = V_i$.
2. Per $j \neq i$, se almeno due dei tre valori registrati per il processo P_j (nei due giri di scambio) coincidono, allora, per impostare il valore di $A_{i,j}$, s'impiega il valore di maggioranza. Altrimenti, per impostare il valore di $A_{i,j}$, s'impiega un valore predefinito, ad esempio *nil*.

17.8 Sommario

In un sistema distribuito senza memoria comune e clock comune è talvolta impossibile determinare l'ordine esatto in cui si verificano due eventi. La relazione *verificato-prima* rappresenta soltanto un ordinamento parziale degli eventi di un sistema distribuito. Per fornire un coerente ordinamento degli eventi si possono usare le marche temporali.

La mutua esclusione in un ambiente distribuito si può realizzare in diversi modi. Nel caso del metodo centralizzato si sceglie uno tra i processi nel sistema per coordinare gli accessi alla sezione critica. Nel caso del metodo totalmente distribuito il procedimento di decisione è distribuito su tutto il sistema; un algoritmo distribuito applicabile alle reti strutturate ad anello è quello con passaggio di contrassegno.

Affinché l'atomicità sia garantita, tutti i siti nei quali si esegue una transazione T devono accordarsi sull'esito finale dell'esecuzione: successo di T in tutti i siti o fallimento di T in tutti i siti. Per assicurare tale proprietà il coordinatore delle transazioni di T de-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ve eseguire un protocollo di conferma. Il protocollo di conferma più diffuso è il protocollo 2PC.

I vari schemi di controllo della concorrenza che si possono usare in un sistema centralizzato si possono modificare per essere impiegati in un ambiente distribuito. Nel caso dei protocolli d'accesso bloccante, l'unica modifica necessaria riguarda il modo in cui si realizza il gestore dei diritti d'accesso bloccante. Nel caso di schemi con marche temporali e di validazione l'unico cambiamento necessario è lo sviluppo di un meccanismo per la generazione di marche temporali globali uniche. Tale meccanismo può concatenare una marca temporale locale con l'identificatore di sito o far avanzare i clock locali ogniqualvolta arriva un messaggio con una marca temporale maggiore.

Il metodo principale per affrontare le situazioni di stallo in un ambiente distribuito è quello di rilevarle. Il problema principale è quello di decidere come mantenere il grafo d'attesa. I differenti metodi di organizzazione del grafo d'attesa comprendono il metodo centralizzato e il metodo totalmente distribuito.

Alcuni algoritmi distribuiti richiedono un coordinatore. Se il coordinatore non funziona correttamente a causa di un guasto al sito in cui risiede, il sistema può continuare l'esecuzione soltanto se si avvia una nuova copia del coordinatore in qualche altro sito; ciò si può realizzare mantenendo un coordinatore di riserva pronto ad assumere il controllo se il coordinatore non funziona correttamente. Un altro metodo prevede che la scelta del nuovo coordinatore avvenga in seguito al verificarsi di malfunzionamenti nel coordinatore corrente. Gli algoritmi che determinano dove si dovrebbe avviare la copia del coordinatore si chiamano algoritmi di elezione. Per eleggere un nuovo coordinatore, nel caso di malfunzionamenti, si possono usare due algoritmi: l'algoritmo dello spaccone e l'algoritmo ad anello.

17.9 Esercizi

17.1 Discutete vantaggi e svantaggi dei due metodi presentati per la generazione di marche temporali globali uniche.

17.2 Una società sta costruendo una rete di calcolatori e richiede un algoritmo per ottenere la mutua esclusione distribuita. Dite quale schema conviene impiegare e spiegate la risposta.

17.3 Dite perché il rilevamento delle situazioni di stallo è molto più oneroso in un ambiente distribuito di quanto non lo sia in un ambiente centralizzato.

17.4 Una società che sta costruendo una rete di calcolatori richiede lo sviluppo di uno schema per la gestione delle situazioni di stallo.

a) Dite se conviene usare uno schema di rilevamento, oppure uno schema di prevenzione delle situazioni di stallo.

b) Dite quale schema di prevenzione conviene usare e spiegate la risposta.

c) Dite quale schema di rilevamento conviene usare e spiegate la risposta.

17.5 Considerate il seguente algoritmo *gerarchico* di rilevamento delle situazioni di stallo nel quale il grafo d'attesa globale è distribuito fra un certo numero di *controllori* diversi che sono organizzati ad albero. Ogni controllore che non sia una foglia conserva un grafo d'attesa che contiene informazioni concernenti i grafi conservati dai controllori che si trovano nel sottoalbero sottostante. In particolare, siano S_A, S_B e S_C controllori tali che S_C risulta essere l'antenato comune più vicino a S_A e S_B (S_C deve essere unico, poiché si tratta di un albero). Si supponga che il nodo T_i compaia nel grafo d'attesa locale dei controllori S_A e S_B. Allora T_i deve comparire anche nei seguenti grafi d'attesa locali:

- del controllore S_C;
- di ogni controllore che si trovi sul percorso da S_C a S_A;
- di ogni controllore che si trovi sul percorso da S_C a S_B.

Inoltre, se T_i e T_j compaiono nel grafo d'attesa del controllore S_D ed esiste un cammino da T_i a T_j nel grafo d'attesa di uno dei figli di S_D, allora nel grafo d'attesa di S_D deve esistere un arco $T_i \rightarrow T_j$.

Mostrate che se in uno qualsiasi dei grafi d'attesa esiste un ciclo, allora il sistema è in stallo.

17.6 Ricavate un algoritmo di elezione per anelli bidirezionali che sia più efficiente di quello presentato in questo capitolo. Dite quanti messaggi sono necessari per n processi.

17.7 Considerate un guasto che si può verificare durante un 2PC per una transazione. Per ogni possibile guasto, spiegate in che modo il 2PC assicura l'atomicità della transazione nonostante il guasto.

17.10 Note bibliografiche

L'algoritmo distribuito per l'estensione della relazione *verificato-prima* a un ordinamento totale di tutti gli eventi del sistema è stato sviluppato da [Lamport 1978b].

Anche il primo algoritmo generale per la realizzazione della mutua esclusione in un ambiente distribuito è stato sviluppato da [Lamport 1978b]. Lo schema di Lamport richiede $3 \times (n - 1)$ messaggi per ogni ingresso in sezione critica. Successivamente, [Ricart e Agrawala 1981] hanno proposto un algoritmo distribuito che richiede solo $2 \times (n - 1)$ messaggi. Il loro algoritmo è presentato nel Paragrafo 17.2.2. Un algoritmo radice-quadratico per la mutua esclusione distribuita è presentato da [Maekawa 1985]. L'algoritmo con passaggio di contrassegno per sistemi con struttura ad anello, presentato nel Paragrafo 17.2.3, è stato sviluppato da [Lann 1977]. Discussioni riguardanti la mutua esclusione in reti di calcolatori sono presentate da [Carvalho e Roucairol 1983]. Una soluzione efficiente e tollerante i guasti del problema della mutua esclusione distribuita è presentata da [Agrawal e Abbadi 1991]. Una semplice classificazione degli algoritmi per la mutua esclusione distribuita è presentata da [Raynal 1991].

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il problema della sincronizzazione distribuita è trattato da [Reed e Kanodia 1979] (ambiente a memoria condivisa), [Lamport 1978a], [Lamport 1978b] e [Schneider 1982] (processi totalmente disgiunti). Una soluzione distribuita al problema dei cinque filosofi è proposta da [Chang 1980].

Il protocollo 2PC è stato sviluppato da [Lampson e Sturgis 1976] e [Gray 1978]. [Mohan e Lindsay 1983] tratta due versioni modificate del protocollo 2PC, chiamate successo presunto e fallimento presunto, che riducono il carico del protocollo 2PC predefinendo un'ipotesi sul futuro delle transazioni.

Studi sui problemi della realizzazione del concetto di transazione in una base di dati distribuita sono presentati da [Gray 1981], [Traiger et al. 1982], e [Spector e Schwarz 1983]. Esaurienti discussioni riguardanti il controllo della concorrenza in ambienti distribuiti sono offerte da [Bernstein et al. 1987].

[Rosenkrantz et al. 1978] presenta l'algoritmo distribuito di prevenzione delle situazioni di stallo con marche temporali. Lo schema totalmente distribuito di rilevamento delle situazioni di stallo presentato nel Paragrafo 17.5.2 è stato sviluppato da [Obermarck 1982]. Lo schema gerarchico di rilevamento delle situazioni di stallo dell'Esercizio 17.5 si trova in [Menasce e Muntz 1979]. Una rassegna sul rilevamento delle situazioni di stallo nei sistemi distribuiti è data da [Knapp 1987] e [Singhal 1989].

Il problema dei generali bizantini è discusso da [Lamport et al. 1982] e [Pease et al. 1980]. L'algoritmo dello spaccone è presentato da [Garcia-Molina 1982]. L'algoritmo di elezione per un sistema con struttura ad anello è stato scritto da [Lann 1977].

Parte sesta

Protezione e sicurezza

Protezione
Sicurezza

I meccanismi di protezione offrono un accesso controllato limitando i tipi d'accesso ai file permessi agli utenti. La protezione deve anche assicurare che solo i processi che hanno ottenuto l'autorizzazione dal sistema operativo possano usare i segmenti di memoria, la CPU e altre risorse.

La protezione è assicurata da un meccanismo che controlla l'accesso dei programmi, processi o utenti alle risorse di un sistema di calcolo. Questo meccanismo deve offrire un mezzo per specificare quali siano i controlli da imporre, e un sistema che permetta di farli rispettare.

La sicurezza garantisce l'autenticazione degli utenti del sistema per proteggere l'integrità delle informazioni (dati e codice) in esso memorizzate e delle risorse fisiche di cui il sistema di calcolo dispone. Il sistema di sicurezza impedisce gli accessi non autorizzati al sistema, i tentativi dolosi di distruzione o alterazione dei dati, e l'introduzione accidentale d'incoerenze.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 18

Protezione

I processi di un sistema operativo devono essere protetti dalle attività degli altri. Molteplici meccanismi provvedono al raggiungimento di tale scopo, vale a dire che file, segmenti di memoria, CPU e altre risorse possono essere adoperate solo dai processi che hanno ottenuto l'autorizzazione dal sistema operativo.

La protezione riguarda il meccanismo per il controllo dell'accesso alle risorse definite da un sistema di calcolo da parte di programmi, processi o utenti. Questo meccanismo deve fornire un mezzo per specificare i controlli da imporre e alcuni mezzi di costrizione. È necessario fare una distinzione tra protezione e sicurezza; quest'ultima è una misura della fiducia sul mantenimento dell'integrità di un sistema e dei suoi dati. La garanzia della sicurezza è un argomento più generico rispetto alla protezione; esso è trattato nel Capitolo 19.

18.1 Scopi della protezione

I calcolatori sono divenuti più complessi e le loro applicazioni sono cresciute a dismisura, perciò è aumentata anche la necessità di proteggere la loro integrità. La protezione era originariamente concepita come un accessorio dei sistemi operativi con multiprogrammazione tale che utenti non fidati potessero condividere tranquillamente uno spazio di nomi logici comuni, come una directory di file, oppure uno spazio di nomi fisici comuni, come la memoria. I moderni concetti di protezione si sono evoluti aumentando l'affidabilità di qualsiasi sistema complesso che usi risorse condivise.

È necessario disporre di sistemi di protezione per diversi motivi. Tra i più ovvi, c'è la necessità di prevenire la violazione intenzionale e dannosa di un vincolo d'accesso da parte di un utente. Tuttavia, ha un'importanza più generale la necessità di assicurare che ogni componente del programma attivo in un sistema impieghi le risorse del sistema in modo coerente con i criteri stabiliti; per un sistema affidabile, questo requisito è assolutamente necessario.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La protezione può migliorare l'affidabilità rilevando errori latenti nelle interfacce tra sottosistemi componenti. Un'individuazione precoce di errori d'interfaccia riesce spesso a impedire la contaminazione di un sottosistema intatto da parte di un sottosistema malfunzionante. Una risorsa non protetta non può difendersi contro l'uso, o l'abuso, da parte di un utente non autorizzato o incompetente. Un sistema orientato alla protezione offre i mezzi per distinguere tra uso autorizzato e non autorizzato.

Il ruolo della protezione è quello di offrire un meccanismo d'imposizione di criteri che controllino l'uso delle risorse. Questi criteri si possono fissare in vari modi: alcuni sono stati fissati nella fase di progettazione del sistema, altri sono determinati dalla gestione di un sistema; altri ancora sono definiti dai singoli utenti per proteggere i loro file e i loro programmi. Un sistema di protezione deve avere una flessibilità tale da consentire d'imporre i diversi criteri che si possono dichiarare per esso.

I criteri d'uso di una risorsa possono variare secondo l'applicazione e possono cambiare nel tempo. Per queste ragioni la protezione non è più una questione riguardante solamente i progettisti di un sistema operativo; anche i programmatori di applicazioni hanno bisogno di meccanismi per la protezione delle risorse create e gestite dai sottosistemi di applicazione. In questo capitolo si descrivono i meccanismi di protezione che il sistema operativo dovrebbe fornire affinché i progettisti di applicazioni possano impiegarli nella progettazione del proprio sistema di protezione.

I *criteri* sono distinti dai *meccanismi*. I meccanismi determinano *come* qualcosa si debba eseguire; i criteri, decidono *cosa* si debba fare. La distinzione tra criteri e meccanismi è importante ai fini della flessibilità. I criteri sono soggetti a cambiamenti connessi al tempo e al luogo. Nel peggiore dei casi qualsiasi cambiamento di criterio richiede un cambiamento anche nel meccanismo sottostante. Sono preferibili meccanismi generali, poiché, in questi casi, un cambiamento di criterio implica soltanto la modifica di alcuni parametri o tabelle del sistema.

18.2 Domìni di protezione

Un sistema di calcolo è un insieme di processi e oggetti. Col nome di *oggetti* si designano sia gli elementi fisici, come la CPU, i segmenti di memoria, le stampanti, i dischi e le unità a nastri, sia gli oggetti logici, come i file, i programmi e i semafori. Ogni oggetto ha un nome unico che lo distingue da tutti gli altri oggetti del sistema; è inoltre possibile accedere a ciascuno di essi solo tramite operazioni ben definite e significative. Gli oggetti sono fondamentalmente tipi di dati astratti.

Le operazioni possibili dipendono dall'oggetto: ad esempio, una CPU può compiere solo elaborazioni; i segmenti di memoria si possono leggere e scrivere; mentre da un lettore di CD-ROM o DVD-ROM si può soltanto leggere; le unità a nastri si possono leggere, scrivere e riavvolgere; i file di dati si possono creare, aprire, leggere, scrivere, chiudere e cancellare; i file di programmi si possono leggere, scrivere, eseguire e cancellare.

A un processo si deve permettere l'accesso alle sole risorse per le quali ha l'autorizzazione. Inoltre, un processo deve poter accedere alle sole risorse di cui ha correntemente bisogno per eseguire il proprio compito. Questo requisito, noto con il nome di *principio del privilegio minimo*, è utile per limitare i danni che possono essere causati al sistema da un processo difettoso. Se, ad esempio, il processo P richiede la procedura A, alla procedura si deve permettere l'accesso solo alle proprie variabili e ai parametri che le vengono passati; non deve poter accedere a tutte le variabili del processo P. Analogamente, si consideri il caso in cui il processo P richieda la compilazione di un particolare file. Al compilatore non si deve permettere l'accesso a qualsiasi file, ma solo al ben definito sottoinsieme di file associato al file da compilare. Viceversa, il compilatore può avere dei file privati, che impiega per scopi contabili o di ottimizzazione, ai quali il processo P non deve aver accesso.

18.2.1 Struttura dei domìni di protezione

Per facilitare questo schema, un processo opera all'interno di un **dominio di protezione**, il quale specifica le risorse cui il processo può accedere.

Ogni dominio definisce un insieme di oggetti e i tipi di operazioni che si possono compiere su ciascun oggetto. La possibilità di eseguire un'operazione su un oggetto è detta **diritto d'accesso**. Un **dominio** è dunque un insieme di diritti d'accesso, ciascuno dei quali è composto di una coppia ordinata *<nome oggetto, insieme dei diritti>*.

Ad esempio, se il dominio D ha il diritto d'accesso *<file F*, {read, write}>, allora un processo in esecuzione nel dominio D può leggere e scrivere il file F, ma non può eseguire altre operazioni su quello stesso oggetto.

I domìni non devono necessariamente essere disgiunti; essi possono condividere diritti d'accesso. Nella Figura 18.1, ad esempio, sono illustrati tre domìni: D_1, D_2 e D_3. Il diritto d'accesso $<O_4$, {print}> è condiviso da D_2 e D_3, implicando che un processo in esecuzione su uno dei due domìni può stampare l'oggetto O_4. Occorre notare che un processo, per leggere e scrivere l'oggetto O_1, deve essere in esecuzione nel dominio D_1. D'altra parte, soltanto i processi che si trovano nel dominio D_3 possono eseguire l'oggetto O_1.

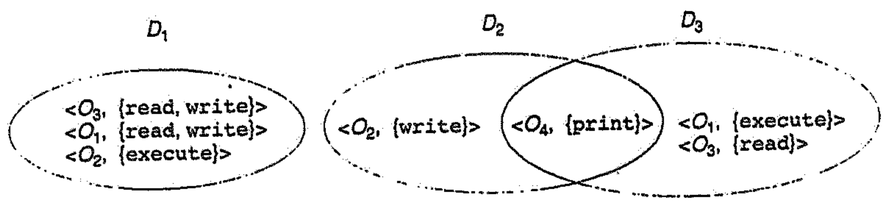


Figura 18.1 Sistema con tre domìni di protezione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

L'associazione tra un processo e un dominio può essere **statica**, se l'insieme delle risorse disponibili per un processo è fissato per tutta la durata del processo, o **dinamica**. Com'è prevedibile, la determinazione dei domìni di protezione dinamici è più complicata della determinazione dei domìni di protezione statici.

Se l'associazione tra processi e domìni è fissa, per aderire al principio del privilegio minimo è necessario disporre di un meccanismo che permetta di modificare il contenuto di un dominio. L'esecuzione di un processo si può dividere in due fasi. Ad esempio, il processo può richiedere l'accesso per lettura in una fase e l'accesso per scrittura in un'altra. Se un dominio è statico, occorre definire un dominio che contenga sia l'accesso per la lettura sia quello per la scrittura. Tuttavia questa disposizione fornisce più diritti di quanti siano necessari in ciascuna delle due fasi, poiché è disponibile il diritto d'accesso per la lettura nella fase in cui è necessario il solo accesso per la scrittura e viceversa; quindi il principio del privilegio minimo è violato. È necessario permettere che il contenuto di un dominio sia modificato, in modo che il dominio rifletta sempre i minimi diritti d'accesso necessari.

Se l'associazione è dinamica, deve essere disponibile un meccanismo per permettere a un processo di passare da un dominio all'altro. Si può anche permettere la modifica al contenuto di un dominio. Se non è possibile modificare il contenuto di un dominio, si può ottenere lo stesso effetto creando un nuovo dominio con il contenuto modificato e passando al nuovo dominio quando sia necessario modificare il contenuto del dominio originario.

Si noti che un dominio si può realizzare in diversi modi:

- Ogni *utente* può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere dipende dall'identità dell'utente. Il **cambio di dominio** avviene quando cambia l'utente, generalmente quando un utente chiude una sessione di lavoro e un altro la comincia.
- Ogni *processo* può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere dipende dall'identità del processo. Il cambio di dominio corrisponde all'invio di un messaggio da un processo a un altro processo e quindi all'attesa di una risposta.
- Ogni *procedura* può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere corrisponde alle variabili locali definite all'interno della procedura. Il cambio di dominio avviene quando s'invoca una procedura.

Il cambio di dominio è discusso in modo più dettagliato nel Paragrafo 18.3.

Si consideri l'ordinario duplice modo di funzionamento (modo di sistema e modo d'utente) dei sistemi operativi. Quando si esegue un processo nel modo di sistema, esso può impiegare istruzioni privilegiate e quindi acquisire il controllo completo del calcolatore. D'altra parte, se è eseguito nel modo d'utente, il processo può impiegare solo istruzioni non privilegiate; di conseguenza, può essere eseguito solo all'interno del proprio spazio di memoria predefinito. Questi due modi proteggono il sistema operativo, che è

eseguito nel dominio di sistema, dai processi utenti, che si eseguono nel dominio d'utente. In un sistema operativo con multiprogrammazione due domìni di protezione sono insufficienti, poiché gli utenti richiedono anche la protezione reciproca. Serve quindi uno schema più elaborato, che s'illustra esaminando due influenti sistemi operativi, lo UNIX e il MULTICS, e osservando come in essi sono stati realizzati questi concetti.

18.2.2 Un esempio: UNIX

Nel sistema operativo UNIX un dominio è associato all'utente. Il cambio di dominio corrisponde al cambio temporaneo dell'identificatore dell'utente. Questo cambio si compie tramite il file system. A ogni file sono associati un'identificatore di proprietario e un bit di dominio (noto con il nome di *setuid bit*). Quando un utente, con identificatore = *A*, avvia l'esecuzione di un file posseduto da *B*, il cui bit di dominio è *off*, l'identificatore del processo viene impostato ad *A*. Quando il *setuid bit* è *on*, l'identificatore d'utente viene impostato a quello del proprietario del file, in questo caso *B*. Quando il processo termina, cessa anche quest'impostazione temporanea dell'identificatore d'utente.

Per cambiare domìni nei sistemi operativi in cui i domìni sono definiti dagli identificatori degli utenti s'impiegano anche altri metodi. Quasi tutti i sistemi hanno bisogno di un tale meccanismo, che si usa per rendere disponibile a tutti gli utenti una funzione che richiede diversi privilegi. Ad esempio, è auspicabile che tutti gli utenti possano ottenere il permesso di accedere a una rete senza che ciascuno debba scrivere il proprio programma d'inserimento nella rete. In tal caso in un sistema UNIX si attiva il bit *setuid* di un programma di rete con la conseguenza che l'identificatore d'utente cambia quando il programma è in esecuzione: l'identificatore di un utente con il privilegio d'accesso alla rete (come *root*, l'utente più importante) sostituisce il corrente identificatore d'utente. Un problema che si pone con questo metodo è dovuto al fatto che se un utente riesce a creare un file con identificatore *root* e con il *setuid* bit posto a *on*, può assumere l'identità *root* e fare qualsiasi operazione sul sistema. Il meccanismo del *setuid* è ulteriormente discusso nell'Appendice A.

Un metodo alternativo, usato in altri sistemi operativi, prevede l'inserimento dei programmi privilegiati in una directory speciale. In questo caso il sistema operativo è progettato in modo da cambiare al momento dell'esecuzione l'identificatore d'utente di ogni programma residente in questa directory, rendendola equivalente a *root* o all'identificatore d'utente del proprietario della directory. Ciò elimina il problema dei programmi segreti di *setuid*, poiché tutti questi programmi risiedono in un'unica locazione. Questo metodo è comunque meno flessibile di quello usato nello UNIX.

Ancora più restrittivi, e quindi più protettivi, sono i sistemi che semplicemente non permettono di modificare l'identificatore d'utente. In questi casi è necessario ricorrere a speciali tecniche per permettere agli utenti di accedere a funzioni privilegiate. Ad esempio, si può attivare un **processo demone** nella fase d'avviamento del sistema ed eseguirlo con un identificatore d'utente speciale. Gli utenti quindi eseguono un programma distinto che invia richieste a questo processo ogni volta che hanno bisogno di usare la relativa funzione. Questo metodo è impiegato dal sistema operativo TOPS-20.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

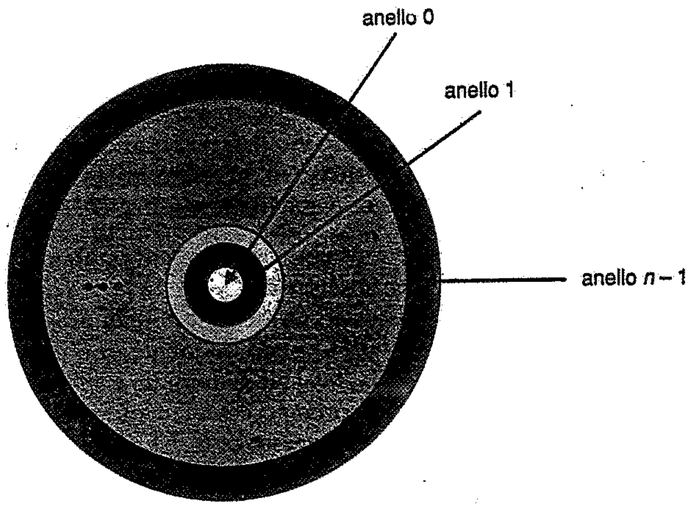


Figura 18.2 Struttura ad anelli del sistema MULTICS.

In ciascuno di questi sistemi la scrittura dei programmi privilegiati si deve realizzare con molta cura: qualsiasi svista può causare una totale mancanza di protezione del sistema. Di solito questi programmi sono i primi a essere attaccati dalle persone che tentano di penetrare in un sistema; sfortunatamente tali attacchi hanno spesso successo; ad esempio, la sicurezza è stata infranta in molti sistemi UNIX attraverso la funzionalità del *setuid*. La sicurezza è trattata nel Capitolo 19.

18.2.3 Un esempio: MULTICS

Nel sistema MULTICS i domìni di protezione sono organizzati gerarchicamente in una struttura ad anelli, numerati da 0 a 7. Ciascun anello corrisponde a un dominio (Figura 18.2). Si supponga che D_i e D_j siano due anelli di dominio. Se $j < i$, D_i è un sottoinsieme di D_j; ciò significa che un processo in esecuzione nel dominio D_j ha più privilegi di quanti ne abbia uno in esecuzione nel dominio D_i. Un processo in esecuzione nel dominio D_0 ha più privilegi di tutti. Se ci sono due soli anelli, questo schema corrisponde al modo d'esecuzione di sistema e d'utente, dove il modo di sistema corrisponde a D_0 e il modo d'utente a D_1.

MULTICS ha uno spazio d'indirizzi segmentato; ogni segmento è un file ed è associato a uno degli anelli. Un descrittore del segmento include un elemento che identifica il numero dell'anello. Inoltre contiene tre bit d'accesso per il controllo della lettura, della scrittura e dell'esecuzione. L'associazione tra segmenti e anelli è una scelta che riguarda i criteri e non rientra nello scopo di questo testo. A ogni processo è associato un contatore del *numero corrente d'anello*, che identifica l'anello nel quale il processo è

attualmente in esecuzione. Quando un processo è in esecuzione nell'anello i, non può accedere a un segmento associato all'anello j, tale che $j < i$, ma può accedere a un segmento associato all'anello k, tale che $k \geq i$. Comunque, il tipo d'accesso è limitato dai bit d'accesso associati a quel segmento.

Nel MULTICS il cambio dei domìni avviene quando un processo passa da un anello all'altro invocando una procedura in un anello diverso. Naturalmente questo cambio si deve compiere in modo controllato, altrimenti un processo potrebbe iniziare l'esecuzione nell'anello 0, senza che sia garantita alcuna protezione. Per permettere un cambio di dominio controllato, occorre modificare il campo concernente l'anello del descrittore del segmento inserendovi le seguenti informazioni:

- **Un intervallo d'accesso.** Una coppia d'interi b_1 e b_2 tali che $b_1 \leq b_2$.
- **Un limite.** Un intero b_3 tale che $b_3 > b_2$.
- **Una lista degli ingressi (o porte).** Identifica i punti d'accesso (ingressi) da cui si possono invocare i segmenti.

Se un processo in esecuzione nell'anello i invoca una procedura (segmento) con intervallo d'accesso (b_1, b_2), la chiamata è ammessa se $b_1 \leq i \leq b_2$ e il numero corrente d'anello del processo rimane i. Altrimenti s'invia un segnale di eccezione al sistema operativo e la situazione viene gestita come segue:

- Se $i < b_1$, si può eseguire la chiamata, poiché si ha un trasferimento a un anello (dominio) con meno privilegi. Tuttavia, se si passano parametri che si riferiscono a segmenti in un anello inferiore (vale a dire segmenti che non sono accessibili alla procedura invocata), allora questi segmenti devono essere copiati in un'area alla quale può accedere anche la procedura invocata.
- Se $i > b_2$, si può eseguire la chiamata solo se b_3 è minore o uguale a i e la chiamata è indirizzata a uno dei punti d'accesso stabiliti nella lista. Questo schema permette a processi con diritti d'accesso limitati di invocare, ma solo in modo attentamente controllato, procedure che si trovano in anelli inferiori e che hanno più diritti d'accesso.

Lo svantaggio maggiore legato alla struttura (gerarchica) ad anelli consiste nel fatto che non consente l'applicazione del principio del privilegio minimo. In particolare, se un oggetto deve essere accessibile nel dominio D_j, ma non nel dominio D_i, allora deve essere $j < i$, ma ciò significa che ogni segmento accessibile in D_i è accessibile anche in D_j.

Il sistema di protezione del MULTICS è in generale più complesso e meno efficiente di quelli adoperati nei correnti sistemi operativi. Se la protezione interferisce con la comodità d'uso del sistema, o ne riduce significativamente le prestazioni, il suo impiego deve essere attentamente valutato rispetto agli scopi del sistema. Ad esempio, si potrebbe volere un complesso sistema di protezione in un calcolatore usato in un'università per gestire i voti degli studenti, e usato anche dagli studenti per i loro compiti. Un simile sistema di protezione non sarebbe adatto a un calcolatore che s'impiega per elaborazioni nu-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

meriche in cui le prestazioni sono della massima importanza. Sarebbe quindi preferibile separare i meccanismi di protezione dai criteri di protezione, permettendo allo stesso sistema di avere una complessa o semplice protezione secondo le necessità dei suoi utenti. Per separare i meccanismi dai criteri sono necessari modelli di protezione più generali.

18.3 Matrice d'accesso

Il modello di protezione qui descritto si può considerare in modo astratto come una matrice, chiamata **matrice d'accesso**. Le righe della matrice rappresentano i domìni, e le colonne gli oggetti. Ciascun elemento della matrice consiste di un insieme di diritti d'accesso. Poiché le colonne definiscono esplicitamente gli oggetti, si possono omettere i nomi degli oggetti dai diritti d'accesso. L'elemento `access(i, j)` definisce l'insieme di operazioni che un processo in esecuzione nel dominio D_i può richiamare sull'oggetto O_j.

Per spiegare questi concetti si considera la matrice d'accesso riportata nella Figura 18.3, nella quale sono presenti quattro domìni e quattro oggetti: tre file (F_1, F_2, F_3) e una stampante. Quando viene eseguito nel dominio D_1, un processo può leggere i file F_2 ed F_3. Un processo in esecuzione nel dominio D_4 ha gli stessi privilegi di un processo in esecuzione nel dominio D_1, ma in più può scrivere anche sui file F_1 e F_3. Solo un processo in esecuzione nel dominio D_2 può accedere alla stampante.

Lo schema della matrice d'accesso offre un meccanismo che permette di specificare diversi criteri. Il meccanismo consiste nella realizzazione della matrice d'accesso e nella garanzia che le caratteristiche semantiche sottolineate siano sempre valide. Più specificamente, occorre assicurare che un processo in esecuzione nel dominio D_i possa accedere solo agli oggetti specificati nella riga i e solo nel modo indicato dagli elementi della matrice d'accesso.

dominio \ oggetto	F_1	F_2	F_3	stampante
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figura 18.3 Matrice d'accesso.

Con la matrice d'accesso si possono attuare i criteri riguardanti la protezione. Tali criteri implicano la scelta dei diritti da inserire nell'(i, j)-esimo elemento. Occorre anche stabilire il dominio nel quale avviene l'esecuzione di ogni processo. Questo criterio è generalmente stabilito dal sistema operativo.

Normalmente sono gli utenti a decidere il contenuto degli elementi della matrice d'accesso. Quando un utente crea un nuovo oggetto O_j, si aggiunge la colonna O_j alla matrice d'accesso con gli elementi di inizializzazione stabiliti dal creatore. L'utente può decidere di inserire alcuni diritti in determinati elementi della colonna j e altri diritti in altri elementi, secondo le necessità.

La matrice d'accesso fornisce un meccanismo adeguato alla definizione e realizzazione di uno stretto controllo sia per l'associazione statica sia per l'associazione dinamica tra processi e domìni. Quando un processo passa da un dominio all'altro, si esegue un'operazione (`switch`) su un oggetto (il dominio). Il passaggio da un dominio all'altro si può controllare inserendo i domìni tra gli oggetti della matrice d'accesso. Analogamente, quando si modifica il contenuto della matrice d'accesso, si esegue un'operazione su un oggetto: la matrice d'accesso. Anche in questo caso si possono controllare le modifiche considerando la stessa matrice d'accesso come un oggetto. In effetti, poiché si può modificare singolarmente, ogni elemento della matrice d'accesso si deve considerare come un oggetto da proteggere.

A questo punto si devono considerare solo le operazioni possibili su questi nuovi oggetti, domìni e matrice d'accesso, e occorre decidere come debbano essere eseguite dai processi.

I processi devono poter passare da un dominio all'altro. Il passaggio dal dominio D_i al dominio D_j è permesso se e solo se l'operazione `switch` $\in$ `access(i, j)`. Quindi, com'è illustrato nella Figura 18.4, un processo in esecuzione nel dominio D_2 può passare al dominio D_3 oppure al dominio D_4. Un processo del dominio D_4 può passare al dominio D_1, e uno del dominio D_1 può passare al dominio D_2.

Permettere la modifica controllata del contenuto degli elementi della matrice d'accesso richiede altre tre operazioni: `copy`, `owner` e `control`.

dominio \ oggetto	F_1	F_2	F_3	stampante	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figura 18.4 Matrice d'accesso della Figura 18.3 con domìni come oggetti.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dominio \ oggetto	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

dominio \ oggetto	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Figura 18.5 Matrice d'accesso con diritti *copy*.

La possibilità di copiare un diritto d'accesso da un dominio (o riga) a un altro della matrice d'accesso è indicata da un asterisco (*) apposto sul diritto d'accesso. Il diritto *copy* permette di copiare il diritto d'accesso solo all'interno della colonna (cioè per l'oggetto) per la quale il diritto stesso è definito. Ad esempio, nella Figura 18.5(a), un processo in esecuzione nel dominio D_2 può copiare l'operazione read in un elemento qualsiasi associato al file F_2. Quindi, la matrice d'accesso della Figura 18.5(a) si può modificare nella matrice d'accesso illustrata nella Figura 18.5(b). Questo schema ha due varianti:

- Un diritto viene copiato da access(i, j) ad access(k, j) e viene successivamente rimosso da access(i, j); quest'azione è un *trasferimento* di un diritto piuttosto che una copiatura.
- La propagazione del diritto *copy* può essere limitata. Ciò significa che quando si copia il diritto R^* da access(i, j) ad access(k, j), si crea solo il diritto R e non R^*. Un processo in esecuzione nel dominio D_k non può copiare ulteriormente il diritto R.

Un sistema può scegliere uno tra questi tre diritti *copy*, oppure può fornirli tutti e tre identificandoli come diritti separati: *copy*, *transfer* e *limited copy*.

dominio \ oggetto	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write*
D_3	execute		

(a)

dominio \ oggetto	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

Figura 18.6 Matrice d'accesso con diritti *owner*.

Occorre anche un meccanismo che permetta di aggiungere nuovi diritti e rimuoverne altri; queste operazioni sono controllate dal diritto *owner*. Se access(i, j) contiene l'operazione corrispondente al diritto *owner*, un processo in esecuzione nel dominio D_i può aggiungere e rimuovere qualsiasi diritto di qualsiasi elemento della colonna j. Ad esempio, nella Figura 18.6(a) il dominio D_1 è il proprietario di F_1 e quindi può aggiungere e cancellare qualsiasi diritto valido nella colonna di F_1. Analogamente, il dominio D_2 è proprietario di F_2 e F_3, quindi può aggiungere e rimuovere qualsiasi diritto valido che si trovi all'interno di queste due colonne. Così, la matrice d'accesso della Figura 18.6(a) si può modificare nella matrice d'accesso illustrata nella Figura 18.6(b).

I diritti *copy* e *owner* permettono a un processo di modificare gli elementi di una colonna. Occorre anche un meccanismo per modificare gli elementi di una riga. Il diritto *control* si può applicare solo a oggetti di dominio. Se access(i, j) contiene il diritto *control*, allora un processo in esecuzione nel dominio D_i può rimuovere qualsiasi diritto d'accesso dalla riga j. Si supponga, prendendo come esempio la Figura 18.4, di inserire il

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dominio \ oggetto	F_1	F_2	F_3	stampante	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	control
D_3		read	execute					
D_4	write		write		switch			

Figura 18.7 Matrice d'accesso modificata della Figura 18.4.

diritto *control* in access(D_2, D_4). Allora un processo in esecuzione nel dominio D_2 può modificare il dominio D_4, com'è illustrato nella Figura 18.7.

I diritti *copy* e *owner* forniscono un meccanismo per limitare la propagazione dei diritti d'accesso, però non forniscono strumenti adeguati per impedire la propagazione delle informazioni. Il problema di garantire che nessuna informazione, tenuta inizialmente in un oggetto, possa migrare all'esterno del proprio ambiente d'esecuzione si chiama **problema della reclusione**. Tale problema è in generale insolubile (si vedano le Note bibliografiche).

Queste operazioni sui domìni e sulla matrice d'accesso non sono di per sé importanti, ma spiegano come il modello della matrice d'accesso consenta la realizzazione e il controllo dei requisiti di protezione dinamica. Nuovi oggetti e nuovi domìni si possono creare dinamicamente ed essere inseriti nel modello della matrice d'accesso. Tuttavia, in questo paragrafo è spiegato soltanto il meccanismo di base; chi progetta e usa il sistema deve scegliere i criteri riguardanti quali domìni, e in che modo, abbiano accesso a determinati oggetti.

18.4 Realizzazione della matrice d'accesso

A questo punto occorre realizzare efficacemente la matrice d'accesso. Tale matrice generalmente è sparsa, ciò significa che molti suoi elementi sono vuoti. A causa del modo in cui si usa la funzione di protezione, le tecniche di strutturazione dei dati per la rappresentazione delle matrici sparse non sono particolarmente utili per quest'applicazione.

18.4.1 Tabella globale

La realizzazione più semplice della matrice d'accesso è una tabella globale costituita di un insieme di triple ordinate <*dominio, oggetto, insieme dei diritti*>. Ogni volta che si esegue un'operazione M su un oggetto O_j del dominio D_i, si cerca una tripla $<D_i, O_j, R_k>$ nella tabella globale, tale che $M \in R_k$. Se si trova questa tripla, l'operazione può continuare, altrimenti si verifica una condizione di eccezione (errore). Questa tipo di realizzazione ha, però, parecchi svantaggi. Generalmente la tabella è grande e non può essere conservata nella memoria centrale, perciò sono richieste ulteriori operazioni di I/O. Per gestire questa tabella spesso si usano tecniche di memoria virtuale; inoltre è difficile anche avvalersi di speciali raggruppamenti di oggetti o domìni. Ad esempio, se chiunque può leggere un oggetto particolare, esso deve avere un elemento distinto in ogni dominio.

18.4.2 Liste d'accesso per oggetti

Ogni colonna della matrice d'accesso si può realizzare come una lista d'accesso per un oggetto (Paragrafo 11.6.2). Naturalmente gli elementi vuoti si possono scartare. Per ogni oggetto la lista risultante è composta di coppie ordinate <*dominio, insieme dei diritti*> e definisce tutti i domìni il cui insieme di diritti d'accesso per quell'oggetto risulta non vuoto.

Questo metodo si può estendere facilmente definendo una lista più un insieme *predefinito* dei diritti d'accesso. Quando nel dominio D_i si tenta un'operazione M su un oggetto O_j, si cerca un elemento $<D_i, R_k>$, con $M \in R_k$, nella lista d'accesso relativa all'oggetto O_j. Se si trova quest'elemento, l'operazione è permessa; altrimenti, si controlla l'insieme predefinito. Se M si trova in questo insieme, l'accesso è permesso, altrimenti l'accesso viene negato e si verifica una condizione di eccezione. Per motivi di efficienza, conviene controllare prima l'insieme predefinito e poi cercare nella lista d'accesso.

18.4.3 Liste delle abilitazioni per domìni

Anziché associare le colonne della matrice d'accesso agli oggetti formando liste d'accesso, è possibile associare ogni riga della matrice al suo dominio. La **lista delle abilitazioni** per un dominio è una lista d'oggetti insieme con le operazioni ammesse su quegli oggetti. Un oggetto è spesso rappresentato dal suo nome fisico o indirizzo, detto **abilitazione** (*capability*). Per eseguire l'operazione M sull'oggetto O_j, il processo esegue l'operazione M, specificando l'abilitazione (puntatore) per l'oggetto O_j come parametro. Il semplice possesso dell'abilitazione indica che l'accesso è permesso.

La lista delle abilitazioni è associata a un dominio, ma non è mai direttamente accessibile a un processo che si trova in esecuzione in quel dominio: è un oggetto protetto, mantenuto dal sistema operativo e al quale gli utenti possono accedere solo indirettamente. La protezione basata sulle abilitazioni si fonda sul presupposto che non è mai permessa la migrazione delle abilitazioni in qualsiasi spazio d'indirizzi direttamente accessibile a un processo utente, dove queste potrebbero essere modificate. Se tutte le abilitazioni sono sicure, è sicuro anche l'oggetto da esse protetto contro accessi non autorizzati.

Le abilitazioni sono state originariamente proposte come un tipo di puntatore sicuro, per soddisfare la necessità di protezione delle risorse dovuta alla diffusione dei calcolatori con multiprogrammazione. L'idea di un puntatore intrinsecamente protetto (dal punto di vista dell'utente di un sistema) getta le basi per una protezione che si può estendere fino al livello delle applicazioni.

Per fornire una protezione intrinseca occorre distinguere le abilitazioni dagli altri oggetti, e interpretarle attraverso una macchina astratta sulla quale si eseguono program-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

mi di livello superiore. Generalmente le abilitazioni si distinguono dagli altri dati in uno dei modi seguenti:

- Ogni oggetto ha un'**etichetta** (*tag*) che ne indica il tipo: abilitazione o dati accessibili. Le etichette non devono essere direttamente accessibili ai programmi d'applicazione. Per imporre tale limitazione si può ricorrere all'ausilio dell'architettura. Anche se per distinguere tra abilitazioni e altri oggetti è sufficiente un bit, spesso se ne adoperano di più. Questa estensione permette all'architettura di applicare a tutti gli oggetti etichette indicanti i rispettivi tipi. Quindi l'architettura può distinguere interi, numeri in virgola mobile, puntatori, valori booleani, caratteri, istruzioni, abilitazioni e valori non inizializzati grazie alle rispettive etichette.
- Lo spazio d'indirizzi associato a un programma si può dividere in due parti: una contenente i dati e le istruzioni del programma, accessibile al programma; l'altra, contenente la lista delle abilitazioni, accessibile solo al sistema operativo. Lo spazio di memoria segmentato è un utile ausilio di questo metodo (Paragrafo 9.5).

Sono stati sviluppati parecchi sistemi di protezione basati sulle abilitazioni, brevemente descritti nel Paragrafo 18.6. Anche il sistema operativo Mach, descritto in modo dettagliato nell'Appendice B, fa uso di una versione di protezione basata sulle abilitazioni.

18.4.4 Meccanismo chiave-serratura

Lo **schema chiave-serratura** rappresenta un compromesso tra le liste d'accesso e le liste di abilitazioni. Ogni oggetto ha una lista di sequenze di bit uniche, chiamate **serrature**; analogamente, ogni dominio ha una lista di sequenze di bit uniche, chiamate **chiavi**. Un processo in esecuzione in un dominio può accedere a un oggetto solo se quel dominio ha una chiave che corrisponde a una delle serrature dell'oggetto.

Come le liste delle abilitazioni, anche la lista delle chiavi per un dominio deve essere gestita dal sistema operativo per conto del dominio. Gli utenti non possono esaminare o modificare direttamente la lista delle chiavi o delle serrature.

18.4.5 Confronto

Le liste d'accesso corrispondono direttamente alle necessità degli utenti. Quando un utente crea un oggetto, può specificare quali domìni hanno accesso a quell'oggetto e quali operazioni sono permesse. Tuttavia, poiché le informazioni sui diritti d'accesso per un particolare dominio non sono localizzate, è difficile stabilire l'insieme dei diritti d'accesso per ogni dominio. Ogni accesso all'oggetto deve inoltre essere controllato, il che richiede una ricerca nella lista d'accesso. In un grande sistema con lunghe liste d'accesso, questa ricerca può causare un notevole spreco di tempo.

Le liste delle abilitazioni non corrispondono direttamente alle necessità degli utenti, ma sono utili per localizzare le informazioni per un dato processo. Un processo che tenti un accesso deve presentare la relativa abilitazione, quindi il sistema di protezione deve verificare soltanto che l'abilitazione sia valida. La revoca delle abilitazioni può essere inefficiente; questo problema è trattato nel Paragrafo 18.5.

Il meccanismo chiave-serratura rappresenta un compromesso tra questi due schemi. Il meccanismo può essere efficace e flessibile, secondo la lunghezza delle chiavi, che possono essere passate liberamente da dominio a dominio. Inoltre i privilegi d'accesso si possono revocare in modo semplice ed efficiente modificando alcune chiavi associate all'oggetto; anche questo problema è trattato nel Paragrafo 18.5.

La maggior parte dei sistemi adopera una combinazione di liste d'accesso e liste di abilitazioni. Quando un processo tenta per la prima volta di accedere a un oggetto, si fa una ricerca nella lista d'accesso. Se l'accesso viene negato, si verifica una condizione di eccezione, altrimenti si crea un'abilitazione che si associa al processo. I riferimenti successivi si servono dell'abilitazione per dimostrare rapidamente che l'accesso è consentito. Dopo l'ultimo accesso, l'abilitazione viene distrutta. Questa strategia è usata nel sistema MULTICS e nel sistema CAL; questi sistemi impiegano sia le liste d'accesso sia quelle delle abilitazioni.

Come esempio, si consideri un file system in cui a ogni file è associata una lista d'accesso. Quando un processo apre un file, si fa una ricerca nella directory per trovare il file, si controlla il permesso d'accesso e si assegnano le aree di memoria per l'I/O. Tutte queste informazioni si registrano in un nuovo elemento della tabella dei file associata al processo. L'operazione riporta un indice in questa tabella per il file appena aperto, tramite il quale si compiono tutte le operazioni sul file. Quindi l'elemento della tabella dei file punta al file e alle sue aree di memoria per l'I/O. Quando il file viene chiuso, si cancella l'elemento della tabella dei file. Poiché la tabella dei file è mantenuta dal sistema operativo, gli utenti non possono alterarla, quindi gli utenti possono accedere ai soli file che sono stati aperti. Poiché l'accesso viene controllato al momento dell'apertura del file, la protezione è assicurata. Nel sistema operativo UNIX si adopera questa strategia.

Il diritto d'accesso si *deve* controllare per ogni accesso e l'elemento della tabella dei file contiene l'abilitazione per le sole operazioni ammesse. Se si apre un file per la lettura, nell'elemento della tabella dei file viene inserita un'abilitazione d'accesso alla lettura. Se si tenta di scrivere in quel file, il sistema rileva questa violazione della protezione confrontando l'operazione richiesta con l'abilitazione presente nell'elemento della tabella dei file.

18.5 Revoca dei diritti d'accesso

In un sistema di protezione dinamica talvolta può essere necessario revocare i diritti d'accesso a oggetti condivisi da diversi utenti. A proposito della revoca si possono presentare diverse questioni:

- **Immediata o ritardata.** Occorre stabilire se la revoca si presenta immediatamente o con ritardo. Se la revoca è ritardata, occorre stabilire quando avverrà.
- **Selettiva o generale.** Quando si revoca un diritto d'accesso a un oggetto, occorre stabilire se la revoca vale per *tutti* gli utenti che hanno un diritto d'accesso a quell'oggetto, oppure se si può specificare un gruppo di utenti cui si devono revocare diritti d'accesso.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- **Parziale o totale.** Occorre stabilire se si può revocare un sottoinsieme di diritti associati a un oggetto, oppure se si devono revocare tutti i diritti d'accesso a quest'oggetto.
- **Temporanea o permanente.** Occorre stabilire se l'accesso si può revocare in modo permanente, cioè il diritto d'accesso non sarà più disponibile, oppure se può essere nuovamente ottenuto.

Disponendo di uno schema con lista d'accesso, fare una revoca è facile. Si cerca il diritto, o i diritti, d'accesso da revocare nella lista d'accesso, e lo si cancella dalla lista. La revoca è immediata e può essere generale o selettiva, totale o parziale, permanente o temporanea.

La revoca delle abilitazioni, invece, è molto più difficile. Poiché le abilitazioni sono distribuite su tutto il sistema, occorre prima trovarle e poi revocarle. Tra gli schemi che realizzano la revoca delle abilitazioni ci sono i seguenti:

- **Riacquisizione.** Le abilitazioni vengono cancellate periodicamente da ogni dominio. Se un processo intende usare un'abilitazione, può scoprire che quell'abilitazione è stata cancellata. Il processo allora può tentare di riacquisirla. Se l'accesso è stato revocato, il processo non è più in grado di riacquisire l'abilitazione.
- **Puntatori indietro.** Insieme a ciascun oggetto si conserva una lista di puntatori a tutte le abilitazioni a esso associate. Quando si richiede una revoca, si possono seguire questi puntatori, modificando le abilitazioni secondo le necessità. Questo schema era adottato nel sistema MULTICS, ed è abbastanza generale, anche se la sua realizzazione è onerosa.
- **Riferimento indiretto.** Le abilitazioni non puntano direttamente agli oggetti. Ogni abilitazione punta all'elemento di una tabella globale che a sua volta punta all'oggetto. La revoca si realizza cercando nella tabella globale l'elemento desiderato e cancellandolo. Quando si tenta l'accesso, si riscontra che l'abilitazione punta a un elemento non ammesso della tabella. Gli elementi della tabella si possono riutilizzare senza difficoltà per altre abilitazioni, poiché sia l'abilitazione sia l'elemento della tabella contengono il nome unico dell'oggetto. L'oggetto deve corrispondere a un'abilitazione e al proprio elemento della tabella. Questo schema, che è stato adottato nel sistema CAL, non permette la revoca selettiva.
- **Chiavi.** Una chiave è una sequenza unica di bit associabile a ogni abilitazione. Tale chiave viene definita al momento della creazione dell'abilitazione e non può essere modificata né esaminata dal processo proprietario dell'abilitazione stessa. È possibile definire una **chiave principale** associata a ogni oggetto, che si può sostituire con l'operazione `set-key`. Quando si crea un'abilitazione, a questa si associa il valore corrente della chiave principale. Quando l'abilitazione viene esercitata, si confronta la sua chiave con la chiave principale. Se le due chiavi coincidono, l'operazione può continuare, altrimenti si verifica una condizione di eccezione. La revoca sostituisce la chiave principale con un valore nuovo tramite l'operazione `set-key`, che invalida tutte le abilitazioni precedenti per quest'oggetto.

Questo schema non permette la revoca selettiva, poiché a ogni oggetto è associata solo una chiave principale. Se a ogni oggetto si associa una lista di chiavi, si può realizzare la revoca selettiva. Infine, tutte le chiavi si possono raggruppare in una tabella globale di chiavi. Un'abilitazione è valida solo se la sua chiave coincide con una delle chiavi della tabella globale. La revoca si realizza rimuovendo dalla tabella la chiave coincidente. In questo schema una chiave si può associare a più oggetti, e più chiavi si possono associare a ciascun oggetto, offrendo la massima flessibilità.

Negli schemi basati sulle chiavi, le operazioni di definizione, inserimento in liste e cancellazione dalle liste delle chiavi stesse non devono essere a disposizione di tutti gli utenti. In particolare, è ragionevole permettere soltanto al proprietario di un oggetto di impostare le chiavi per quell'oggetto. Questa scelta, in ogni modo, riguarda i criteri che il sistema di protezione può realizzare ma non deve definire.

18.6 Sistemi basati su abilitazioni

In questo paragrafo si esaminano due sistemi di protezione basati su abilitazioni. Questi sistemi sono diversi nel grado di complessità e nel tipo di criteri che su di essi si possono realizzare. Nessuno dei due è molto usato, ma entrambi sono interessanti banchi di prova delle teorie sulla protezione.

18.6.1 Un esempio: Hydra

Il sistema Hydra è un sistema di protezione basato sulle abilitazioni, caratterizzato da una notevole flessibilità. Il sistema fornisce un prefissato insieme di possibili diritti d'accesso noto al sistema e da esso interpretato. Tra questi diritti sono presenti le principali forme d'accesso come il diritto per la lettura, la scrittura o l'esecuzione di un segmento di memoria. Inoltre, il sistema offre agli utenti (del sistema di protezione) gli strumenti necessari per dichiarare altri diritti. I diritti definiti dagli utenti sono interpretati esclusivamente dai programmi d'utente, ma il sistema fornisce la protezione degli accessi nell'uso di questi diritti e dei diritti definiti dal sistema. Queste funzioni costituiscono un notevole passo avanti nella tecnologia della protezione.

Le operazioni sugli oggetti sono definite in modo procedurale, e le procedure che realizzano tali operazioni sono a loro volta oggetti, accessibili indirettamente attraverso abilitazioni. I nomi delle procedure definite dagli utenti si devono comunicare al sistema di protezione, che deve gestire oggetti del tipo definito dagli utenti. Quando si comunica la definizione di un oggetto al sistema Hydra, i nomi delle operazioni sul tipo diventano **diritti ausiliari**. Questi diritti ausiliari si possono descrivere in un'abilitazione per un'istanza del tipo. Affinché un processo possa eseguire un'operazione su un oggetto tipizzato, l'abilitazione per quell'oggetto deve contenere tra i diritti ausiliari il nome dell'operazione invocata. Questa limitazione permette la distinzione dei diritti d'accesso secondo l'istanza e secondo il processo.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il sistema Hydra offre anche l'**amplificazione dei diritti**. Questo schema permette di certificare che una procedura è *fidata* per agire su un parametro formale di un tipo specificato, per conto di qualsiasi processo che abbia un diritto d'esecuzione della procedura. I diritti posseduti da una procedura fidata sono indipendenti dai diritti posseduti dal processo chiamante e possono addirittura superarli. Tuttavia tale procedura non deve essere considerata universalmente fidata (ad esempio, alla procedura non è permesso di agire su altri tipi), e la fidatezza non deve essere estesa a qualsiasi altra procedura o altro segmento di programma che può essere eseguito da un processo.

L'amplificazione permette alle procedure di accedere alle variabili che rappresentano il tipo di dato astratto. Se, ad esempio, un processo possiede un'abilitazione a un oggetto tipizzato *A*, quest'abilitazione può comprendere un diritto ausiliario a richiamare una generica operazione *P*, ma non può comprendere nessuno dei cosiddetti diritti del nucleo, come i diritti per la lettura, la scrittura o l'esecuzione, sul segmento che rappresenta *A*. Tale abilitazione offre a un processo un mezzo per accedere indirettamente, tramite l'operazione *P*, alla rappresentazione di *A*, ma solo per scopi specifici.

D'altra parte, quando un processo impiega l'operazione *P* su un oggetto *A*, l'abilitazione d'accesso ad *A* può essere amplificata quando il controllo passa al corpo del codice di *P*. Quest'amplificazione può essere necessaria per conferire a *P* il diritto d'accesso al segmento di memoria che rappresenta *A*, così da realizzare l'operazione che *P* definisce sul tipo di dati astratto. Si può permettere al corpo del codice di *P* di leggere o scrivere direttamente nel segmento di *A*, anche se il processo richiamante non può farlo. Al termine di *P*, si riporta l'abilitazione per *A* al suo stato originale non amplificato. Questo è un tipico caso in cui i diritti posseduti da un processo per accedere a un segmento protetto devono cambiare dinamicamente, secondo il compito da svolgere. La regolazione dinamica dei diritti si esegue per garantire la coerenza delle astrazioni definite dai programmatori. Nel sistema operativo Hydra l'amplificazione dei diritti si può fissare in modo esplicito nelle dichiarazioni dei tipi astratti.

Quando un utente passa un oggetto come argomento a una procedura, può essere necessario assicurare che la procedura non possa modificare l'oggetto. Questa limitazione si può realizzare facilmente passando un diritto d'accesso senza diritto di modifica (scrittura). Tuttavia, se l'amplificazione è possibile, il diritto di modifica può essere ripristinato, e il requisito di protezione dell'utente può quindi essere aggirato. In generale, naturalmente, un utente può supporre che una procedura esegua correttamente il proprio compito. Tale ipotesi, però, non sempre è corretta, poiché possono verificarsi errori fisici o logici. Il sistema Hydra risolve questo problema limitando le amplificazioni.

Il meccanismo di chiamata di procedura del sistema Hydra è stato progettato come una diretta soluzione al *problema dei sottosistemi mutuamente sospetti*. Questo problema è definito come segue. Si supponga che per un programma sia stabilito che possa essere invocato come servizio da diversi utenti (ad esempio, una procedura di ordinamento, un compilatore, un gioco). Quando gli utenti invocano questo programma di servizio, corrono il rischio che il programma non funzioni bene e possa danneggiare i dati forniti, o trattenere, senza averne l'autorità, qualche diritto d'accesso ai dati da adoperare. Analogamente, il programma di servizio può avere alcuni file privati, ad esempio file di contabilizzazione, cui il programma utente chiamante non deve accedere direttamente. Il sistema Hydra fornisce meccanismi per affrontare questo problema in modo diretto.

Un sottosistema dell'Hydra è costruito sopra il nucleo di protezione e può richiedere la protezione dei propri componenti. Un sottosistema interagisce col nucleo tramite un insieme di primitive stabilite dal nucleo, che definisce i diritti d'accesso alle risorse definite dal sottosistema. I criteri relativi all'uso di queste risorse da parte dei processi utenti possono essere definiti da chi progetta il sottosistema, ma s'impongono tramite l'uso della protezione degli accessi standard offerta dal sistema di abilitazioni.

Un programmatore può servirsi direttamente del sistema di protezione, dopo aver appreso le sue caratteristiche dal relativo manuale. Il sistema Hydra offre un'ampia libreria di procedure definite dal sistema che i programmi utenti possono impiegare. Un utente del sistema Hydra può incorporare esplicitamente le chiamate dirette a queste procedure di sistema nel codice del proprio programma, oppure servirsi di un traduttore di programmi interfacciato al sistema Hydra.

18.6.2 Un esempio: sistema Cambridge CAP

Un diverso orientamento alla protezione basata sulle abilitazioni è stato seguito nella progettazione del sistema Cambridge CAP. Il sistema di abilitazioni del CAP è più semplice e a prima vista meno potente dell'analogo sistema dell'Hydra. Tuttavia, con un esame più attento, è possibile capire che anche questo sistema si può usare per offrire una protezione sicura agli oggetti definiti dagli utenti. Il CAP ha due tipi di abilitazioni. Il tipo ordinario si chiama **abilitazione di dati**, e si può impiegare per fornire l'accesso agli oggetti, ma gli unici diritti forniti sono quelli ordinari di lettura, scrittura o esecuzione dei singoli segmenti di memoria associati all'oggetto. Le abilitazioni di dati sono interpretate dal microcodice della macchina CAP.

Il secondo tipo è la cosiddetta **abilitazione logica**, che è protetta, ma non interpretata, dal microcodice del CAP. L'interpretazione spetta a una procedura *protetta* (cioè privilegiata) che può essere scritta da un programmatore di applicazioni come parte di un sottosistema. A una procedura protetta è associato un particolare tipo di amplificazione di diritti. Quando si esegue il corpo del codice di tale procedura, un processo acquisisce temporaneamente i diritti di lettura o scrittura del contenuto dell'abilitazione logica stessa. Questo particolare tipo di amplificazione di diritti corrisponde a una realizzazione delle primitive `seal` e `unseal` sulle abilitazioni (si vedano le Note bibliografiche). Naturalmente questo privilegio rimane soggetto alla verifica del tipo per assicurare che solo le abilitazioni logiche per uno specificato tipo astratto possano passare a una qualsiasi procedura di quel tipo. Nessun codice gode di totale fiducia, tranne il microcodice della macchina CAP.

L'interpretazione di un'abilitazione logica è lasciata esclusivamente al sottosistema, che la esegue per mezzo delle proprie procedure protette. Questo schema permette di realizzare un gran numero di criteri di protezione. Benché un programmatore possa de-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

finire le proprie procedure protette, la sicurezza del sistema nel suo complesso non può essere compromessa (anche se tali procedure contengono errori). Il sistema di protezione di base non permetterebbe a una procedura protetta non verificata, definita dall'utente, di accedere a qualsiasi segmento di memoria, o abilitazione, che non appartenga all'ambiente di protezione nel quale la procedura stessa risiede. La conseguenza più grave dovuta a una procedura protetta non sicura è la violazione della protezione del sottosistema del quale quella procedura è responsabile.

I progettisti del sistema CAP hanno notato che l'uso delle abilitazioni logiche permette di fare notevoli economie nella formulazione e nella realizzazione di criteri di protezione adeguati ai requisiti delle risorse astratte. Tuttavia un progettista di sottosistemi che voglia usare questa funzione non può semplicemente studiare un manuale, come nel caso del sistema Hydra, ma deve apprenderne i princìpi e le tecniche di protezione, poiché il sistema non offre alcuna libreria di procedure.

18.7 Protezione basata sul linguaggio

Il grado di protezione fornito negli attuali sistemi di calcolo è di solito ottenuto attraverso il nucleo del sistema operativo, il quale si occupa di controllare e convalidare ogni tentativo d'accesso a una risorsa protetta. Poiché una completa convalida degli accessi è potenzialmente una fonte di notevole sovraccarico, è necessario disporre dell'ausilio dell'architettura per ridurre il costo di ogni convalida, o si deve accettare un compromesso rispetto agli obiettivi della protezione. Se la flessibilità di realizzazione dei criteri di protezione è limitata dai meccanismi presenti, o se gli ambienti di protezione sono più grandi di quanto è necessario ad assicurare una maggiore efficienza, soddisfare tutti questi obiettivi è difficile.

Gli scopi della protezione sono stati perfezionati con l'aumentare della complessità dei sistemi operativi e soprattutto con il tentativo di fornire interfacce d'utente di livello superiore. I progettisti dei sistemi di protezione si sono basati in modo determinante su idee nate nell'ambito dei linguaggi di programmazione e, soprattutto, sui concetti di tipi di dati astratti e di oggetti. Attualmente i sistemi di protezione non riguardano solo l'identità di una risorsa alla quale si tenta di accedere, ma anche la natura funzionale di quell'accesso. Nei sistemi di protezione più recenti l'interesse relativo alle funzioni da invocare va oltre un insieme di funzioni definite dal sistema, come gli ordinari metodi per l'accesso ai file, per comprendere anche funzioni definibili dagli utenti.

Anche i criteri concernenti l'uso delle risorse variano secondo l'applicazione e, col passare del tempo, possono essere soggetti a cambiamenti. Per questi motivi la protezione non può più essere considerata come un esclusivo interesse del progettista di un sistema operativo, ma deve essere uno strumento disponibile al progettista di applicazioni per proteggere le risorse di un sottosistema d'applicazione da manomissioni o effetti dovuti a errori.

18.7.1 Imposizione basata sul compilatore

Specificare il controllo degli accessi desiderato per una risorsa condivisa significa fare un'asserzione dichiarativa su tale risorsa. Questo tipo di dichiarazione si può integrare in un linguaggio di programmazione estendendone la funzione di tipizzazione. Quando insieme alla tipizzazione dei dati si dichiara anche la protezione, i progettisti dei sottosistemi possono specificare i propri requisiti di protezione, e la necessità di altre risorse del sistema. Tali specificazioni si dovrebbero fornire direttamente nella fase di stesura dei programmi, e nello stesso linguaggio in cui si scrivono i programmi. Questo metodo ha importanti vantaggi:

1. le necessità di protezione si devono semplicemente dichiarare e non programmare come una sequenza di chiamate di procedure di un sistema operativo;
2. i requisiti di protezione si possono stabilire indipendentemente dalle funzioni fornite da uno specifico sistema operativo;
3. i progettisti dei sottosistemi non devono fornire mezzi d'imposizione;
4. la notazione dichiarativa è naturale, perché i privilegi d'accesso sono strettamente connessi al concetto linguistico di tipi di dati.

La realizzazione di un linguaggio di programmazione può fornire diverse tecniche per imporre protezioni, ma ciascuna di esse deve dipendere in qualche misura dalle funzioni offerte dall'architettura sottostante e dal suo sistema operativo. Si supponga, ad esempio, che un linguaggio sia impiegato per generare codice da eseguire sul sistema Cambridge CAP. In questo sistema ogni riferimento alla memoria effettuato sull'architettura sottostante avviene indirettamente tramite abilitazioni. Questa limitazione impedisce a un processo qualsiasi di accedere a una risorsa esterna al proprio ambiente di protezione. Tuttavia un programma può imporre limitazioni arbitrarie su come una risorsa può essere usata durante l'esecuzione di un particolare segmento di codice da parte di qualunque processo. Tali limitazioni si possono realizzare in modo pressoché immediato ricorrendo alle abilitazioni logiche offerte dal CAP. In un linguaggio di programmazione i criteri di protezione specificabili nel linguaggio stesso si possono realizzare fornendo procedure protette standard per interpretare le abilitazioni. Questo schema permette ai programmatori di specificare i criteri di protezione, senza costringerli a occuparsi dei dettagli riguardanti la realizzazione della relativa imposizione.

Anche se un sistema non offre un nucleo di protezione potente come quelli dell'Hydra o del CAP, esistono meccanismi che permettono la realizzazione delle funzioni di protezione in un linguaggio di programmazione. La differenza principale è dovuta al fatto che la *sicurezza* di questa protezione non è grande quanto quella gestita da un nucleo di protezione, poiché il meccanismo si deve basare su un maggior numero di supposizioni sullo stato operativo del sistema. Un compilatore può separare i riferimenti per i quali non è possibile avere violazioni di protezione da quelli per i quali la violazione può avvenire, e trattarli in modi diversi. La sicurezza offerta da questo tipo di protezione si

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

fonda sul presupposto che il codice generato dal compilatore non venga modificato prima o durante la sua esecuzione.

Di seguito sono elencati i vantaggi dell'imposizione basata esclusivamente su un nucleo, rispetto a quelli che si hanno con l'imposizione fornita da un compilatore:

- **Sicurezza.** L'imposizione effettuata da un nucleo offre un grado di sicurezza del sistema di protezione maggiore di quello offerto dalla generazione, da parte di un compilatore, del codice di controllo della protezione. In uno schema gestito dal compilatore, la sicurezza è basata sulla correttezza del traduttore, oppure su qualche meccanismo di gestione della memoria che protegge i segmenti dai quali si esegue il codice compilato e, in ultima analisi, sulla sicurezza dei file dai quali si carica il programma. Alcune di queste considerazioni valgono anche per un nucleo di protezione gestito dal sistema operativo, ma in questo caso in misura minore poiché il nucleo può risiedere in segmenti fissati di memoria fisica e può essere caricato solo da un file designato. In un sistema di abilitazioni con etichette, nel quale tutto il calcolo degli indirizzi è eseguito dall'architettura o da un microprogramma fisso, si può avere una sicurezza ancora maggiore. La protezione che s'avvale dell'ausilio dell'architettura è relativamente immune dalle violazioni della protezione che possono verificarsi a causa di malfunzionamenti sia fisici sia logici.

- **Flessibilità.** La flessibilità di un nucleo di protezione ha alcuni limiti per quel che riguarda la realizzazione di criteri di protezione definiti dagli utenti, anche se può fornire al sistema le funzioni necessarie per l'imposizione dei propri criteri. Con un linguaggio di programmazione i criteri di protezione si possono dichiarare, così come si può fornire l'imposizione richiesta. Se un linguaggio non offre una sufficiente flessibilità, si può estendere o sostituire con minore perturbazione di un sistema in servizio rispetto a quella che si avrebbe modificando il nucleo di un sistema operativo.

- **Efficienza.** La maggiore efficienza si ha quando l'imposizione della protezione è gestita direttamente dall'architettura. L'imposizione basata sul linguaggio ha il vantaggio di poter verificare l'imposizione dell'accesso statico nella fase di compilazione. Inoltre, poiché un compilatore intelligente può allestire il meccanismo d'imposizione su misura per soddisfare la necessità specificata, spesso si può evitare il carico fisso delle chiamate del nucleo.

Riepilogando, la specificazione della protezione in un linguaggio di programmazione permette la descrizione ad alto livello di criteri per l'assegnazione e l'uso di risorse. L'ambiente di un linguaggio di programmazione può fornire gli strumenti per l'imposizione della protezione quando non è disponibile il controllo automatico gestito dall'architettura; inoltre può interpretare le indicazioni di protezione per generare chiamate a qualsiasi sistema di protezione fornito dall'architettura e dal sistema operativo.

Un modo per rendere disponibile la protezione ai programmi d'applicazione prevede l'uso delle abilitazioni logiche come oggetti di calcolo. Questo concetto è basato sull'idea che alcuni componenti di programmi possano avere il privilegio di creare o esaminare queste abilitazioni. Un programma di creazione di abilitazioni può eseguire un'operazione primitiva (`seal`) che sigilla una struttura di dati, rendendo il contenuto di quest'ultima inaccessibile a qualsiasi componente di programma che non possieda i privilegi *seal* o *unseal*. Questi possono copiare la struttura di dati, o passarne l'indirizzo ad altri componenti del programma, ma non possono ottenere l'accesso al suo contenuto. Tali abilitazioni s'introducono per portare un meccanismo di protezione all'interno del linguaggio di programmazione. L'unico problema che s'incontra seguendo tale metodo riguarda l'uso delle operazioni `seal` e `unseal`; tale uso richiede infatti un orientamento procedurale per specificare la protezione. Per rendere disponibile l'ambiente di protezione ai programmatori di applicazioni sembra preferibile una notazione non procedurale o dichiarativa.

È necessario un meccanismo dinamico sicuro per il controllo degli accessi, per poter distribuire tra i processi utenti le abilitazioni alle risorse del sistema. Per contribuire all'affidabilità generale di un sistema, il meccanismo per il controllo dell'accesso deve essere utilizzabile in modo sicuro, e per essere utile nella pratica deve essere anche ragionevolmente efficiente. Questo requisito ha portato allo sviluppo di un certo numero di costrutti di linguaggio che consentono ai programmatori di dichiarare varie limitazioni riguardanti l'uso di specifiche risorse (si vedano le Note bibliografiche). Questi costrutti forniscono meccanismi per tre funzioni:

1. Distribuire, in modo sicuro ed efficiente, le abilitazioni tra i processi clienti: in particolare i meccanismi assicurano che un processo utente si servirà di una risorsa solo se gli è stata concessa un'abilitazione per essa.

2. Specificare il tipo di operazioni che un processo particolare può compiere su una risorsa assegnata (ad esempio, a un lettore di file si deve permettere soltanto di leggere i file, mentre a uno scrittore si possono concedere sia la lettura sia la scrittura): non dovrebbe essere necessario concedere lo stesso insieme di diritti a ogni processo utente e un processo non dovrebbe avere la possibilità di ampliare il proprio insieme di diritti d'accesso, tranne che abbia ricevuto l'autorizzazione da parte del meccanismo di controllo degli accessi.

3. Specificare l'ordine in cui un processo particolare può compiere le diverse operazioni su una risorsa (ad esempio, un file deve essere aperto prima di essere letto): deve essere possibile dare a due processi limitazioni diverse sull'ordine in cui possono compiere le operazioni sulla risorsa assegnata.

L'incorporazione dei concetti di protezione nei linguaggi di programmazione, intesa come strumento pratico per la progettazione dei sistemi, è in una fase iniziale. La protezione probabilmente acquisterà un crescente interesse da parte dei progettisti di nuovi sistemi con architetture distribuite, e requisiti sempre più severi sulla sicurezza dei dati; sarà quindi riconosciuta più diffusamente anche l'importanza d'idonee notazioni di linguaggio in cui esprimere i requisiti di protezione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

18.7.2 Protezione nel linguaggio Java 2

Il Java è un linguaggio di programmazione orientato agli oggetti introdotto dalla Sun Microsystems. I programmi scritti nel linguaggio Java sono composti da classi, ognuna delle quali è un insieme di campi di dati e di funzioni (chiamate **metodi**) che operano su quei campi. La JVM (*Java virtual machine*) carica una classe come risposta a una richiesta di creazione di istanze (o oggetti) di quella classe. Una tra le caratteristiche più originali e utili del linguaggio Java è la gestione del caricamento dinamico di classi non fidate da una rete e dell'esecuzione all'interno della stessa JVM di classi che si ritengono mutuamente sospette.

Proprio per queste capacità del linguaggio Java, il problema della protezione è di vitale importanza. Le classi eseguite dalla stessa JVM possono provenire da diverse fonti e possono avere diversi gradi di affidabilità. Quindi, è insufficiente imporre la protezione al livello del processo della JVM. Intuitivamente, il fatto che una richiesta d'apertura di file sia accettata dipenderà generalmente da quale classe ha fatto la richiesta d'apertura. Il sistema operativo non ha queste informazioni.

Dunque questo tipo di decisioni di protezione sono gestite all'interno della JVM. Quando la JVM carica una classe, la assegna a un dominio di protezione che fornisce i permessi per quella classe. Il dominio di protezione al quale si assegna una classe dipende dall'URL dal quale la classe è stata caricata e da eventuali firme digitali sul file della classe (le firme digitali sono trattate nel Paragrafo 19.7.1). Un file che permette la configurazione dei criteri di protezione determina i permessi che si dànno al dominio (e alle sue classi). Per esempio, le classi prelevate da un server fidato si potrebbero mettere in un dominio di protezione che permette a tali classi di accedere ai file nelle directory iniziali degli utenti, mentre le classi prelevate da un server ritenuto non fidato potrebbero non avere permessi d'accesso ai file.

Per una JVM può essere difficile stabilire qual è la classe responsabile di una richiesta d'accesso a una risorsa protetta. Gli accessi avvengono spesso in modo indiretto, per mezzo di librerie di sistema o altre classi. Si consideri ad esempio una classe cui non è permesso aprire connessioni di rete. Potrebbe chiamare una libreria di sistema per richiedere il caricamento dei contenuti di un URL. La JVM deve decidere se aprire a no una connessione di rete per questa richiesta. Ma resta il problema di quale classe si dovrebbe considerare per determinare se la connessione si debba concedere o no, l'applicazione o la libreria di sistema.

L'orientamento seguito nel linguaggio Java 2 è quello di richiedere alla classe di libreria di permettere esplicitamente la connessione di rete per il caricamento dell'URL richiesto. Più in generale, per poter accedere a una risorsa protetta, uno dei metodi nella sequenza delle chiamate che ha portato alla richiesta deve esplicitamente asserire il privilegio di accedere alla risorsa. In questo modo, il metodo si *assume la responsabilità* della richiesta e, presumibilmente, eseguirà anche tutti i controlli necessari ad assicurare la sicurezza della richiesta stessa. Chiaramente, non tutti i metodi sono abilitati ad asserire privilegi; lo possono fare solo se la classe d'appartenenza è in un dominio di protezione che ha esso stesso il permesso di esercitare quel privilegio.

Questo metodo di realizzazione si chiama **ispezione della pila**. Ogni thread nella JVM ha una pila associata per le correnti invocazioni di metodi. Quando il suo chiamante è potenzialmente non fidato, un metodo esegue una richiesta d'accesso all'interno di un blocco `doPrivileged`, per accedere direttamente o indirettamente a una risorsa protetta (si tratta del metodo seguito nel JDK 1.2, sebbene per essere più precisi, `doPrivileged` è un metodo della classe `AccessController` al quale si passa una classe con un metodo `run`). Quando l'esecuzione entra nel blocco `doPrivileged`, si annota l'elemento della pila per questo metodo per indicare questo fatto, e successivamente si eseguono le istruzioni nel blocco. Quando, nel seguito, si richiede l'accesso a una risorsa protetta, o da parte di questo metodo o da parte di un metodo da esso chiamato, s'invoca `checkPermissions` per richiedere l'ispezione della pila, allo scopo di determinare se l'accesso richiesto debba essere concesso. L'ispezione consiste nell'esame degli elementi presenti nella pila del thread chiamante, partendo da quello inserito più recentemente e procedendo verso il meno recente. Se prima si trova un elemento che ha l'annotazione `doPrivileged`, `checkPermissions` permette immediatamente l'accesso. Se invece si trova prima un elemento per il quale l'accesso non è consentito secondo il dominio di protezione della classe del metodo, `checkPermissions` genera un'eccezione. Se l'ispezione esaurisce la pila senza trovare né un tipo d'elemento né l'altro, allora la concessione dell'accesso dipende dalla versione (per esempio, l'Internet Explorer 4.0 e il JDK 1.2 permettono l'accesso, mentre il Netscape 4.0 lo nega).

La procedura d'ispezione della pila è illustrata nella Figura 18.8. In quest'esempio, il metodo `gui` di una classe nel dominio di protezione *untrusted applet* compie due operazioni, prima una `get` e poi una `open`. La prima corrisponde all'invocazione del metodo `get` di una classe nel dominio di protezione *URL loader*, che ha i permessi per aprire sessioni nei siti nel dominio `lucent.com`, e in particolare per `proxy.lucent.com` per individuare gli URL. Per questa ragione, l'invocazione di `get` dall'*applet* non fidata andrà a buon fine: la chiamata a `checkPermissions` nella libreria di rete troverà l'elemento della pila relativo al metodo `get` che ha eseguito la sua `open` in un blocco `doPrivileged`. Tut-

dominio di protezione:	*applet* non fidato	caricatore di URL	interconnessione
permesso della socket:	nessuno	`*.lucent.com:80, connect`	qualsiasi

classe:	`gui:`
	`. . .`
	`get(url);`
	`open(addr);`
	`. . .`
		`get(URL u):`
		`. . .`
		`doPrivileged {`
		`open('proxy.lucent.com:80');`
		`}`
		`<request u from proxy>`
		`. . .`
			`open(Addr a):`
			`. . .`
			`checkPermission(a, connect);`
			`connect (a);`
			`. . .`

Figura 18.8 Ispezione della pila.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

tavia, l'invocazione della open da parte dell'*applet* non fidata risulta invece in un'eccezione, poiché la chiamata a `checkPermissions` non trova alcuna annotazione `doPrivileged` prima di raggiungere l'elemento della pila relativo al metodo `gui`.

Ovviamente, affinché l'ispezione della pila possa funzionare, un programma non deve poter modificare le annotazioni sul suo stesso elemento della pila, né compiere altre manipolazioni che interferiscano con l'ispezione della pila. Questa è una delle differenze più importanti tra il linguaggio Java e molti altri linguaggi (compreso il C++). Un programma scritto in Java non può accedere direttamente alla memoria. Piuttosto, può manipolare solo oggetti per i quali ha un riferimento. I riferimenti non si possono contraffare e le manipolazioni si compiono per mezzo d'interfacce ben definite. La conformità a questi criteri è imposta da un raffinato insieme di controlli della fase di caricamento e della fase d'esecuzione. Ne segue che un oggetto non può manipolare la propria pila, poiché non può ottenere un riferimento né a essa né ad altri componenti del sistema di protezione.

Più in generale, i controlli del linguaggio Java nella fase di caricamento e nella fase d'esecuzione impongono la **sicurezza dei tipi** (*type safety*) delle classi. La sicurezza dei tipi garantisce che le classi non possano trattare gli interi come puntatori, scrivere oltre la fine di un vettore, accedere alla memoria in modi arbitrari. Un programma può accedere a un oggetto soltanto attraverso i metodi definiti per quell'oggetto dalla sua classe. Su ciò si fonda il sistema di protezione del linguaggio Java, poiché permette a una classe di incapsulare efficacemente i propri dati e metodi e di proteggerli da altre classi caricate nella stessa JVM. Per esempio, una variabile si può definire *privata*, in modo che solo la classe che la contiene possa accedervi, oppure si può definire *protetta*, in modo che possano accedervi solo la classe che la contiene, le sue sottoclassi e le classi dello stesso *package*. La sicurezza dei tipi garantisce l'imposizione di queste limitazioni.

18.8 Sommario

I sistemi di calcolo contengono molti oggetti, ed essi devono essere protetti contro i possibili abusi. Gli oggetti possono essere fisici (come la memoria, il tempo di CPU o i dispositivi di I/O) o logici (come i file, i programmi e i tipi di dati astratti). Un diritto d'accesso è un permesso per eseguire un'operazione su un oggetto. Un dominio è un insieme di diritti d'accesso. I processi vengono eseguiti in domìni e possono usare tutti i diritti d'accesso del dominio per accedere agli oggetti e manipolarli. Durante il suo ciclo di vita un processo può essere vincolato a un dominio di protezione o può essergli consentito di passare da un dominio a un altro.

La matrice d'accesso è un modello generale di protezione; fornisce un meccanismo per la protezione senza imporre un criterio di protezione particolare al sistema o ai suoi utenti. La separazione tra criteri e meccanismi è un'importante caratteristica di progettazione.

La matrice d'accesso è sparsa e normalmente si realizza per mezzo di liste d'accesso associate a ciascun oggetto, oppure per mezzo di liste di abilitazioni associate a ciascun

dominio. Si può inserire la protezione dinamica nel modello di matrice d'accesso considerando i domìni e la stessa matrice d'accesso come oggetti. La revoca dei diritti d'accesso in un modello di protezione dinamico è di solito più facile da realizzare con lo schema delle liste d'accesso che con le liste di abilitazioni.

I sistemi reali sono molto più limitati e tendono a fornire protezioni solo per i file. Lo UNIX è rappresentativo, poiché per ogni file fornisce le protezioni per lettura, scrittura ed esecuzione, distinte per proprietario, gruppo e chiunque. Il sistema MULTICS, oltre alla protezione dei file, impiega una struttura ad anelli. L'Hydra, il sistema Cambridge CAP e il Mach sono sistemi con abilitazioni che estendono la protezione agli oggetti definiti dagli utenti.

La protezione basata sul linguaggio offre un controllo delle richieste e dei privilegi più selettivo di quello ottenibile con il sistema operativo. Per esempio, una singola JVM può eseguire molti thread, ognuno in un diverso dominio di protezione. La JVM controlla le richieste di risorse attraverso il raffinato meccanismo dell'ispezione della pila e attraverso la sicurezza dei tipi offerta dal linguaggio.

18.9 Esercizi

18.1 Dite quali sono le differenze più importanti fra le liste delle abilitazioni e le liste d'accesso.

18.2 Un file di un MCP Burroughs B7000/B6000 si può etichettare come dati sensibili. Quando si cancella un file di questo tipo, si sovrascrive la sua area di memoria con bit casuali. Dite qual è l'utilità di questa funzione.

18.3 In un sistema con protezione ad anelli, il livello 0 ha i massimi diritti d'accesso agli oggetti e il livello n (maggiore di zero) ha i minori diritti d'accesso. I diritti d'accesso di un programma a un particolare livello della struttura ad anelli sono considerati un insieme di abilitazioni. Dite qual è la relazione fra le abilitazioni rispetto a un oggetto di un dominio al livello j e un dominio al livello i (per $j > i$).

18.4 Considerate un sistema nel quale i videogiochi possono essere usati dagli studenti solo tra le 22 e le 6, dai membri della facoltà tra le 17 e le 8 e dal personale del centro di calcolo a tutte le ore. Suggerite uno schema per realizzare efficacemente questo criterio.

18.5 Nei sistemi RC 4000 e in altri sistemi è definito un albero di processi tale che risorse (oggetti) e diritti d'accesso vengono assegnati a tutti i discendenti di un processo esclusivamente dai loro antenati. Un discendente quindi non può mai avere la possibilità di fare qualcosa che i suoi antenati non possono fare. La radice dell'albero è il sistema operativo, che ha la possibilità di fare qualunque cosa. Considerate un insieme di diritti d'accesso rappresentato da una matrice d'accesso A. $A(x, y)$ definisce i diritti d'accesso del processo x all'oggetto y. Se x è un discendente di z, dite qual è la relazione tra $A(x, y)$ e $A(z, y)$ per un oggetto arbitrario y.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

18.6 Dite quali caratteristiche dell'architettura di un sistema di calcolo sono necessarie per ottenere un'efficiente manipolazione delle abilitazioni. Dite se queste caratteristiche si possono adoperare per la protezione della memoria.

18.7 Considerate un ambiente di calcolo in cui a ciascun processo e a ciascun oggetto del sistema si associa un numero unico. Supponete di permettere al processo col numero n di accedere a un oggetto col numero m solo se $n > m$. Dite di che tipo di struttura di protezione si tratta.

18.8 Dite quali problemi di protezione possono insorgere se si usa una pila condivisa per il passaggio di parametri.

18.9 Considerate un ambiente di calcolo in cui a un processo si assegna il privilegio di accedere a un oggetto solo n volte. Suggerite uno schema per realizzare questo criterio.

18.10 Se si cancellano tutti i diritti d'accesso a un oggetto, questo non è più accessibile. A questo punto occorre cancellare anche l'oggetto, e restituire al sistema lo spazio che occupava. Suggerite una realizzazione efficiente di questo schema.

18.11 Dite cos'è il principio del privilegio minimo, e perché è importante che un sistema di protezione aderisca a questo principio.

18.12 Dite perché è difficile proteggere un sistema in cui gli utenti possono eseguire direttamente le proprie operazioni di I/O.

18.13 Le liste delle abilitazioni si conservano generalmente all'interno dello spazio d'indirizzi d'utente dei rispettivi utenti. Dite in che modo si può garantire che gli utenti non possano modificare il contenuto delle liste.

18.14 Descrivete quali caratteristiche del modello di protezione del linguaggio Java verrebbero meno se si permettesse a un programma scritto in Java di alterare direttamente le annotazioni nel suo elemento della pila.

18.10 Note bibliografiche

Il modello di protezione della matrice d'accesso tra domìni e oggetti è sviluppato in [Lampson 1969] e [Lampson 1971]. [Popek 1974] e [Saltzer e Schroeder 1975] presentano eccellenti rassegne sull'argomento della protezione. [Harrison et al. 1976] si serve di una versione formale di questo modello per dimostrare matematicamente alcune proprietà di un sistema di protezione.

Il concetto di abilitazione è un'evoluzione di quello di *codeword*, di Iliffe e Jodeit, impiegato nel calcolatore della Rice University, [Iliffe e Jodeit 1962]. Il termine *capability* (*abilitazione*) è stato introdotto da [Dennis e Horn 1966].

Il sistema Hydra è descritto in [Wulf et al. 1981]. Il sistema CAP è descritto in [Needham e Walker 1977]. [Organick 1972] discute il sistema di protezione ad anelli del sistema MULTICS.

La revoca è trattata in [Redell e Fabry 1974], [Cohen e Jefferson 1975] e [Ekanadham e Bernstein 1979]. Il principio di separazione tra criteri e meccanismi di protezione è stato sostenuto dal progettista del sistema Hydra, [Levin et al. 1975]. Il problema della reclusione è stato trattato per la prima volta in [Lampson 1973] ed esaminato successivamente in [Lipner 1975].

L'uso di linguaggi di livello più alto per specificare il controllo degli accessi è stato suggerito per la prima volta da [Morris 1973], che ha proposto l'uso delle operazioni `seal` e `unseal`, descritte nel Paragrafo 18.7.1. [Kieburtz e Silberschatz 1978], [Kieburtz e Silberschatz 1983] e [McGraw e Andrews 1979] hanno proposto diversi costrutti di linguaggio per la gestione degli schemi dinamici generali per la gestione delle risorse. [Jones e Liskov 1978] considera il problema dell'incorporazione di uno schema statico di controllo degli accessi in un linguaggio di programmazione che prevede i tipi di dati astratti.

Un'analisi più dettagliata dell'ispezione della pila, che comprende il confronto con altri orientamenti alla sicurezza dell'ambiente Java, si trova in [Wallach et al. 1987] e [Gong et. al 1997].

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 19

Sicurezza

La protezione, così com'è stata esaminata nel Capitolo 18, è un problema strettamente *interno* e riguarda la fornitura di un accesso controllato a programmi e dati memorizzati in un calcolatore. La **sicurezza**, d'altra parte, non richiede solo un adeguato sistema di protezione, ma anche la considerazione dell'ambiente *esterno* nel quale opera il sistema. La protezione interna è inutile se la console del sistema può essere usata da personale non autorizzato, oppure se i file possono essere rimossi dal calcolatore per mezzo di nastri o dischi e portati in un sistema non protetto. Questi problemi di sicurezza riguardano però la gestione piuttosto che il sistema operativo.

Le informazioni memorizzate in un sistema di calcolo (codice e dati) così come le sue risorse fisiche devono essere protette da accessi non autorizzati, distruzioni o alterazioni dolose, e introduzioni accidentali d'incoerenze. In questo capitolo si esaminano i modi in cui le informazioni possono essere usate in modo scorretto o rese incoerenti intenzionalmente, quindi si presentano alcuni meccanismi per la difesa da tali situazioni.

19.1 Problema della sicurezza

Nel Capitolo 18 sono trattati alcuni meccanismi offerti dai sistemi operativi (con l'ausilio di appropriate caratteristiche dell'architettura) per consentire agli utenti di proteggere le loro risorse (di solito programmi e dati). Questi meccanismi funzionano bene fintanto che gli utenti si conformano ai modi d'uso e d'accesso previsti per tali risorse. Si dice che un sistema è **sicuro** se le sue risorse si adoperano e vi si accede soltanto nei modi previsti. Sfortunatamente, non è possibile ottenere una sicurezza totale; ciononostante si devono avere meccanismi che rendano l'elusione della sicurezza un caso raro e non la norma.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le violazioni della sicurezza (abusi) del sistema si possono classificare come intenzionali (dolose) o accidentali. È più semplice proteggere un sistema contro gli abusi accidentali che contro quelli dolosi. Tra le forme di accessi dolosi ci sono le seguenti:

- letture non autorizzate di dati (furto d'informazioni);
- alterazione non autorizzata dei dati;
- distruzione non autorizzata dei dati;
- impedimento del legittimo uso del sistema (rifiuto del servizio).

La realizzazione di un sistema che garantisca un'assoluta protezione dagli abusi dolosi è impossibile, ma il costo di tali azioni si può rendere sufficientemente alto da scoraggiare quasi tutti, se non tutti, i tentativi d'accesso non autorizzati alle informazioni contenute nel sistema. Per proteggere il sistema è necessario prendere misure di sicurezza a quattro livelli:

1. **Fisico.** I siti che ospitano i sistemi di calcolo devono essere protetti fisicamente contro gli accessi armati o furtivi da parte d'intrusi.
2. **Umano.** Occorre vagliare accuratamente gli utenti per ridurre la possibilità di concedere autorizzazioni a utenti che forniscano l'accesso a intrusi (ad esempio in cambio di denaro).
3. **Rete.** Molti dati informatici nei sistemi moderni viaggiano in linee a noleggio private, linee condivise (come quelle della rete Internet), o linee di tipo telefonico. L'intercettazione di questi dati può essere tanto dannosa quanto l'intrusione in un calcolatore. L'interruzione di queste comunicazioni potrebbe essere dovuta a un attacco per **rifiuto del servizio**, che riduce le possibilità d'uso da parte degli utenti e l'affidamento che si può fare sul sistema.
4. **Sistema operativo.** Il sistema deve proteggere se stesso dalle violazioni della sicurezza intenzionali o accidentali.

Per garantire la sicurezza del sistema operativo è necessario mantenere la sicurezza ai primi due livelli: un punto debole ad alto livello della sicurezza (fisico o umano) consente di eludere le misure di sicurezza più rigorose a un livello più basso (sistema operativo).

In molte applicazioni garantire la sicurezza del sistema di calcolo merita uno sforzo considerevole. I grandi sistemi commerciali contenenti dati relativi agli stipendi o ad altre attività finanziarie sono obiettivi invitanti per i ladri. I sistemi che contengono dati riguardanti operazioni aziendali possono interess. i concorrenti privi di scrupoli; inoltre la perdita di questi dati, sia accidentale sia fraudolenta, potrebbe compromettere seriamente il funzionamento dell'azienda.

D'altra parte l'architettura del sistema deve avere caratteristiche di protezione (Capitolo 18) al fine di consentire la realizzazione di funzioni di sicurezza. Ad esempio, l'MS-DOS e le vecchie versioni del Macintosh offrono un basso grado di sicurezza poiché le architetture per cui furono progettati non prevedevano la protezione né della memoria né dell'I/O. Oggi che le architetture hanno raggiunto livelli di raffinatezza tali da avere caratteristiche di protezione, i progettisti di questi sistemi operativi hanno cercato di aggiungere la sicurezza. Sfortunatamente aggiungere una funzione a un sistema già definito è molto più difficile che progettare e realizzare la stessa funzione prima della realizzazione del sistema. Sistemi operativi più recenti, come il Windows NT, sono stati progettati e realizzati per fornire funzioni di sicurezza.

Nel seguito del capitolo si tratta la sicurezza al livello del sistema operativo; la sicurezza ai livelli umano e fisico, benché importante, va oltre gli scopi di questo libro. La sicurezza nei sistemi operativi e tra sistemi operativi si realizza in diversi modi: dalla richiesta di parole d'ordine per l'accesso al sistema fino all'isolamento dei processi concorrenti in esecuzione nel sistema. Anche il file system offre un certo grado di protezione.

19.2 Autenticazione degli utenti

Per i sistemi operativi il principale problema di sicurezza riguarda l'**autenticazione**. Il sistema di protezione dipende dalla capacità d'identificare i programmi e i processi correntemente in esecuzione, che a sua volta è basata sulla capacità di identificare ogni utente del sistema. Normalmente un utente identifica se stesso, quindi si tratta di stabilire se l'identità di un utente è autentica. Generalmente l'autenticazione è basata su uno o più dei seguenti tre elementi: oggetti posseduti dall'utente (una chiave o una scheda); conoscenze dell'utente (un identificatore e una parola d'ordine); un attributo dell'utente (impronta digitale, impronta della retina o firma).

19.2.1 Parole d'ordine

Il metodo più diffuso per identificare un utente è quello che prevede l'uso di **parole d'ordine** (*password*). Quando un utente s'identifica attraverso il proprio identificatore, riceve la richiesta d'immissione di una parola d'ordine. Se la parola d'ordine immessa dall'utente corrisponde a quella memorizzata nel sistema, il sistema presume che l'utente sia legittimo.

Spesso le password si usano per proteggere oggetti del calcolatore quando non esistono schemi di protezione più completi. Le parole d'ordine si possono considerare un caso particolare di chiavi o di abilitazioni. Ad esempio, si potrebbe associare una parola d'ordine a ogni risorsa, come un file; in questo modo ogni volta che si chiede l'uso della risorsa, si deve fornire la relativa parola d'ordine; se questa è giusta, si permette l'accesso. A diritti d'accesso diversi si possono associare parole d'ordine diverse. Si possono ad esempio impiegare parole d'ordine diverse per ciascuna delle seguenti operazioni su file: lettura, aggiunta e aggiornamento.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.2.2 Vulnerabilità delle parole d'ordine

Le parole d'ordine sono assai comuni poiché sono facili da capire e da usare. Sfortunatamente, si possono indovinare, esporre accidentalmente, sottrarre o trasferire illegalmente da un utente autorizzato a uno privo d'autorizzazione.

Ci sono due comuni metodi per indovinare una parola d'ordine. Uno si fonda sulla conoscenza dell'utente (o d'informazioni su di esso) da parte di intrusi (sia umani sia programmi): troppo spesso le persone usano per le parole d'ordine informazioni ovvie (come il nome del loro gatto o del loro consorte). L'altro modo è l'uso della forza bruta: enumerare tutte le possibili combinazioni di lettere, numeri e segni d'interpunzione finché si trova la parola d'ordine cercata. In questo modo è facile indovinare le parole d'ordine brevi; una parola d'ordine di quattro cifre decimali permette solo 10.000 combinazioni. In media occorrono solo 5000 tentativi per indovinare quella giusta. Un programma che può fare un tentativo ogni millisecondo impiegherebbe solo cinque secondi a indovinare una parola d'ordine di quattro cifre. L'enumerazione non è però adatta all'individuazione delle parole d'ordine in sistemi che consentono parole d'ordine lunghe, che distinguono le lettere maiuscole dalle minuscole, e che permettono l'uso dei numeri e dei caratteri di punteggiatura. Naturalmente gli utenti devono approfittare delle maggiori possibilità e non devono usare, ad esempio, soltanto le lettere minuscole.

Il fallimento della sicurezza dovuto all'esposizione delle parole d'ordine può derivare da una sorveglianza visiva o elettronica: un intruso può sbirciare sopra la spalla (*shoulder surfing*) di un utente mentre questo inizia una sessione di lavoro e carpire facilmente la sua parola d'ordine. Alternativamente, chiunque abbia accesso alla rete in cui risiede un calcolatore può inserire un monitor di rete che gli consenta di osservare tutti i dati trasferiti nella rete (*sniffing*), compresi gli identificatori degli utenti e le parole d'ordine. La cifratura dei flussi di dati contenenti le parole d'ordine risolve questo problema.

L'esposizione diventa un problema particolarmente grave se si annota la parola d'ordine dove può essere letta o persa. Alcuni sistemi obbligano gli utenti a scegliere parole d'ordine lunghe o difficili da ricordare; se portato all'estremo, questo metodo può spingere qualche utente a registrare la propria parola d'ordine, determinando una sicurezza assai minore di quella di sistemi che consentono parole d'ordine semplici.

Il metodo finale di compromissione delle parole d'ordine, il trasferimento illecito, deriva dalla natura umana. In molti sistemi di calcolo vige una regola che vieta la condivisione delle utenze; tale regola talvolta si stabilisce per ragioni di contabilizzazione, ma spesso è impiegata per migliorare la sicurezza. Ad esempio, se un identificatore d'utente è condiviso da più utenti e si è verificata una violazione della sicurezza con esso, è impossibile sapere chi l'ha usato al momento della violazione, o anche se era uno degli utenti autorizzati; con un identificatore diverso per ciascun utente, quest'ultimo può essere direttamente interpellato sul suo uso. Talvolta gli utenti violano le regole di condivisione per aiutare loro amici o per raggirare la contabilizzazione; tale comportamento può causare l'accesso al sistema di utenti non autorizzati, e magari pericolosi.

Le parole d'ordine possono essere generate dal sistema o scelte dall'utente. Quelle generate dal sistema possono essere difficili da ricordare e quindi gli utenti potrebbero trascriverle. Quelle scelte dagli utenti sono invece più facili da indovinare, un utente potrebbe ad esempio usare il proprio nome o il nome della sua automobile preferita. Gli amministratori possono controllare periodicamente le parole d'ordine degli utenti e informarli se sono troppo corte o troppo facili da indovinare. Alcuni sistemi prevedono l'*invecchiamento* delle parole d'ordine, costringendo gli utenti a modificarle a intervalli regolari, ad esempio ogni tre mesi. Questo metodo non è del tutto sicuro, poiché gli utenti possono alternare sempre le stesse due parole d'ordine. Questo problema si risolve, come accade in alcuni sistemi, registrando l'elenco delle ultime n parole d'ordine per ciascun utente allo scopo di impedirne il riutilizzo.

Lo schema delle parole d'ordine può avere parecchie varianti. Ad esempio, la parola d'ordine può essere cambiata spesso; al limite a ogni sessione di lavoro. Alla fine di *ogni* sessione di lavoro, il sistema o l'utente sceglie una nuova parola d'ordine che si dovrà usare per la sessione di lavoro successiva. In questo caso, anche se una parola d'ordine viene usata abusivamente, può essere usata una sola volta. Quando l'utente legittimo, all'apertura della sessione di lavoro successiva, prova a usare la parola d'ordine trova che non più valida e scopre la violazione della sicurezza. A questo punto si può reagire con azioni che rimedino alla violazione della sicurezza.

19.2.3 Parole d'ordine cifrate

In tutti questi metodi c'è la difficoltà di mantenere segrete le parole d'ordine all'interno del calcolatore. Si pone il problema di come il sistema possa memorizzare le parole d'ordine in modo sicuro, e permetterne l'uso quando un utente presenta la propria parola d'ordine. Il sistema UNIX si serve della **cifratura** per evitare di mantenere segreto il proprio elenco di parole d'ordine. Il sistema contiene una funzione estremamente difficile da invertire — i progettisti sperano che sia impossibile — ma semplice da calcolare. Cioè, dato un valore x, è facile calcolare il valore della funzione $f(x)$; ma, dato un valore $f(x)$, è 'impossibile' calcolare x. Questa funzione si usa per cifrare tutte le parole d'ordine, che si memorizzano solo in questa forma. Quando un utente immette una parola d'ordine, questa viene cifrata e confrontata con quella già cifrata e memorizzata. Anche se la parola d'ordine cifrata viene vista, non potendo essere decifrata, è impossibile stabilire la parola d'ordine originale. Quindi non è necessario tenere segreto il file che contiene le parole d'ordine. La funzione $f(x)$ si realizza con **algoritmo di cifratura** rigorosamente progettato e verificato (Paragrafo 19.7.2).

La pecca di questo metodo consiste nel fatto che il sistema operativo non ha più il controllo delle parole d'ordine. Anche se le parole d'ordine sono cifrate, chiunque disponga di una copia del file che le contiene può servirsi di efficienti procedure di cifratura contro tale file, ad esempio cifrando ciascuna parola di un dizionario e confrontando il risultato con le parole d'ordine cifrate contenute nel file; se l'utente ha scelto come parola d'ordine una parola contenuta nel dizionario, la parola d'ordine viene scoperta. Se s'impiegano calcolatori sufficientemente veloci, o anche batterie di calcolatori lenti, un simile confronto può richiedere solo poche ore. Poiché i sistemi UNIX fanno uso di un ben noto algoritmo di cifratura, un 'pirata' informatico (*cracker*) potrebbe conservare un insieme di coppie <*parola d'ordine*, *parola d'ordine cifrata*> per individuare rapidamente le parole d'ordine già violate. Per questo motivo le nuove versioni dello UNIX registrano

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

le parole d'ordine cifrate in un file che può essere letto solo dall'amministratore del sistema (*superuser*). I programmi che confrontano le parole d'ordine inserite con quelle registrate eseguono `setuid` per l'utente *root* in modo che possano leggere questo file, ma gli altri utenti non possano farlo.

Un altro punto debole dei sistemi di parole d'ordine dello UNIX è determinato dal fatto che molti sistemi UNIX considerano significativi soltanto i primi otto caratteri, è quindi estremamente importante che gli utenti sfruttino appieno lo spazio delle parole d'ordine disponibile. Per evitare il metodo di cifratura del dizionario alcuni sistemi non consentono l'uso di parole del dizionario come parola d'ordine. Un buon metodo è quello di generare le proprie parole d'ordine usando la prima lettera di ciascuna parola di una frase facile da ricordare, ciò impiegando sia lettere minuscole sia maiuscole intercalate da un adeguato numero di segni d'interpunzione. Ad esempio, la frase "Il nome di mia madre è Teresa." può produrre la parola d'ordine "IndmmèT.!"; che è difficile da scoprire, ma, per l'utente, facile da ricordare.

19.2.4 Parole d'ordine monouso

Uno dei modi in cui un sistema può evitare i tentativi di sottrazione delle parole d'ordine è l'uso di un insieme di **parole d'ordine accoppiate**. All'inizio di una sessione di lavoro, il sistema sceglie a caso una coppia di parole d'ordine dall'insieme delle parole d'ordine accoppiate e presenta all'utente un elemento della coppia selezionata; l'utente deve fornire l'altro elemento. In questo tipo di sistema si *esige* dall'utente la risposta corretta.

Questo metodo si può generalizzare impiegando un algoritmo come parola d'ordine. L'algoritmo potrebbe essere, ad esempio, una funzione definita sui numeri interi; il sistema sceglie a caso un numero intero e lo presenta all'utente che vi applica la funzione e risponde con il risultato ottenuto. Anche il sistema calcola il valore della funzione e, se i risultati coincidono, consente l'accesso.

Tale metodo che implica l'uso di parole d'ordine algoritmiche non incorre nel pericolo dell'esposizione delle parole d'ordine; un utente può cioè immettere una parola d'ordine e nessuna entità che intercetta tale parola d'ordine potrebbe riutilizzarla. In questa variante, l'utente e il sistema condividono un'informazione segreta che non è mai trasmessa in modi che rischino di comprometterne la riservatezza, ma si usa insieme con un seme, anch'esso condiviso, come argomento di una funzione. Il seme (*seed*) è un numero casuale o una sequenza alfanumerica e costituisce la richiesta d'identificazione da parte del sistema. L'informazione segreta e il seme si usano come argomenti della funzione *f*(*informazione*, *seme*), il valore di questa funzione è la parola d'ordine che si comunica al calcolatore. Poiché il calcolatore è a conoscenza sia del seme sia dell'informazione segreta, può anch'esso calcolare il valore della funzione. Se i due risultati coincidono, l'identità dell'utente è autenticata. All'accesso successivo il sistema genera un altro seme e si ripete il procedimento. Questa volta la parola d'ordine è diversa.

In questo sistema a **parole d'ordine monouso**, la parola d'ordine cambia a ogni richiesta d'accesso: chiunque scopra la parola d'ordine usata in una sessione di lavoro e

cerchi di sfruttarla in un'altra non avrà successo. Le parole d'ordine monouso sono uno fra i pochissimi metodi capaci d'impedire l'errata autenticazione di un utente dovuta all'esposizione della parola d'ordine. Prodotti commerciali che realizzano le parole d'ordine monouso, ad esempio SecurID, impiegano specifiche calcolatrici elettroniche; molte di queste hanno la forma delle carte di credito e sono dotate di una tastiera e di un visore, alcune usano l'ora corrente come seme casuale. L'utente immette l'informazione segreta, nota anche come **numero d'identificazione personale** (*personal identification number* — PIN), per mezzo della tastiera, e il visore mostra la parola d'ordine monouso. L'uso di un generatore di parole d'ordine monouso e di un PIN è una forma di **autenticazione secondo due fattori**. In questo caso sono necessari due diversi tipi di componenti. L'autenticazione secondo due fattori offre un'autenticazione molto più sicura di quella che considera un solo fattore.

Una variante di questi sistemi usa un **libro dei codici**, o **taccuino monouso**, cioè un elenco di parole d'ordine usa e getta. In questo caso ogni parola d'ordine dell'elenco si usa, nell'ordine, una sola volta quindi si cancella dalla lista. Il diffuso sistema S/Key impiega come fonte delle parole d'ordine monouso uno specifico programma di calcolo o un libro di codici compilato in base a esso.

19.2.5 Tecniche biometriche

Ci sono molte altre varianti riguardo all'impiego delle parole d'ordine per l'autenticazione. Per garantire l'accesso sicuro ad ambienti fisici protetti, ad esempio, a centri d'elaborazione di dati, spesso si usano strumenti per la rilevazione dell'impronta del palmo o dell'intera mano. Questi lettori confrontano i parametri memorizzati con quelli che rileva la tavoletta per la lettura dei parametri della mano. Tali parametri possono comprendere una mappa della temperatura, la lunghezza delle dita, la loro larghezza e il disegno delle linee. Questi dispositivi sono però ancora troppo ingombranti e costosi per essere usati per la normale autenticazione da parte di un calcolatore.

I lettori d'impronte digitali sono diventati accurati e sufficientemente economici e dovrebbero diffondersi maggiormente. Questi dispositivi leggono il motivo formato dalle increspature della pelle sulle dita e lo convertono in una sequenza di numeri. Di solito memorizzano un insieme di sequenze, per adattarsi alla posizione del dito sulla tavoletta di lettura e ad altri fattori. Uno specifico programma può a questo punto eseguire la scansione del dito e confrontare i dati ottenuti con quelli memorizzati, determinando se il dito sulla tavoletta corrisponde a quello relativo ai dati memorizzati. Chiaramente, si possono mantenere profili di molti utenti e il dispositivo di scansione consente di distinguerli. Uno schema di autenticazione secondo due fattori molto accurato è quello che richiede sia un nome d'utente e la relativa parola d'ordine sia la scansione dell'impronta digitale. Se per il trasferimento si cifrano queste informazioni, il sistema può essere molto resistente alle tecniche d'imitazione delle identità e di reimmissione d'informazioni autentiche precedentemente intercettate (*replay attack*).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.3 Minacce ai programmi

Quando un utente può usare un programma scritto da un altro utente, si possono verificare abusi e comportamenti inattesi. Nei Paragrafi 19.3.1, 19.3.2 e 19.3.3 sono descritti alcuni metodi comuni per compiere tali abusi: cavalli di Troia e trabocchetti, e attacchi basati sull'alterazione della pila.

19.3.1 Cavalli di Troia

Molti sistemi dispongono di un meccanismo che permette agli utenti di usare programmi scritti da altri utenti. Se questi programmi si eseguono in un dominio che fornisce i diritti d'accesso dell'utente che esegue il programma, gli altri utenti possono abusare di questi diritti. Un elaboratore di testi, ad esempio, può includere il codice per la ricerca di parole o sezioni di testo; se le parole vengono trovate, tutto il file può essere copiato in un'area speciale accessibile al creatore dell'elaboratore di testi. Un segmento di codice che abusi del suo ambiente è detto **cavallo di Troia**. I lunghi percorsi di ricerca, come quelli diffusi nei sistemi UNIX, aggravano il problema dei cavalli di Troia. Il percorso di ricerca elenca l'insieme delle directory in cui compiere la ricerca quando si sottopone un nome di programma ambiguo. Tutte le directory che si trovano nel percorso di ricerca devono essere sicure, altrimenti un cavallo di Troia può insinuarsi nel percorso dell'utente ed essere eseguito accidentalmente.

Si consideri, ad esempio, l'uso del carattere '.' in un percorso di ricerca; tale carattere indica all'interprete dei comandi di includere la directory corrente nella ricerca. Quindi, se un utente ha il carattere '.' nel suo percorso di ricerca, ha impostato la sua directory corrente a una directory di un amico e inserisce il nome di un normale comando di sistema, il comando potrebbe essere eseguito dalla directory dell'amico. Il programma sarebbe eseguito all'interno del dominio dell'utente, consentendo al programma stesso di fare tutto ciò che è consentito fare all'utente, compresa, ad esempio, la cancellazione dei suoi file (che sono effettivamente nella directory dell'amico).

Una variante del cavallo di Troia è un programma che emula una procedura d'inizio di una sessione di lavoro: l'ignaro utente, nella fase d'accesso a un terminale, crede di aver scritto erroneamente la propria parola d'ordine; prova ancora e, questa volta, ha successo. Ciò che realmente accade in un caso come questo è che un emulatore della procedura d'accesso alla sessione di lavoro, sottrae il nome d'utente e la parola d'ordine dell'utente. L'emulatore registra la parola d'ordine e mostra un messaggio d'errore nell'inserimento dei dati, quindi interrompe la propria esecuzione e lascia l'utente di fronte alla vera procedura d'accesso. Questo tipo d'attacco può essere respinto dal sistema operativo mostrando un abituale messaggio alla fine di una sessione di lavoro interattiva o attraverso una sequenza di caratteri non 'catturabile' inviata dalla tastiera, come la sequenza Ctrl-Alt-Canc impiegata nel sistema operativo Windows NT.

19.3.2 Trabocchetti

Il progettista di un programma o di un sistema può lasciare nel programma un 'buco' segreto. Questo tipo di violazione della sicurezza, detto **trabocchetto** (*trap door*), è stato mostrato nel film *War Games*. Il codice può ad esempio cercare uno specifico identificatore d'utente, o una parola d'ordine specifica, e può quindi aggirare le normali procedure di sicurezza. Alcuni programmatori sono stati arrestati per aver truffato banche inserendo errori d'arrotondamento nel loro codice per ottenere l'accredito nei propri conti degli occasionali mezzi centesimi di dollaro. Considerato il numero di transazioni eseguite da una grande banca, con tali accrediti si possono raggiungere delle cospicue quantità di denaro.

Un abile trabocchetto si potrebbe inserire in un compilatore, che in questo caso potrebbe generare sia un codice oggetto normale sia un codice oggetto contenente un trabocchetto, a prescindere dal codice sorgente da compilare. Si tratta di un'attività particolarmente nefasta, poiché un'ispezione del codice sorgente del programma non rivelerebbe alcun problema. L'informazione è contenuta solo nel codice sorgente del compilatore. I trabocchetti costituiscono un problema difficile: per individuarli è necessario analizzare tutto il codice sorgente dei componenti di un sistema. Poiché i sistemi possono essere composti da milioni di righe di codice, queste analisi non si fanno spesso.

19.3.3 Attacchi basati sull'alterazione della pila

L'attacco basato sull'alterazione della pila (che si compie con la tecnica nota come *buffer-overflow*) è il modo più diffuso col quale un utente esterno a un sistema può ottenere un accesso non autorizzato a esso, attraverso una rete. Un utente autorizzato potrebbe servirsi dello stesso metodo per ottenere privilegi d'accesso che in realtà non gli spettano (*privilege escalation*).

Sostanzialmente questi attacchi sfruttano un errore in un programma. L'errore può essere dovuto semplicemente alla bassa qualità della programmazione, ad esempio trascurare di controllare che si rispettino i limiti di un elemento da riempire con dati in ingresso. In questo caso l'aggressore invia più dati di quelli che il programma si aspetta. Con una serie di tentativi, oppure esaminando il codice sorgente del programma da attaccare, se questo è disponibile, l'aggressore determina i punti vulnerabili e scrive un programma che permette di compiere le seguenti azioni:

1. superare (*overflow*) i limiti di un campo da riempire immettendo dati, o di un argomento della riga di comando, o di un vettore che riceve dati in ingresso (*buffer*) — ad esempio in un demone di rete — fino a invadere la pila;
2. l'invasione della pila si deve compiere in modo da sovrascrivere il corrente indirizzo di ritorno annotato nella pila con l'indirizzo del segmento di codice che compie l'attacco;
3. scrivere il semplice segmento di codice, per l'area successiva nella pila, che include i comandi che l'aggressore desidera eseguire, ad esempio l'attivazione di un interprete di comandi.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il risultato dell'esecuzione di questo programma d'attacco sarà la disponibilità di un'interprete dei comandi o l'esecuzione di un altro comando con i privilegi di amministratore.

Ad esempio, se un modulo in una pagina Web prevede che il nome d'utente sia inserito in un apposito campo, l'aggressore potrebbe immettere col nome d'utente un insieme di caratteri aggiuntivi (con l'obiettivo di superare uno dei limiti del vettore che li riceve e di invadere la pila), un nuovo indirizzo di ritorno da caricare nella pila, e il codice che l'aggressore vuole eseguire. Quando la funzione per la lettura dei dati contenuti nel vettore termina la sua esecuzione, l'indirizzo di ritorno è quello del codice dell'aggressore, che quindi viene eseguito.

Questo tipo d'attacco è particolarmente dannoso perché si può eseguire all'interno di un sistema e si può anche trasmettere nei canali di comunicazione disponibili. Un attacco di questo tipo avviene attraverso protocolli che di solito si usano per comunicare con il calcolatore, quindi può essere molto difficile da individuare e da prevenire. Questi attacchi possono anche oltrepassare le barriere di sicurezza (i cosiddetti *firewall*), si veda il Paragrafo 19.5.

Una soluzione di questo problema consiste nel dotare la CPU di una funzione che le consenta d'impedire l'esecuzione di codice presente in una sezione della memoria assegnata alla pila. Le versioni recenti delle CPU SPARC della Sun includono questa funzione e le versioni recenti del sistema operativo Solaris la utilizzano. In questo caso, l'indirizzo di ritorno della funzione attaccata si può ancora modificare, ma se l'indirizzo di ritorno è all'interno della pila e si cerca di eseguire quel codice, la CPU genera un segnale di eccezione cui segue l'arresto del programma e la comunicazione dell'errore.

19.4 Minacce ai sistemi

La maggior parte dei sistemi operativi fornisce ai processi gli strumenti per generare altri processi. Negli ambienti di questo tipo si possono determinare situazioni di abuso delle risorse del sistema e dei file degli utenti. Quella che segue è una presentazione dei due metodi d'abuso più comuni: i cosiddetti 'worm' e i 'virus'.

19.4.1 Worm

Un **worm** è un processo che sfrutta gli effetti della **proliferazione** per minare le prestazioni del sistema; esso genera continuamente copie di se stesso logorando le risorse del sistema, talvolta fino a renderlo inutilizzabile da tutti gli altri processi. I worm diventano particolarmente efficaci nelle reti di trasmissione, poiché hanno la possibilità di riprodursi in diversi sistemi a esse collegati e quindi di far crollare l'intera rete. Una situazione di questo tipo si è verificata nel 1988 tra sistemi UNIX connessi alla rete Internet, causando milioni di dollari di perdita in tempo di sistema e di programmazione.

Al termine del giorno lavorativo del 2 novembre 1988, Robert Tappan Morris Jr., uno studente del primo anno della Cornell University, inserì un programma worm in uno o più calcolatori connessi alla rete Internet. Questo programma aveva come obiettivo le stazioni di lavoro Sun 3 della Sun Microsystems e i calcolatori VAX che eseguivano varianti della versione 4 del BSD UNIX. Il programma si propagò rapidamente per grandi distanze, ed entro poche ore dalla sua immissione consumò le risorse dei sistemi infetti fino a causarne il crollo.

Sebbene Robert Morris avesse progettato il programma affinché si riproducesse e distribuisse rapidamente, furono alcune caratteristiche dell'ambiente di rete dello UNIX che fornirono al programma i mezzi per propagarsi per tutto il sistema. Probabilmente Morris scelse per l'attivazione iniziale un calcolatore connesso alla rete Internet accessibile dagli utenti esterni. Da qui il worm sfruttò le lacune nelle procedure di sicurezza dello UNIX e ingannò le funzioni che semplificano la condivisione delle risorse in una rete locale per ottenere accessi non autorizzati a migliaia di altri siti connessi.

Il worm era composto da due programmi, un **rampino d'arrembaggio** (*grappling hook*, detto anche **avviamento** o **vettore**) e un programma principale. Il rampino d'arrembaggio, di nome `11.c`, era costituito da 99 righe di codice in Linguaggio C da compilare ed eseguire in ogni calcolatore raggiunto. Una volta stabilitosi nel calcolatore sotto attacco, il rampino d'arrembaggio si connetteva al calcolatore nel quale era stato generato e caricava una copia del programma principale nel sistema *agganciato* (Figura 19.1). Il programma principale iniziava quindi la ricerca di altri calcolatori cui il sistema appena infettato poteva connettersi con facilità. Per questa operazione, Morris sfruttò il comando `rsh`; si tratta di un comando di rete del sistema operativo UNIX che facilita le esecuzioni remote. Mediante l'impostazione di file speciali contenenti un elenco di coppie di nomi *<calcolatore, nome d'utente>* gli utenti possono omettere l'inserimento della parola d'ordine a ogni accesso ai calcolatori remoti presenti nell'elenco di coppie. Il worm analizzava questi file speciali alla ricerca dei nomi dei siti che avrebbero consentito l'esecuzione re-

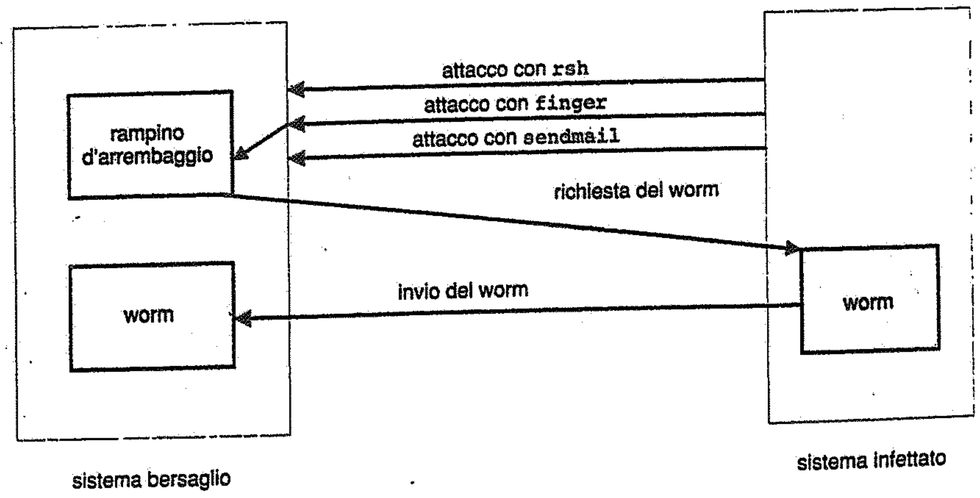


Figura 19.1 L'*Internet worm* di Robert Morris.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

mota senza chiedere l'immissione della parola d'ordine, attivava in questi sistemi un interprete dei comandi remoto, quindi vi caricava il programma principale e ricominciava l'esecuzione.

L'attacco per mezzo dell'accesso remoto era solo uno dei tre metodi d'infezione impiegati. Gli altri due sfruttavano bachi del sistema operativo presenti nei programmi `finger` e `sendmail` del sistema operativo UNIX. Il programma `finger` funziona come una guida telefonica elettronica; il comando

`finger` *`nome_d'utente@nome_del_calcolatore`*

riporta il nome reale e per l'accesso al sistema di una persona, e altre informazioni eventualmente specificate dall'utente, come gli indirizzi e i numeri telefonici di casa e ufficio, l'ambito di lavoro, o interessanti citazioni. Il comando `finger` viene eseguito come processo in sottofondo (demone) in ciascun sito BSD e risponde alle richieste inviategli da qualsiasi sito della rete Internet. Il punto vulnerabile alle violazioni dolose era costituito dal fatto che la lettura dei dati ricevuti in ingresso era eseguita senza controllare i limiti del vettore di memoria che avrebbe dovuto contenerli: il codice eseguiva un **attacco per alterazione della pila**. Il programma di Morris inviava al comando `finger` una richiesta costituita di una sequenza di 536 byte, appositamente costruita affinché oltrepassasse i limiti del vettore per i dati in ingresso e sovrascrivesse l'elemento della pila del programma. Anziché restituire il controllo alla funzione `main` nella quale il programma si trovava prima della chiamata di Morris, il controllo del demone `finger` veniva deviato a una procedura contenuta nella sequenza di 536 byte ora residente nella pila. Questa procedura eseguiva il comando `/bin/sh`, che, nel caso di successo, dava al worm un interprete dei comandi remoto nella macchina presa di mira.

Anche il baco sfruttato in `sendmail` comportava l'uso di un demone a scopi dolosi. Il programma `sendmail` instrada i messaggi di posta elettronica in un ambiente di rete. Alcune istruzioni d'ausilio all'individuazione e alla correzione degli errori contenute nel programma consentono di controllare e vedere lo stato del sistema di gestione della posta. Quest'operazione di controllo è utile agli amministratori di un sistema ed è spesso lasciata in attività come processo in sottofondo. Morris incluse nel proprio arsenale una chiamata al comando `debug`, la quale, invece di specificare un indirizzo d'utente come nei normali casi di verifica, inviava un insieme di comandi che spedivano ed eseguivano una copia del programma avente la funzione di rampino d'arrembaggio.

Una volta giunto sul posto, il programma principale del worm intraprendeva una serie di tentativi sistematici per scoprire le parole d'ordine degli utenti. Inizialmente cercava utenti privi di parole d'ordine o con parole d'ordine costituite da combinazioni tra i veri nomi e i nomi d'accesso, quindi provava con le parole contenute in un dizionario interno di 432 lemmi, e infine provava come possibile parola d'ordine ogni parola del dizionario in linea dello UNIX. Questo raffinato quanto efficiente algoritmo di ricerca in tre fasi delle parole d'ordine consentiva al worm di ottenere ulteriori accessi con altri nomi d'utente al sistema infettato. Poi il worm cercava in ogni utenza violata nuovi file di dati di `rsh`; quindi faceva dei tentativi su tutti gli elementi di `rsh` e, come si è descritto precedentemente, poteva ottenere l'accesso a utenze in sistemi remoti.

A ogni nuovo accesso, il worm cercava copie di se stesso già attive. Se ne trovava una, la nuova copia terminava la propria esecuzione, a esclusione di ogni settima istanza.

Se fosse stato progettato in modo da terminare ogniqualvolta si propagava su una macchina già infettata, probabilmente non sarebbe mai stato scoperto. Consentendo a ogni settima copia di proseguire (probabilmente per disorientare i tentativi di arrestarne la diffusione attraverso l'adescamento con worm *fasulli*) portò alla totale infestazione di sistemi Sun e VAX connessi alla rete Internet.

Le stesse caratteristiche dell'ambiente di rete del sistema operativo UNIX che favorirono la propagazione del worm contribuirono anche al suo arresto. La semplicità nelle comunicazioni elettroniche, i meccanismi per la copiatura di file sorgenti e binari in calcolatori remoti e la possibilità di accedere sia al codice sorgente sia all'esperienza degli altri consentirono, grazie a sforzi congiunti, il rapido sviluppo di una soluzione. Già dalla sera del giorno successivo, il 3 novembre, cominciarono a circolare via Internet, tra gli amministratori dei vari sistemi, metodi per arrestare il programma invasore. Nel giro di pochi giorni erano disponibili specifici programmi per la correzione provvisoria dei difetti nella sicurezza sfruttati dal worm.

Ci si può chiedere perché Morris diffuse il suo *Internet worm*. L'azione è stata considerata sia come bravata, in seguito degenerata, sia come seria azione criminale. Vista la complessità del metodo d'attacco, è improbabile che l'immissione del worm o la dimensione della sua diffusione non fossero intenzionali. Il programma compiva operazioni elaborate per nascondere le proprie tracce e per respingere ogni tentativo di arrestarne la diffusione. D'altra parte il programma non conteneva codice rivolto al danneggiamento o alla distruzione dei sistemi in cui veniva eseguito. L'autore chiaramente possedeva l'esperienza necessaria a includere questo genere di comandi; in realtà nel codice d'avviamento erano contenute alcune strutture di dati utilizzabili per trasferire un cavallo di Troia o un virus (Paragrafi 19.3.1 e 19.4.2). Il comportamento del programma potrebbe condurre a interessanti osservazioni, anche se non fornisce fondati elementi per dedurne le motivazioni. Di definitivo resta comunque il risvolto legale dell'azione: una corte federale ha dichiarato Robert Morris colpevole e gli ha comminato una condanna a tre anni di libertà vigilata, 400 ore di servizio alla comunità e un'ammenda di 10.000 dollari; le sue spese legali superarono probabilmente i 100.000 dollari.

19.4.2 Virus

Un'altra forma d'attacco ai calcolatori è rappresentata dai **virus**. Come i worm, i virus sono progettati per diffondersi e seminare distruzione in un sistema, modificando o distruggendo file e causando crolli dei sistemi e il malfunzionamento dei programmi. Mentre un worm è strutturato come un programma completo, un virus è un frammento di codice che s'inserisce in un programma legittimo. I virus costituiscono un grave problema per gli utenti di calcolatori, specialmente per quelli di microcalcolatori. Generalmente i calcolatori multiutente non sono soggetti alla contaminazione da parte dei virus, poiché il sistema operativo protegge dalle scritture i programmi eseguibili. Anche se un virus riuscisse a infettare un programma, i suoi poteri resterebbero limitati poiché altri aspetti del sistema sono comunque protetti. I sistemi per singolo utente non hanno questo tipo di protezioni, perciò i virus sono liberi d'essere eseguiti.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Di solito i virus sono diffusi da utenti che prelevano programmi infetti dalle bacheche elettroniche pubbliche (BBS) o a causa dello scambio di dischi infetti. Nel febbraio 1992, ad esempio, due studenti della Cornell University crearono tre giochi per il sistema operativo Macintosh contenenti un virus e li distribuirono negli archivi internazionali attraverso la rete Internet. Il virus fu scoperto quando un professore di matematica del Galles, che aveva prelevato i giochi, fu avvertito dell'infezione da un programma antivirus. Altri 200 utenti avevano già prelevato i giochi. Pur non essendo stato progettato per distruggere dati, il virus si diffondeva nei file delle applicazioni causando ritardi nell'esecuzione e il malfunzionamento dei programmi. Gli autori furono facilmente rintracciati, poiché i giochi erano stati spediti per posta elettronica da un indirizzo della Cornell University. Le autorità dello stato di New York ordinarono l'arresto degli studenti per il reato di manomissione di calcolatori.

In un altro incidente, un programmatore californiano diede alla moglie, dalla quale stava divorziando, un dischetto da usare in un calcolatore conteso. Il disco conteneva un virus che cancellò tutti i file del sistema. Il marito fu arrestato e accusato di danneggiamenti.

Negli ultimi anni, una comune forma di trasmissione dei virus è lo scambio di file della serie di programmi Microsoft Office, ad esempio documenti prodotti col Microsoft Word. Questi documenti possono contenere le cosiddette *macro* (o programmi scritti in Visual Basic) che i programmi inclusi nel pacchetto Microsoft Office (Word, PowerPoint, Excel, Access) eseguono automaticamente. Poiché questi programmi sono eseguiti nell'ambito del profilo dell'utente, le macro sono eseguite in modo assai incontrollato (potrebbero ad esempio cancellare a loro piacimento i file dell'utente).

Occasionalmente, alcune infezioni virali imminenti vengono annunciate con un certo rilievo dai *mass media*. Questo fu il caso del virus *Michelangelo*, programmato per cancellare i file dei dischi infetti il 6 marzo 1992, il giorno del cinquecentodiciassettesimo anniversario della nascita dell'artista del Rinascimento. Data l'enorme pubblicità che lo circondò, il virus fu individuato e distrutto nella maggior parte dei siti prima della sua attivazione, perciò causò poco o nessun danno. Casi di questo tipo sensibilizzano l'opinione pubblica sul problema dei virus, tanto che i programmi antivirus si vendono ottimamente. La maggior parte dei prodotti commerciali è efficace solo contro virus specifici e noti, e opera cercando nei file di un sistema la specifica sequenza di istruzioni che caratterizza ciascun virus. Quando tali programmi trovano una sequenza nota, rimuovono le istruzioni che compongono il virus, *disinfettando* il programma. Questi prodotti commerciali contengono elenchi di centinaia di virus ai quali fanno riferimento durante la ricerca. I virus e i programmi antivirus diventano sempre più raffinati. Alcuni tipi di virus nell'infettare i programmi modificano se stessi in modo da evitare il suddetto metodo di base della ricerca delle sequenze di istruzioni note che caratterizzano ciascun virus, impiegato dai programmi antivirus. Questi a loro volta, per individuare i virus, cercano famiglie di sequenze di istruzioni anziché singole sequenze.

La migliore protezione contro i virus informatici è la prevenzione, o la pratica dell'**elaborazione sicura**. Acquistare programmi sigillati dai rivenditori autorizzati ed evitare copie illegali, provenienti da fonti pubbliche o da scambi di dischi, è il modo più sicuro per prevenire le infezioni. D'altra parte, anche i programmi legittimi non sono immuni dalle infezioni virali: ci sono stati casi di impiegati di società che, per causare danni economici, hanno infettato le copie matrici di programmi commerciali. Per quel che riguarda i virus di macro, una misura di prevenzione consiste nello scambiare i documenti prodotti col programma Microsoft Word in un formato di file detto RTF (*rich text format*); diversamente dal formato proprio di tale programma, l'RTF non consente l'inclusione delle macro.

Una misura di difesa consiste nell'evitare l'apertura di ogni allegato a un messaggio di posta elettronica proveniente da indirizzi sconosciuti. Sfortunatamente la storia ha dimostrato che i punti vulnerabili appaiono tanto rapidamente quanto vengono risolti. Nel 2000, ad esempio, il virus *love bug* si è ampiamente diffuso apparendo come un messaggio d'amore inviato da un amico del ricevente. Una volta attivato, aprendo l'allegato scritto in Visual Basic, si propagava inviando se stesso ai primi indirizzi presenti nella rubrica. Fortunatamente, eccetto l'intasamento dei sistemi di posta elettronica e delle cartelle della posta in arrivo degli utenti, era relativamente innocuo. D'altra parte il suo comportamento nega l'assoluta efficacia della misura difensiva che consiste nell'aprire soltanto gli allegati provenienti da utenti conosciuti. Un metodo di difesa più efficace consiste nell'evitare l'apertura di tutti gli allegati che contengono codice eseguibile. Alcune aziende impongono tale metodo come criterio di sicurezza generale, rimuovendo tutti gli allegati di questo tipo dai messaggi in arrivo.

Un'altra precauzione, pur non prevenendo le infezioni, ne consente un'individuazione precoce. Si devono riformattare i dischi del sistema; ciò è particolarmente utile per i settori d'avviamento, spesso soggetti ad attacchi da parte dei virus. Si caricano solamente programmi sicuri e si calcola una somma di controllo per ciascun file copiato. L'elenco delle somme di controllo si deve conservare in un luogo immune da accessi non autorizzati. A ogni avviamento del sistema, un programma ricalcolerà le somme di controllo e le confronterà con quelle dell'elenco originale; ogni differenza sarà un allarme per possibili infezioni.

Poiché di solito operano tra sistemi, e non tra programmi, processi o utenti, worm e virus generalmente pongono problemi di sicurezza piuttosto che di protezione.

19.4.3 Attacchi per rifiuto del servizio

L'ultima categoria di attacchi, quelli per **rifiuto del servizio** (*denial of service*), non prevede l'acquisizione d'informazioni o la sottrazione di risorse, ma piuttosto il blocco dell'utilizzo legittimo di un sistema o di un servizio. Ad esempio, un attacco malevolo con intrusione potrebbe eliminare tutti i file in un sistema; la maggior parte degli attacchi per rifiuto del servizio invece si riferiscono a sistemi che non sono stati violati con attacchi intrusivi. Infatti, scatenare un attacco che impedisce l'uso legittimo è di solito più semplice rispetto a penetrare in un sistema.

Questi attacchi generalmente si compiono tramite la rete e si dividono in due categorie. La prima è quella degli attacchi che impiegano talmente tante risorse del sistema attaccato da non lasciarne libere per svolgere elaborazioni utili. Ad esempio, la pressione di un pulsante in un sito Web potrebbe portare al caricamento di un'*applet* Java che quando è in esecuzione usa tutto il tempo di CPU disponibile.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

La seconda include gli attacchi che mirano a mettere in crisi la rete dell'organizzazione obiettivo dell'attacco. Sono stati portati con successo diversi attacchi ai maggiori siti Web mondiali. Questi attacchi sfruttano o meglio abusano di alcune funzioni fondamentali dei protocolli TCP/IP. Ad esempio, se l'aggressore invia la parte del protocollo che dice "Voglio stabilire una connessione TCP", ma a questa non fa mai seguire la parte che all'altro capo ci si aspetta e cioè "La connessione è ora completa", ci si trova con molte sessioni TCP parzialmente avviate che possono portare ad assorbire tutte le risorse di rete del sistema, impedendo qualsiasi altra connessione TCP legittima. Questi attacchi, che possono durare ore o anche giorni, hanno impedito del tutto o parzialmente l'uso dei sistemi di server attaccati. Per fermarli, si agisce di solito al livello della rete, fino a quando si possono aggiornare i sistemi operativi in modo da ridurre la loro vulnerabilità.

In generale, poiché si servono degli stessi meccanismi delle normali richieste di servizio, gli attacchi per rifiuto del servizio non si possono impedire. Spesso è difficile capire se un rallentamento del sistema è dovuto a un sovraccarico di richieste oppure a un attacco.

19.5 Migliorare la sicurezza dei sistemi

La possibilità di rendere sicuri i sistemi è strettamente correlata al rilevamento delle intrusioni (Paragrafo 19.6). Entrambe le tecniche devono operare insieme per assicurare che un sistema sia sicuro e che, se si verifica una violazione della sicurezza, questa sia individuata.

Un metodo per migliorare la sicurezza di un sistema consiste in una sua scansione periodica alla ricerca di varchi nella sicurezza. Queste scansioni si possono eseguire quando il sistema è relativamente poco utilizzato, in modo da avere minori effetti della registrazione (*logging*), e possono controllare vari aspetti del sistema:

- parole d'ordine brevi o facili da indovinare;
- programmi privilegiati non autorizzati, come i programmi di *setuid*;
- programmi non autorizzati nelle directory di sistema;
- processi dall'esecuzione inaspettatamente lunga;
- improprie protezioni delle directory sia degli utenti sia di sistema;
- improprie protezioni dei file di dati del sistema, come il file delle parole d'ordine, i driver dei dispositivi, o anche dello stesso nucleo del sistema operativo;
- elementi pericolosi nel percorso di ricerca dei programmi (ad esempio, cavalli di Troia — Paragrafo 19.3.1);
- modifiche ai programmi di sistema individuate con somme di controllo;
- demoni di rete inattesi o nascosti.

Ogni problema individuato dalla scansione di sicurezza può essere risolto automaticamente o riferito al gestore del sistema.

I calcolatori collegati in rete sono più soggetti ad attacchi contro la sicurezza di quanto lo siano i sistemi indipendenti. In questo caso gli attacchi arrivano da un insieme sconosciuto e vasto di punti d'accesso invece che da un insieme noto di punti d'accesso, come terminali direttamente connessi. Anche se in misura minore, anche i sistemi collegati tramite modem alle linee telefoniche sono più esposti dei sistemi indipendenti.

Il governo degli USA infatti considera i sistemi sicuri tanto quanto è sicuro il loro collegamento più esteso. Ad esempio, a un sistema di massima segretezza è possibile accedere solo dall'interno di un edificio considerato di massima segretezza. Il sistema perde il proprio grado di massima segretezza se dall'esterno di tale ambiente può avvenire un qualsiasi tipo di comunicazione. Alcuni servizi governativi prendono misure di sicurezza estreme; quando un terminale non è in uso i connettori in cui viene inserito per comunicare con un calcolatore sicuro vengono depositati in un luogo sicuro nell'ufficio. Per ottenere l'accesso al calcolatore è necessario conoscere la combinazione della serratura, così come le informazioni di autenticazione per il calcolatore stesso.

Sfortunatamente per gli amministratori di sistemi e per i professionisti della sicurezza dei calcolatori, è spesso impossibile confinare una macchina in una stanza e disabilitare ogni accesso remoto. La rete Internet, ad esempio, connette milioni di calcolatori e sta diventando una risorsa indispensabile alla realizzazione degli scopi di molte società e persone. Se si considera Internet come una comunità, allora come accade in ogni comunità formata di milioni di membri, alcuni di essi sono disonesti. Gli utenti disonesti di Internet dispongono di molti strumenti che consentono loro di tentare l'accesso ai calcolatori collegati, proprio come fece Robert Morris col proprio *Internet worm*.

Si pone quindi il problema di come i calcolatori fidati si possano collegare in modo sicuro a una rete non fidata. Una soluzione è data dall'uso di una **barriera di sicurezza** (*firewall*), che separa i sistemi fidati da quelli non fidati; si tratta di un calcolatore o di un instradatore situato tra queste due categorie di sistemi. Esso limita l'accesso di rete tra i due **domìni di sicurezza** e controlla e registra tutte le connessioni. Può anche limitare le connessioni secondo gli indirizzi di sorgente o di destinazione, le porte di sorgente o di destinazione, o la direzione delle connessioni. I Web server, ad esempio, impiegano il protocollo http per comunicare con i programmi di consultazione del Web; in tal caso la barriera di sicurezza dovrebbe consentire l'accesso col protocollo http da tutti i sistemi oltre la barriera di sicurezza soltanto verso il Web server, collocato all'interno della barriera di sicurezza ma, ad esempio, impedire l'accesso tramite il protocollo `finger`, usato dall'*Internet worm* di Morris per inserirsi nei calcolatori. Una barriera di sicurezza può così separare una rete in più domìni. Uno schema comune considera la rete Internet come dominio non fidato; prevede una rete parzialmente fidata, la cosiddetta **zona smilitarizzata** (*demilitarized zone* — DMZ), come secondo dominio; e un terzo dominio che comprende i calcolatori aziendali (Figura 19.2). Le connessioni sono consentite da Internet ai calcolatori della DMZ e dai calcolatori aziendali alla DMZ e alla rete Internet, ma non lo sono né da Internet né dalla DMZ ai calcolatori aziendali. In questo modo ogni accesso è controllato, e qualsiasi sistema della DMZ, anche se violato tramite un protocollo consentito dalla barriera di sicurezza, non può in ogni caso accedere ai calcolatori aziendali.

Ovviamente, lo stesso sistema che costituisce la barriera di sicurezza deve essere sicuro e resistente agli attacchi, altrimenti le sue capacità di fornire connessioni sicure pos-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

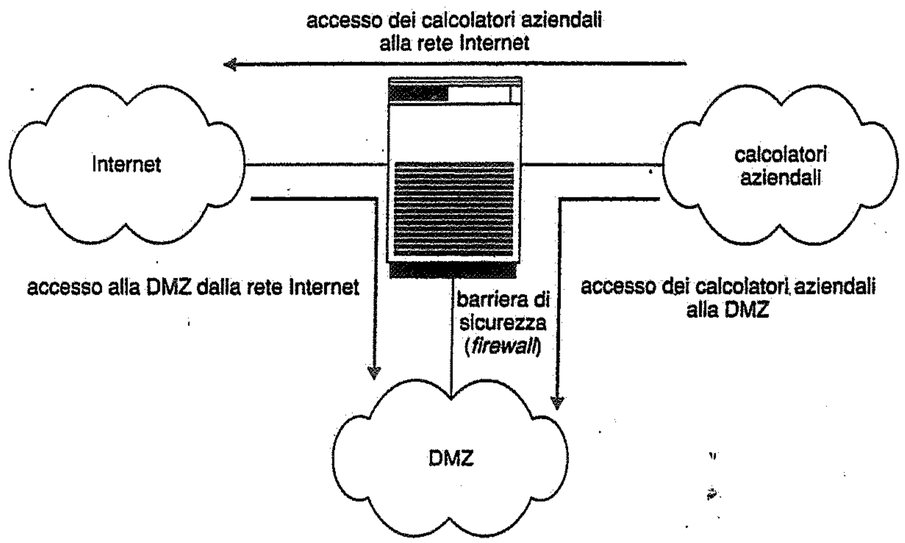


Figura 19.2 Sicurezza di rete con separazione in domìni tramite una barriera di sicurezza (*firewall*).

sono essere compromesse. Le barriere di sicurezza non riescono a prevenire gli attacchi che si aprono un varco o si trasmettono all'interno di protocolli e connessioni che le stesse barriere di sicurezza permettono. Un attacco per alterazione della pila diretto a un Web server non viene bloccato da una barriera di sicurezza perché le connessioni http sono permesse, ed è proprio il contenuto della connessione http che ospita l'attacco. Allo stesso modo gli attacchi per rifiuto del servizio possono colpire i calcolatori che costituiscono le barriere di sicurezza proprio come gli altri calcolatori. Un altro punto vulnerabile delle barriere di sicurezza è quello che si sfrutta con le tecniche di contraffazione delle identità (*spoofing*), tramite le quali un sistema non autorizzato finge di esserlo, riuscendo a soddisfare alcuni criteri di autorizzazione. Ad esempio, se una regola della barriera di sicurezza permette una connessione da uno specifico sistema e lo identifica secondo il suo indirizzo IP, un altro sistema potrebbe inviare pacchetti servendosi proprio di quell'indirizzo e quindi ottenere il permesso d'accesso.

19.6 Rilevamento delle intrusioni

Il **rilevamento delle intrusioni**, come suggerisce il nome, è quell'attività volta a scoprire tentativi di intrusione o intrusioni già avvenute nei sistemi di calcolo e ad attivare azioni appropriate in risposta alle intrusioni stesse. Per il **rilevamento delle intrusioni** s'impiegano molte tecniche che si differenziano secondo vari *fattori*. Alcuni di questi sono i seguenti:

- Fase in cui è avvenuto il rilevamento dell'intrusione: mentre si stava verificando, o solo successivamente.
- Le informazioni esaminate per scoprire l'attività intrusiva. Queste potrebbero comprendere comandi per l'interprete impartiti dall'utente, chiamate del sistema da parte dei processi, oltre a intestazioni e contenuto dei pacchetti di rete. Alcune forme d'intrusione si possono rilevare solo attraverso una correlazione delle informazioni acquisite da più di una di queste sorgenti.
- L'ampiezza della capacità di risposta. Alcune semplici forme di risposta consistono nell'informare l'amministratore del sistema della potenziale intrusione oppure nel bloccare in qualche modo la potenziale attività intrusiva, ad esempio arrestando un processo impegnato in un'attività apparentemente intrusiva. Con una forma di risposta più raffinata, un sistema potrebbe sviare, in modo trasparente, l'attività dell'intruso, portandolo verso una **trappola**, cioè una risorsa fittizia mostrata all'aggressore allo scopo di controllare l'attacco e ottenere maggiori informazioni; ovviamente all'aggressore la risorsa sembrerebbe reale.

Questi gradi di libertà nella progettazione di sistemi che impiegano tecniche per il rilevamento delle intrusioni hanno portato a un'ampia gamma di soluzioni che vanno sotto il nome di **sistemi di rilevamento delle intrusioni** (*intrusion-detection systems* — IDS).

19.6.1 Elementi che caratterizzano un'intrusione

È difficile definire con esattezza come s'identifica un'intrusione, quindi i sistemi IDS attuali di solito seguono uno dei due metodi meno ambiziosi. Secondo il primo di questi, chiamato **rilevamento basato sulle tracce**, si esaminano i dati d'ingresso al sistema o il traffico della rete alla ricerca di particolari schemi o sequenze di azioni, chiamati tracce (*signature*), che si ritengono indizi di attacchi. Un esempio banale di rilevamento basato su tracce è il controllo dei tentativi di autenticazione ripetuti, poiché questo è il tipico indizio di un utente che tenta d'indovinare una parola d'ordine per poter accedere al sistema. Un altro esempio semplice è la ricerca nella rete di pacchetti che contengono la sequenza `/etc/passwd/` indirizzati a un sistema UNIX. Un altro esempio ancora è offerto da un programma antivirus, che analizza i file binari alla ricerca di virus conosciuti.

Il secondo metodo di solito si chiama **rilevamento di anomalie**, e si riferisce alle tecniche che cercano d'identificare comportamenti anomali in un sistema. Ovviamente, non tutte le attività di sistema anomale indicano un'intrusione, ma l'ipotesi che si fa è che le intrusioni generino spesso un comportamento anomalo. Un esempio di rilevamento di anomalie è il controllo delle chiamate del sistema usate da un demone allo scopo di identificare se l'uso di queste chiamate differisce significativamente da quello usuale, indicando in questo caso la possibilità che il sistema sia stato oggetto di un attacco per alterazione della pila. Un altro esempio è il controllo dei comandi impartiti all'interprete per individuare comandi anomali da parte di un certo utente, oppure il riconoscimento di una procedura d'accesso che si svolge in un orario anomalo per un certo utente, che potrebbe indicare che un aggressore è riuscito a ottenere l'accesso a nome di quell'utente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il rilevamento basato su tracce e il rilevamento di anomalie si possono considerare come due facce della stessa medaglia: il rilevamento basato su tracce cerca di caratterizzare comportamenti pericolosi e li identifica quando questi si presentano nel sistema, mentre il rilevamento di anomalie cerca di caratterizzare il comportamento normale (o non pericoloso) del sistema e identifica qualsiasi comportamento che differisce da questo.

I differenti metodi tuttavia portano a IDS con caratteristiche molto diverse tra loro. In particolare, il rilevamento di anomalie può riuscire a scoprire metodi d'intrusione che in precedenza erano sconosciuti. Il rilevamento basato su tracce identifica invece solo gli attacchi conosciuti e che si possono codificare in uno specifico schema di azioni riconoscibili. Per questo motivo, gli attacchi che non erano stati considerati al momento della generazione delle tracce non potranno essere identificati. Il problema è ben noto alle aziende che commercializzano programmi antivirus, che sono costrette a rilasciare frequentemente gli aggiornamenti delle tracce a mano a mano che nuovi virus vengono creati e identificati con procedure manuali.

Il rilevamento di anomalie non è necessariamente superiore a quello basato su tracce. Infatti analizzare e caratterizzare accuratamente quello che è il comportamento 'normale' di un sistema è un problema considerevole per i sistemi che si basano sul rilevamento di anomalie. Se l'intrusione è già avvenuta quando il sistema viene analizzato, l'attività intrusiva potrebbe essere inclusa nel comportamento 'normale'. Anche se il sistema viene analizzato correttamente, senza l'influenza di comportamenti intrusivi, l'analisi deve comprendere un'immagine piuttosto completa del comportamento normale, altrimenti il numero di falsi allarmi sarebbe eccessivo.

Per illustrare l'impatto che può avere una frequenza di falsi allarmi anche solo moderatamente alta, si consideri un'installazione composta da alcune decine di stazioni di lavoro con sistema operativo UNIX dalle quali si registrano dati relativi a eventi legati alla sicurezza allo scopo di rilevare le intrusioni. Un'installazione di piccole dimensioni come questa può facilmente generare un milione di annotazioni di verifica al giorno. Tuttavia, solo una o due di queste potrebbero essere tali da meritare un'analisi più approfondita da parte dell'amministratore. Se si suppone, in modo ottimistico, che ogni attacco si rifletta in dieci annotazioni di verifica, si può calcolare con una certa approssimazione la frequenza di annotazioni di verifica che riflettono le vere attività intrusive per mezzo della formula

$$\frac{2\,\dfrac{intrusioni}{giorno}\cdot 10\,\dfrac{annotazioni}{intrusione}}{10^6\,\dfrac{annotazioni}{giorno}} = 0{,}00002$$

Interpretando il valore che si ottiene come una 'probabilità di occorrenza di annotazioni d'intrusione', la si denota con $P(I)$; cioè, l'evento I rappresenta l'occorrenza di un'annotazione che riflette un reale comportamento intrusivo. Poiché $P(I) = 0{,}00002$, si trova che $P(\neg I) = 1 - P(I) = 0{,}99998$. Si supponga ora che A denoti l'evento dell'IDS che

attiva un allarme. Un IDS accurato dovrebbe massimizzare sia $P(I|A)$ sia $P(\neg I|\neg A)$, cioè, sia la probabilità che un allarme indichi un'intrusione sia la probabilità che l'assenza di un allarme indichi che non vi sono intrusioni. Considerando per il momento $P(I|A)$, si può calcolare tale valore usando la **formula di Bayes:**

$$P(I|A) = \frac{P(I)\cdot P(A|I)}{P(I)\cdot P(A|I) + P(\neg I)\cdot P(A|\neg I)} = \frac{0{,}00002\cdot P(A|I)}{0{,}00002\cdot P(A|I) + 0{,}99998\cdot P(A|\neg I)}$$

Si consideri ora l'impatto della frequenza di falsi allarmi $P(A|\neg I)$ su $P(I|A)$. Anche nel caso di una frequenza di veri allarmi molto buona, con $P(A|I) = 0{,}8$, da una frequenza di falsi allarmi piuttosto buona, $P(A|\neg I) = 0{,}0001$, si ottiene $P(I|A) \approx 0{,}14$. Questo significa che meno di uno su sette allarmi indica una vera intrusione. Nei sistemi in cui un amministratore della sicurezza investiga su ogni allarme, questa frequenza di falsi allarmi porterebbe a un enorme spreco di tempo e convincerebbe molto presto l'amministratore a ignorare tutti gli allarmi.

Quest'esempio mostra un principio generale per i sistemi IDS: per essere utilizzabile proficuamente, un IDS deve offrire una frequenza di falsi allarmi estremamente bassa. Per i sistemi di rilevamento di anomalie, il raggiungimento di una frequenza di falsi allarmi sufficientemente bassa è un serio problema, proprio per le difficoltà che si hanno nell'analizzare e caratterizzare adeguatamente il comportamento normale dei sistemi. Tuttavia, la ricerca nel campo del rilevamento di anomalie procede positivamente.

19.6.2 Verifica e registrazione

Un metodo comune per il rilevamento delle intrusioni è l'**elaborazione dei tracciati di verifica** (*audit-trail processing*), in cui gli eventi rilevanti per la sicurezza si memorizzano in file di sistema formando tracciati di verifica che poi si confrontano con tracce d'attacchi (nel rilevamento basato su tracce), oppure si analizzano per scoprire comportamenti anomali (nel rilevamento di anomalie). Nei sistemi UNIX, ci sono due semplici programmi, `syslog` e `swatch`, che si possono usare per creare e analizzare tracciati di verifica e generare risposte. Il programma `syslog` crea tracciati di verifica e fornisce una funzione d'invio di messaggi rilevanti per la sicurezza. Il programma `swatch` applica semplici tecniche di rilevamento basato su tracce ai tracciati di verifica generati da `syslog` mentre questi vengono creati e produce opportuni risultati nel caso in cui si rilevino intrusioni.

Quando `syslog` è installato in un sistema UNIX, viene creato all'avvio del sistema un processo demone chiamato `syslogd`. Questo processo attende messaggi da varie possibili sorgenti e li dispone com'è prescritto nel file di configurazione `syslog.conf` (che di solito si trova nella directory `/etc/`). Il processo `syslogd` si può configurare in modo che accetti messaggi da molte sorgenti diverse, che includono in modo predefinito il driver di registrazione `/dev/log`, ma anche da altre sorgenti, incluse quelle di altri calcolatori raggiungibili via rete. Il file `syslog.conf` contiene un insieme di coppie, ognuna con un campo selettore e un campo di azione. Il campo selettore identifica i messaggi sui quali si deve compiere l'azione. I tipi d'azione che si possono compiere sui messaggi

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

comprendono l'invio del messaggio a un file, a una lista di utenti, ai processi demoni `syslogd` in altri calcolatori, oppure a un programma attraverso una pipe del sistema UNIX. Come tale, `syslogd` offre una funzione centralizzata attraverso la quale si possono gestire i vari messaggi relativi alla sicurezza.

Il programma `swatch`, al momento della scrittura di questo testo, è disponibile nel Web all'indirizzo `http://www.stanford.edu/~atkins/swatch/`. Questo programma elabora i messaggi inviatigli dal demone `syslogd` (o da altri programmi di registrazione) e intraprende determinate attività quando riconosce specifici schemi. I messaggi non devono necessariamente raggiungere `swatch` direttamente, ma possono essere scritti su file e successivamente letti e analizzati dal programma `swatch`. Questo programma confronta ogni messaggio di verifica con le espressioni regolari memorizzate in un file di configurazione. Il file memorizza, insieme con ogni espressione regolare, anche le azioni che si devono intraprendere se il confronto ha esito positivo. Le azioni permesse comprendono l'emissione di una riga o di un segnale sonoro al terminale di controllo del programma `swatch`, l'invio di un messaggio a una lista di utenti, per posta elettronica o con il comando `write`, oppure l'esecuzione di un programma con valori presi dal messaggio di verifica come argomenti.

Ogni espressione regolare contenuta nel file di configurazione del programma `swatch` si può considerare una traccia primitiva, utilizzabile in un rilevamento d'intrusioni basato su tracce. Ad esempio, una traccia utile potrebbe essere rappresentata da tutte le sequenze di caratteri che contengono le parole "accesso negato" e determinare un'azione corrispondente che manda un segnale sonoro al terminale di controllo del programma `swatch`. Tuttavia, le tracce gestite da `swatch` sono piuttosto rudimentali, poiché si applicano a un solo messaggio di verifica alla volta. Questo metodo non riesce a trattare le tracce di attacchi che comprendono molte azioni che si rifletterebbero in molteplici messaggi di verifica, nessuno dei quali singolarmente potrebbe indicare l'attacco. Alcuni tra gli strumenti d'elaborazione dei tracciati di verifica disponibili in commercio offrono capacità d'elaborazione più raffinate.

19.6.3 Tripwire

Un esempio di semplice strumento per il rilevamento di anomalie è il **Tripwire**, uno strumento per il controllo dell'integrità del file system del sistema operativo UNIX, sviluppato alla Purdue University. Il Tripwire opera seguendo l'ipotesi che una vasta classe d'intrusioni generi modifiche anomale nei file e nelle directory di sistema. Ad esempio, un aggressore potrebbe modificare il file `/etc/passwd` di un sistema per permettere in futuro a un intruso di accedere facilmente al sistema. Un intruso potrebbe modificare alcuni programmi di sistema, magari inserendo delle copie con cavalli di Troia, o inserendo nuovi programmi nelle directory che si trovano di solito nei percorsi di ricerca degli interpreti di comandi degli utenti. Oppure, un intruso potrebbe rimuovere file di registrazione (*log*) del sistema per nascondere le proprie tracce. Il Tripwire è uno strumento che serve a tenere sotto controllo il file system rispetto alle operazioni di aggiunta, cancellazione e modifica dei file e per avvertire gli amministratori di queste modifiche.

Il comportamento del Tripwire è controllato da un file di configurazione, `tw.config`, che elenca le directory e i file per i quali si vogliono tenere sotto controllo le modifiche, le cancellazioni o le aggiunte. Ogni elemento di questo file di configurazione comprende una maschera di selezione che specifica quali attributi del file (attributi di *inode*) si devono controllare. Ad esempio, la maschera di selezione potrebbe specificare che si rilevano i cambiamenti relativi al contenuto di un file, o in modo più specifico, al risultato dell'applicazione di una funzione di hash predefinita al contenuto del file. Il Tripwire a questo scopo offre diverse funzioni di hash resistenti alle collisioni, dove una funzione di hash si dice resistente alle collisioni se, dato il valore $f(x)$ che si ottiene applicando f a un file x, è praticamente impossibile, dal punto di vista computazionale, il calcolo di un diverso file x' tale che $f(x') = f(x)$. Quindi, tenere sotto controllo i cambiamenti del valore di hash di un file equivale a tenere sotto controllo il file stesso, ma la memorizzazione dei valori di hash richiede assai meno spazio della copiatura dei file veri e propri.

La prima volta che si attiva, il Tripwire prende in ingresso il file `tw.config` e calcola una traccia per ciascun file o directory, contenente gli attributi che si devono controllare (attributi di *inode* e valori di hash). Queste tracce si memorizzano in una base di dati. Le volte successive, il Tripwire considera come ingresso sia `tw.config` sia la base di dati precedentemente memorizzata, ricalcola la traccia per ciascun file o directory elencato in `tw.config` e confronta questa traccia con quella (se c'è) memorizzata nella base di dati. Gli eventi che si segnalano all'amministratore comprendono una diversa traccia di un file o directory rispetto a quella memorizzata precedentemente nella base di dati (un file modificato), ogni file o directory per i quali non esisteva una traccia nella base di dati (un file aggiunto) e qualsiasi traccia nella base di dati per la quale non esiste più un file o una directory (un file eliminato).

Sebbene funzioni egregiamente per un'ampia classe di attacchi, il Tripwire ha alcuni limiti. Forse il più ovvio è la necessità di proteggere da modifiche non autorizzate i file di programma dello stesso Tripwire e i file di dati associati, inclusa la base di dati. Per questa ragione, il Tripwire e i file associati, dovrebbero risiedere in qualche mezzo a prova d'attacco, come un disco protetto dalle scritture oppure un server sicuro dove le connessioni al sistema sono controllate rigorosamente. Sfortunatamente questo rende più scomodo l'aggiornamento della base di dati dopo modifiche autorizzate alle directory e ai file tenuti sotto controllo. Una seconda limitazione è dovuta al fatto che alcuni file rilevanti per la sicurezza, ad esempio i file di registrazione di sistema, *devono* cambiare nel tempo e il Tripwire non fornisce un modo per distinguere tra modifiche autorizzate e non autorizzate. Quindi, ad esempio, un attacco che modifica (senza cancellarlo) un file di registrazione di sistema, che sarebbe stato comunque modificato dalla normale attività di sistema, sfuggirebbe alla capacità di rilevamento del Tripwire. Il meglio che potrebbe fare in questo caso è il rilevamento di determinate banali incoerenze (ad esempio, se il file di registrazione si riduce di dimensioni). Di questo strumento esistono sia versioni commerciali sia libere.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.6.4 Controllo delle chiamate del sistema

Il controllo delle chiamate del sistema è una forma più raffinata e recente di rilevamento di anomalie. Con questo metodo si controllano le chiamate del sistema per rilevare in tempo reale se un processo mostra un comportamento difforme da quello atteso. Si sfrutta il fatto che un programma definisce implicitamente le sequenze di chiamate del sistema che può eseguire, secondo i possibili cammini d'esecuzione all'interno del programma stesso. Per programmi complessi e di grandi dimensioni, l'insieme di possibili sequenze di chiamate del sistema è enorme e di non facile individuazione e a maggior ragione non può essere memorizzato esplicitamente. Ciononostante, questo metodo cerca di caratterizzare il comportamento 'usuale', secondo le chiamate del sistema, in una forma abbreviata, in modo che le tracce delle chiamate del sistema osservate si possano confrontare con questa caratterizzazione per rilevare comportamenti anomali. Un tipo d'intrusione che si può rilevare in questo modo sono gli attacchi che s'impossessano di un processo e che quindi fanno in modo che questo esegua il codice dell'aggressore anziché il codice del programma originale; ad esempio sfruttando una vulnerabilità, dovuta alla mancanza di controllo della quantità di dati immessi in un vettore che riceve dati in ingresso nel programma del processo, che consente un attacco per alterazione della pila (Paragrafo 19.3.3). Se il codice dell'aggressore ha una sequenza di chiamate del sistema anomala rispetto a quella del programma originale, allora il sistema di rilevamento delle intrusioni dovrebbe essere capace di individuarlo e di intraprendere le opportune azioni (ad esempio, terminare il processo).

Un esempio di questo metodo, per i processi del sistema operativo UNIX, è stato inizialmente proposto alla University of New Mexico. In questo metodo la sequenza usuale di chiamate del sistema per un certo programma viene generata eseguendo il programma su diversi dati d'ingresso, provenienti sia da utenti reali sia progettati per indurre un insieme di comportamenti da parte del programma. Le sequenze di chiamate del sistema generate in questo modo vengono poi memorizzate (ignorando i parametri). Ad esempio, una di queste sequenze potrebbe essere:

```
open, read, mmap, mmap, open, getrlimit, mmap, close.
```

Questa sequenza è successivamente usata per riempire una tabella che indica, per ogni chiamata del sistema, quali chiamate del sistema la possono seguire a distanza uno, due, ecc., fino a distanza k. Ad esempio, si supponga di scegliere $k = 3$. Esaminando la prima open, si nota che read segue a distanza uno e mmap segue a distanze due e tre. In modo analogo, mmap segue read a distanze uno e due, mentre open segue a distanza tre. Facendo quest'esercizio per ogni chiamata del sistema, si ottiene la tabella riportata nella Figura 19.3.

Questa tabella viene memorizzata e durante le esecuzioni successive del programma, la sequenza di chiamate del sistema viene confrontata con la tabella per rilevare eventuali discrepanze. Ad esempio, se si osserva la sequenza

```
open, read, mmap, open, open, getrlimit, mmap, close
```


Chiamata di sistema	distanza = 1	distanza = 2	distanza = 3
open	read getrlimit	mmap	mmap close
read	mmap	mmap	open
mmap	mmap open close	open getrlimit	getrlimit mmap
getrlimit	mmap	close	
close			

Figura 19.3 Struttura di dati derivata dalla sequenza di chiamate del sistema.

(cioè, una chiamata di mmap è sostituita da una chiamata di open), si notano le seguenti differenze: open segue open a una distanza di tre; open segue read a una distanza di due; open segue open a una distanza di uno; e getrlimit segue open a una distanza di due. Il fatto che il sistema IDS determini o no che questo comportamento sia dovuto a un'intrusione potrebbe dipendere dal numero di discrepanze osservate (magari come frazione del numero totale delle possibili discrepanze) o dalle specifiche chiamate del sistema coinvolte.

Questo metodo e altri simili sono stati applicati principalmente a processi di root, come sendmail, cercando di rilevare intrusioni. I processi di root sono tra i candidati ideali per questa tecnica, poiché di solito hanno un insieme limitato di comportamenti e non cambiano spesso. Inoltre, sono spesso obiettivi di attacchi per il loro stato privilegiato e perché, nel caso dei servizi di rete che accettano comunicazioni dalla rete stessa, gli aggressori ne possono verificare la presenza e altre caratteristiche in modo remoto.

Sebbene le tecniche di controllo delle chiamate del sistema possano essere utili, un aggressore potrebbe costruire un attacco che non modifichi la sequenza delle chiamate del sistema tanto da mettere in allarme il sistema IDS. Ad esempio, l'aggressore potrebbe avere una copia del programma che intende attaccare e potrebbe tentare varie possibilità d'intrusione fino a quando non ne trova una che lasci sostanzialmente intatta la sequenza di chiamate del sistema. È possibile comunque ottenere un ulteriore livello di sicurezza estendendo queste tecniche di rilevamento in modo che prendano in considerazione anche i parametri delle chiamate del sistema.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.7 Crittografia

In un calcolatore isolato il sistema operativo può determinare con sicurezza chi è il mittente e chi il destinatario di qualsiasi comunicazione tra processi, poiché esso controlla tutti i canali di comunicazione all'interno del calcolatore. In una rete di calcolatori la situazione è diversa. Un calcolatore in una rete riceve bit dai *cavi*, senza avere alcun modo immediato e affidabile per determinare quale calcolatore o applicazione abbia inviato quei bit. In modo analogo, un calcolatore invia bit nella rete senza sapere con certezza chi effettivamente li riceverà.

Di solito, per risalire ai potenziali mittenti e destinatari dei messaggi si usano gli indirizzi di rete. I pacchetti di rete arrivano con un indirizzo di sorgente, come un indirizzo IP, e quando un calcolatore invia un messaggio nomina esplicitamente il destinatario specificandolo nell'indirizzo di destinazione. Tuttavia, per le applicazioni per le quali la sicurezza è importante, non è possibile fidarsi del fatto che gli indirizzi di sorgente e destinazione di un pacchetto identifichino con certezza chi lo ha inviato o chi lo riceve. Un calcolatore dal quale si sta compiendo un attacco potrebbe inviare un messaggio con un indirizzo sorgente falsificato, e molti altri calcolatori, oltre a quello specificato nell'indirizzo di destinazione, potrebbero ricevere il pacchetto (come di solito effettivamente avviene). Ad esempio, tutti gli instradatori nel cammino tra sorgente e destinazione *riceveranno* il pacchetto. Dunque, il sistema operativo non può decidere se concedere l'accesso per la scrittura a un file se non può considerare fidata l'identità del richiedente. E non può imporre la protezione per la lettura a un file se non è in grado di determinare con esattezza chi riceverà i contenuti di un file che lui invia nella rete.

In generale, si considera impossibile costruire una rete di qualsiasi dimensione in cui gli indirizzi di sorgente e di destinazione dei pacchetti si possano considerare *fidati* in questo senso. Quindi, l'unica possibilità è cercare di eliminare in qualche modo la necessità di considerare come fidata la rete. Per questo è essenziale il ruolo della crittografia. Da un punto di vista astratto, la **crittografia** si usa per imporre i potenziali mittenti e destinatari di un messaggio. La crittografia moderna si fonda su codici segreti, chiamati **chiavi**, che si distribuiscono selettivamente ai calcolatori di una rete e si usano per gestire i messaggi. La crittografia permette al destinatario di un messaggio di verificare che il messaggio sia stato creato da un calcolatore che possiede una certa chiave (la chiave è la *sorgente* del messaggio). In modo analogo, un processo mittente può codificare il suo messaggio in modo tale che solo un calcolatore in possesso di una certa chiave possa decifrare il messaggio, divenendo così la *destinazione* del messaggio stesso. Tuttavia, a differenza degli indirizzi di rete, le chiavi sono progettate in modo che sia computazionalmente impossibile derivarle a partire dai messaggi generati tramite esse e da qualsiasi altra informazione pubblica. Dunque, questo meccanismo costituisce un mezzo molto più fidato per imporre i mittenti e i destinatari dei messaggi.

19.7.1 Autenticazione

Imporre l'insieme dei potenziali mittenti di un messaggio è un processo che si chiama anche autenticazione. Un algoritmo di autenticazione permette al ricevente di un messaggio di verificare che questo sia stato creato da un calcolatore che possiede una certa chiave. Più precisamente, un algoritmo di autenticazione comprende i seguenti componenti:

- Un insieme K di chiavi.
- Un insieme M di messaggi.
- Un insieme A di autenticatori.
- Una funzione $S: K \rightarrow (M \rightarrow A)$. Cioè, per ogni $k \in K$, $S(k)$ è una funzione che genera autenticatori a partire da messaggi. Sia S sia $S(k)$ devono essere funzioni calcolabili in modo efficiente per ogni k.
- Una funzione $V: K \rightarrow (M \times A \rightarrow \{\texttt{true}, \texttt{false}\})$. Cioè, per ogni $k \in K$, $V(k)$ è una funzione che verifica gli autenticatori sui messaggi. Sia V sia $V(k)$ devono essere funzioni calcolabili in modo efficiente per ogni k.

La proprietà fondamentale che un algoritmo di autenticazione deve possedere è la seguente: dato un messaggio m, un calcolatore può generare un autenticatore $a \in A$ tale che $V(k)(m, a) = \texttt{true}$ soltanto se esso possiede $S(k)$. Quindi, un calcolatore che possiede $S(k)$ può generare autenticatori su messaggi in modo che tutti gli altri calcolatori che possiedono $V(k)$ li possano verificare. Tuttavia, un calcolatore che non ha $S(k)$ non può generare autenticatori su messaggi che possono essere verificati tramite $V(k)$. Poiché gli autenticatori sono generalmente visibili (ad esempio, s'inviano nella rete con i messaggi stessi), $S(k)$ non si deve poter derivare a partire dagli autenticatori.

Gli algoritmi di autenticazione si dividono in due tipi principali. In un **codice di autenticazione di messaggi** (*message authentication code* — MAC), la conoscenza di $V(k)$ o di $S(k)$ è equivalente: uno si può derivare dall'altro. Per questa ragione è importante proteggere $V(k)$ tanto quanto $S(k)$. Un semplice esempio di MAC definisce $S(k)(m) = f(k, m)$ dove f è una funzione non invertibile sul suo primo argomento (cioè, il primo argomento non si può derivare dal valore della funzione) e resistente alle collisioni sul suo secondo argomento (cioè, è praticamente impossibile trovare un $m' \neq m$ tale che $f(k, m) = f(k, m')$). Un appropriato algoritmo di verifica è dato da $V(k)(m, a) = (f(k, m) = a)$. Si noti che k è necessario per calcolare sia $S(k)$ sia $V(k)$; cioè, chiunque può calcolare uno, può calcolare l'altro.

Il secondo tipo di algoritmo di autenticazione è un algoritmo basato sulla **firma digitale** e quindi gli autenticatori prodotti in questo caso sono chiamati *firme digitali*. In un algoritmo basato sulla firma digitale è praticamente impossibile dal punto di vista computazionale derivare $S(k)$ da $V(k)$ e, in particolare, V è una funzione non invertibile. Dunque, non è necessario che $V(k)$ sia tenuta segreta e può essere liberamente distribuita. Per questa ragione, $V(k)$ si chiama **chiave pubblica**; viceversa, $S(k)$ (o semplicemente k) si

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

chiama chiave privata. Nel seguito si descrive un algoritmo basato sulla firma digitale, noto come RSA, dalle iniziali dei cognomi dei suoi inventori (Ronald Rivest, Adi Shamir e Leonard Adleman). Nello schema RSA la chiave k è una coppia $\langle d, N\rangle$, dove N è il prodotto di due grandi numeri primi p e q scelti casualmente (ad esempio, p e q potrebbero essere rappresentati da 512 bit ciascuno). L'algoritmo di firma è $S(\langle d, N\rangle)(m) = f(m)^d \bmod N$, dove f è una funzione resistente alle collisioni. L'algoritmo di verifica è dunque $V(\langle d, N\rangle)(m, a) \equiv (a^e \bmod N = f(m))$ dove e soddisfa $ed \bmod (p-1)(q-1) = 1$. Si considera praticamente impossibile, dal punto di vista computazionale, per un calcolatore che possiede $V(\langle d, N\rangle)$ (cioè, che possiede $\langle e, N\rangle$ ma non d) generare $S(\langle d, N\rangle)$ (cioè, d).

19.7.2 Cifratura

La cifratura è un modo per vincolare i possibili destinatari di un messaggio. Dunque è complementare all'autenticazione, e per sottolineare questo punto si cercherà di darne un'illustrazione parallela a quella dell'autenticazione. Un algoritmo di cifratura permette al mittente di un messaggio di imporre che solo un calcolatore che possiede una certa chiave possa leggere il messaggio. Più precisamente, un algoritmo di cifratura comprende i seguenti componenti:

- Un insieme K di chiavi.
- Un insieme M di messaggi.
- Un insieme C di testi cifrati.
- Una funzione $E : K \rightarrow (M \rightarrow C)$. Cioè, per ogni $k \in K$, $E(k)$ è una funzione che genera testi cifrati a partire dai messaggi. Sia E sia $E(k)$ devono essere funzioni calcolabili in modo efficiente per ogni k.
- Una funzione $D : K \rightarrow (C \rightarrow M)$. Cioè, per ogni $k \in K$, $D(k)$ è una funzione che genera messaggi a partire dai testi cifrati. Sia D sia $D(k)$ devono essere funzioni calcolabili in modo efficiente per ogni k.

La proprietà fondamentale che un algoritmo di autenticazione deve possedere è la seguente: dato un testo cifrato $c \in C$, un calcolatore può calcolare m in modo che $E(k)(m) = c$ solo se possiede $D(k)$. Quindi, un calcolatore che ha $D(k)$ può decifrare i testi cifrati ottenendo i testi in chiaro usati per produrli. Tuttavia, un calcolatore che non conosce $D(k)$ non può decifrarli. Poiché i testi cifrati sono di solito visibili (ad esempio, s'inviano in una rete), è essenziale che sia impossibile derivare $D(k)$ dai testi cifrati.

Proprio come ci sono due tipi principali di algoritmi di autenticazione, ci sono due tipi principali di algoritmi di cifratura. Nel primo tipo, chiamato **algoritmo di cifratura simmetrica**, $D(k)$ si può derivare da $E(k)$ e viceversa. Quindi, la segretezza di $D(k)$ deve essere protetta tanto quanto quella di $E(k)$. Durante gli ultimi 20 anni, l'algoritmo di cifratura simmetrica più usato negli USA per applicazioni civili è stato il DES (*data encryption standard*), adottato dal National Institute of Standards and Technology (NIST). Tuttavia, il DES non è considerato sicuro per molte applicazioni, poiché le chiavi impiegate, di 56 bit di lunghezza, si possono sottoporre a una ricerca esaustiva usando risorse di calcolo moderatamente potenti. Il NIST sta dunque procedendo all'adozione di un nuovo algoritmo di cifratura, chiamato AES (*advanced encryption standard*), che sostituirà il DES. La scelta dello specifico algoritmo AES è attualmente in corso.

In un algoritmo di cifratura asimmetrica, risulta computazionalmente impossibile derivare $D(k)$ da $E(k)$ e quindi $E(k)$ non ha ragione d'essere tenuto segreto e si può distribuire senza limiti; $E(k)$ è la chiave pubblica e $D(k)$ (o semplicemente k) è la chiave privata. È interessante notare che il meccanismo sottostante l'algoritmo di firma RSA si può usare anche per ottenere un algoritmo di cifratura asimmetrica. Di nuovo, la chiave k è una coppia $\langle d, N\rangle$ dove N è il prodotto di due grandi numeri primi p e q scelti casualmente. L'algoritmo di cifratura è dato da $E(\langle d, N\rangle)(m) = m^e \bmod N$ dove e, d ed N sono definiti come in precedenza. L'algoritmo di decifrazione è dunque dato da $D(\langle d, N\rangle)(c) = c^d \bmod N$.

19.7.3 Un esempio: SSL

L'SSL 3.0 è un protocollo crittografico che permette a due calcolatori di comunicare in modo sicuro, cioè, in modo che ognuno dei due possa determinare il mittente e il destinatario dei messaggi diretti all'altro. Si tratta forse del protocollo crittografico oggi più comunemente usato nella rete Internet, poiché è il protocollo standard per mezzo del quale i programmi di consultazione del Web possono comunicare in modo sicuro con i Web server. L'SSL è un protocollo complesso con molte opzioni e in questo testo si presenta una sola delle sue varianti, e solo in una forma astratta e molto semplificata, in modo tale da considerare principalmente l'uso delle sue primitive crittografiche.

Il protocollo SSL è avviato da un *client* allo scopo di comunicare in modo sicuro con un *server*. Prima dell'avvio del protocollo, s'ipotizza che il server s abbia ottenuto un certificato, denotato con cert_s, da una terza entità chiamata **autorità di certificazione** (*certification authority* → CA). Questo certificato è una struttura di dati che contiene i seguenti elementi:

- vari attributi attrs del server, come il *nome unico che lo contraddistingue* e il suo *nome comune* (DNS);
- un algoritmo di cifratura pubblico $E(k_s)$ per il server;
- un intervallo di validità interval durante il quale il certificato si può considerare valido;
- una firma digitale a sulle informazioni di cui sopra da parte della CA, cioè, $a = S(k_{CA})(\langle \text{attrs}, E(k_s), \text{interval}\rangle)$.

Prima dell'attivazione del protocollo, si presume che il client abbia ottenuto l'algoritmo pubblico di verifica $V(k_{CA})$ per la specifica CA. Nel caso del Web, il programma di consultazione impiegato dall'utente viene consegnato dal relativo produttore insieme con gli algoritmi di verifica di alcune autorità di certificazione. L'utente può aggiungere o cancellare a proprio piacimento gli algoritmi di verifica per le autorità di certificazione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Quando un client c si connette al server s, gli invia un valore casuale n_c di 28 byte. Il server s risponde inviando al client un proprio valore casuale n_s oltre al suo certificato cert_s. Il client verifica che $V(k_{CA})(\langle \text{attrs}, E(k_s), \text{interval}\rangle, a) = \text{true}$ e che la data corrente sia compresa nell'intervallo di validità interval del certificato. Se entrambe queste verifiche sono soddisfatte, il client genera un valore casuale di 46 byte, detto *premaster secret* e denotato da pms, e invia al server il valore $\text{cpms} = E(k_s)(\text{pms})$: Il server ricostruisce $\text{pms} = D(k_s)(\text{cpms})$. A questo punto sia il client sia il server sono in possesso di n_c, n_s e pms, e possono autonomamente calcolare un valore condiviso di 48 byte, detto *master secret* e denotato da ms, con $\text{ms} = f(n_c, n_s, \text{pms})$, dove f è una funzione non invertibile e resistente alle collisioni. Soltanto il server e il client possono calcolare ms, poiché soltanto essi conoscono pms. Inoltre, la dipendenza di ms da n_c e n_s assicura che ms sia un valore *nuovo*, cioè che non sia stato usato in una precedente esecuzione del protocollo. A questo punto sia il client sia il server calcolano le seguenti chiavi a partire dal *master secret* ms:

- una chiave di cifratura simmetrica k_{cs}^{crypt} per cifrare i messaggi dal client al server;
- una chiave di cifratura simmetrica k_{sc}^{crypt} per cifrare i messaggi dal server al client;
- una chiave di generazione di MAC k_{cs}^{mac} per generare autenticatori sui messaggi dal client al server;
- una chiave di generazione di MAC k_{sc}^{mac} per generare autenticatori sui messaggi dal server al client.

Per inviare un messaggio m al server, il client invia

$$c = E(k_{cs}^{\text{crypt}})(\langle m, S(k_{cs}^{\text{mac}})(m)\rangle).$$

Al ricevimento di c, il server ricostruisce

$$\langle m, a\rangle = D(k_{cs}^{\text{crypt}})(c)$$

e accetta m se $V(k_{cs}^{\text{mac}})(m, a) = \text{true}$. Analogamente, per inviare un messaggio m al client, il server invia

$$c = E(k_{sc}^{\text{crypt}})(\langle m, S(k_{sc}^{\text{mac}})(m)\rangle).$$

e il client ricostruisce

$$\langle m, a\rangle = D(k_{sc}^{\text{crypt}})(c)$$

e accetta m se $V(k_{sc}^{\text{mac}})(m, a) = \text{true}$.

Questo protocollo permette al server di limitare i riceventi dei suoi messaggi al client che ha generato pms, e i mittenti dei messaggi che esso accetta a quello stesso client. Analogamente, il client può limitare i destinatari dei messaggi che invia e il mittente dei messaggi che accetta alla parte che conosce $S(k_s)$ (cioè, la parte capace di decifrare pms). In molte applicazioni, come le transazioni che si svolgono nel Web, il client deve verificare l'identità della parte che conosce $S(k_s)$. Questo è uno degli scopi del certificato cert_s; in particolare, il campo attrs contiene informazioni che il client può usare per determinare l'identità (ad esempio, il nome del dominio) del server con il quale sta comunicando. Per applicazioni nelle quali anche il server deve avere informazioni sul client, l'SSL ha un'opzione tramite la quale un client può inviare un certificato al server.

19.7.4 Uso della crittografia

I protocolli di rete sono di solito organizzati in strati, in cui ciascuno strato agisce come un client rispetto allo strato sottostante. Cioè, quando un protocollo genera un messaggio per il protocollo analogo nel calcolatore di destinazione, consegna il suo messaggio al protocollo sottostante nella pila dei protocolli di rete in modo che questo lo consegni al rispettivo protocollo in quella macchina. Ad esempio, in una rete IP, il TCP (un protocollo dello *strato di trasporto*) agisce come un client dell'IP (un protocollo dello *strato di rete*): i pacchetti del TCP sono passati all'IP, lo strato sottostante, che in modo analogo li passa allo *strato di collegamento dei dati* sottostante affinché siano trasmessi attraverso la rete al corrispondente livello IP nel calcolatore di destinazione. Questo livello IP a sua volta consegnerà i pacchetti TCP al livello TCP su quella macchina. Complessivamente, il modello di riferimento OSI, che è stato adottato quasi universalmente per le reti di trasmissione di dati, definisce sette livelli di protocolli. Qualsiasi buon libro sulle reti presenta questi argomenti in modo dettagliato.

La crittografia si può includere quasi in ogni livello di questo modello. Il protocollo SSL (Paragrafo 19.7.3) fornisce sicurezza al livello di trasporto. La sicurezza al livello di rete, che è stata standardizzata (IPSec), definisce formati dei pacchetti IP che permettono l'inserimento di autenticatori e la cifratura del contenuto dei pacchetti. L'IPSec si sta diffondendo come base per la realizzazione di **reti private virtuali** (*virtual private network* — VPN). Sono stati anche sviluppati numerosi protocolli che permettono alle applicazioni di utilizzarlo.

In generale, non c'è una risposta definitiva alla domanda su quale sia lo strato più adatto in una pila di protocolli per collocare la protezione crittografica. Da una parte, se si colloca la protezione negli strati bassi della pila, può beneficiarne un maggior numero di protocolli. Ad esempio, poiché i pacchetti IP incapsulano i pacchetti TCP, la cifratura dei pacchetti IP (ad esempio con l'IPSec) nasconde anche il contenuto del pacchetto TCP incapsulato. Analogamente, gli autenticatori su pacchetti IP rilevano le modifiche delle informazioni nelle intestazioni TCP contenute.

D'altra parte, la protezione collocata negli strati bassi della pila dei protocolli potrebbe non offrire una protezione sufficiente ai protocolli dei livelli più alti. Ad esempio, un server d'applicazioni eseguito sopra l'IPSec potrebbe autenticare i calcolatori client dai quali riceve le richieste. Tuttavia, per autenticare un utente in uno specifico calcolatore, si deve poter usare un protocollo del livello d'applicazione (che include, ad esempio, l'inserimento della parola d'ordine da parte dell'utente).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.8 Classificazione della sicurezza dei sistemi di calcolo

Nella pubblicazione intitolata *Department of Defence Trusted Computer System Evaluation Criteria* (dicembre 1985), del Dipartimento della Difesa degli USA, sono specificate quattro categorie di sicurezza: *A*, *B*, *C* e *D*. La categoria di livello più basso è la *D*, e corrisponde a una protezione minimale. Non ha sottocategorie, e si assegna ai sistemi che non soddisfano i criteri di nessuna delle categorie rimanenti. A essa appartengono, ad esempio, i sistemi operativi MS-DOS e Windows 3.1.

Il livello successivo, la categoria *C*, offre forme di protezione discrezionale insieme con la possibilità d'identificare e controllare gli utenti e le loro azioni per mezzo di strumenti di verifica. La categoria *C* è suddivisa nelle due classi *C1* e *C2*. Un sistema appartenente alla prima di esse incorpora qualche forma di controllo che permette agli utenti di proteggere le loro informazioni private e che impedisce ad altri utenti di leggerle accidentalmente o di distruggerle. La classe *C1* è un ambiente in cui più utenti che collaborano accedono ai dati allo stesso livello di riservatezza. La maggior parte delle versioni del sistema operativo UNIX appartiene a questa classe.

L'insieme dei sistemi di protezione di un sistema di calcolo (architettura, programmi) che assicura l'imposizione dei criteri di sicurezza è noto come **piattaforma di calcolo fidata** (*trusted computer base* — TCB). La TCB di un sistema di classe *C1* controlla l'accesso ai file permettendo agli utenti di regolamentare la condivisione dei loro oggetti con individui identificati e gruppi definiti. Inoltre, la TCB richiede che ogni utente dichiari la propria identità prima che egli inizi una qualunque attività mediata dalla TCB stessa. Il processo d'autenticazione si basa su un meccanismo di protezione o sulle parole d'ordine; la TCB protegge i dati d'autenticazione rendendoli inaccessibili agli utenti non autorizzati.

Un sistema appartenente alla classe *C2* unisce alle caratteristiche di un sistema di classe *C1* la capacità di controllare gli accessi a un livello individuale. Ad esempio, i diritti d'accesso a un file si possono specificare fino al livello del singolo individuo. Inoltre, l'amministratore del sistema può verificare le azioni di uno o più utenti, individuandoli attraverso il sistema d'identificazione individuale. La TCB protegge anche il proprio codice e le proprie strutture di dati da eventuali tentativi di modifica. Inoltre, nessuna informazione precedentemente prodotta da un utente è disponibile a un altro utente che tenti di accedere a un oggetto contenente tali informazioni lasciato nel sistema. Alcune versioni dello UNIX particolarmente sicure appartengono alla classe *C2*.

I sistemi a protezione tassativa della categoria *B* hanno tutte le proprietà dei sistemi di classe *C2* e applicano a ogni oggetto un'etichetta che indica il livello di riservatezza. Il TCB della classe *B1* impiega tali etichette per prendere decisioni riguardanti il controllo tassativo degli accessi. Ad esempio, un utente classificato al livello 'confidenziale' non può avere accesso a un file che sia classificato al più riservato livello 'segreto'. La TCB specifica anche il livello di riservatezza all'inizio e alla fine di ogni pagina emessa in forma leggibile. Oltre le normali informazioni di convalida (nome d'utente e parola d'ordine), la TCB gestisce le autorizzazioni e i permessi dei singoli utenti e incorpora almeno due livelli di sicurezza. Questi livelli sono gerarchici, cosicché un utente può avere accesso a un

qualunque oggetto cui è assegnata un'etichetta di riservatezza di livello pari o inferiore alla sua autorizzazione. Ad esempio, un utente di livello 'segreto', in assenza di prescrizioni più dettagliate, può accedere a un file di livello 'confidenziale'. I processi sono inoltre isolati l'uno dall'altro tramite l'uso di spazi d'indirizzi distinti.

Un sistema di classe *B2* estende l'uso di etichette di riservatezza a ogni risorsa del sistema. Ai dispositivi fisici sono assegnati livelli minimi e massimi di sicurezza che il sistema usa per garantire il rispetto dei vincoli imposti dall'ambiente fisico nel quale tali dispositivi sono situati. Inoltre, un sistema di classe *B2* permette la protezione dei canali di comunicazione e la verifica degli eventi che potrebbero portare allo sfruttamento illegittimo di un canale protetto.

Un sistema di classe *B3* permette la creazione di liste di controllo degli accessi che identificano utenti o gruppi ai quali *non* è consentito l'accesso a un oggetto specificato. La TCB contiene anche un meccanismo di controllo di quegli eventi che potrebbero indicare una violazione del protocollo di sicurezza. Il meccanismo informa l'amministratore del sistema e, se è necessario, induce la conclusione forzata degli eventi nella maniera meno dannosa possibile.

La categoria *A* è il grado più alto della classificazione. Un sistema di classe *A1* è dal punto di vista funzionale e dell'architettura equivalente a un sistema di classe *B3*, ma si deve progettare e verificare impiegando metodi formali di definizione e verifica, garantendo con un'elevata probabilità che la TCB sia stata correttamente realizzata. Un sistema appartiene a una classe superiore ad A1 se, ad esempio, è stato progettato e realizzato in un impianto di produzione affidabile da personale affidabile.

La TCB assicura solo che il sistema possa garantire il rispetto dei vincoli previsti dai criteri di sicurezza, ma non specifica il contenuto dei criteri stessi. Tipicamente, per un dato sistema informatico si sviluppano i criteri di sicurezza per la **certificazione**, che sono poi vagliati e **accreditati** da un'agenzia specializzata — ad esempio l'NCSC (*National Computer Security Center*). Per alcuni sistemi informatici si potrebbe aver bisogno di altre certificazioni, come quelle rilasciate dal TEMPEST, che garantiscono la protezione dal trafugamento elettronico delle informazioni. Ad esempio, i terminali di un sistema certificato dal TEMPEST sono schermati in modo da evitare la propagazione esterna del campo elettromagnetico. Questa schermatura assicura che un'attrezzatura posta fuori della stanza o dell'edificio in cui i terminali sono situati non possa rilevare alcuna informazione inviata al terminale.

19.9 Un esempio: Windows NT

Il Microsoft Windows NT è un sistema operativo relativamente recente progettato per fornire una gamma di garanzie di sicurezza che variano da un livello minimale al livello *C2* della classificazione della sicurezza del governo degli Stati Uniti. Il livello di sicurezza impiegato normalmente dal sistema Windows NT è quello minimale, ma l'amministratore del sistema può facilmente portare le garanzie di sicurezza al livello desiderato. Per aiutare l'amministratore nella scelta dei parametri di sicurezza richiesti, è disponibile un

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

programma d'utilità; si tratta di `C2config.exe`. In questo paragrafo si esaminano gli strumenti impiegati da questo sistema operativo per garantire i diversi livelli di sicurezza; per maggiori informazioni si veda il Capitolo 21.

Il modello di sicurezza si basa sulla nozione di utente accreditato del sistema (*user account*). Il sistema operativo Windows NT permette la creazione di un numero arbitrario di utenti del sistema, i quali si possono raggruppare in un qualunque modo. L'accesso agli oggetti del sistema si può poi consentire o negare secondo i modi desiderati. Il sistema identifica gli utenti per mezzo di un identificatore di sicurezza *unico*. Quando un utente accede al sistema, si crea un **contrassegno d'accesso di sicurezza** che include l'identificatore di sicurezza dell'utente, gli identificatori di sicurezza per tutti i gruppi dei quali l'utente è membro, e un elenco di tutti i permessi speciali di cui l'utente gode. Alcuni esempi di permessi speciali sono la possibilità di fare copie di riserva di file e directory, di spegnere il calcolatore, di accedere al sistema in modo interattivo e di regolare l'orologio del sistema. Ogni processo che il sistema esegue per conto di un utente riceve una copia del contrassegno d'accesso. Ogniqualvolta l'utente — direttamente o per mezzo di un processo — tenta di accedere a oggetti del sistema, l'accesso è negato o concesso secondo gli identificatori di sicurezza contenuti nel contrassegno. L'autenticazione è di solito portata a termine rispetto a un nome d'utente e a una parola d'ordine, anche se la struttura modulare del sistema Windows NT permette lo sviluppo di mezzi di autenticazione specifici. Ad esempio, si potrebbe usare un analizzatore elettronico dell'impronta della retina per verificare la reale identità dell'utente.

Il sistema Windows NT usa l'idea del soggetto per far sì che i programmi eseguiti per conto di un utente non ottengano modi d'accesso al sistema meno restrittivi di quelli dell'utente stesso. Un **soggetto** (*subject*) si usa per identificare e gestire i permessi relativi a ogni programma eseguito per conto di un utente, ed è composto dal contrassegno d'accesso e dal programma che agisce per conto dell'utente. Poiché il sistema Windows NT si basa su un modello client-server, per controllare gli accessi si usano due classi di soggetti. Un esempio di **soggetto semplice** è il tipico programma d'applicazione che un utente esegue dopo aver ottenuto l'accesso al sistema. A un soggetto semplice si assegna un **contesto di sicurezza** definito secondo il contrassegno d'accesso dell'utente. Un **soggetto server** è un processo realizzato come un server protetto che usa il contesto di sicurezza del client quando agisce per suo conto. Quando un processo assume gli attributi di sicurezza di un altro processo si dice che lo **impersona**.

Come si è detto nel Paragrafo 19.6, la verifica è un utile strumento per migliorare la sicurezza. Il sistema Windows NT è dotato di un sistema di verifica, e permette il controllo di molte comuni minacce per la sicurezza del sistema. Alcuni esempi di casi in cui la verifica è utile per localizzare minacce alla sicurezza sono la registrazione degli accessi non riusciti per individuare i tentativi d'accesso con parole d'ordine casuali e quella degli accessi riusciti per scoprire attività all'interno del sistema in orari insoliti; inoltre, al fine di prevenire il diffondersi di un virus, è utile tenere traccia dei tentativi di scrittura — riusciti e falliti — su file eseguibili; infine, la registrazione dei tentativi d'accesso a un file permette d'individuare gli accessi ai file riservati.

Gli attributi di sicurezza di un oggetto del sistema Windows NT sono descritti da un **descrittore di sicurezza**. Esso contiene l'identificatore di sicurezza del proprietario dell'oggetto (il quale può cambiare i permessi d'accesso, o gli identificatori di sicurezza dei gruppi), che è usato solo dal sottosistema POSIX; una lista di controllo discrezionale degli accessi che stabilisce quali utenti o gruppi abbiano o non abbiano possibilità d'accesso; e una lista di sistema di controllo degli accessi che controlla quali messaggi di verifica saranno generati. Ad esempio, il descrittore di sicurezza del file `foo.bar` potrebbe essere di proprietà di avi e avere la seguente lista di controllo discrezionale degli accessi:

- avi — consentito ogni accesso;
- gruppo cs — consentiti accessi per la lettura e la scrittura;
- utente cliff — nessun accesso consentito.

Inoltre, il descrittore potrebbe includere una lista di sistema di controllo degli accessi che permetta di verificare le scritture di ogni utente. Una lista di controllo degli accessi contiene elementi composti dell'identificatore di sicurezza dell'individuo e della maschera d'accesso che definisce tutte le azioni permesse sull'oggetto insieme con un valore convenzionale d'*accesso consentito* o d'*accesso negato* per ogni azione. I file di Windows NT possono avere i seguenti tipi d'accesso: `ReadData`, `WriteData`, `AppendData`, `Execute`, `ReadExtendedAttribute`, `WriteExtendedAttribute`, `ReadAttributes`, e `WriteAttributes`. È chiaro che ciò permette un preciso grado di controllo dei modi d'accesso agli oggetti.

Il sistema Windows NT classifica gli oggetti come contenitori e non contenitori. Gli **oggetti contenitori**, ad esempio le directory, possono contenere in senso logico altri oggetti. Di norma, quando si crea un oggetto all'interno di un oggetto contenitore, il nuovo oggetto eredita i permessi dell'oggetto genitore; così pure, se l'utente copia un file da una directory a un'altra, il file eredita i permessi della directory di destinazione. Gli **oggetti non contenitori** non ereditano nessun altro permesso.

Tuttavia, se si cambiano i permessi di una directory, i nuovi permessi non si applicano automaticamente alle directory e ai file in essa già contenuti; l'utente, se lo desidera, può richiedere esplicitamente che siano applicati i nuovi permessi. Inoltre, se un utente sposta un file da una directory a un'altra, i permessi del file rimangono invariati.

L'amministratore del sistema può anche inibire l'uso di una delle stampanti del sistema per un intero giorno o per parte di esso; e può avvalersi del Monitor delle Prestazioni per individuare i problemi incipienti. In generale, un punto di forza del sistema Windows NT è la capacità di mettere a disposizione strumenti che aiutano a fornire un ambiente informatico sicuro. Molti di questi strumenti però di solito non sono attivi; in questo modo il generico utente di un PC può lavorare in un ambiente consueto. Nel caso di un effettivo sistema multiutente l'amministratore dovrebbe progettare e applicare un piano di sicurezza sfruttando i servizi offerti dal sistema operativo.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

19.10 Sommario

La protezione è un problema interno. La sicurezza deve prendere in considerazione sia il sistema di calcolo sia l'ambiente — persone, edifici, affari, oggetti preziosi, minacce — all'interno del quale si usa il sistema.

I dati memorizzati in un sistema di calcolo devono essere protetti da accessi non autorizzati, distruzioni o alterazioni dolose, e dall'introduzione accidentale d'incoerenze. È più semplice proteggere un sistema dalla perdita accidentale di coerenza dei dati che da accessi dolosi ai dati. Un'assoluta protezione dagli abusi dolosi delle informazioni memorizzate in un sistema di calcolo non è possibile, ma il costo di tali azioni può essere reso sufficientemente alto da scoraggiare quasi tutti, se non tutti, i tentativi d'accesso non autorizzati alle informazioni contenute nel sistema.

Le varie misure d'autorizzazione presenti in un sistema di calcolo possono essere una protezione insufficiente per dati molto riservati. In questi casi i dati possono essere cifrati: non è possibile che i dati cifrati siano letti, salvo che il lettore sappia come decifrarli.

Vi sono vari metodi di autenticazione oltre all'ordinaria protezione basata su identificatore d'utente e parola d'ordine. Le parole d'ordine monouso cambiano ogni volta i dati inviati per evitare gli attacchi per reimmissione d'informazioni autentiche precedentemente intercettate (*replay attack*). La doppia autenticazione richiede due forme di autenticazione, ad esempio, una calcolatrice e un PIN d'attivazione. Insieme con i dispositivi biometrici, questi metodi diminuiscono enormemente la possibilità di contraffazione dei dati di autenticazione.

Ci sono vari tipi di attacchi che si possono condurre contro singoli calcolatori o in modo generalizzato. Virus e worm si autoriproducono, a volte infettando migliaia di calcolatori. Gli attacchi basati sull'alterazione della pila permettono a un utente ostile di cambiare il suo livello d'accesso al sistema. Gli attacchi per rifiuto del servizio impediscono l'uso legittimo dei sistemi colpiti.

Esistono vari modi per prevenire o rilevare gli incidenti concernenti la sicurezza. Questi modi includono i sistemi di rilevamento delle intrusioni, il rilevamento e la registrazione di eventi di sistema, il controllo delle modifiche ai programmi di sistema e il controllo delle chiamate del sistema. La crittografia si può impiegare sia all'interno di un singolo sistema sia tra diversi sistemi per prevenire l'intercettazione di informazioni.

19.11 Esercizi

19.1 Una parola d'ordine può divenire nota ad altri utenti in diversi modi. Dite se esiste un metodo semplice per individuare un evento di questo tipo e spiegate la risposta.

19.2 L'elenco di tutte le parole d'ordine è conservato all'interno del sistema operativo. Quindi, se un utente riesce a leggere quest'elenco, la protezione delle parole d'ordine non è più garantita. Proponete uno schema per impedire il verificarsi del problema. (Suggerimento: si usino diverse rappresentazioni interne ed esterne.)

19.3 Un'estensione sperimentale del sistema operativo UNIX permette a un utente di associare un programma 'guardiano' (*watchdog*) a un file in modo che tale programma sia attivato ogni volta che un programma richiede l'accesso al file. Il guardiano allora permette o nega l'accesso al file. Discutete i pro e i contro di questo metodo ai fini della sicurezza.

19.4 Il programma COPS del sistema UNIX, scandisce un dato sistema alla ricerca di eventuali varchi nella sicurezza e avverte l'utente della presenza di possibili problemi. Elencate i rischi potenziali nell'uso di tale sistema di sicurezza. Dite in che modo tali problemi si possono limitare o eliminare.

19.5 Discutete i mezzi con cui i gestori dei sistemi connessi alla rete Internet possono aver progettato i loro sistemi per limitare o eliminare i danni causati dall'*Internet worm* di Robert Morris. Dite quali sono gli inconvenienti dovuti a tali modifiche al modo in cui il sistema funziona.

19.6 Esprimetevi pro o contro la condanna comminata a Robert Morris Jr. per aver prodotto ed eseguito l'*Internet worm*.

19.7 Elencate sei elementi riguardanti la sicurezza di un sistema di calcolo per una banca. Per ogni elemento nell'elenco dite se riguarda la sicurezza fisica, umana, o del sistema operativo.

19.8 Esponete due vantaggi offerti dalla cifratura dei dati memorizzati in un sistema di calcolo.

19.12 Note bibliografiche

Una trattazione generale della sicurezza si trova in [Hsiao et al. 1979], [Landwehr 1981], [Denning 1982], [Pfleeger 1989] e [Russel et al. 1991]; anche [Lobel 1986] è un testo d'interesse generale.

Argomenti relativi alla progettazione e alla verifica dei sistemi di sicurezza sono discussi in [Rushby 1981] e [Silverman 1983]. Un nucleo di sicurezza per un microcalcolatore con più unità d'elaborazione è descritto in [Schell 1983]. Un sistema distribuito sicuro è descritto in [Rushby e Randell 1983].

La sicurezza delle parole d'ordine è discussa in [Morris e Thompson 1979]. I metodi per combattere i ladri di parole d'ordine sono discussi in [Morshedian 1986]. L'autenticazione di parole d'ordine con comunicazione non sicura è esaminata in [Lamport 1981]. La questione della violazione delle parole d'ordine è trattata in [Seely 1989]. Il problema delle intrusioni nei calcolatori è trattato in [Lehmann 1987] e [Reid 1987].

Discussioni sulla sicurezza del sistema operativo UNIX si trovano in [Grampp e Morris 1984], [Wood e Kochan 1985], [Farrow 1986a], [Farrow 1986b], [Filipski e Hanko 1986], [Hecht et al. 1988], [Kramer 1988] e [Garfinkel e Spafford 1991]. [Bershad e Pinkerton 1988] presentano l'estensione dei programmi guardiani al BSD UNIX. Il pacchetto di scansione della sicurezza COPS per UNIX è stato scritto da Dan

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Farmer alla Purdue University; gli utenti della rete Internet possono prelevarlo, usando il programma `ftp`, dalla directory `/pub/security/cops` all'indirizzo `ftp.uu.net`.

[Spafford 1989] presenta una dettagliata discussione tecnica dell'*Internet worm*; l'articolo di Spafford è apparso, insieme con altri tre articoli sull'argomento, in una sezione speciale dedicata all'*Internet worm* di *Communications of the ACM*, Volume 32, Numero 6, giugno 1989.

[Diffie e Hellman 1976], [Diffie e Hellman 1979] sono stati i primi ricercatori a proporre l'uso dello schema di cifratura a chiave pubblica. L'algoritmo presentato nel Paragrafo 19.7.2, basato su tale schema, è stato sviluppato da [Rivest et al. 1978]. [Lempel 1979], [Simmons 1979], [Gifford 1982], [Denning 1982] e [Ahituv et al. 1987] indagano l'uso della crittografia nei sistemi di calcolo. Discussioni riguardanti la protezione delle firme digitali sono presentate in [Akl 1983], [Davies 1983], [Denning 1983] e [Denning 1984].

Il governo federale degli USA è ovviamente interessato alla sicurezza, e ha pubblicato il [*Department of Defense Trusted Computer System Evaluation Criteria* 1985], anche noto come *Orange Book*, nel quale sono descritti una serie di livelli di sicurezza e le caratteristiche che un sistema operativo deve avere per essere qualificato a ciascun grado di sicurezza. La sua lettura è un buon punto di partenza per la comprensione degli aspetti riguardanti la sicurezza dei sistemi di calcolo. Il testo *Microsoft Windows NT Workstation Resource Kit* [Microsoft 1996] descrive le caratteristiche e l'uso del modello di sicurezza del sistema operativo Windows NT.

L'algoritmo RSA è presentato in [Rivest et al. 1978]. Nel sito `http://www.nist.gov/aes/` si trovano informazioni sulle attività del NIS riguardanti l'AES; nello stesso sito si trovano anche informazioni su altri standard di cifratura per gli USA. Una trattazione più completa dell'SSL 3.0 si trova all'indirizzo `http://home.netscape.com/eng/ssl3/`. Nel 1999 l'SSL 3.0 è stato leggermente modificato e presentato col nome TLS in una RFC (*request for comments*) dell'IETF.

L'esempio del Paragrafo 19.6.1 che illustra l'impatto del tasso di falsi allarmi sull'efficacia degli IDS è basato su [Axelsson 1999]. Una descrizione più completa del programma `swatch` e il suo uso con `syslog` si trova in [Hansen e Atkins 1993]. La descrizione del Tripwire del Paragrafo 19.6.3 si basa su [Kim e Spafford 1993]. La descrizione e l'esempio presenti nel Paragrafo 19.6.4 sono tratti da [Forrest et al. 1996].

Parte settima

Casi di studio

Sistema operativo LINUX
Windows 2000
Prospettiva storica

A questo punto si possono riunire i concetti illustrati in questo libro descrivendo alcuni sistemi operativi reali. Due sono discussi nei dettagli; LINUX e Microsoft Windows 2000. Il sistema LINUX è stato scelto per diverse ragioni: è diffuso, è liberamente disponibile, e rappresenta un sistema UNIX completo nelle sue funzioni. Ciò offre a chi studia i sistemi operativi l'opportunità di esaminare — e modificare — il codice sorgente di un *vero* sistema operativo.

Anche il sistema Windows 2000 è discusso nei particolari. Questo recente sistema operativo prodotto dalla Microsoft sta ottenendo consensi non solo sul mercato dei PC e delle stazioni di lavoro ma anche su quello dei server per gruppi di lavoro. Lo si è scelto perché offre la possibilità di studiare un sistema operativo moderno che differisce radicalmente dallo UNIX sia nel progetto sia nella realizzazione.

Sono inoltre brevemente discussi altri sistemi operativi molto influenti. L'ordine di presentazione è stato scelto in modo da evidenziare le analogie e le differenze tra i vari sistemi; non è strettamente cronologico e non riflette l'importanza relativa dei sistemi.

Infine sono disponibili tre appendici, prelevabili tramite la rete Internet, che trattano, rispettivamente, FreeBSD, Mach e Nachos. Il sistema operativo FreeBSD è un altro sistema UNIX. Ma mentre LINUX adotta l'orientamento che prevede la combinazione di caratteristiche di diversi tipi di sistemi UNIX, FreeBSD si basa sul modello BSD dello UNIX. Anche il codice sorgente del FreeBSD è liberamente disponibile. Mach è un sistema operativo moderno compatibile col BSD UNIX. Nachos è un sistema operativo didattico che consente agli studenti di costruire da sé parti significative del sistema operativo e di osservare gli effetti del loro lavoro.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 20

Sistema operativo LINUX

Nell'Appendice A si tratta il funzionamento interno del sistema operativo 4.3BSD, uno fra i sistemi operativi di tipo UNIX. LINUX è un altro sistema operativo ispirato allo UNIX che negli ultimi anni ha guadagnato molti consensi. In questo capitolo è trattata la storia e lo sviluppo del LINUX, e sono discusse le sue interfacce per il programmatore e l'utente: esse devono molto alla tradizione dello UNIX. Si analizzano anche i metodi adottati per realizzare queste interfacce, anche se in ciò LINUX ha molto in comune con le esistenti versioni dello UNIX, perché è stato concepito allo scopo di permettere l'esecuzione del massimo numero possibile di applicazioni per il sistema UNIX; le informazioni fondamentali fornite sullo UNIX nell'Appendice A non saranno ripetute in questo capitolo.

LINUX è un sistema operativo in rapida evoluzione: il nucleo descritto in questo capitolo è la versione 2.2, pubblicata nel gennaio 1999.

20.1 Storia

Il sistema operativo LINUX assomiglia molto a un qualunque altro sistema UNIX, e in effetti la compatibilità con UNIX è stata uno degli obiettivi principali nella sua progettazione; tuttavia, il sistema LINUX è molto più recente della maggior parte dei sistemi UNIX. Il suo sviluppo ha avuto inizio nel 1991 quando Linus Torvalds, uno studente finlandese, scrisse LINUX, un nucleo piccolo ma autosufficiente per la CPU 80386, la prima vera CPU a 32 bit della gamma Intel per i PC-compatibili.

Già fin dagli inizi del suo sviluppo il codice sorgente del LINUX fu reso gratuitamente disponibile tramite la rete Internet: ne è risultata una storia ricca di contributi da parte di utenti di tutto il mondo che comunicavano quasi esclusivamente tramite Internet. Da un nucleo iniziale che realizzava parzialmente un ridotto sottoinsieme dei servizi di sistema dello UNIX, il sistema LINUX è cresciuto fino a includere molte funzioni dello UNIX.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I primi stadi di sviluppo del LINUX avevano a che fare in gran parte con il nucleo centrale del sistema operativo, il programma privilegiato che interagisce direttamente con l'architettura del calcolatore e gestisce tutte le risorse del sistema; chiaramente per ottenere un sistema operativo completo occorre molto più del solo nucleo. È utile distinguere il nucleo del LINUX da un sistema LINUX. Il **nucleo** è un programma del tutto originale sviluppato interamente dalla comunità LINUX; il **sistema** LINUX, nella sua forma odierna, incorpora una moltitudine di componenti, alcuni scritti *ex novo*, altri presi in prestito da diversi progetti di sviluppo e altri ancora creati in collaborazione con altri gruppi di programmatori.

Il sistema LINUX di base è un ambiente standard per le applicazioni e per il codice scritto dagli utenti, ma non impone convenzioni rigide sulla gestione complessiva delle risorse. Un ulteriore livello organizzativo si è reso necessario con la progressiva maturazione del LINUX: un **pacchetto di distribuzione** include tutti i componenti ordinari del sistema operativo più una serie di strumenti di gestione che semplifica l'installazione iniziale, gli aggiornamenti successivi, e l'installazione o la rimozione di altri pacchetti. Un moderno pacchetto di distribuzione, inoltre, fornisce di solito strumenti per la gestione del file system, degli utenti, dei servizi di rete, e così via.

20.1.1 Nucleo del sistema LINUX

La prima versione del nucleo del sistema operativo LINUX resa disponibile al pubblico fu la 0.01, datata 14 maggio 1991; non forniva alcun servizio di rete, funzionava solo su PC con CPU Intel 80386-compatibili, e la sua gestione dei driver dei dispositivi era estremamente limitata. Anche il sottosistema per la memoria virtuale era abbastanza elementare, e non dava la possibilità di associare i file a parti di spazi d'indirizzi di memoria (*memory mapping*); tuttavia, anche questa primissima versione permetteva la condivisione delle pagine con copiatura su scrittura. Il solo file system disponibile era Minix, perché i primi nuclei LINUX furono sviluppati su piattaforme Minix. Ciononostante il nucleo era in grado di gestire veri e propri processi UNIX con spazi d'indirizzi protetti.

La successiva versione importante, LINUX 1.0, fu diffusa solo il 14 marzo 1994; rappresentava il culmine di tre anni di rapido sviluppo del nucleo. Forse la più importante nuova caratteristica riguardava la gestione dei servizi di rete: incorporava i protocolli standard dello UNIX, TCP/IP, e un'interfaccia a socket compatibile con il BSD UNIX per la programmazione di rete; grazie ai nuovi driver era anche possibile eseguire il protocollo IP su una rete Ethernet o su linee seriali e modem usando i protocolli PPP o SLIP.

Il *Kernel 1.0* includeva anche un nuovo file system molto migliorato che non soffriva delle limitazioni del file system originario Minix; inoltre, funzionava con diversi tipi di controller SCSI per l'accesso ai dischi ad alte prestazioni. Anche il sottosistema per la memoria virtuale era stato migliorato, e dava ora la possibilità di gestire la paginazione con file d'avvicendamento (*swapping*) e di associare a parti di spazi d'indirizzi di memoria file arbitrari (anche se solo per operazioni di lettura).

Questa versione forniva inoltre l'accesso a una serie di ulteriori dispositivi che, seppur ancora limitati alla piattaforma Intel-PC, comprendevano le unità a dischetti e CD-ROM, schede audio, vari tipi di mouse e tastiere internazionali. Il nucleo era anche in grado di simulare le operazioni in virgola mobile per i calcolatori basati sulla CPU 80386 che non erano dotate del coprocessore matematico 80387, e realizzava inoltre la **comunicazione tra processi** (IPC — *interprocess communication*) nello stile dello UNIX System V, compresa la condivisione della memoria, i semafori e le code di messaggi. Infine, il nucleo metteva a disposizione alcuni semplici strumenti per il caricamento dinamico di moduli aggiuntivi del nucleo.

A questo punto si cominciò a sviluppare la versione 1.1 del nucleo, ma fu necessario distribuire in seguito numerose correzioni per la versione 1.0, nella quale erano stati riscontrati errori. Come convenzione di numerazione per le versioni del nucleo fu adottato lo schema seguente: le versioni con un numero dispari dopo il punto sono **nuclei di sviluppo**; quelle con numero pari dopo il punto sono invece **nuclei di produzione**. Gli aggiornamenti rispetto alle versioni stabili sono da intendersi solo come rimedi temporanei, mentre i nuclei di sviluppo potrebbero includere nuovi servizi relativamente poco collaudati.

Il *Kernel 1.2* fu diffuso nel marzo 1995; non rappresentava un miglioramento paragonabile a quello della versione1.0, ma in ogni modo permetteva la gestione di una ben più ampia gamma di dispositivi, compreso l'allora nuovo bus PCI; la possibilità di usare il modo virtuale 8086 della CPU 80386 — un'altra funzione specificamente dedicata ai PC — che permetteva di simulare il sistema operativo DOS per PC. Le funzioni di rete erano state aggiornate in modo da permettere l'uso del protocollo IPX, e rendere le funzioni del protocollo IP più complete, includendo i servizi di contabilizzazione delle risorse (*accounting*) e di barriera di sicurezza (*firewall*).

Il *Kernel 1.2* fu anche l'ultima versione dedicata ai soli PC: i file sorgenti del LINUX 1.2 comprendevano già alcune versioni parziali per le CPU SPARC, Alpha e MIPS, ma la piena integrazione di queste altre architetture doveva attendere la diffusione della versione stabile 1.2.

LINUX 1.2 si concentrava sul problema di gestire un maggior numero di dispositivi e di fornire realizzazioni più complete delle funzioni esistenti. Molti nuovi servizi erano a quel tempo in via di sviluppo, ma l'integrazione del nuovo codice all'interno del nucleo fu rimandata fino alla distribuzione della versione stabile del *Kernel 1.2*; di conseguenza, la versione 1.3 portò con sé una gran quantità di nuovi servizi.

Il materiale di cui s'è detto fu infine distribuito come LINUX 2.0 nel giugno 1996. Il passaggio dalle versioni numerate con l'unità prima del punto alla versione 2.0 era giustificato da due nuove caratteristiche di basilare importanza: la possibilità di gestire architetture diverse, compresa un'effettiva versione per CPU Alpha a 64 bit, e quella di gestire sistemi con più unità d'elaborazione. Le versioni basate sul *Kernel 2.0* sono anche disponibili per le CPU Motorola della serie 68000 e per i sistemi SPARC della Sun. Una versione derivata del LINUX che sfrutta il micronucleo Mach è anche eseguibile sui PC e PowerMac.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Ma le novità introdotte dalla versione 2.0 non terminano qui. Il codice relativo alla gestione della memoria era stato sostanzialmente migliorato per fornire una cache unificata per i dati del file system che fosse indipendente dai servizi di caching relativi ai dispositivi a blocchi. In conseguenza di questo cambiamento il nucleo otteneva dal file system e dalla memoria virtuale prestazioni molto migliori. Per la prima volta, inoltre, i servizi di caching per il file system erano stati estesi ai file system di rete, ed erano disponibili le regioni associate a parti di spazi d'indirizzi nella memoria scrivibili.

Il *Kernel 2.0* forniva anche prestazioni dei protocolli TCP/IP molto migliori, oltre a permettere l'uso di un certo altro numero di protocolli, fra i quali AppleTalk, AX.25 per i radioamatori, e ISDN; era anche possibile montare volumi remoti Netware e SMB (Microsoft LanManager).

Altre migliorie fondamentali erano la realizzazione di thread interni al nucleo, la gestione delle dipendenze fra i moduli caricabili, e il caricamento automatico dei moduli richiesti. La configurazione dinamica del nucleo in fase d'esecuzione era stata non poco migliorata grazie a una nuova interfaccia standard di configurazione. Altre due novità comprendevano la divisione in quote del file system e l'adozione delle classi di scheduling dei processi per l'elaborazione in tempo reale, compatibili con lo standard POSIX.

Nel gennaio 1999 è stato pubblicato LINUX 2.2 proseguendo i miglioramenti introdotti dal LINUX 2.0. È stato introdotto anche un adattamento ai sistemi UltraSPARC. I servizi di rete sono stati affinati con un servizio di barriera di sicurezza più flessibile, una migliore gestione del traffico e dell'instradamento, e l'uso di ampie finestre TCP e conferme (*ack*) selettive. Si possono leggere i dischi Acorn, Apple Macintosh, ed NTFS, è stato migliorato l'NFS ed è stato aggiunto un demone, eseguito nel modo del nucleo, per l'NFS. La gestione dei segnali, delle interruzioni, e di alcuni aspetti dell'I/O è ora eseguita impiegando un sistema di bloccaggio più selettivo per migliorare le prestazioni dell'SMP.

20.1.2 Sistema LINUX

Per molti aspetti il nucleo rappresenta la parte centrale del progetto LINUX, ma altri componenti concorrono a costituire un sistema operativo LINUX completo. Mentre il nucleo è composto di codice scritto *ex novo* specificamente per il progetto LINUX, molti programmi aggiuntivi che completano il sistema non sono stati espressamente concepiti per il sistema LINUX, ma sono condivisi da un certo numero di sistemi operativi ispirati allo UNIX. In particolare il sistema LINUX adotta molti strumenti sviluppati come parti del sistema operativo BSD di Berkeley, del sistema X Window del MIT, e del progetto GNU della Free Software Foundation.

Questa condivisione è stata a doppio senso: lo sviluppo delle principali librerie di sistema del LINUX ha avuto origine dal progetto GNU, ma la comunità LINUX le ha poi notevolmente migliorate affrontandone le omissioni e inefficienze, ed eliminando gli errori. Altri componenti, ad esempio il compilatore per il Linguaggio C del progetto GNU, gcc (*GNU C compiler*), erano già di qualità sufficientemente alta per essere direttamente usati dal LINUX. Gli strumenti necessari per la gestione dei servizi di rete sono stati derivati da codice originariamente scritto per 4.3BSD, ma discendenti più recenti del BSD come FreeBSD hanno preso a prestito codice dal sistema LINUX, ad esempio la libreria di simulazione delle operazioni in virgola mobile per le CPU Intel e i driver dei dispositivi audio per PC.

La manutenzione complessiva del sistema LINUX è curata da una rete più o meno salda di programmatori che collaborano tramite la rete Internet, e la responsabilità dell'integrità di alcune specifiche componenti è assegnata a piccoli gruppi o individui. Un limitato numero di siti Internet ad accesso pubblico tramite il protocollo ftp (*file transfer protocol*) agiscono da custodi dello standard *de facto* di questi componenti.

Anche il documento *File System Hierarchy Standard* è curato dalla comunità LINUX al fine di preservare la compatibilità fra le varie parti del sistema: esso specifica la struttura generale di un file system LINUX, stabilendo sotto quali nomi di directory i file di configurazione, le librerie, i file eseguibili di sistema e i file di dati dovrebbero essere archiviati.

20.1.3 Pacchetti di distribuzione

In teoria chiunque potrebbe installare un sistema LINUX compilando le ultime versioni del codice sorgente ottenibili dai siti ftp; questo era esattamente ciò che ogni utente doveva fare durante i primi anni di vita del LINUX. A mano a mano che il sistema maturava, però, diversi individui o gruppi hanno cercato di rendere quest'incombenza meno pesante fornendo pacchetti precompilati standard di facile installazione.

Queste raccolte, o pacchetti di distribuzione, comprendono molto di più del sistema LINUX di base: generalmente forniscono strumenti aggiuntivi per l'installazione e la gestione del sistema, e anche pacchetti precompilati e pronti per l'installazione che forniscono molti comuni strumenti del sistema UNIX, ad esempio server di rete, programmi di consultazione del Web, elaboratori di testi e persino giochi.

I primi pacchetti di distribuzione erano organizzati semplicemente fornendo un modo per collocare i file nei luoghi appropriati; la gestione dei pacchetti è uno dei contributi importanti delle moderne versioni: esse comprendono un programma d'archiviazione dei pacchetti che permette di installare, aggiornare e rimuovere senza difficoltà i pacchetti desiderati. Nei primi anni di vita del LINUX la versione SLS fu la prima raccolta di pacchetti LINUX classificabile come un vero e proprio pacchetto di distribuzione; tuttavia, anche se poteva essere installata in un'unica soluzione, non forniva gli strumenti di gestione dei pacchetti che ci si può aspettare oggi. La versione **Slackware** rappresentò complessivamente un gran salto di qualità, nonostante la sua gestione dei pacchetti fosse piuttosto scadente, essa è ancora oggi assai diffusa tra la comunità LINUX.

Dalla distribuzione della Slackware, sono divenute disponibili molte nuove versioni commerciali e non commerciali. Due versioni di gran successo sono **Red Hat** e **Debian**, la prima prodotta da un'impresa commerciale, la seconda fornita dalla comunità LINUX; altre versioni commerciali sono prodotte da **Caldera**, **Craftworks** e **WorkGroup Solutions**. Il gran numero di seguaci del LINUX in Germania ha portato alla diffusione di diverse versioni commerciali in lingua tedesca, tra le quali **SuSE** e **Unifix**. I pacchetti di

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

distribuzione del LINUX attualmente in circolazione sono troppi per essere qui elencati; questa gran varietà non impedisce la compatibilità fra versioni diverse. Il formato RPM per i file è adottato o almeno compreso dalla maggior parte delle versioni, e le applicazioni commerciali distribuite in questo formato si possono installare ed eseguire con qualunque versione del LINUX che accetti i file RPM.

20.1.4 Licenze

Il nucleo del LINUX è distribuito in accordo alla GNU General Public License (GPL), i cui termini sono stati stabiliti dalla Free Software Foundation. Il sistema LINUX non è di **pubblico dominio**: questa locuzione implica la rinuncia ai diritti d'autore, mentre i diritti sul codice del LINUX sono tuttora detenuti dai vari autori. Tuttavia, il sistema LINUX è *libero* nel senso che si può copiare, modificare e usare in qualunque modo si desideri, e si può far circolare senza alcuna limitazione.

La principale conseguenza dei termini della licenza del LINUX è che chiunque lo usi o crei un prodotto da esso derivato (un legittimo esercizio) non può reclamare diritti di proprietà sul prodotto. I programmi diffusi in accordo con la GPL, inoltre, non si possono ridistribuire in forma puramente binaria, ma si deve rendere disponibile il codice sorgente, e ciò vale per ogni componente di un pacchetto di distribuzione che sia soggetto alla GPL. (Si noti che questa limitazione non impedisce la diffusione gratuita o la vendita di versioni puramente binarie, purché chiunque riceva i file eseguibili abbia la possibilità di ottenere il codice sorgente a un prezzo ragionevole.)

20.2 Princìpi di progettazione

Dal punto di vista del progetto complessivo il sistema LINUX assomiglia a qualunque altro sistema UNIX tradizionale non basato su micronucleo. È un sistema multiutente a più processi dotato di una completa serie di strumenti compatibili con UNIX; il suo file system aderisce alla tradizionale semantica UNIX, e anche il modello standard di comunicazione in rete dello UNIX è pienamente realizzato. I dettagli interni della progettazione del LINUX sono stati assai influenzati dalla storia del suo sviluppo.

Sebbene oggi il sistema LINUX possa essere eseguito su un'intera gamma di piattaforme, esso fu inizialmente concepito esclusivamente per l'architettura dei PC, e gran parte di questa prima fase di sviluppo fu portata a termine da appassionati e non da professionisti ben finanziati e dotati di raffinati strumenti di ricerca: il risultato fu che, fin dall'inizio, LINUX ottenne da risorse limitate il massimo possibile di capacità funzionali. Anche se oggi funziona perfettamente su calcolatori con più unità d'elaborazione con centinaia di megabyte di memoria centrale e molti gigabyte di spazio di dischi, LINUX può ancora funzionare con meno di 4 MB di RAM.

A mano a mano che i PC divenivano più potenti e la memoria e i dischi più economici, l'originario nucleo 'minimalista' del LINUX s'arricchiva di nuovi servizi tipici del sistema UNIX; la velocità e l'efficienza sono ancora oggi obiettivi importanti, ma molta ricerca recente e attualmente in corso si concentra su un terzo obiettivo centrale: la

costituzione di uno standard. Uno dei prezzi che occorre pagare per la diversità delle versioni dello UNIX disponibili è infatti l'impossibilità, almeno in generale, di trasferire codice sorgente fra due sue versioni: anche quando le chiamate del sistema sono le medesime nei due sistemi, non è detto che il loro comportamento sia identico. Gli standard POSIX comprendono un insieme di specifiche riguardanti diversi aspetti del comportamento di un sistema operativo, ed esistono documenti POSIX relativi ai servizi più comuni dei sistemi operativi e a estensioni quali i thread dei processi e le operazioni in tempo reale. Il sistema LINUX è concepito al fine di rispettare i documenti POSIX, e almeno due sue versioni ufficiali hanno ottenuto la certificazione POSIX.

Presentando interfacce standard sia al programmatore sia all'utente, il sistema operativo LINUX dovrebbe riservare poche sorprese a chiunque conosca già il sistema UNIX; di conseguenza, queste interfacce non saranno qui trattate nei dettagli: il contenuto dei paragrafi sull'interfaccia per il programmatore (Paragrafo A.3) e per l'utente (Paragrafo A.4) del sistema 4.3BSD vale anche per il sistema LINUX. Si noti solo che ordinariamente l'interfaccia per il programmatore del LINUX segue la semantica dello UNIX SVR4 e non quella del BSD; per ottenere la semantica del BSD è disponibile una raccolta di librerie.

Esistono molti altri standard nel mondo UNIX, ma la certificazione del LINUX in conformità a essi è a volte rallentata dal fatto che di solito questi sistemi sono disponibili solo a pagamento, e la certificazione di un sistema operativo rispetto alla maggior parte degli standard è un processo molto costoso. Ciononostante, poiché la capacità di eseguire un'ampia gamma di applicazioni è importante per qualunque sistema operativo, la realizzazione di nuovi standard è un obiettivo fondamentale nello sviluppo del LINUX anche in assenza di certificazioni formali. Oltre allo standard di base POSIX, LINUX attualmente comprende le estensioni POSIX ai thread e un sottoinsieme delle estensioni POSIX al controllo dei processi in tempo reale.

20.2.1 Componenti del sistema

Il sistema LINUX è composto da tre blocchi principali di codice, in linea con le versioni più tradizionali del sistema UNIX:

1. **Nucleo.** Il nucleo è responsabile di tutte le astrazioni importanti messe in atto dal sistema operativo, come la memoria virtuale e i processi.
2. **Librerie di sistema.** Le librerie di sistema definiscono un insieme standard di funzioni grazie alle quali le applicazioni interagiscono col nucleo, e realizzano la maggior parte dei servizi forniti dal sistema operativo che non necessitano dei pieni privilegi del nucleo.
3. **Utilità di sistema.** Le utilità di sistema sono programmi che eseguono singoli compiti specializzati di gestione. Alcune di esse potrebbero essere chiamate solo una volta per inizializzare e configurare qualche aspetto del sistema; altre — note come *demoni* nella terminologia UNIX — possono rimanere permanentemente in esecuzione per eseguire compiti come la gestione delle richieste di nuove connessioni, la gestione delle richieste d'accesso dai terminali o l'aggiornamento dei file di registrazione (*log*).

http://www.staldpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

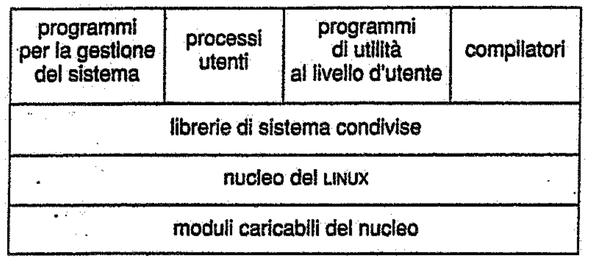


Figura 20.1 Componenti del sistema LINUX.

Nella Figura 20.1 sono illustrati i vari elementi che costituiscono un sistema LINUX completo. La distinzione più importante in questo contesto è quella fra il nucleo e tutto il resto. Tutto il codice del nucleo è eseguito dalla CPU in modo privilegiato e con piena possibilità d'accesso alle risorse fisiche del calcolatore: nel contesto del sistema LINUX ci si riferisce a questo modo privilegiato come al **modo del nucleo**, e questo è equivalente al modo monitor descritto nel Paragrafo 2.5.1. Il nucleo del LINUX non contiene codice da eseguire nel modo d'utente: tutto il codice aggiuntivo necessario che non ha bisogno di essere eseguito nel modo del nucleo si trova nelle librerie di sistema.

Anche se parecchi sistemi operativi moderni hanno adottato per i loro nuclei un'architettura basata sullo scambio di messaggi, LINUX si conforma al modello storico dello UNIX: il nucleo è un unico blocco monolitico di codice binario. La ragione principale di ciò è dovuta alla volontà di migliorare le prestazioni: poiché tutto il codice e le strutture di dati del nucleo sono contenute in un unico spazio d'indirizzi, non vi è necessità d'alcun cambio di contesto quando un processo invoca una funzione del sistema operativo o nel caso dei segnali d'interruzione. Non è solo il codice di base relativo allo scheduling e alla gestione della memoria virtuale a occupare questo spazio d'indirizzi; *tutto* il codice del nucleo, inclusi i driver dei dispositivi, il file system e il codice relativo alla gestione della rete, si trova nello stesso spazio d'indirizzi.

Il fatto che tutto il codice del nucleo sia messo nello stesso calderone non significa che non vi sia spazio per la modularità: così come le applicazioni possono caricare librerie condivise durante l'esecuzione per ottenere le parti di codice aggiuntivo di cui hanno bisogno, il nucleo può caricare (o scaricare) dinamicamente moduli durante l'esecuzione. Non è necessario che il nucleo sappia in anticipo quali moduli si potranno caricare: i moduli sono veri e propri componenti caricabili in modo indipendente.

Il nucleo costituisce la parte centrale del sistema operativo LINUX: fornisce tutti i servizi necessari per eseguire i processi e gestire l'accesso alle risorse fisiche sia in modo arbitrario sia in modo protetto. In effetti, fa tutto ciò che un sistema operativo dovrebbe essere in grado di fare. Preso isolatamente, però, il sistema operativo fornito dal nucleo non assomiglia affatto a un sistema UNIX: mancano molte sue caratteristiche, e anche quelle fornite non hanno il formato che le applicazioni si aspettano in ambiente UNIX. L'interfaccia del sistema operativo vista dalle applicazioni in esecuzione non fa propriamente parte del nucleo: le applicazioni impiegano funzioni delle librerie di sistema, che eventualmente si servono a loro volta degli opportuni servizi del nucleo.

Le librerie di sistema forniscono molti tipi di funzioni: al livello più semplice permettono alle applicazioni di far richiesta di certi servizi del nucleo. L'esecuzione di una chiamata del sistema richiede il passaggio dal modo d'utente al modo privilegiato del nucleo; i dettagli di questo passaggio variano secondo l'architettura del sistema. Le librerie si occupano di raccogliere gli argomenti della chiamata del sistema e, se è necessario, organizzano gli argomenti nella forma richiesta dall'esecuzione della chiamata del sistema.

Le librerie possono anche fornire versioni più complesse delle chiamate del sistema di base: ad esempio, le funzioni del Linguaggio C di gestione dei file con memorizzazione transitoria (*buffering*) sono tutte codificate nelle librerie di sistema; esse forniscono un più efficace controllo dell'I/O su file rispetto alle chiamate del sistema di base del nucleo. Le librerie forniscono anche funzioni che non corrispondono per niente a chiamate del sistema, ad esempio algoritmi d'ordinamento, funzioni matematiche e procedure per la manipolazione di sequenze di caratteri. Tutte le funzioni necessarie per l'esecuzione di applicazioni UNIX o POSIX sono codificate nelle librerie di sistema.

Il sistema LINUX include un'ampia varietà di programmi eseguibili nel modo d'utente: si tratta di programmi d'utilità di sistema o d'utente. I primi comprendono tutto il codice necessario per inizializzare il sistema, ad esempio per configurare i dispositivi di rete o per caricare i moduli del nucleo. Anche i server permanentemente in esecuzione sono considerati programmi d'utilità del sistema: essi gestiscono le richieste d'accesso al sistema da parte degli utenti, le richieste di connessione e le code di stampa.

Non tutti i programmi standard d'utilità assolvono funzioni chiave d'amministrazione del sistema: l'ambiente UNIX per l'utente contiene un gran numero di programmi d'utilità che aiutano a eseguire attività quotidiane come elencare il contenuto delle directory, spostare o cancellare i file, o mostrare il contenuto di un file. Programmi d'utilità più complessi si possono usare per elaborare un testo, ad esempio per ordinarlo o per cercare fra i suoi elementi certe sequenze di caratteri. Nel loro insieme questi programmi d'utilità costituiscono una serie di strumenti che ogni utente si aspetta di trovare in un sistema UNIX; questi, pur non eseguendo alcuna funzione del sistema operativo, sono una parte importante del sistema LINUX.

20.3 Moduli del nucleo

Il nucleo del LINUX è in grado di caricare e scaricare su richiesta proprie parti di codice: questi moduli caricabili sono eseguiti in modo privilegiato, e hanno quindi pieno accesso alle risorse fisiche del calcolatore. In teoria non c'è limite a ciò che un modulo del nucleo può fare; tipicamente un modulo è la codifica di un driver di dispositivo, un file system o un protocollo di comunicazione.

I moduli del nucleo sono convenienti per diverse ragioni. Il codice sorgente del LINUX è gratuito, e quindi chiunque volesse scrivere codice del nucleo potrebbe compilare il nucleo modificato e riavviare il sistema per caricare la nuova funzione, ma la ricom-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

pilazione, il collegamento e il ricaricamento dell'intero nucleo è un processo laborioso che si dovrebbe ripetere molte volte durante la stesura di un driver. Usando i moduli del nucleo non è necessario costituire un nuovo nucleo per provare un nuovo driver: il driver si può compilare separatamente e può essere caricato dal nucleo già in uso. Naturalmente, dopo che un nuovo driver è stato scritto, si può distribuire come modulo cosicché anche altri utenti potranno trarne beneficio senza dover ricompilare integralmente il nucleo.

Quest'ultimo punto ha altre implicazioni: essendo soggetto a licenza GPL, il nucleo del LINUX non può essere distribuito insieme con nuovi componenti, a meno che anch'essi non siano soggetti a licenza GPL e il loro codice sorgente non sia disponibile su richiesta. L'interfaccia a moduli del nucleo permette a terzi di scrivere e distribuire alle loro condizioni driver di dispositivi o file system che non si potrebbero distribuire secondo le norme GPL.

I moduli del nucleo permettono a un sistema LINUX di funzionare grazie a un nucleo minimale standard privo di driver incorporati: i necessari driver dei dispositivi si possono caricare in modo esplicito nella fase d'avviamento del sistema, o possono essere caricati automaticamente dal sistema su richiesta e scaricati dopo l'uso. Un driver per lettore di CD-ROM, ad esempio, può essere caricato nel momento in cui si monta un CD nel file system, e scaricato nuovamente dalla memoria quando si smonta il CD dal file system. Gli elementi principali della struttura a moduli del LINUX sono tre:

1. **La gestione dei moduli** permette il caricamento dei moduli nella memoria e la loro comunicazione con il resto del sistema
2. **La registrazione dei driver** permette di comunicare al resto del nucleo la disponibilità di un nuovo driver.
3. **La risoluzione dei conflitti** è un meccanismo che permette a un driver di riservarsi l'uso di certe risorse fisiche e di proteggerle da un uso accidentale da parte di un altro driver.

20.3.1 Gestione dei moduli

Il caricamento di un modulo è un processo più complesso del semplice trasferimento nella memoria del codice binario che lo costituisce: il sistema deve anche accertarsi del fatto che ogni riferimento fatto dal modulo a simboli o punti d'ingresso del nucleo sia aggiornato per puntare alle corrette locazioni di memoria all'interno dello spazio d'indirizzi del nucleo. Il sistema LINUX affronta questo problema dividendo il caricamento in due parti: la gestione di sezioni di codice del modulo nella memoria del nucleo, e quella dei simboli cui i moduli possono fare riferimento.

Il sistema LINUX mantiene una tabella interna dei simboli nel nucleo: essa non contiene l'intero insieme di simboli definiti durante la compilazione del nucleo, ma solo i simboli esplicitamente esportati dal nucleo. L'insieme di simboli esportati costituisce una ben definita interfaccia tramite la quale un modulo può interagire con il nucleo.

Sebbene l'esportazione di un simbolo da una funzione del nucleo presupponga la richiesta esplicita del programmatore, nessuno sforzo aggiuntivo è necessario per importare questi simboli in un modulo; chi scrive il modulo usa semplicemente il meccanismo

standard di collegamento esterno del Linguaggio C: tutti i simboli esterni cui il modulo fa riferimento ma che non sono dichiarati dal modulo stesso sono contrassegnati come irrisolti nel codice binario finale. Quando un modulo deve essere caricato nella memoria, un programma d'utilità del sistema lo esamina per rilevare questi riferimenti irrisolti: il valore d'ogni simbolo esterno irrisolto si cerca nella tabella dei simboli del nucleo, e gli indirizzi corretti per quei simboli in base al nucleo attualmente in esecuzione sono sostituiti nel codice del modulo. Solo a questo punto il modulo può essere caricato dal nucleo: se il programma d'utilità del sistema non riesce a risolvere un riferimento cercandone l'indirizzo nella tabella dei simboli del nucleo, il modulo non è accettato dal sistema.

Il caricamento di un modulo avviene in due fasi. In primo luogo, il caricatore, un programma d'utilità del sistema, richiede al nucleo di riservare per il modulo una regione continua di memoria virtuale di nucleo; il nucleo restituisce l'indirizzo dell'area di memoria assegnata, e il caricatore può usare quest'indirizzo per rilocare il codice di macchina del modulo rispetto all'indirizzo di caricamento corretto. Una seconda chiamata del sistema passa al nucleo il nuovo modulo, insieme con ogni tabella di simboli esterni che il modulo chiede di esportare: il modulo è copiato nello spazio di memoria precedentemente riservato, e si aggiorna la tabella dei simboli del nucleo in base ai nuovi simboli per un eventuale uso da parte d'altri moduli non ancora caricati.

L'ultimo componente della gestione dei moduli è il meccanismo di richiesta dei moduli. Il nucleo definisce un'interfaccia di comunicazione alla quale il programma di gestione dei moduli si può collegare; grazie a ciò il nucleo potrà comunicare al programma di gestione le richieste d'uso relative al driver di un dispositivo, a un file system o a un servizio di rete i cui moduli non sono ancora stati caricati, dando così al gestore la possibilità di caricare il modulo appropriato; le richieste saranno soddisfatte solo dopo il completamento di quest'operazione. Il programma di gestione interroga il nucleo a intervalli regolari per appurare quali moduli dinamicamente caricati non sono più in uso, scaricando i moduli che non sono correntemente necessari.

20.3.2 Registrazione dei driver

Un modulo che sia stato caricato rimane una regione isolata della memoria fino a quando non comunica al nucleo quali servizi è in grado di fornire. Il nucleo mantiene tabelle dinamiche di tutti i driver noti, e fornisce un insieme di procedure che permettono di aggiungere o rimuovere un driver da queste tabelle in qualunque momento. Il nucleo assicura l'avviamento della procedura d'inizializzazione d'ogni nuovo modulo caricato, e l'attivazione della procedura di chiusura prima che esso sia scaricato: queste procedure sono responsabili della registrazione dei servizi forniti dal modulo.

Un modulo può registrare molti tipi di driver; un certo modulo può scegliere fra questi e, se lo desidera, registrare più di un driver. Un certo driver, ad esempio, potrebbe voler registrare due meccanismi distinti per l'accesso a un dispositivo. Le tabelle di registrazione includono i seguenti elementi:

- **Driver dei dispositivi.** Questi driver comprendono i dispositivi a caratteri (come stampanti, terminali e mouse), i dispositivi a blocchi (fra i quali tutte le unità a disco), e i dispositivi d'interfacciamento alla rete.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- **File system.** Un file system è un'entità che comprende le procedure di sistema del LINUX relative a un file system virtuale: può trattarsi della gestione di un formato da usare per la memorizzazione di file nei dischi, oppure di un file system di rete come l'NFS, o ancora di un file system virtuale i cui contenuti sono generati su richiesta, come il file system *proc*.
- **Protocolli di rete.** Un modulo può essere la concreta realizzazione di un intero protocollo di rete, come ad esempio IPX, o più semplicemente un insieme di norme per il filtraggio dei pacchetti di dati per una barriera di sicurezza di rete.
- **Formato binario.** Questo formato specifica un modo per riconoscere e caricare un nuovo tipo di file eseguibile.

Inoltre, un modulo può registrare un nuovo insieme di elementi nelle tabelle `sysctl` e `/proc`, in modo che lo stesso modulo si possa configurare dinamicamente (Paragrafo 20.7.3).

20.3.3 Risoluzione dei conflitti

Le versioni commerciali del sistema operativo UNIX sono di solito concepite per essere eseguite sui calcolatori di un singolo produttore: un vantaggio di questa situazione è che chi sviluppa programmi ha un'idea piuttosto precisa delle possibili configurazioni dei calcolatori. L'architettura di un PC, d'altro canto, è disponibile in un gran numero di configurazioni diverse, e con molti possibili driver per dispositivi come le schede di rete, controller SCSI e schede grafiche. Il problema di trattare le differenti configurazioni dei calcolatori si aggrava non appena si adotta un criterio modulare di gestione dei driver dei dispositivi, perché l'insieme dei dispositivi attivi diviene variabile dinamicamente.

Il sistema LINUX fornisce un meccanismo centrale di risoluzione dei conflitti che aiuta a regolare l'accesso a certe risorse fisiche. I suoi obiettivi sono i seguenti:

- impedire che moduli diversi entrino in conflitto per l'accesso alle risorse fisiche;
- impedire che una procedura di autoverifica di un certo driver — cioè i tentativi del driver di determinare la configurazione del dispositivo — interferisca con driver già presenti;
- risolvere i conflitti che possono nascere fra diversi driver che tentino di accedere alle stesse risorse fisiche; ad esempio, sia il driver di una stampante collegata a una porta parallela sia il driver di rete IP relativo a una porta parallela (*paralle-line IP* — PLIP) potrebbero cercare di comunicare con la porta parallela.

Per raggiungere questi scopi il nucleo mantiene una lista delle risorse fisiche assegnate. Il PC ha un numero limitato di possibili porte di I/O (indirizzi nel suo spazio d'indirizzi di I/O), linee per le interruzioni e canali DMA; da un driver di un dispositivo che voglia accedere a una di queste risorse ci si aspetta che prima di tutto prenoti la risorsa presso il nucleo. Questo prerequisito, fra l'altro, permette all'amministratore del sistema di determinare esattamente quali risorse siano state assegnate da quale driver a ogni dato istante.

Ci si attende che un modulo usi questo meccanismo per prenotare ogni risorsa fisica che intenda impiegare. Se la richiesta di prenotazione non è accettata, ad esempio perché la risorsa non esiste o è già in uso, sta al modulo decidere quale tipo d'azione intraprendere: potrebbe dichiarare fallita la sua procedura d'inizializzazione e richiedere di essere scaricato, oppure continuare tentando di usare risorse fisiche alternative.

20.4 Gestione dei processi

Un processo è il contesto fondamentale all'interno del quale tutte le attività richieste dagli utenti sono assistite dal sistema operativo. Per essere compatibile con altri sistemi UNIX, il sistema operativo LINUX deve adottare un modello dei processi simile a quello di altre versioni del sistema UNIX: tuttavia, LINUX si comporta in modo diverso in alcuni punti chiave. In questo paragrafo si richiama il tradizionale modello UNIX dei processi descritto nel Paragrafo A.3.2, e si presenta il modello LINUX basato sui thread.

20.4.1 Modello `fork-exec` dei processi

Il principio fondamentale della gestione dei processi nello UNIX è di separare il problema in due parti: la creazione di processi e l'esecuzione di un nuovo programma. La chiamata del sistema `fork` assolve il primo compito, mentre l'esecuzione di un nuovo programma è avviata dalla chiamata del sistema `execve`: queste due funzioni sono nettamente differenti, tanto che si può creare un nuovo processo tramite la `fork` senza che si debba necessariamente avviare l'esecuzione di un nuovo programma — il processo figlio continua a eseguire esattamente lo stesso programma del processo genitore; similmente, l'esecuzione di un nuovo programma non richiede la creazione di un nuovo processo: un qualunque processo può impartire la chiamata del sistema `execve` in qualunque momento, nel qual caso il programma correntemente in esecuzione è immediatamente forzato a terminare, e il nuovo programma è avviato nel contesto del processo esistente.

Il vantaggio di questo modello è la sua gran semplicità: invece di dover specificare ogni dettaglio riguardante l'ambiente di un nuovo programma tramite la chiamata del sistema che ne avvia l'esecuzione, i nuovi programmi sono semplicemente eseguiti nell'ambiente esistente. Se un processo genitore desidera modificare l'ambiente d'esecuzione di un nuovo programma, può eseguire una `fork` e poi, continuando a eseguire il programma originale nell'ambito del processo figlio, eseguire tutte le chiamate del sistema necessarie per modificare quel processo figlio prima di avviare infine il nuovo programma.

Nel sistema UNIX, quindi, un processo racchiude tutte le informazioni di cui il sistema operativo necessita per gestire il contesto di una singola esecuzione di un singolo programma; nel LINUX, questo contesto può essere suddiviso in un certo numero di sezioni specifiche. A grandi linee, le proprietà di un processo si dividono in tre gruppi: l'identità, l'ambiente e il contesto del processo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.4.1.1 Identità dei processi

L'identità di un processo consiste principalmente dei seguenti elementi:

- **Identificatore del processo** (*process identifier* — PID). A ogni processo è associato un identificatore unico detto PID: esso si usa per segnalare un certo processo al sistema operativo quando un'applicazione invoca una chiamata del sistema riferita a quel processo. Ulteriori identificatori associano un processo a un gruppo di processi (tipicamente un albero di processi generati da `fork` a partire da un unico comando d'utente) o a una sessione di lavoro.
- **Credenziali.** Ogni processo deve avere un identificatore d'utente e uno o più identificatori di gruppo associati che determinano per un processo i diritti d'accesso alle risorse del sistema e ai file (i gruppi di utenti sono discussi nel Paragrafo 11.6.2; i gruppi di processi non sono trattati).
- **Personalità.** Il concetto di personalità di un processo non fa parte del tradizionale bagaglio dello UNIX, ma nel sistema LINUX ogni processo ha un identificatore di personalità associato che può modificare lievemente la semantica di certe chiamate del sistema: questi identificatori sono principalmente usati da librerie di simulazione per ottenere la compatibilità di certe chiamate del sistema con certe specifiche versioni dello UNIX.

Un processo ha un controllo limitato dei suoi stessi identificatori: gli identificatori del gruppo e della sessione di lavoro possono essere cambiati se il processo desidera avviare un nuovo gruppo o una nuova sessione; le credenziali possono essere cambiate subordinatamente a certi controlli di sicurezza; il PID di un processo, invece, non è modificabile, e lo identifica in modo unico fino alla sua terminazione.

20.4.1.2 Ambiente di un processo

L'ambiente di un processo è ereditato dal suo genitore, ed è composto di due vettori: il vettore degli argomenti e il vettore d'ambiente. Il **vettore degli argomenti** elenca semplicemente gli argomenti della riga di comando usata per invocare il programma in esecuzione, e comincia per convenzione con il nome del programma stesso. Il **vettore d'ambiente** è un elenco di coppie della forma "*NOME* = *VALORE*" che associano alle variabili d'ambiente valori arbitrari. La descrizione dell'ambiente non è contenuta nella memoria del nucleo ma nello spazio d'indirizzi nel modo d'utente del processo, ed è il primo dato sulla pila del processo.

Questi due vettori non sono alterati a seguito della creazione di un nuovo processo: il processo figlio eredita l'ambiente del processo genitore. L'avvio di un nuovo programma, invece, implica la costituzione di un ambiente interamente nuovo: un processo che impartisca la chiamata `execve` deve anche specificare l'ambiente del nuovo programma tramite variabili d'ambiente che sono passate dal nucleo al nuovo programma, e sostituiscono l'ambiente attuale del processo. A parte questa circostanza, i vettori d'ambiente e degli argomenti sono ignorati dal nucleo; la loro interpretazione è lasciata interamente alle librerie e alle applicazioni.

Il passaggio delle variabili d'ambiente da un processo all'altro e il meccanismo d'ereditarietà costituiscono un modo flessibile di scambiare informazioni fra i componenti del sistema eseguiti nel modo d'utente. Alcune importanti variabili d'ambiente hanno un significato convenzionale correlato a certe parti del sistema: i valori della variabile `TERM`, ad esempio, denotano il tipo di terminale usato durante una sessione di lavoro, e molti programmi sfruttano questo fatto per stabilire i modi d'esecuzione di certe operazioni che hanno effetti sullo schermo dell'utente (spostamento del cursore, scorrimento di parti di testo, e così via). I programmi in grado di comunicare in più lingue usano la variabile `LANG` per stabilire in quale lingua mostrare i messaggi.

Il meccanismo delle variabili d'ambiente personalizza il sistema operativo localmente, processo per processo, invece che globalmente: ogni utente può scegliere una lingua o un programma particolare indipendentemente dalle scelte altrui.

20.4.1.3 Contesto di un processo

L'identità di un processo e le proprietà dell'ambiente sono di solito stabilite al momento della creazione di un processo, e non mutano fino alla sua terminazione, anche se un processo potrebbe decidere di cambiare il suo ambiente o alcuni aspetti della sua identità. Il contesto di un processo, d'altra parte, è lo stato del programma in esecuzione in ogni istante: esso cambia continuamente.

- **Contesto di scheduling.** La parte più importante del contesto di un processo è il suo contesto di scheduling: le informazioni di cui lo scheduler necessita per sospendere o riavviare il processo, comprese le copie di tutti i registri del processo. I registri in virgola mobile sono memorizzati separatamente, e ripristinati solo quando è necessario, per rendere più efficiente il salvataggio di stato dei processi che non usano l'aritmetica in virgola mobile. Il contesto di scheduling comprende anche informazioni sulla priorità di scheduling e su ogni segnale che attende d'essere recapitato al processo. Una parte cruciale del contesto di scheduling è la pila di nucleo del processo: un'area di memoria nello spazio d'indirizzi del nucleo riservata esclusivamente all'uso di codice eseguito nel modo del nucleo; sia le chiamate del sistema sia la gestione delle interruzioni occorrenti durante l'esecuzione del processo usano questa pila.
- **Contabilizzazione delle risorse.** Il nucleo mantiene informazioni sulle risorse correntemente impiegate da ciascun processo, e anche sulla quantità totale di risorse impiegate durante la sua intera esistenza.
- **Tabella dei file.** È un vettore di puntatori alle strutture di dati del nucleo relative ai file. Quando un processo impartisce una chiamata del sistema di I/O relativa a un file, lo identifica tramite il suo indice in questa tabella.
- **Contesto del file-system.** Mentre la tabella dei file elenca i file aperti esistenti, il contesto del file system riguarda le richieste di apertura di nuovi file. Le directory da cui avviare la ricerca di un nuovo file sono memorizzate qui.

http://www.chtpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

- **Tabella dei gestori dei segnali.** I sistemi UNIX possono recapitare a un processo segnali asincroni come risposta a un evento esterno. Questa tabella definisce quali procedure nello spazio d'indirizzi del processo si devono invocare quando si riceve un certo segnale.
- **Contesto della memoria virtuale.** Il contesto della memoria virtuale descrive esaurientemente i contenuti dello spazio d'indirizzi di un processo, ed è trattato nel Paragrafo 20.6.

20.4.2 Processi e thread

La maggior parte dei moderni sistemi operativi è in grado di gestire sia processi sia thread. Anche se le differenze fra il significato dei due termini variano da caso a caso, si possono distinguere come segue: i *processi* rappresentano l'esecuzione di un unico programma, mentre i *thread* rappresentano contesti d'esecuzione distinti e concorrenti all'interno di un singolo processo che esegue un singolo programma.

Due qualsiasi processi distinti hanno due spazi d'indirizzi distinti e indipendenti, anche quando condividono una parte dei contenuti delle proprie memorie virtuali; in particolare, questa condivisione non è mai totale. Per contro due thread all'interno di uno stesso processo condividono il *medesimo*, anziché un *simile* spazio d'indirizzi. Qualunque cambiamento apportato da un thread alla memoria virtuale è immediatamente visibile agli altri thread dello stesso processo, perché essi sono eseguiti all'interno di un solo spazio d'indirizzi.

Ci sono diversi modi per realizzare i thread: possono essere oggetti all'interno del nucleo posseduti da un processo, oppure entità completamente indipendenti; possono anche non essere realizzati affatto all'interno del nucleo, ma al livello d'utente tramite applicazioni e librerie con l'aiuto della gestione fornita dal nucleo dei segnali d'interruzione provenienti dal temporizzatore.

Il nucleo del LINUX affronta le differenze fra thread e processi in maniera molto semplice: adotta esattamente la stessa rappresentazione interna per entrambi. Un thread è solo un nuovo processo che condivide lo stesso spazio d'indirizzi del genitore. La differenza fra thread e processo diviene però evidente al momento della creazione di un nuovo thread, poiché la chiamata del sistema da usare non è più la `fork` ma la `clone`: essa crea un nuovo processo che ha la sua specifica identità, ma cui è permesso condividere le strutture di dati del genitore; una `fork` avrebbe creato per il processo figlio un contesto interamente nuovo.

Questo modo di procedere è praticabile perché il sistema LINUX non mantiene l'intero contesto di un processo all'interno delle strutture di dati principali di un processo, ma memorizza separatamente ogni sottocontesto: il contesto del file system, la tabella dei descrittori di file, la tabella dei gestori dei segnali e il contesto della memoria virtuale sono contenuti in strutture di dati separate. La struttura di dati di un processo contiene semplicemente puntatori a queste altre strutture, cosicché qualunque altro processo può facilmente condividere uno dei sottocontesti puntando alla relativa area di memoria.

La chiamata del sistema `clone` accetta un argomento che specifica quali sottocontesti condividere e quali copiare, al momento della creazione del nuovo processo, il quale ha sempre una nuova identità e un nuovo contesto di scheduling; secondo gli argomenti passati, però, può creare una nuova struttura di dati per un certo sottocontesto il cui contenuto è una copia del corrispondente sottocontesto del processo genitore, oppure condividere il sottocontesto del genitore. La chiamata del sistema `fork` non è altro che un caso particolare di `clone` nel quale si copiano tutti i sottocontesti e nessuno è condiviso. La chiamata del sistema `clone` permette alle applicazioni di controllare nei dettagli la condivisione di strutture di dati fra due thread.

Il documento POSIX.1c definisce un'interfaccia di programmazione standard che permette alle applicazioni di eseguire thread multipli: le librerie di sistema del LINUX realizzano quest'interfaccia in due modi differenti. Un'applicazione può scegliere di usare il pacchetto di realizzazione dei thread basato sul modo d'utente, oppure quello basato sul modo del nucleo: il primo evita i ritardi dovuti allo scheduling e alle chiamate del sistema che occorrono durante l'interazione dei thread, ma è limitato dal fatto che tutti i thread sono eseguiti all'interno di un unico processo; il secondo usa la chiamata del sistema `clone` per realizzare la stessa interfaccia di programmazione ma, poiché in questo modo si creano contesti di scheduling multipli, ha il vantaggio di permettere alle applicazioni di eseguire più thread contemporaneamente in sistemi con più unità d'elaborazione; i thread multipli possono inoltre compiere più chiamate del sistema simultaneamente.

20.5 Scheduling

Lo scheduling consiste nell'assegnazione del tempo di CPU ai diversi task all'interno di un sistema operativo. Di norma si pensa che l'attività di scheduling consista nell'esecuzione e nella sospensione dei processi, ma un altro aspetto dello scheduling importante per il sistema LINUX è l'esecuzione dei vari task del nucleo, che include sia i task richiesti da un processo in esecuzione, sia i task interni eseguiti per conto di un driver di dispositivo.

20.5.1 Sincronizzazione del nucleo

Il modo in cui il nucleo esegue lo scheduling delle sue stesse operazioni è essenzialmente diverso dal modo in cui esegue lo scheduling dei processi. La richiesta d'esecuzione di codice nel modo del nucleo può avvenire in due modi: un programma in esecuzione può richiedere un servizio del sistema operativo esplicitamente, tramite una chiamata del sistema, o implicitamente, ad esempio nel caso di un'eccezione di pagina mancante; alternativamente, il driver di un dispositivo può generare un segnale d'interruzione che induce l'esecuzione di un gestore di tale interruzione definito dal nucleo.

Il problema che il nucleo si trova a dover affrontare è che questi task potrebbero tentare di accedere alle stesse strutture di dati interne: se una procedura di gestione di un segnale d'interruzione interrompe un task del nucleo che sta usando certe strutture di dati, la procedura in questione non può accedere a quelle stesse strutture di dati se non a rischio di alterarne i dati. Quest'osservazione è connessa all'idea delle sezioni critiche,

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

cioè parti di codice che condividono certe strutture di dati e che non si possono eseguire in modo concorrente.

In considerazione di quanto s'è detto, la sincronizzazione del nucleo è un problema più complicato dello scheduling dei processi: è necessario un meccanismo che impedisca l'interruzione di una sezione critica del nucleo da parte di un'altra sezione critica.

La prima parte della soluzione adottata dal sistema LINUX consiste nell'impedire la sospensione dell'ordinario codice di nucleo: di solito il nucleo, quando riceve un segnale d'interruzione proveniente dal temporizzatore, chiama lo scheduler dei processi, il quale almeno potenzialmente può sospendere l'esecuzione del processo corrente e riprendere l'esecuzione di un altro — la normale partizione del tempo d'ogni sistema UNIX. Quando però il nucleo riceve un segnale d'interruzione proveniente dal temporizzatore durante l'esecuzione di una procedura di servizio del nucleo, l'operazione di scheduling non avviene immediatamente: s'imposta l'indicatore (*flag*) need_resched del nucleo in modo da indicare al nucleo di eseguire lo scheduler dopo che la chiamata del sistema ha terminato e il controllo sta per essere restituito al codice utente.

Una volta che una parte di codice del nucleo è in esecuzione si può garantire che nessun'altra parte di codice del nucleo sarà eseguita fino a quando non si verificherà uno dei fatti seguenti:

- un'interruzione;
- un'eccezione di pagina mancante;
- una chiamata allo scheduler da parte dello stesso codice del nucleo.

I segnali d'interruzione sono un problema solo se contengono essi stessi sezioni critiche: le interruzioni provenienti dal temporizzatore non provocano mai direttamente lo scheduling di un processo, ma richiedono solamente che l'operazione di scheduling sia eseguita appena possibile, quindi una qualunque interruzione non può influenzare l'ordine d'esecuzione di quel codice di nucleo che non sia una procedura di gestione di un segnale d'interruzione. Quando la procedura di gestione dell'interruzione ha terminato, l'esecuzione riprende semplicemente dallo stesso codice del nucleo che era in esecuzione al momento dell'arrivo dell'interruzione.

Le eccezioni di pagina mancante rappresentano un potenziale problema; se tenta di leggere o scrivere nella memoria d'utente, una procedura del nucleo potrebbe generare un'eccezione di pagina mancante la cui gestione richiede l'esecuzione di un'operazione di I/O da un disco, con la conseguente sospensione del processo correntemente in esecuzione fino al completamento dell'operazione di I/O. Analogamente, se la procedura d'esecuzione di una chiamata del sistema chiama lo scheduler mentre il sistema opera nel modo del nucleo, cosa che può avvenire o esplicitamente tramite una chiamata diretta al codice dello scheduler o implicitamente a causa della chiamata di una funzione che determini l'attesa del completamento di un'operazione di I/O, il processo sarà sospeso e avverrà un'operazione di scheduling. L'esecuzione del processo continuerà più tardi sempre nel modo del nucleo, a partire dall'istruzione che segue la chiamata allo scheduler.

Il codice del nucleo può allora assumere che non sarà mai interrotto da un altro processo, e che non c'è quindi necessità di prendere particolari precauzioni per proteggere le sezioni critiche: l'unico aspetto di cui occorre assicurarsi è che le sezioni critiche non contengano riferimenti alla memoria d'utente o attese per il completamento d'operazioni di I/O.

La seconda tecnica di protezione che LINUX usa si applica alle sezioni critiche incluse nelle procedure di gestione dei segnali d'interruzione. Lo strumento fondamentale in questo caso è l'architettura di controllo delle interruzioni della CPU: disabilitando le interruzioni durante l'esecuzione di una sezione critica, il nucleo si assicura di poter procedere senza rischiare accessi concorrenti a strutture di dati condivise.

Disabilitare le interruzioni comporta comunque una penalizzazione: le istruzioni di abilitazione e disabilitazione delle interruzioni sono onerose nella maggior parte delle architetture; inoltre ogni operazione di I/O è sospesa fino a quando le interruzioni rimangono disabilitate, e tutti i dispositivi che attendono di essere serviti dovranno aspettare la riabilitazione delle interruzioni, cosa che ha effetti negativi sulle prestazioni. Il nucleo del LINUX usa un'architettura di sincronizzazione che permette l'esecuzione di lunghe regioni critiche senza richiedere la disabilitazione delle interruzioni per tutta la durata della loro esecuzione. Questa possibilità torna particolarmente utile per il codice di rete: un segnale d'interruzione relativo a un driver di un dispositivo di rete può comunicare l'arrivo di un intero pacchetto di dati, e quindi implicare l'esecuzione di una gran quantità di codice per decodificare, instradare e inoltrare il pacchetto tramite la procedura di gestione dell'interruzione.

Il sistema LINUX realizza in pratica quest'architettura dividendo le procedure di gestione delle interruzioni in due parti: la parte superiore e quella inferiore. La parte superiore è un'ordinaria procedura di gestione delle interruzioni, ed è eseguita solo dopo la disabilitazione condizionale delle interruzioni (segnali d'interruzione di priorità più alta possono interrompere la procedura, ma segnali d'interruzione di priorità uguale o inferiore sono disabilitati). La parte inferiore della procedura di gestione, invece, è eseguita, con tutte le interruzioni abilitate, da un minischeduler in grado di assicurare che le parti inferiori non s'interrompano mai; il minischeduler è automaticamente chiamato ogni volta che una procedura di gestione di un segnale d'interruzione termina.

Questa divisione permette al nucleo di completare l'esecuzione di complesse elaborazioni come risposta a un'interruzione senza che il nucleo debba preoccuparsi di essere interrotto. Se un'interruzione si verifica durante l'esecuzione di una parte inferiore, anche se l'interruzione può richiedere l'esecuzione della stessa parte inferiore, essa sarà differita fino alla terminazione della parte inferiore correntemente in esecuzione. L'esecuzione di una parte inferiore può essere interrotta da una parte superiore, ma non può mai essere interrotta da una parte inferiore simile.

L'architettura a parti è completata da un meccanismo di disabilitazione selettiva delle parti inferiori durante l'esecuzione di codice di nucleo ordinario. Usando questo sistema il nucleo può codificare facilmente le sezioni critiche: le sezioni critiche dei gestori dei segnali d'interruzione sono poste nelle parti inferiori, e quando il codice di nucleo ordi-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nario vuole eseguire una sua sezione critica, può disabilitare ogni attinente parte inferiore per evitare di essere interrotto da altre sezioni critiche. Alla fine della sezione critica il nucleo riabilita le parti inferiori ed esegue le parti inferiori che sono state poste in attesa dall'esecuzione delle loro corrispondenti parti superiori nel corso della sezione critica.

Nella Figura 20.2 sono riassunti i vari livelli di protezione delle interruzioni all'interno del nucleo; ogni livello può essere interrotto da codice eseguito a un livello superiore, ma non sarà mai interrotto da codice eseguito allo stesso livello o a livelli inferiori, eccezion fatta, naturalmente, per il codice eseguito nel modo d'utente: un processo utente può sempre essere sospeso da un altro processo utente a causa dello scadere del quanto di tempo.

20.5.2 Scheduling dei processi

Quando il nucleo si trova a compiere un'operazione di scheduling, deve scegliere quale processo eseguire; questa circostanza si verifica o quando scade il quanto di tempo assegnato a un processo, o quando un processo di nucleo in esecuzione si è bloccato nell'attesa dell'arrivo di un segnale di riattivazione. Il sistema LINUX adotta due distinti algoritmi di scheduling: uno è un algoritmo a partizione del tempo per l'equa condivisione del tempo di CPU da parte di più processi, e l'altro è concepito per elaborazioni in tempo reale dove le priorità assolute sono più importanti dell'equità.

Una parte dell'identità di ogni processo è costituita dalla classe di scheduling, in base alla quale si sceglie quale di questi algoritmi applicare al processo; le classi adottate dal sistema LINUX sono definite dalle estensioni dello standard POSIX all'elaborazione in tempo reale (POSIX.4, oggi noto come POSIX.1b).

Per i processi eseguiti a partizione del tempo, il sistema LINUX usa un algoritmo a priorità basato sui crediti: ogni processo possiede un certo numero di crediti di scheduling; quando occorre assegnare la CPU a un nuovo processo, la scelta cade su chi ha il massimo numero di crediti. Ogni volta che si presenta un segnale d'interruzione proveniente dal temporizzatore, il processo attualmente in esecuzione perde un credito; quando i suoi crediti sono ridotti a zero viene sospeso e si sceglie un altro processo per l'esecuzione.

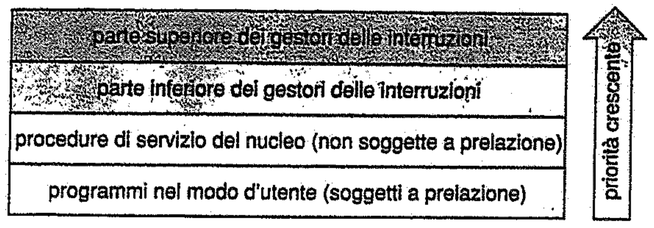


Figura 20.2 Livelli di protezione delle interruzioni.

Se tutti i processi eseguibili hanno un numero di crediti pari a zero il sistema LINUX esegue un'operazione di riassegnamento dei crediti, aggiungendo crediti a *tutti* i processi del sistema, e non solo a quelli eseguibili, secondo la regola seguente:

$$crediti = \frac{crediti}{2} + priorità$$

Quest'algoritmo pondera due fattori: la storia e la priorità dei processi. La metà dei crediti già in possesso di un processo gli sono riassegnati; si tratta di un modo di tener conto del comportamento recente del processo stesso: i processi che sono spesso in esecuzione esauriscono in fretta la loro scorta di crediti, ma quelli che rimangono sospesi per la maggior parte del tempo accumulano crediti grazie alle operazioni di riassegnamento dei crediti, e finiscono quindi con l'avere un numero di crediti maggiore dopo ogni operazione di questo tipo. Questo sistema dà automaticamente un'alta priorità ai processi interattivi o con prevalenza di I/O, e in effetti in questo caso un basso tempo di risposta è importante.

L'uso della priorità di un processo nel calcolo dei nuovi crediti permette di regolare con precisione il grado di priorità assegnato a un processo; ai programmi eseguiti in sottofondo si può assegnare una bassa priorità, e questi riceveranno automaticamente meno crediti rispetto ai processi interattivi degli utenti, usufruendo così di un tempo di CPU più breve rispetto a processi anche simili ma con più alta priorità. Questa strategia a priorità è il modo in cui il sistema LINUX realizza il meccanismo *nice* di gestione delle priorità dello UNIX.

Lo scheduling in tempo reale del LINUX è ancora più semplice; le due classi di scheduling richieste dal documento POSIX.1b, la classe FCFS e quella RR (Paragrafi 6.3.1 e 6.3.4 rispettivamente), sono entrambe disponibili, e in tutti e due i casi a ogni processo si assegna una priorità. Nello scheduling a partizione del tempo, però, processi con priorità diverse sono almeno in una certa misura in competizione, mentre nello scheduling in tempo reale lo scheduler esegue sempre il processo con priorità più alta, oppure, a parità di priorità, il processo che attende da più tempo. L'unica differenza fra lo scheduling FCFS e quello RR è che un processo FCFS continua l'esecuzione fino alla terminazione o fino a che esso stesso si blocchi, mentre un processo RR può essere sospeso allo scadere del suo quanto di tempo, nel qual caso è posto alla fine della coda di scheduling; la conseguenza di ciò è che i processi RR della stessa priorità si comportano mutuamente come in un ordinario sistema a partizione del tempo.

Lo scheduling in tempo reale del LINUX è in tempo reale debole, non in tempo reale stretto; lo scheduler offre rigide garanzie sulle priorità relative dei processi in tempo reale, ma il nucleo non fornisce alcuna garanzia sulla rapidità con cui un processo in tempo reale pronto per l'esecuzione sarà effettivamente eseguito una volta che sia stato scelto dallo scheduler per l'esecuzione: si ricordi a questo proposito che l'esecuzione del codice del nucleo non può mai essere sospesa a causa di codice d'utente in attesa, perciò se un'interruzione dovesse rendere eseguibile un processo in tempo reale mentre il nucleo sta eseguendo una chiamata del sistema per conto di un altro processo, il processo in tempo reale dovrà rassegnarsi ad attendere la terminazione o il blocco della chiamata del sistema attualmente in esecuzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.5.3 Multielaborazione simmetrica

La versione del *Kernel 2.0* fu il primo nucleo stabile del LINUX in grado di gestire le architetture per la multielaborazione simmetrica (SMP): processi o thread distinti si possono eseguire in parallelo su unità d'elaborazione distinte; tuttavia, al fine di tutelare le necessità di sincronizzazione del nucleo, la realizzazione dell'SMP nel LINUX stabilisce che una sola unità d'elaborazione alla volta possa eseguire codice nel modo del nucleo. Per far rispettare questa prescrizione l'SMP adotta per il nucleo un unico semaforo ad attesa attiva (*spinlock*); esso non rappresenta un problema per quei processi che non richiedono molti servizi al nucleo, mentre i processi che richiedono l'esecuzione di grandi quantità di codice di nucleo possono assistere a drastici crolli delle loro prestazioni.

Il *Kernel 2.2* rende la realizzazione dell'SMP maggiormente flessibile dividendo il singolo semaforo ad attesa attiva del nucleo in bloccaggi multipli in modo che ognuno di essi protegga un piccolo sottoinsieme delle strutture di dati del nucleo. Attualmente, la gestione dei segnali, delle interruzioni e alcune procedure di I/O impiegano bloccaggi multipli per consentire a più unità d'elaborazione di eseguire simultaneamente codice nel modo del nucleo. Il nucleo di sviluppo *Kernel 2.3* (LINUX 2.4) sembra aver rimosso completamente il singolo semaforo ad attesa attiva del nucleo.

20.6 Gestione della memoria

Il sistema di gestione della memoria del sistema LINUX ha due componenti: il primo si occupa dell'assegnazione e del rilascio di pagine, gruppi di pagine e piccoli blocchi di memoria; il secondo si occupa della gestione della memoria virtuale, la memoria indirizzabile dai processi in esecuzione.

Nel seguito si descrivono entrambi questi meccanismi, e si esamina poi il modo in cui i componenti caricabili di un nuovo programma sono portati all'interno della memoria virtuale di un processo in conseguenza di una chiamata del sistema exec.

20.6.1 Gestione della memoria fisica

Il principale strumento della gestione della memoria fisica nel nucleo del LINUX è l'**assegnatore delle pagine**: esso è responsabile dell'assegnazione e del rilascio delle pagine fisiche, ed è in grado di assegnare su richiesta gruppi di pagine fisicamente contigue. L'assegnatore usa un algoritmo detto *buddy-heap* per tener traccia delle pagine fisiche disponibili; un assegnatore di questo tipo associa coppie di unità adiacenti di memoria assegnabile: ogni regione di memoria assegnabile ha una compagna adiacente (*buddy*, appunto), e ogni volta che due regioni compagne assegnate si liberano entrambe, vengono combinate per formare una regione più ampia. Anche questa più ampia regione ha una compagna con la quale potenzialmente potrà formare una regione assegnabile ancora più ampia. Alternativamente se una richiesta per una piccola regione di memoria non

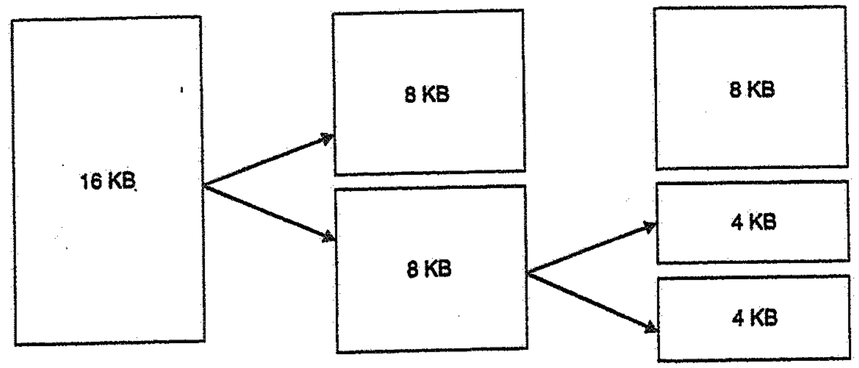


Figura 20.3 Divisione delle regioni di memoria secondo l'algoritmo *buddy-heap*.

può essere soddisfatta assegnando una regione libera esistente, una regione libera più ampia sarà suddivisa in due regioni compagne al fine di soddisfare la richiesta. Per tenere traccia delle regioni di memoria libere di ogni data dimensione permessa si usano liste concatenate distinte; nel sistema LINUX la più piccola dimensione assegnabile tramite questo meccanismo è di una singola pagina fisica. Nella Figura 20.3 è mostrato un esempio di assegnazione secondo l'algoritmo *buddy-heap*: si vuole assegnare una regione di 4 KB, ma la più piccola regione disponibile è di 16 KB. La regione è allora ricorsivamente divisa fino a ottenere la dimensione richiesta.

In ultima analisi tutte le operazioni di assegnazione di memoria nel nucleo del LINUX avvengono o staticamente, perché i driver riservano aree di memoria contigua durante l'avvio del sistema, o dinamicamente tramite l'assegnatore delle pagine; tuttavia le funzioni del nucleo non devono necessariamente usare l'assegnatore di base per riservare memoria: esistono molti sottosistemi specializzati di gestione della memoria che usano il soggiacente assegnatore delle pagine per gestire le loro proprie risorse di memoria. I più importanti sono quello della memoria virtuale, descritto nel Paragrafo 20.6.2; l'assegnatore a lunghezza variabile `kmalloc`; e le due cache permanenti del nucleo, la *buffer cache* e la cache delle pagine.

Molti componenti del sistema operativo LINUX hanno bisogno di assegnare su richiesta intere pagine, ma spesso c'è anche la necessità di blocchi di memoria più piccoli: il nucleo fornisce un assegnatore aggiuntivo per le richieste di dimensione arbitraria, in questo caso la dimensione di una richiesta non è nota in anticipo e potrebbe anche essere solo di qualche byte invece che di un'intera pagina. Questo servizio è offerto dalla funzione `kmalloc`, analoga alla `malloc` del Linguaggio C, la quale assegna su richiesta intere pagine, ma le divide poi in blocchi più piccoli. Il nucleo mantiene un insieme di liste

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

delle pagine usate dalla funzione `kmalloc`, e tutte le pagine su una data lista vengono suddivise in blocchi di una fissata dimensione. L'assegnazione della memoria implica in questo caso l'individuazione della lista appropriata e il successivo uso del primo elemento libero nella lista, o l'assegnazione di una nuova pagina e la sua suddivisione.

Sia l'assegnatore delle pagine sia la `kmalloc` sono protetti dalle interruzioni: una funzione che voglia assegnare memoria passa una certa priorità di richiesta alla funzione d'assegnazione; le procedure di gestione delle interruzioni usano una priorità atomica che garantisce o il soddisfacimento della richiesta o, nel caso non vi sia più memoria, il suo immediato rigetto. Per contro i normali processi utenti che richiedono l'assegnazione di memoria cominceranno a cercare memoria libera esistente e rimarranno in attesa fino a che non ve ne sia di disponibile. La priorità d'assegnazione può anche essere usata per specificare che la memoria richiesta serve per un'operazione DMA, il che torna utile in architetture come quelle dei PC, dove certe richieste DMA possono essere relative solo ad alcune specifiche pagine di memoria fisica.

Le regioni di memoria richieste tramite la `kmalloc` sono assegnate permanentemente fino a che non siano esplicitamente rilasciate: questa funzione non è in grado di riassegnare regioni già in uso per sopperire a carenze di memoria.

Gli altri tre sottosistemi principali che svolgono la loro propria gestione delle pagine di memoria fisica sono strettamente correlati: si tratta della *buffer cache*, della cache delle pagine e del sistema per la memoria virtuale. La *buffer cache* è la cache principale del nucleo per i dispositivi a blocchi come le unità a disco, ed è il meccanismo principale attraverso il quale sono eseguite le operazioni di I/O relative a questi dispositivi. Sia il file system proprio del sistema LINUX sia l'NFS impiegano la cache delle pagine; questa memorizza pagine intere di dati contenuti nei file e il suo uso non è limitato ai dispositivi a blocchi, ma può anche essere usata per memorizzare dati relativi alla rete. Il *sistema per la memoria virtuale* gestisce i contenuti dello spazio d'indirizzi virtuali di ciascun processo.

Questi tre sistemi interagiscono strettamente: la lettura di una pagina di dati nella cache delle pagine richiede il passaggio temporaneo attraverso la *buffer cache*; le pagine nella cache delle pagine possono anche essere indirizzate dal sistema della memoria virtuale se un processo ha associato un file a una regione del suo spazio d'indirizzi. Il nucleo tiene il conto dei riferimenti che i tre sottosistemi fanno a ogni pagina di memoria fisica, cosicché le pagine condivise da due o più di essi possono essere rilasciate quando non sono più usate da alcun sottosistema.

20.6.2 Memoria virtuale

Il sistema per la memoria virtuale del LINUX si occupa dello spazio d'indirizzi visibile a ogni processo. Esso crea pagine di memoria virtuale su richiesta e gestisce il loro caricamento da disco o il loro trasferimento nell'area d'avvicendamento su disco quando è richiesto. Nel LINUX il gestore della memoria virtuale considera lo spazio d'indirizzi di un processo da due punti di vista diversi: come insieme di regioni distinte e come insieme di pagine.

Il primo punto di vista è di natura logica, e riflette le istruzioni che il sistema per la memoria virtuale ha ricevuto riguardo all'organizzazione dello spazio d'indirizzi; quest'ultimo è visto come un insieme di regioni non intersecantesi, e ogni regione rappresenta un sottoinsieme continuo e allineato alle pagine dello spazio d'indirizzi. Ogni regione è descritta internamente da un'unica struttura di dati `vm_area_struct` che ne definisce le caratteristiche, compresi i permessi di lettura, scrittura ed esecuzione del processo, e le informazioni riguardanti tutti i file a essa associati. Le regioni riguardanti un dato spazio d'indirizzi sono organizzate in una struttura ad albero binario bilanciato che permette una rapida ricerca della regione corrispondente a un indirizzo virtuale.

Il nucleo adotta anche un secondo punto di vista riguardante lo spazio d'indirizzi, questa volta di natura fisica, realizzato grazie alle tabelle delle pagine fisiche del processo. Gli elementi della tabella delle pagine determinano l'esatta posizione corrente di ogni pagina della memoria virtuale, sia che essa si trovi in un disco sia che risieda nella memoria fisica; la gestione della memoria in accordo con questo secondo punto di vista è messa in atto per mezzo di un insieme di procedure invocate dai gestori dei segnali di eccezione del nucleo, ogniqualvolta un processo tenta di accedere a una pagina che non è in quel momento puntata da alcun elemento della tabella delle pagine. Ogni struttura `vm_area_struct` nella descrizione dello spazio d'indirizzi contiene un campo che punta a una tabella di funzioni che realizzano i servizi chiave di gestione delle pagine per ogni data regione di memoria virtuale. Tutte le richieste di operazioni relative a una pagina non disponibile sono recapitate in ultima analisi all'appropriato gestore nella tabella di funzioni della struttura `vm_area_struct`, cosicché le procedure di gestione della memoria centrale non devono essere a conoscenza dei dettagli riguardanti la gestione di ogni possibile tipo di regione di memoria.

20.6.2.1 Regioni di memoria virtuale

Il sistema LINUX opera con diversi tipi di regioni di memoria virtuale. La prima proprietà caratterizzante un tipo di memoria virtuale è la sua memoria ausiliaria, cioè la descrizione dell'origine delle pagine; nella maggior parte dei casi, si tratta di un file o non è presente. Una regione priva di memoria ausiliaria è il tipo più semplice di memoria virtuale: essa rappresenta memoria a *valori nulli*, nel senso che quando un processo tenta di leggere una pagina di questa regione ottiene come risposta semplicemente una pagina di memoria riempita di zeri.

Una regione la cui memoria ausiliaria sia un file agisce come una finestra sui contenuti di quel file: quando un processo tenta di accedere a una pagina della regione, nella tabella delle pagine è scritto l'indirizzo di una pagina della cache delle pagine del nucleo corrispondente allo scostamento appropriato all'interno del file. La stessa pagina di memoria fisica è usata sia dalla cache delle pagine sia dalle tabelle delle pagine del processo, cosicché ogni cambiamento apportato al file dal file system è immediatamente visibile a ogni processo che ha la locazione di quel file associata a una regione del suo spazio d'indirizzi. Il numero di processi che possono usare la stessa regione dello stesso file è arbitrario, e a questo scopo ciascuno di essi userà la stessa pagina di memoria fisica.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un'altra caratteristica di un tipo di regione di memoria virtuale è la sua reazione alle operazioni di scrittura; la visibilità di una regione di memoria dallo spazio d'indirizzi di un processo può infatti essere *privata* o *condivisa*. Nel primo caso, se il processo scrive in quella regione, la procedura di paginazione rileva la necessità di una copiatura su scrittura per garantire l'effetto locale dei cambiamenti; nel secondo caso, d'altra parte, l'operazione comporta l'aggiornamento dell'oggetto associato a quella regione, in modo che i cambiamenti siano immediatamente visibili a ogni altro processo che condivide la regione.

20.6.2.2 Durata di uno spazio d'indirizzi virtuale

Il nucleo crea un nuovo spazio d'indirizzi virtuale precisamente in due situazioni: l'avviamento di un nuovo programma tramite la chiamata del sistema `exec`; la creazione di un nuovo processo per mezzo della chiamata del sistema `fork`. Nel primo caso l'operazione è semplice: si assegna al nuovo programma un nuovo spazio d'indirizzi completamente vuoto; è compito delle procedure che caricano il programma preparare lo spazio d'indirizzi con regioni di memoria virtuale.

Nel secondo caso, la creazione di un nuovo processo tramite la `fork` comporta la creazione di una copia completa dello spazio d'indirizzi virtuali esistente del processo genitore. Il nucleo copia i descrittori `vm_area_struct` del processo genitore, e crea poi un nuovo insieme di tabelle delle pagine per il figlio; le tabelle del genitore sono copiate direttamente in quelle del figlio, incrementando solo il conteggio dei riferimenti a ogni pagina coinvolta: ne segue che dopo la `fork` i processi genitore e figlio condividono le stesse pagine di memoria fisica nei loro spazi d'indirizzi.

Un caso particolare si ha quando durante l'operazione di copiatura s'incontra una regione di memoria virtuale privata; tutte le pagine appartenenti a questa regione sulle quali il processo ha scritto qualcosa sono private, e gli eventuali successivi cambiamenti apportati dal genitore o dal figlio non devono ripercuotersi sulla pagina corrispondente nello spazio d'indirizzi dell'altro processo. Durante la copiatura delle tabelle si stabilisce che le pagine in questione siano soltanto leggibili e le si contrassegna per la copiatura su scrittura: fino a che nessuno dei due processi modifica queste pagine, i due processi condividono le stesse pagine di memoria fisica, ma quando un processo tenta di scrivere in una di esse, si controlla il conteggio dei riferimenti alla pagina, e se essa è ancora condivisa il processo ne copia i contenuti in una nuova pagina di memoria fisica, usando poi questa copia in luogo dell'originale. Si tratta di un meccanismo che assicura il più a lungo possibile la condivisione tra processi di pagine di dati privati; le copie si creano solo se è assolutamente necessario.

20.6.2.3 Paginazione e avvicendamento dei processi

Uno dei compiti importanti che il sistema per la memoria virtuale deve assolvere è spostare pagine di memoria dalla memoria fisica al disco quando la memoria fisica in questione è richiesta per altri scopi. I primi sistemi UNIX raggiungevano lo scopo spostando nei dischi i contenuti di interi processi in un'unica soluzione, ma le versioni moderne sfruttano maggiormente la paginazione, cioè il trasferimento di singole pagine di memoria virtuale dalla memoria fisica al disco e viceversa. Il sistema LINUX non usa l'avvicendamento d'interi processi, ma adotta esclusivamente i più recenti meccanismi di paginazione.

Il sistema di paginazione si può dividere in due parti: l'**algoritmo di scelta** decide quali pagine trasferire su disco, e quando farlo; il **meccanismo di paginazione** compie il trasferimento ed esegue anche l'operazione inversa non appena ve ne sia bisogno.

L'algoritmo di scelta del LINUX è una versione modificata dell'algoritmo dell'orologio (con seconda chance), descritto nel Paragrafo 10.4.5.2. Il sistema LINUX adotta una scansione a passo multiplo e ogni pagina ha un'*età* ritoccata a ogni passo. L'età, per la precisione, è una misura della giovinezza di una pagina, cioè di quanto la pagina è stata usata di recente: le pagine per le quali è stato frequentemente richiesto l'accesso avranno un'età maggiore, mentre l'età delle pagine meno richieste diminuirà a ogni passo. L'algoritmo di paginazione sceglie per il trasferimento le pagine meno frequentemente usate (*least frequently used* — LFU).

Il meccanismo di paginazione è in grado di paginare sia in specifici dispositivi e in partizioni, sia in normali file, anche se quest'ultima operazione è assai più lenta a causa dei rallentamenti indotti dal file system. L'assegnazione di blocchi che risiedono in dispositivi di avvicendamento è eseguita attraverso una mappa di bit dei blocchi usati che si trova sempre nella memoria fisica. L'assegnatore adotta un algoritmo che tenta di scrivere le pagine secondo successioni ininterrotte di blocchi del disco al fine di migliorare le prestazioni. L'assegnatore registra che una pagina è stata trasferita nel disco usando una peculiarità delle tabelle delle pagine delle moderne CPU: il bit di pagina assente dell'elemento della tabella relativo alla pagina trasferita è posto a uno, il che permette al resto dell'elemento della tabella d'essere sovrascritto con un indice che identifica il luogo in cui la pagina è stata trasferita.

20.6.2.4 Memoria virtuale del nucleo

Il sistema LINUX riserva per usi interni una regione costante e dipendente dall'architettura dello spazio d'indirizzi virtuali di ogni processo. Gli elementi della tabella delle pagine che si riferiscono a queste pagine del nucleo sono contrassegnati come protetti, cosicché le pagine non sono né visibili né modificabili quando la CPU è in esecuzione nel modo d'utente. Quest'area di memoria virtuale del nucleo è costituita di due regioni. La prima è statica e contiene riferimenti a ogni pagina di memoria fisica disponibile nel sistema, fornendo un semplice modo di tradurre indirizzi fisici in indirizzi virtuali durante l'esecuzione di codice di nucleo: la parte centrale del nucleo insieme con tutte le pagine assegnate dall'ordinario assegnatore delle pagine risiedono in questa regione.

Il resto della parte riservata al nucleo dello spazio d'indirizzi non è dedicato ad alcuno scopo specifico: gli elementi della tabella delle pagine che si riferiscono a questa regione possono essere modificati dal nucleo in modo da puntare a qualunque altra area di memoria desiderata. Il nucleo fornisce due funzioni che permettono ai processi di usare questa memoria virtuale: `vmalloc` assegna un numero arbitrario di pagine di memoria fisica e le associa a un'unica regione della memoria virtuale del nucleo, dando la possibilità di assegnare grandi sezioni di memoria contigua anche quando non vi sia sufficiente spazio fisico contiguo per soddisfare la richiesta. La funzione `vremap` associa una sequenza d'indirizzi virtuali a un'area di memoria usata da un driver di dispositivo per compiere l'I/O associato alla memoria.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.6.3 Caricamento ed esecuzione dei programmi utenti

L'esecuzione dei programmi utenti da parte del nucleo del LINUX si attiva per mezzo della chiamata del sistema exec, la quale chiede al nucleo di eseguire un nuovo programma all'interno del processo corrente in modo tale da sovrascrivere integralmente il contesto d'esecuzione attuale con il contesto iniziale del nuovo programma. Il primo obiettivo di questo servizio di sistema è verificare che il processo chiamante abbia diritto d'esecuzione sul file interessato; risolta tale questione, il nucleo invoca una procedura di caricamento per avviare l'esecuzione del programma. Il caricatore non trasferisce necessariamente i contenuti del file da eseguire nella memoria fisica, ma per lo meno associa il programma alla memoria virtuale.

Nel LINUX non c'è una singola procedura per il caricamento dei programmi: il sistema operativo mantiene invece una tabella dei possibili caricatori, e dà a ognuno di essi l'opportunità di tentare di caricare il file in questione al momento dell'esecuzione di una exec. Il motivo originario dell'esistenza di una tale tabella era che nel passaggio dalla versione 1.0 alla versione 1.2 del nucleo il formato dei file binari del LINUX è stato modificato: i primi nuclei adottavano il formato a.out, che è relativamente semplice e comune ai vecchi sistemi UNIX; i più recenti sistemi LINUX usano il più moderno formato ELF, adottato adesso dalla maggior parte delle attuali versioni dello UNIX. Il formato ELF offre un certo numero di vantaggi rispetto ad a.out, fra i quali si segnalano la flessibilità e l'estendibilità: si possono aggiungere nuove sezioni a un file binario ELF (ad esempio per fornire ulteriori informazioni utili all'individuazione e alla correzione degli errori), senza che le procedure di caricamento vadano incontro a problemi. Grazie alla registrazione di più procedure di caricamento il sistema LINUX gestisce facilmente i formati binari ELF e a.out all'interno di un unico sistema.

Nei Paragrafi 20.6.3.1 e 20.6.3.2 sono trattati esclusivamente il caricamento e l'esecuzione di programmi in formato ELF: le procedure di caricamento concernenti il formato a.out sono simili anche se più semplici.

20.6.3.1 Associazione dei programmi alla memoria

Il caricamento di un file binario nella memoria fisica non è eseguito dal caricatore binario; le pagine del file binario sono invece associate a regioni della memoria virtuale: solo quando il programma tenterà di accedere a una data pagina, la conseguente eccezione di pagina mancante causerà il caricamento di quella pagina nella memoria fisica.

L'iniziale associazione del file a regioni di memoria virtuale è compito del caricatore binario del nucleo. Un file in formato binario ELF consiste in un'intestazione seguita da diverse sezioni allineate alle pagine: il caricatore ELF lavora leggendo l'intestazione e associando le sezioni del file a regioni distinte della memoria virtuale.

Nella Figura 20.4 è mostrata l'organizzazione tipica delle regioni di memoria preparate dal caricatore ELF. Il nucleo si trova nella regione riservata a un estremo dello spazio d'indirizzi; si tratta di una regione privilegiata di memoria virtuale inaccessibile agli ordinari programmi eseguiti nel modo d'utente. Il resto della memoria virtuale è disponibile per le applicazioni che possono usare le funzioni del nucleo che creano regioni associate a porzioni di un file o disponibili per i dati delle applicazioni.

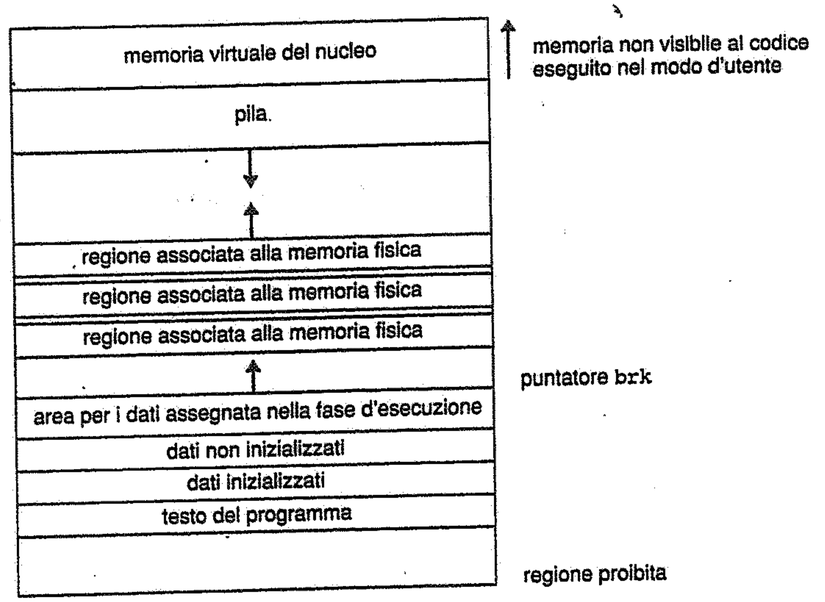


Figura 20.4 Organizzazione della memoria per i programmi ELF.

Il compito del caricatore è di instaurare le associazioni iniziali per permettere l'avvio dell'esecuzione del programma: fra le regioni da inizializzare ci sono la pila e i segmenti di testo e di dati del programma.

La pila è posta in cima alla memoria virtuale d'utente, e cresce verso il basso impiegando indirizzi via via inferiori. Include copie delle variabili d'ambiente che costituiscono gli argomenti passati al programma dalla chiamata del sistema exec. Le altre regioni sono poste vicino all'estremo inferiore della memoria virtuale; le sezioni del file binario che contengono il testo del programma o dati solamente leggibili sono assegnate a regioni protette dalla scrittura. I dati scrivibili inizializzati sono quindi associati alla memoria virtuale, e infine i dati non inizializzati sono associati a regioni private a valori nulli.

Appena sotto queste regioni di dimensione fissata si trova una regione a dimensione variabile che i programmi possono espandere secondo le necessità al fine di memorizzare dati assegnati nella fase d'esecuzione. Ogni processo ha un puntatore brk che punta all'estensione attuale di questa regione di dati, e con una singola chiamata del sistema un processo può ampliare o contrarre la sua regione relativa a brk.

Una volta che queste operazioni sono state completate il caricatore inizializza il contatore di programma del processo secondo il punto d'inizio registrato nell'intestazione ELF, e il processo è pronto per lo scheduling.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.6.3.2 Collegamento statico e dinamico

Una volta che il programma è stato caricato e avviato, tutti i dati necessari estratti dal file binario sono stati trasferiti nello spazio d'indirizzi virtuale del processo; tuttavia, la maggior parte dei programmi deve eseguire funzioni delle librerie di sistema, e anche queste devono essere caricate. Nel caso più semplice, quando un programmatore scrive un'applicazione, le funzioni di libreria necessarie sono direttamente incorporate nel file binario eseguibile del programma: si dice in questo caso che l'operazione di collegamento del programma alle librerie è eseguita staticamente e programmi di questo tipo possono cominciare l'esecuzione non appena siano stati caricati.

Lo svantaggio principale del collegamento statico è che ogni programma deve contenere copie diverse delle stesse funzioni di libreria: sia in termini di memoria fisica sia di spazio di disco è più efficiente caricare nella memoria le librerie di sistema una sola volta. L'operazione di collegamento dinamico permette di eseguire un unico caricamento.

Il sistema LINUX realizza il collegamento dinamico nel modo d'utente grazie a una speciale libreria di collegamento. Ogni programma il cui collegamento alle librerie sia stato eseguito dinamicamente contiene una piccola funzione collegata staticamente che è chiamata all'avviamento del programma. Questa funzione non fa altro che associare la libreria di collegamento alla memoria virtuale ed eseguire il codice contenuto nella funzione; la libreria di collegamento legge l'elenco delle librerie dinamiche richieste dal programma e i nomi delle variabili e delle funzioni di libreria di cui il programma si vuole avvalere: queste informazioni sono contenute nelle sezioni del file binario ELF. La libreria di collegamento associa poi le librerie richieste alla memoria virtuale, e risolve i riferimenti ai singoli contenuti in queste librerie. Non è importante l'esatto punto della memoria dove risiedono queste librerie condivise: si tratta di codice compilato in modo d'essere **codice indipendente dalla posizione** (*position independent code* — PIC), che si può eseguire a partire da qualunque indirizzo di memoria.

20.7 File system

Il sistema LINUX adotta il modello standard di file system dello UNIX. Nello UNIX un file non è necessariamente un oggetto memorizzato in un disco o prelevato da un file server di rete: un file è qualunque elemento sia in grado di gestire l'immissione o l'emissione di un flusso di dati. I driver dei dispositivi possono apparire come file agli occhi dell'utente, e così pure i canali di comunicazione fra processi o le connessioni di rete.

Il nucleo del LINUX gestisce tutti questi tipi di file nascondendone i dettagli relativi alla struttura interna sotto uno strato di programmi: il file system virtuale (VFS).

20.7.1 File system virtuale

Il VFS del LINUX è concepito secondo princìpi di progettazione orientata agli oggetti. È costituito da due componenti: un insieme di definizioni che stabiliscono cosa può essere un oggetto di tipo file, e uno strato di programmi che serve a manipolare questi oggetti. I tre tipi d'oggetti principali definiti dal VFS sono le strutture **oggetto-inode** e **oggetto-file**, che rappresentano file singoli, e l'**oggetto-file system**, che denota invece un intero file system.

Per ciascuno dei tre tipi di oggetti menzionati il VFS definisce un insieme di operazioni che devono essere codificate da tale struttura: ogni oggetto contiene un puntatore a una tabella di funzioni, e questa elenca gli indirizzi delle funzioni che realizzano le operazioni richieste per l'oggetto in questione. Lo strato di programmi del VFS può quindi eseguire un'operazione su uno di questi oggetti chiamando una funzione appropriata scelta dalla tabella delle funzioni dell'oggetto, senza dover sapere in anticipo con che tipo d'oggetto ha a che fare: il VFS non sa né ha bisogno di sapere se un *inode* rappresenta un file di rete, un file in un disco, una directory o una socket di rete; in ogni caso, la funzione che realizza l'operazione di lettura dei dati per quell'oggetto si trova in un punto ben determinato e invariabile della tabella delle funzioni dell'oggetto, e lo strato di programmi del VFS chiamerà la funzione in questione senza interessarsi del modo in cui i dati saranno effettivamente letti.

L'oggetto-file system rappresenta un insieme connesso di file che forma una gerarchia di directory autonoma. Il nucleo mantiene un unico oggetto-file system per ogni unità a disco montata come un file system e per ogni file system in rete attualmente collegato. La principale responsabilità di quest'oggetto è di garantire la possibilità di accedere agli *inode*: il VFS identifica ogni *inode* tramite un'unica coppia <*file system, numero di inode*>, e individua l'*inode* corrispondente a un certo numero di *inode* chiedendo all'oggetto-file system di riportare l'*inode* con tale numero.

Gli oggetti *inode* e file costituiscono il meccanismo d'accesso ai file: il primo rappresenta il file globalmente, mentre il secondo identifica un punto d'accesso ai dati del file; un processo non può quindi avere accesso ai dati contenuti in un file relativo a un *inode* se non ottiene prima un oggetto-file corrispondente a tale *inode*. L'oggetto-file tiene traccia del punto del file nel quale il processo sta correntemente leggendo o scrivendo, cosa che permette di gestire l'I/O sequenziale sui file; esso registra inoltre l'eventuale richiesta d'accesso per scrittura da parte del processo al momento dell'apertura del file, e tiene traccia dell'attività del processo quando ciò è necessario per eseguire letture anticipate adattive (cioè il trasferimento anticipato di dati nella memoria al fine di migliorare le prestazioni).

Di solito gli oggetti-file appartengono a un solo processo, mentre gli oggetti-*inode* sono condivisi, e anche quando un certo file non è usato da nessun processo il suo *inode* può essere trasferito nella cache dal VFS per ottenere migliori prestazioni nell'ipotesi di un prossimo accesso al file relativo. Tutti i dati contenuti nel file trasferiti nella cache

http://www.pdfdump.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sono organizzati in una lista all'interno dell'oggetto-*inode* del file; l'*inode* registra anche alcune informazioni sul file, ad esempio il proprietario, la dimensione e la data dell'ultima modifica.

Le directory sono trattate in modo leggermente diverso dagli altri file; l'interfaccia per il programmatore dello UNIX definisce un certo numero di operazioni eseguibili sulle directory, come la creazione, rimozione o il cambiamento di nome di un file contenuto in una directory: al contrario delle operazioni di lettura e scrittura, che necessitano della preliminare apertura del file, le chiamate del sistema relative a queste operazioni non richiedono l'apertura da parte dell'utente dei file coinvolti. Il VFS perciò definisce queste operazioni sulle directory all'interno dell'oggetto-*inode*, e non nell'oggetto-file.

20.7.2 File system *ext2fs*

Il file system residente su dischi ordinariamente usato dal LINUX si chiama *ext2fs* per ragioni storiche. Originariamente LINUX adottava un file system compatibile con quello del sistema Minix allo scopo di facilitare lo scambio di dati con la piattaforma di sviluppo Minix; quel file system, tuttavia, soffriva di gravi limitazioni quali la massima lunghezza dei nomi dei file, pari a 14 caratteri, e la massima dimensione del file system, pari a 64 MB. Il file system del Minix fu sostituito da un nuovo file system chiamato *extfs* (*extended file system*); successive modifiche apportate al fine di migliorare le prestazioni e aggiungere qualche servizio mancante portarono all'*ext2fs* (*second extended file system*).

Il file system *ext2fs* ha molto in comune con l'*ffs* (*fast file system*) del sistema BSD descritto nel Paragrafo A.7.7; adotta un meccanismo simile per individuare i blocchi di dati appartenenti a un certo file, memorizzando puntatori a blocchi di dati in blocchi indiretti sparsi per il file system, fino a impiegare blocchi indiretti a tre livelli. Come nell'*ffs*, le directory sono memorizzate nei dischi esattamente come ogni altro file, anche se i loro contenuti sono interpretati diversamente: ogni blocco di una directory consiste di una lista concatenata d'elementi ognuno contenente la lunghezza dell'elemento stesso, il nome di un file e il numero dell'*inode* cui l'elemento si riferisce.

Le differenze principali tra l'*ext2fs* e l'*ffs* riguardano le strategie d'assegnazione dello spazio dei dischi: nell'*ffs* lo spazio nei dischi si assegna ai file in blocchi di 8 KB, con blocchi che possono essere ulteriormente suddivisi in frammenti di 1 KB al fine di memorizzare piccoli file o blocchi riempiti solo parzialmente alla fine di un file; per contro, l'*ext2fs* non usa affatto i frammenti ma esegue tutte le assegnazioni secondo unità più piccole. La dimensione predefinita di un blocco nell'*ext2fs* è di 1 KB, anche se c'è la possibilità di gestire blocchi di 2 KB o 4 KB.

Per ottenere buone prestazioni il sistema operativo deve tentare di eseguire trasferimenti di I/O in blocchi di grandi dimensioni, ogniqualvolta ciò sia possibile, raggruppando richieste di I/O fisicamente adiacenti. Il raggruppamento riduce il ritardo per richiesta indotto dai driver dei dispositivi, dai dischi e dal controller del disco. Blocchi di 1 KB sono troppo piccoli perché garantiscano buone prestazioni, quindi l'*ext2fs* adotta strategie d'assegnazione concepite per collocare blocchi di un file logicamente adiacenti in blocchi fisicamente adiacenti nel disco, rendendo così possibile l'accorpamento di molte richieste relative a blocchi differenti in un'unica operazione.

Questo criterio d'assegnazione si attua in due fasi. Come nell'*ffs*, un file system *ext2fs* è suddiviso in **gruppi di blocchi** multipli; l'*ffs* usa il simile concetto di **gruppi di cilindri**, dove ogni gruppo corrisponde a un singolo cilindro del disco fisico. Tuttavia, la tecnologia delle moderne unità a disco distribuisce i settori nei dischi con diverse densità, e quindi anche con diverse dimensioni dei cilindri, secondo la distanza dal centro del disco, perciò gruppi di cilindri di dimensione costante non corrispondono necessariamente alla geometria del disco.

Prima di poter assegnare lo spazio a un file, l'*ext2fs* deve scegliere un gruppo di blocchi per quel file. Per quel che riguarda i blocchi di dati tenta di scegliere lo stesso gruppo di blocchi nel quale l'*inode* del file è stato assegnato; per assegnare un *inode* che non sia relativo a una directory, sceglie lo stesso gruppo di blocchi della directory cui il file appartiene. Le directory, invece di essere raggruppate, sono distribuite su tutti i blocchi disponibili. Queste strategie hanno l'obiettivo di mantenere informazioni correlate all'interno di uno stesso gruppo di blocchi, ma anche di distribuire il carico d'informazioni su tutti i gruppi di blocchi del disco, così da ridurre la frammentazione d'ogni area del disco stesso.

All'interno di un gruppo di blocchi, l'*ext2fs* tenta se è possibile compiere le assegnazioni in modo fisicamente contiguo riducendo quindi la frammentazione. Esso mantiene una mappa di bit di tutti i blocchi liberi di un gruppo di blocchi: durante l'assegnazione dei primi blocchi relativi a un nuovo file, l'*ext2fs* comincia la ricerca di un blocco libero dall'inizio del gruppo di blocchi; quando invece si tratta di ampliare un file, continua la ricerca dall'ultimo blocco assegnato al file. La ricerca si svolge in due fasi: innanzi tutto si cerca un intero byte libero nella mappa di bit; se ciò non riesce, si cerca un bit libero. La ricerca di interi byte liberi ha lo scopo di tentare di assegnare lo spazio nei dischi in porzioni di almeno 8 blocchi.

Una volta che si sia identificato un blocco libero, si estende la ricerca all'indietro fino a incontrare un blocco assegnato. Quando si trova un byte libero nella mappa di bit, quest'estensione a ritroso impedisce che l'*ext2fs* lasci un buco fra l'ultimo blocco assegnato nel precedente byte non nullo e il byte libero trovato; una volta individuato il prossimo blocco da assegnare tramite una ricerca per byte o per bit, l'*ext2fs* estende l'assegnazione in avanti fino a un massimo di otto blocchi **preassegnando** questo spazio supplementare al file. La preassegnazione concorre a diminuire il grado di frammentazione nel corso di scritture intercalate in file diversi, e riduce anche il costo dell'assegnazione in termini d'impegno della CPU grazie all'assegnazione simultanea di più blocchi; i blocchi preassegnati sono nuovamente riportati sulla mappa di bit dei blocchi liberi al momento della chiusura del file.

Nella Figura 20.5 sono illustrati i criteri d'assegnazione. Ogni riga rappresenta una sequenza di bit posti a uno o a zero nella mappa di bit dell'assegnazione; questi bit contrassegnano i blocchi liberi e i blocchi in uso del disco. La prima possibilità è che si trovino blocchi liberi sufficientemente vicini al punto d'inizio della ricerca, in tal caso essi sono assegnati indipendentemente dal fatto che il loro insieme possa essere frammentato; infatti, dato che i blocchi sono vicini fra loro, è probabile che possano essere letti

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

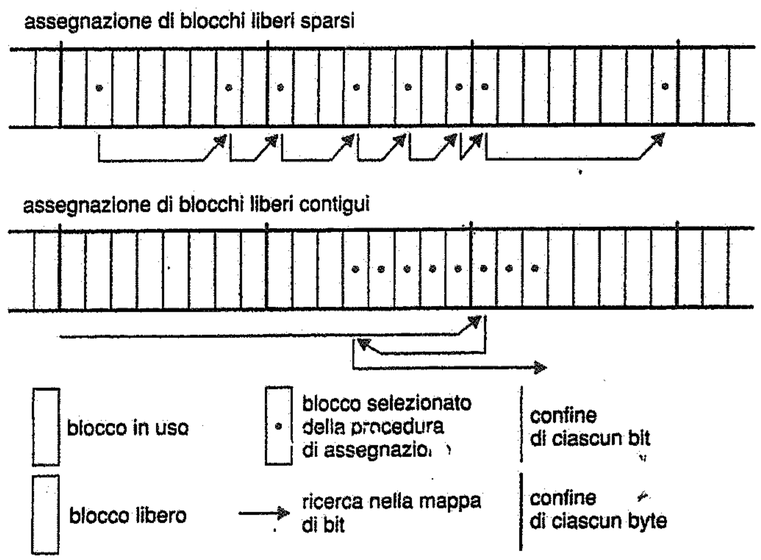


Figura 20.5 Criteri di assegnazione dell'*ext2fs*.

senza dover eseguire operazioni di ricerca nel disco, cosa che compensa parzialmente la frammentazione. Inoltre, assegnare tutti questi blocchi a un solo file è una scelta che a lungo termine si rivela migliore dell'assegnazione di blocchi isolati a file distinti, perché a lungo andare regioni ampie di spazio libero su disco divengono rare. La seconda possibilità è che non si sia trovato un blocco libero nelle vicinanze del punto d'inizio della ricerca, quindi si procede nella ricerca di un intero byte libero nella mappa di bit; se si assegnasse questo byte senza prendere alcuna precauzione si creerebbe un frammento di spazio libero prima di esso, quindi è necessario tornare indietro per colmare lo spazio rimasto fra il primo blocco assegnato precedentemente e il blocco in questione. Infine, per rispettare la dimensione predefinita di un'assegnazione, che è di otto blocchi, si estende l'assegnazione in avanti fin quando è necessario.

20.7.3 File system *proc*

Il file system VFS è sufficientemente flessibile da permettere la realizzazione di un file system che non archivi in modo permanente alcun dato, ma semplicemente funga da interfaccia per qualche altro servizio. Il *process file system* del LINUX, noto come *proc*, è un esempio di file system i cui contenuti non sono in realtà memorizzati in alcun luogo, ma sono calcolati su richiesta secondo le richieste degli utenti.

Un file system *proc* non è una caratteristica esclusiva del LINUX: UNIX SVR4 adottava un file system di questo tipo come efficiente interfaccia alle funzioni del nucleo d'ausilio alle attività di messa a punto dei processi; ogni sottodirectory del file system corrispondeva non a una directory in qualche disco, ma piuttosto a un processo attivo del sistema, cosicché un elenco dei contenuti del file system presentava una directory per processo, essendo il nome della directory la rappresentazione decimale nel formato ASCII dell'identificatore unico del processo (PID).

Il sistema LINUX realizza un file system di questo tipo, ma lo amplia notevolmente aggiungendo un certo numero di ulteriori directory e file di testo all'interno della directory radice del file system. Questi nuovi oggetti corrispondono a diverse statistiche relative al nucleo e ai driver caricati; il file system *proc* fornisce ai programmi la possibilità di accedere a queste informazioni leggendo semplici file di testo che nell'ambiente UNIX è possibile trattare con strumenti complessi. Per fare un esempio, il classico comando ps dello UNIX, che elenca gli stati di tutti i processi in esecuzione, è stato in passato realizzato come un processo privilegiato che leggeva gli stati dei processi direttamente dalla memoria virtuale del nucleo; nel LINUX, questo comando è realizzato da un programma del tutto ordinario che semplicemente esamina e compone in forma maggiormente leggibile le informazioni contenute nel *proc*.

Il file system *proc* deve costruire una struttura di directory e i contenuti dei file in esso residenti; giacché un file system UNIX è per definizione un insieme di *inode* relativi a file e directory identificati dal loro numero di *inode*, il file system *proc* deve definire in modo unico un numero di *inode* permanente per ogni directory e per i file in essa contenuti. Una volta instaurata questa corrispondenza si può usare il numero di *inode* per identificare esattamente l'operazione richiesta da un utente che tenti di leggere un certo file o di esaminare una certa directory. Quando si leggono dati da uno di questi file, il file system *proc* estrae l'informazione appropriata, la compone in forma leggibile e la colloca nella sezione di memoria (*buffer*) di lettura del processo richiedente.

La corrispondenza fra un numero di *inode* e il tipo d'informazione divide il numero di *inode* in due parti: nel sistema LINUX un PID è lungo 16 bit, ma un numero di *inode* è lungo 32 bit; i primi 16 bit del numero di *inode* sono interpretati come un PID, e i rimanenti definiscono il tipo d'informazione richiesta.

Un PID pari a zero non è ammesso, quindi un campo PID pari a zero nel numero di *inode* significa che questo *inode* contiene informazioni globali e non relative allo specifico processo: *proc* contiene file globali distinti che forniscono informazioni come la versione del nucleo, la memoria libera, statistiche sulle prestazioni, e i driver attualmente attivi.

Non tutti i numeri di *inode* usati da un tale file system sono riservati; il nucleo può assegnare nuovi *inode* del *proc* dinamicamente, mantenendo una mappa dei numeri di *inode* assegnati. Esso mantiene anche una struttura di dati ad albero degli elementi globali registrati del file system *proc*: ogni elemento contiene il numero di *inode* del file, il nome del file, i permessi d'accesso e le funzioni speciali usate per generare i contenuti del file; i driver possono registrare o rimuovere elementi da quest'albero in qualunque

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

momento, e una parte speciale dell'albero che appare nella directory /proc/sys è riservata alle variabili del nucleo. I file contenuti in quest'albero sono trattati per mezzo di un insieme di gestori comuni che permettono sia la lettura sia la scrittura di queste variabili, cosicché un amministratore del sistema può regolare i valori dei parametri del nucleo semplicemente scrivendoli in decimali ASCII nel file appropriato.

Per permettere alle applicazioni di accedere efficientemente a queste variabili, il sottoalbero /proc/sys è reso disponibile per il tramite di una speciale chiamata del sistema, sysctl, che legge e scrive le stesse variabili in formato binario anziché sotto forma di testo, evitando i ritardi indotti dal file system; questa chiamata del sistema non fornisce una funzione aggiuntiva ma legge semplicemente l'albero dinamico per decidere a quali variabili l'applicazione si riferisce.

20.8 I/O

Il sistema per l'I/O del sistema operativo LINUX appare all'utente in modo molto simile a quello di un qualunque sistema UNIX: tutti i driver dei dispositivi, nei limiti del possibile, sono rappresentati come file ordinari. L'utente apre un canale d'accesso a un dispositivo nello stesso modo in cui apre un file qualunque: i dispositivi possono comparire come oggetti del file system. L'amministratore del sistema può creare file speciali all'interno del file system che contengono riferimenti a uno specifico driver di dispositivo, e un utente che apra un tale file potrà leggere e scrivere nel dispositivo associato. Grazie all'ordinario sistema di protezione dei file, che stabilisce chi può accedere a quali file, l'amministratore può controllare l'accesso ai dispositivi.

Il sistema LINUX suddivide i dispositivi in tre classi: i dispositivi a blocchi, i dispositivi a caratteri, e i dispositivi di rete. Nella Figura 20.6 è illustrata la struttura globale del sistema dei driver dei dispositivi. I **dispositivi a blocchi** comprendono tutti i dispositivi capaci di fornire l'accesso diretto ai blocchi di dati di dimensione fissa completamente indipendenti; fra questi vi sono i dischi, i dischetti e i dischi ottici. I dispositivi a

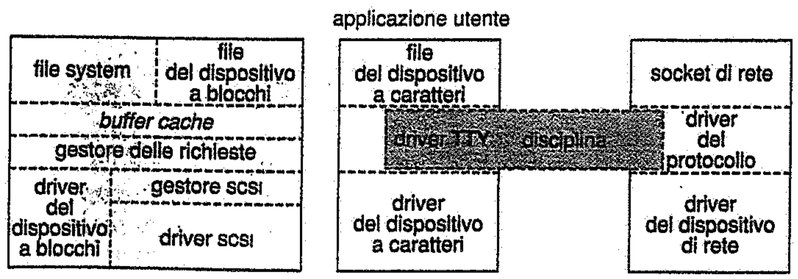


Figura 20.6 Struttura del sistema dei driver dei dispositivi.

blocchi sono generalmente usati per contenere interi file system, ma c'è anche la possibilità di accedere direttamente a questi dispositivi per permettere a programmi specializzati di creare e riparare i file system in essi contenuti. Anche le applicazioni possono accedere direttamente a questi dispositivi se lo desiderano; una base di dati, ad esempio, potrebbe voler strutturare i dati nei dischi in modo appropriato ai propri scopi specifici, invece di usare il file system generale.

I **dispositivi a caratteri** comprendono la maggior parte degli altri dispositivi, con l'importante eccezione dei dispositivi di rete. I dispositivi a caratteri non devono necessariamente fornire le piene funzioni dei file ordinari: un dispositivo relativo a un altoparlante, ad esempio, permette la scrittura di dati, ma non certo la lettura; analogamente, la localizzazione di un certo punto del file potrebbe essere un servizio fornito a un'unità a nastro magnetico, mentre nel caso di un mouse tale operazione non avrebbe senso.

I **dispositivi di rete** sono trattati diversamente dai dispositivi delle altre due classi: gli utenti non possono trasferire direttamente dati ai dispositivi di rete, ma devono comunicare indirettamente tramite il sottosistema di rete del nucleo. L'interfaccia ai dispositivi di rete è discussa separatamente nel Paragrafo 20.10.

20.8.1 Dispositivi a blocchi

I dispositivi a blocchi costituiscono l'interfaccia principale a tutte le unità a disco del sistema. Le prestazioni sono una questione di particolare rilevanza per i dischi, e il sistema per i dispositivi a blocchi deve quindi fornire servizi che permettano di accedere ai dischi il più rapidamente possibile. I due componenti principali che s'impiegano per raggiungere questo scopo sono la *block buffer cache* e il *gestore delle richieste*.

20.8.1.1 *Block buffer cache*

La *block buffer cache* del LINUX assolve due compiti principali: funge sia da insieme di memorie di transito (*buffer*) per l'I/O in esecuzione, sia da cache per l'I/O già completato. La *buffer cache* è costituita da due parti: le aree di memoria per l'I/O, un insieme di pagine di dimensione dinamicamente variabile assegnato direttamente dal gruppo di pagine della memoria centrale del nucleo (ogni pagina è divisa in un insieme di aree d'uguale dimensione); e un insieme corrispondente di descrittori di tali aree, detti buffer_head, uno per ogni area di memoria della cache.

I buffer_head contengono tutte le informazioni che il nucleo mantiene riguardo alle aree di memoria per l'I/O. L'informazione principale è l'identificazione, ogni area di memoria per l'I/O è identificata da una tripla: il dispositivo a blocchi associato, lo scostamento dei dati all'interno del dispositivo a blocchi, e la dimensione dell'area di memoria. Tali aree di memoria per l'I/O sono anche contenute in un certo numero di liste distinte: le aree il cui contenuto ha subito modifiche, quelle il cui contenuto non ha subito modifiche, e quelle protette; oltre alla lista di quelle libere. Una di queste aree è posta nella lista delle aree libere o da un file system (ad esempio quando si cancella un file), o da una funzione refill_freelist chiamata ogniqualvolta il nucleo necessita di altre aree di

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

memoria per l'I/O. Il nucleo allunga la lista delle aree di memoria per l'I/O libere o espandendo l'insieme di tali aree o riciclando aree già esistenti, secondo la memoria libera disponibile. Infine ogni area non presente nella suddetta lista è indicizzata da una funzione hash dei suoi numeri di dispositivo e di blocco, ed è inserita in una corrispondente lista concatenata di ricerca hash.

La gestione delle aree di memoria per l'I/O del nucleo risolve automaticamente, sfruttando l'aiuto di due demoni, il problema della scrittura nei dischi del contenuto di tali aree di memoria che hanno subito modifiche. Il primo si attiva semplicemente a intervalli regolari per chiedere che siano scritti nei dischi tutti i dati le cui modifiche, avvenute da un certo intervallo di tempo, non vi siano ancora state riportate; l'altro è un thread del nucleo che si attiva ogniqualvolta la funzione `refill_freelist` trova che una parte troppo estesa dei dati della *buffer cache* non è ancora stata riportata nei dischi.

20.8.1.2 Gestore delle richieste

Il **gestore delle richieste** è lo strato di programmi che gestisce la lettura e la scrittura del contenuto di un'area di memoria per l'I/O da e su un driver di un dispositivo a blocchi. Il meccanismo di base è incentrato su una funzione, `ll_rw_block`, che esegue le letture e le scritture a basso livello relativamente ai dispositivi a blocchi; questa funzione ha come argomento una lista di descrittori `buffer_head` e di indicatori (*flag*) di lettura e scrittura e appronta le operazioni di I/O per tutte queste aree, senza attendere il loro completamento.

Le richieste di I/O in attesa sono registrate nelle strutture `request`, che rappresentano una richiesta per la scrittura o la lettura di un insieme contiguo di settori in un singolo dispositivo a blocchi. Dato che più di un'area di memoria per l'I/O può essere interessata dal trasferimento, la struttura contiene un puntatore al primo elemento di una lista di `buffer_head` da usare durante l'operazione.

Una lista distinta di richieste è mantenuta per ogni driver di un dispositivo a blocchi, e lo scheduling di queste richieste avviene in accordo all'algoritmo C-SCAN che sfrutta l'ordine nel quale le richieste sono inserite o rimosse dalle liste relative ai dispositivi. Le liste di richieste sono mantenute in ordine crescente di settore iniziale. Quando una richiesta è servita da un driver, non è rimossa dalla lista; la rimozione avviene solo quando l'operazione di I/O è terminata: a questo punto il driver prosegue con la prossima richiesta nella lista anche se nuove richieste sono state inserite nella lista prima della richiesta attiva.

A mano a mano che arrivano nuove richieste di I/O, il gestore delle richieste tenta di accorparle prima di porle nella lista di un dispositivo: il passaggio di una sola richiesta di grandi dimensioni in luogo di molte piccole richieste è spesso molto più efficiente in termini dei ritardi indotti dai dispositivi. Tutti i `buffer_head` in una tale richiesta aggregata sono riservati non appena è impartita la richiesta iniziale di I/O. A mano a mano che la richiesta è servita dal driver, i singoli `buffer_head` che costituiscono la richiesta aggregata sono rilasciati uno alla volta: non è necessario che un processo che attende un'area di memoria per l'I/O debba aspettare il rilascio di tutte le altre. Questa considerazione è particolarmente importante perché assicura un'esecuzione efficiente delle letture anticipate.

Un'ultima caratteristica del gestore delle richieste è che è possibile impartire richieste di I/O che scavalchino totalmente la *buffer cache*: una funzione di I/O di basso livello, `brw_page`, crea un insieme di `buffer_head` temporanei che etichettano i contenuti di una pagina di memoria al fine di presentare le richieste di I/O al gestore delle richieste; questi descrittori temporanei, tuttavia, non sono collegati alla *buffer cache*, e quando l'ultima area della pagina ha terminato il proprio I/O, si rilascia l'intera pagina e si eliminano i `buffer_head`.

Questo meccanismo è usato dal nucleo ogniqualvolta il chiamante esegue indipendentemente il caching dei dati o quando si sa che i dati non possono più essere soggetti a caching. La cache delle pagine riempie le sue pagine in questo modo per evitare il doppio caching degli stessi dati sia da parte della *buffer cache* sia da parte della cache delle pagine stesse. Anche il sistema per la memoria virtuale scavalca la cache quando esegue operazioni di I/O verso dispositivi per l'avvicendamento dei processi.

20.8.2 Dispositivi a caratteri

Un driver di un dispositivo a caratteri può essere qualsiasi driver che non offra l'accesso diretto a blocchi di dati di dimensioni fissate. Ogni driver di un dispositivo a caratteri registrato dal nucleo del LINUX deve anche registrare un insieme di funzioni che realizzino le operazioni di I/O su file gestite dal driver. Il nucleo non esegue quasi alcuna operazione preliminare su una richiesta di lettura o scrittura su file relativa a un dispositivo a caratteri, ma passa semplicemente la richiesta al dispositivo in questione lasciandogli il compito di servirla.

L'eccezione principale a questa regola è costituita dai driver dei dispositivi a caratteri relativi ai terminali: il nucleo mantiene un'interfaccia uniforme a questi driver per mezzo di un insieme di strutture `tty_struct`: ognuna di esse fornisce il controllo del flusso e la memorizzazione transitoria dei dati provenienti dal terminale e passa questi dati a un'interprete.

L'interprete delle informazioni provenienti da un terminale segue una determinata disciplina, la più comune delle quali è la `tty`, che incolla il flusso dei dati del terminale ai flussi di I/O standard dei processi utenti in esecuzione, permettendo a questi processi di comunicare direttamente con il terminale dell'utente. Questo compito è complicato dal fatto che ci può essere più di un processo contemporaneamente in esecuzione, e la disciplina `tty` è responsabile dell'annessione e della rimozione dell'I/O del terminale dai vari processi a esso collegati a mano a mano che questi processi sono sospesi o riattivati dall'utente.

Il sistema LINUX dispone di altre discipline che non hanno nulla a che fare con l'I/O di un processo utente: i protocolli di rete PPP e SLIP rappresentano un modo di codificare un collegamento di rete su un dispositivo terminale come una linea seriale. Questi protocolli sono realizzati nel sistema LINUX come driver che da un lato appaiono al sistema della gestione dei terminali come interpreti con una determinata disciplina, e dall'altro appaiono al sistema per la gestione della rete come driver di dispositivi di rete: dopo che una di queste discipline è stata attivata in un dispositivo terminale, ogni dato che appare su quel terminale sarà direttamente instradato all'appropriato driver del dispositivo di rete.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.9 Comunicazione fra processi

Il sistema UNIX fornisce un ricco ambiente per la comunicazione fra processi. La comunicazione può limitarsi alla notificazione del verificarsi di un evento, da parte di un processo a un altro processo, ma può anche coinvolgere il trasferimento di dati fra processi.

20.9.1 Sincronizzazione e segnali

Il meccanismo standard dello UNIX usato per comunicare a un processo che un evento si è verificato è il segnale. Eccetto alcune limitazioni che riguardano i segnali inviati a processi di proprietà di un altro utente, i segnali si possono inviare da un processo a ogni altro processo. Tuttavia è disponibile solo un limitato numero di segnali, che per di più non veicolano al processo destinatario altra informazione se non il mero fatto che un segnale è stato generato. Non è necessario che un segnale sia generato da un altro processo: anche il nucleo genera internamente dei segnali; ad esempio, può inviare un segnale a un processo server quando arrivano dati in un canale di rete, a un processo genitore quando un figlio termina, o quando è trascorso un intervallo di tempo impostato in un temporizzatore.

Internamente il nucleo del LINUX non usa segnali per comunicare con i processi eseguiti nel modo del nucleo: se uno di essi attende il verificarsi di un evento, non userà di norma i segnali per ricevere la comunicazione dell'evento. Infatti, le comunicazioni riguardanti gli eventi asincroni sono realizzate per mezzo degli stati di scheduling e delle strutture `wait_queue`. Questo meccanismo permette ai processi eseguiti nel modo del nucleo di informarsi vicendevolmente sugli eventi rilevanti, e dà la possibilità ai driver e al sistema per la gestione della rete di generare eventi. Ogniqualvolta un processo desidera attendere la terminazione di un evento, si sposta alla struttura `wait_queue` associata a quell'evento e comunica allo scheduler la sua non disponibilità all'esecuzione; l'evento in questione riattiva tutti i processi nella `wait_queue` al momento della sua terminazione. Si tratta di una procedura che permette a più processi di attendere il verificarsi di un unico evento: ad esempio, se diversi processi stanno tentando di leggere un file da un disco, essi saranno tutti riattivati quando i dati sono stati trasferiti correttamente nella memoria.

Anche se i segnali sono sempre stati il principale meccanismo di comunicazione asincrona fra processi, il sistema LINUX dispone anche del meccanismo basato sui semafori dello UNIX System V: un processo si pone in attesa a un semaforo con la stessa facilità con la quale aspetterebbe un segnale, ma i semafori hanno due vantaggi: poter essere condivisi in gran numero da diversi processi indipendenti, e l'esecuzione atomica delle operazioni sui semafori. Internamente il meccanismo standard `wait_queue` sincronizza i processi che comunicano tramite semafori.

20.9.2 Passaggio di dati fra processi

Il sistema LINUX mette a disposizione diversi metodi per il passaggio di dati fra processi. Il meccanismo standard dello UNIX, noto come *pipe*, permette a un processo figlio di ereditare dal genitore un canale di comunicazione; i dati scritti a un'estremità di tale canale di comunicazione possono essere letti all'altra estremità. Nel sistema LINUX si tratta ancora una volta di un tipo particolare di *inode* del file system virtuale, ed è dotato di una coppia di strutture `wait_queue` per sincronizzare il lettore e lo scrittore. Il sistema UNIX definisce anche un insieme di servizi di rete che possono inviare flussi di dati a processi locali o remoti. I servizi di rete sono trattati nel Paragrafo 20.10.

Sono disponibili altri due metodi per la condivisione dei dati fra processi. In primo luogo la memoria condivisa offre un modo estremamente rapido di comunicare quantità arbitrarie di dati: qualsiasi informazione scritta da un processo in una regione di memoria condivisa può essere immediatamente letta da ogni altro processo che abbia quella regione associata al suo spazio d'indirizzi. Il principale svantaggio della memoria condivisa è che di per sé non offre alcun metodo di sincronizzazione: un processo non può chiedere al sistema operativo se qualche dato sia stato scritto in una regione di memoria condivisa, né può sospendere l'esecuzione fino a che una tale operazione di scrittura si sia verificata. La memoria condivisa diviene uno strumento particolarmente potente quando è usata insieme con un altro meccanismo di comunicazione fra processi che fornisca la sincronizzazione mancante.

Nel LINUX una regione di memoria condivisa è un oggetto permanente che può essere creato o cancellato dai processi; un tale oggetto è trattato come se fosse un piccolo spazio d'indirizzi indipendente: gli algoritmi di paginazione possono scegliere di trasferire nel disco alcune pagine di memoria condivisa allo stesso modo in cui possono scegliere di trasferire una pagina di dati di un processo. L'oggetto che rappresenta la memoria condivisa agisce come memoria ausiliaria della regione di memoria condivisa proprio come un file può essere la memoria ausiliaria di una regione di memoria virtuale. Quando un file è associato a una regione di memoria virtuale, il verificarsi di un'eccezione di pagina mancante causa l'associazione alla memoria virtuale dell'appropriata pagina del file; similmente le associazioni relative alla memoria condivisa fanno sì che dalle eccezioni di pagina mancante risulti il caricamento nella memoria di pagine prelevate da un oggetto permanente di memoria condivisa. Sempre in analogia con i file, gli oggetti che rappresentano la memoria condivisa conservano i loro contenuti anche se essi non sono associati alla memoria virtuale di alcun processo.

20.10 Strutture di rete

I servizi di rete sono uno dei punti di forza del LINUX: questo sistema operativo non solo gestisce i protocolli standard della rete Internet per la comunicazione fra sistemi UNIX, ma realizza anche altri protocolli originariamente sviluppati per sistemi operativi diversi dallo UNIX. Essendo stato originariamente concepito per i PC e non per grandi stazioni di lavoro o server, gestisce molti protocolli usati nelle reti di PC, ad esempio l'AppleTalk e l'IPX.

Internamente il nucleo del LINUX realizza i servizi di rete per mezzo di tre strati di programmi:

1. l'interfaccia a socket;
2. i driver dei protocolli;
3. i driver dei dispositivi di rete.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le applicazioni degli utenti richiedono tutti i servizi di rete tramite l'interfaccia a socket; essa è progettata per somigliare allo strato di socket del 4.3BSD, cosicché i programmi scritti per far uso delle socket di Berkeley potranno essere eseguiti dal sistema LINUX senza che sia necessario apportare modifiche al codice sorgente. Quest'interfaccia è descritta nel Paragrafo 4.6.1. L'interfaccia a socket BSD è sufficientemente generale da poter rappresentare gli indirizzi di rete di un'ampia gamma di protocolli; il sistema LINUX usa quindi questa sola interfaccia per realizzare tutti i protocolli forniti, e non solo quelli dei sistemi standard BSD.

Lo strato di programmi seguente è quello dei protocolli, organizzato in maniera simile al livello proprio del BSD. Si richiede che tutti i dati che arrivino a questo livello siano etichettati da un identificatore che specifichi secondo quale protocollo devono essere trattati; ciò vale sia per i dati provenienti dalla socket di un'applicazione, sia per quelli provenienti dal driver di un dispositivo di rete. I protocolli possono eventualmente comunicare fra di loro: all'interno dell'insieme di protocolli per la rete Internet, ad esempio, protocolli distinti gestiscono l'instradamento, la comunicazione degli errori e la ritrasmissione affidabile dei dati persi.

Lo strato dei protocolli può riscrivere i pacchetti, crearne di nuovi, dividerli in frammenti o riassemblarli da frammenti, o anche semplicemente scartare i dati in arrivo; in ultima analisi, quando ha terminato l'elaborazione di un gruppo di pacchetti, li passa all'interfaccia socket se la destinazione dei dati è locale, oppure a un driver di un dispositivo di rete, se i pacchetti devono essere inviati lungo la rete. Lo strato di protocolli decide a quale socket o dispositivo inviare il pacchetto.

Tutta la comunicazione che avviene fra gli strati di programmi che realizzano i servizi di rete è eseguita passando singole strutture dette `skbuff`; esse contengono un insieme di puntatori a un'unica regione contigua di memoria all'interno della quale i pacchetti di rete possono essere assemblati. I dati significativi puntati da una struttura `skbuff` non devono necessariamente trovarsi all'inizio di tali aree di memoria, e non devono neanche estendersi fino alla fine: il codice dei servizi di rete può aggiungere o togliere dati agli estremi del pacchetto, a patto che il risultato sia ancora contenibile dall'area di memoria. Questa possibilità è particolarmente importante per le moderne CPU, i cui miglioramenti in velocità hanno abbondantemente superato le prestazioni della memoria centrale: l'architettura basata sulle strutture `skbuff` rende flessibile la manipolazione delle intestazioni e dei codici per il controllo degli errori dei pacchetti evitando però duplicazioni non necessarie dei dati.

L'insieme di protocolli più importante nel LINUX è la serie di protocolli IP, composta di un certo numero di protocolli distinti. Il protocollo IP realizza l'instradamento da un calcolatore all'altro ovunque nella rete; su questo protocollo d'instradamento sono costruiti i protocolli UDP, TCP e ICMP. Il protocollo UDP trasferisce singoli datagrammi arbitrari fra i calcolatori; il protocollo TCP instaura connessioni affidabili con consegna garantita nell'ordine originario, e ritrasmissione automatica dei dati persi; il protocollo ICMP si usa per la trasmissione di messaggi di stato e di vari tipi di messaggi d'errore.

Si assume che i pacchetti (`skbuff`) che raggiungano la pila di protocolli siano già etichettati da un identificatore interno che indica a quale protocollo è attinente il pacchetto. Driver dei dispositivi di rete diversi codificano il tipo di protocollo in modo differente sul loro mezzo di trasmissione, quindi l'identificazione del protocollo va eseguita all'interno dei driver stessi, i quali usano a questo fine una tabella hash degli identificatori di protocollo noti, e passano poi il pacchetto al protocollo appropriato; è possibile aggiungere nuovi protocolli alla tabella hash sfruttando il meccanismo di caricamento dei moduli del nucleo.

I pacchetti IP che giungono al sistema sono consegnati al driver IP, il cui compito è quello di eseguire l'instradamento: esso determina la destinazione del pacchetto e lo inoltra al driver del protocollo interno appropriato per la consegna locale, oppure lo reinserisce nella coda del driver di un dispositivo di rete se deve essere inoltrato a un altro calcolatore. La decisione riguardante l'instradamento viene presa sulla base di due tabelle: la base permanente di informazioni sull'instradamento (*forwarding information base* — FIB), e una cache delle più recenti decisioni di instradamento. La FIB contiene informazioni sulle configurazioni d'instradamento e può specificare percorsi basati su indirizzi di destinazione determinati o su parametri che rappresentano più destinazioni; è organizzata come un insieme di tabelle hash indicizzate dagli indirizzi di destinazione, e la ricerca parte sempre dalle tabelle che contengono i percorsi più precisi. Quando una ricerca di questo tipo ha successo, il percorso individuato è posto nella cache dei percorsi, la quale contiene solo destinazioni specifiche prive di parametri, in modo che le ricerche siano più rapide. Un elemento della cache dei percorsi viene eliminato dopo un certo tempo d'inutilizzo.

In diverse fasi il protocollo IP passa i pacchetti a una sezione distinta di codice per la **gestione di barriere di sicurezza** (*firewall management*), cioè il filtraggio selettivo dei pacchetti secondo criteri arbitrari ma di solito relativi alle strategie di sicurezza. Il gestore delle funzioni di barriera di sicurezza mantiene un certo numero di **catene di barriere di sicurezza** distinte, e permette la comparazione di una struttura `skbuff` con una qualunque di esse; catene distinte assolvono funzioni distinte: una si usa per inoltrare i pacchetti, una per la ricezione dei pacchetti, e una per i dati generati dal calcolatore. Ogni catena è costituita da un insieme ordinato di prescrizioni, ciascuna delle quali specifica una funzione di filtro tra le molte possibili, oltre a dati arbitrari che devono trovare riscontro nei pacchetti.

Due altre funzioni eseguite dal driver IP sono la scomposizione e il riassemblaggio dei pacchetti di grandi dimensioni: un pacchetto da spedire troppo grande per essere posto nella coda di un dispositivo viene semplicemente diviso in **frammenti** più piccoli che possono essere posti in coda; il calcolatore ricevente si occuperà di riassemblare i frammenti. Il driver IP mantiene un oggetto `ipfrag` per ogni frammento che attende il riassemblaggio, e un oggetto `ipq` per ogni datagramma in corso d'assemblaggio. I frammenti in arrivo sono confrontati con ogni `ipq`, e nel caso di riscontro positivo il frammento è aggiunto all'oggetto; altrimenti, si crea un nuovo `ipq`. Quando l'ultimo frammento di un `ipq` è arrivato si costruisce una struttura `skbuff` interamente nuova per alloggiare il pacchetto, e si passa di nuovo il pacchetto al driver IP.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I pacchetti che l'IP identifica come destinati al calcolatore locale sono passati a uno degli altri driver di protocollo. I protocolli TCP e UDP adottano lo stesso metodo per associare ogni pacchetto alle relative socket mittenti e destinatarie: ogni coppia di socket collegate è identificata in modo unico dagli indirizzi del mittente e del destinatario e dai corrispondenti numeri di porta. Gli elenchi delle socket sono riuniti in una tabella hash la cui chiave è costituita da questi quattro valori e che può essere usata per individuare le socket relative ai pacchetti in arrivo. Il protocollo TCP deve anche occuparsi delle connessioni non affidabili, e a tal fine mantenere liste ordinate dei pacchetti trasmessi, ma privi di una ricevuta di ritorno, che saranno ritrasmessi dopo un certo tempo, e liste dei pacchetti ricevuti in modo disordinato, che saranno presentati alla socket una volta ricevuti i dati mancanti.

20.11 Sicurezza

Il modello di sicurezza del sistema LINUX è strettamente correlato ai modelli di sicurezza tipici dello UNIX. Le questioni relative alla sicurezza sono classificabili in due gruppi:

1. **Autenticazione.** Assicurare che nessuno possa accedere al sistema senza prima dimostrare di averne diritto.
2. **Controllo dell'accesso.** Fornire un meccanismo che permetta di controllare se un utente abbia diritto d'accesso a un certo oggetto, e che impedisca l'accesso se l'esito del controllo è negativo.

20.11.1 Autenticazione

L'usuale meccanismo di autenticazione nello UNIX è sempre stato basato su un file di parole d'ordine leggibile da tutti: la parola d'ordine di un utente è combinata con un valore casuale, e il risultato è codificato tramite una funzione di codifica non invertibile e infine registrato nel file delle parole d'ordine; l'uso di una funzione di codifica non invertibile significa che la parola d'ordine originale non si può ricavare dal file delle parole d'ordine se non per prove ed errori. Quando un utente presenta una parola d'ordine al sistema essa viene ricombinata con il valore casuale memorizzato nel file delle parole d'ordine; al risultato si applica la funzione di codifica non invertibile, e se ciò che si ottiene coincide con il contenuto del file delle parole d'ordine l'utente è ammesso al sistema.

Storicamente le realizzazioni dello UNIX di questo meccanismo hanno incontrato diversi problemi: le parole d'ordine erano spesso limitate alla lunghezza di otto caratteri, e il numero dei possibili valori casuali era così basso che si potevano facilmente combinare le parole d'ordine più comuni con ogni possibile valore casuale e avere buone possibilità d'individuare una o più parole d'ordine contenute nel file delle parole d'ordine, ottenendo così l'accesso non autorizzato alle utenze corrispondenti. Sono state introdotte estensioni del meccanismo delle parole d'ordine al fine di mantenerle segrete memorizzandole in un file non accessibile al pubblico; altre estensioni permettono l'uso di parole

d'ordine più lunghe o adottano metodi di codifica più sicuri. Sono stati introdotti altri meccanismi di autenticazione che limitano il tempo di collegamento al sistema oppure il tempo durante il quale l'utente può distribuire informazioni di autenticazione a sistemi collegati tramite la rete.

Per affrontare questi problemi un certo numero di produttori ha sviluppato un nuovo meccanismo di sicurezza: il sistema PAM (*pluggable authentication modules*), basato su una libreria condivisa che può essere usata da ogni componente del sistema che abbia bisogno di autenticare utenti. Una versione di questo sistema è disponibile per il sistema LINUX. Il sistema PAM permette di caricare su richiesta moduli di autenticazione secondo le indicazioni contenute in un file di configurazione valido per tutto il sistema. Un nuovo meccanismo di autenticazione aggiunto in seguito si può registrare nel file di configurazione in modo che tutti i componenti del sistema possano immediatamente usufruirne. I moduli PAM sono in grado di specificare metodi di autenticazione, limitazioni alle attività degli utenti, funzioni di avvio di una sessione di lavoro o funzioni di cambio di una parola d'ordine. In quest'ultimo caso, quando un utente cambia la propria parola d'ordine, tutti i necessari meccanismi di autenticazione possono essere aggiornati in un'unica soluzione.

20.11.2 Controllo degli accessi

Nei sistemi UNIX, e così anche nel sistema LINUX, il controllo degli accessi è realizzato per mezzo d'identificatori numerici unici: un identificatore d'utente (*uid*) individua un singolo utente o un singolo insieme di diritti d'accesso, un identificatore di gruppo (*gid*) è un identificatore aggiuntivo che si può usare per determinare i diritti di più di un utente.

Il controllo degli accessi si applica a vari oggetti del sistema: ogni file disponibile nel sistema è protetto dal meccanismo standard del controllo degli accessi, e ciò vale anche per altri oggetti condivisi come le regioni di memoria condivise e i semafori.

Ogni oggetto in un sistema UNIX sottoposto al controllo degli accessi di singoli utenti o gruppi dispone di un unico uid e un unico gid associati; anche i processi utenti hanno un unico uid, ma possono avere più di un gid. Se l'uid di un processo corrisponde all'uid di un oggetto, quel processo gode dei **diritti d'utente** o dei **diritti di proprietà** di quell'oggetto; se gli uid non corrispondono ma uno dei gid di un processo corrisponde al gid di un oggetto, il processo gode dei **diritti di gruppo** su quell'oggetto; altrimenti, il processo ha i **diritti generici** (*world right*) sull'oggetto.

Il sistema LINUX esegue il controllo degli accessi assegnando agli oggetti una **maschera di protezione** che specifica quali modi d'accesso — lettura, scrittura o esecuzione — si devono concedere ai processi con diritti d'accesso proprietari, di gruppo o generici. Il proprietario di un oggetto, ad esempio, potrebbe avere accesso a un file per la sua lettura, scrittura ed esecuzione; gli utenti di un certo gruppo potrebbero avere accesso per la lettura ma non per la scrittura; e chiunque altro potrebbe non avere alcun diritto d'accesso.

L'unica eccezione a quanto detto è costituita dallo uid privilegiato *root*: un processo dotato di questo speciale uid gode automaticamente dei diritti d'accesso a qualunque oggetto del sistema, scavalcando i normali controlli; processi di questo tipo hanno anche il permesso di eseguire operazioni privilegiate come la lettura di qualunque regione della

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

memoria fisica o aprire socket di rete riservate. Questo meccanismo permette al nucleo di impedire agli utenti ordinari di accedere a determinate risorse del sistema: la maggior parte delle risorse chiave interne del nucleo sono implicitamente di proprietà dell'uid *root*.

Il sistema LINUX adotta il meccanismo standard `setuid` dello UNIX descritto nel Paragrafo A.3.2; esso permette a un programma di godere, durante una certa esecuzione, di privilegi diversi da quelli dell'utente che esegue il programma: ad esempio, il programma `lpr` che pone un file in coda di stampa ha accesso alle code di stampa del sistema, anche se l'utente che ne richiede l'esecuzione non lo ha. La realizzazione nello UNIX di `setuid` distingue fra lo uid *reale* e lo uid *effettivo* di un processo: il primo è quello dell'utente che richiede l'esecuzione del programma; il secondo è quello del proprietario del file.

Nel LINUX questo meccanismo è esteso in due direzioni. In primo luogo realizza il meccanismo `saved user-id` secondo le specifiche POSIX; si tratta di permettere a un processo di perdere e riacquisire ripetutamente il suo uid effettivo. Per ragioni di sicurezza, infatti, un programma potrebbe voler eseguire la maggior parte delle sue operazioni in un modo sicuro, rinunciando ai privilegi conferitigli dal suo `setuid` status, ma potrebbe voler usufruire di tutti i suoi privilegi durante le esecuzioni di alcune operazioni. Le versioni standard dello UNIX riescono a fornire questo servizio solo scambiando gli uid reali ed effettivi; lo uid effettivo precedente è ricordato, ma lo uid reale del programma non sempre corrisponde allo uid dell'utente che ne richiede l'esecuzione. Il meccanismo `saved user-id` permette a un processo di rendere il suo uid effettivo uguale al suo uid reale e di ritornare poi al valore precedente del suo uid effettivo senza dover mai modificare il valore dello uid reale.

Il secondo miglioramento apportato dal sistema LINUX è l'aggiunta di una caratteristica dei processi che permette di usufruire di un sottoinsieme dei diritti conferiti dallo uid effettivo: le proprietà *fsuid* e *fsgid* di un processo sono usate quando è consentito l'accesso a un file, e sono attive ogniqualvolta lo uid effettivo o il gid lo sono; tuttavia, fsuid e fsgid possono essere attivate indipendentemente dagli identificatori effettivi, cosa che permette a un processo di accedere ai file per conto di un altro utente senza dover assumere in alcun senso l'identità di quell'utente. In particolare, i processi server possono impiegare questo meccanismo per fornire file a un certo utente senza divenire suscettibili di terminazione o sospensione da parte di quell'utente.

Il LINUX offre un altro metodo per il passaggio flessibile dei diritti da un programma a un altro, divenuto comune nelle moderne versioni dello UNIX. Una volta che un collegamento fra due processi del sistema sia stato istituito per mezzo di una socket locale, ognuno dei due processi può inviare all'altro un descrittore di file relativo a uno dei suoi file aperti; l'altro processo riceve quindi un descrittore duplicato dello stesso file: ciò permette a un client di offrire a un processo server l'accesso selettivo a un solo file senza conferirgli alcun altro privilegio. Ad esempio, non è più necessario che un server di stampa sia in grado di leggere tutti i file di un utente che chieda la stampa di alcuni file; l'utente, e cioè il client, potrebbe semplicemente comunicare al server i descrittori dei file dei quali richiede la stampa, negando al server la possibilità di accedere agli altri file di sua proprietà.

20.12 Sommario

LINUX è un moderno sistema operativo gratuito basato sugli standard UNIX. È stato progettato per essere eseguito efficientemente e in modo affidabile sui comuni PC, ma può anche essere eseguito su diverse altre piattaforme. Fornisce un'interfaccia per il programmatore e un'interfaccia per l'utente compatibili con i sistemi UNIX standard, e può eseguire moltissime applicazioni per il sistema UNIX, compreso un crescente numero di applicazioni commerciali.

Il sistema LINUX non si è sviluppato dal nulla: nel suo complesso comprende molti componenti originariamente sviluppati indipendentemente da esso. La parte centrale del nucleo del sistema operativo è interamente originale, ma permette l'esecuzione di molti programmi esistenti per il sistema UNIX; ciò rende il tutto un completo sistema operativo compatibile con UNIX e non soggetto a limitazioni legali imposte dagli autori del codice.

Per ragioni legate alle prestazioni il nucleo del LINUX è realizzato secondo schemi tradizionali come un blocco monolitico di codice, ma è sufficientemente modulare da permettere alla maggior parte dei driver di essere caricati o rimossi dinamicamente.

Il sistema LINUX è un sistema multiutente che fornisce meccanismi di protezione dei processi ed è in grado di eseguire più processi a partizione del tempo. Un nuovo processo può condividere selettivamente parti del suo ambiente d'esecuzione con il processo genitore, permettendo di realizzare la programmazione multithread. La comunicazione fra processi sfrutta sia i meccanismi del System V — code di messaggi, semafori e memoria condivisa — sia l'interfaccia a socket del BSD. È possibile usufruire simultaneamente di più protocolli di rete differenti per mezzo dell'interfaccia a socket.

Dal punto di vista dell'utente, il file system è un albero di directory conforme alla semantica UNIX; internamente LINUX usa uno strato d'astrazione per gestire molti file system differenti. Gestisce file system virtuali, di rete e orientati ai dispositivi. I file system orientati ai dispositivi accedono al disco attraverso due cache: i dati passano attraverso la cache delle pagine unificata con il sistema della memoria virtuale, mentre i metadati passano attraverso la *buffer cache*, una cache distinta indicizzata dai blocchi fisici del disco.

Il sistema per la gestione della memoria usa la condivisione delle pagine e la copiatura su scrittura per minimizzare la duplicazione dei dati condivisi da processi diversi. Le pagine sono caricate quando si genera un riferimento a esse, e sono trasferite di nuovo nella memoria ausiliaria secondo un algoritmo LFU quando è necessario riappropriarsi di regioni della memoria fisica.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

20.13 Esercizi

20.1 Il sistema operativo LINUX è eseguibile su diverse piattaforme. Dite cosa devono fare gli sviluppatori del LINUX per assicurare l'adattabilità del sistema a diverse CPU e diverse architetture per la gestione della memoria, e per minimizzare la quantità richiesta di codice del nucleo specifico per le singole architetture.

20.2 I moduli del nucleo caricabili dinamicamente consentono di aggiungere driver al sistema in maniera flessibile. Dite se hanno anche qualche svantaggio. Dite in quali circostanze sarebbe preferibile compilare un nucleo in un unico file eseguibile, e quando invece sarebbe più opportuno dividerlo in moduli indipendenti. Motivate le risposte.

20.3 La programmazione multithread è una tecnica diffusa; descrivete tre possibilità di realizzazione dei thread, e confrontate queste proposte con il meccanismo `clone` del LINUX, spiegando in quali circostanze ognuno dei meccanismi alternativi risulterebbe migliore di quello adottato dal LINUX.

20.4 Dite quali sono i costi aggiuntivi implicati dalla creazione e dallo scheduling di un processo rispetto al costo di un thread ottenuto col meccanismo `clone`.

20.5 Lo scheduler del LINUX realizza lo scheduling in tempo reale debole; dite quali funzioni, necessarie per certi compiti di programmazione per elaborazioni in tempo reale, mancano, e come si potrebbero includere nel nucleo.

20.6 Il nucleo del LINUX non permette il trasferimento nei dischi della memoria del nucleo; individuate le conseguenze di questa limitazione sul progetto del nucleo, ed enunciate due vantaggi e due svantaggi di questa scelta progettuale.

20.7 Nel sistema LINUX le librerie condivise eseguono molte operazioni fondamentali per il sistema operativo. Dite qual è il vantaggio di scorporare dal nucleo il codice relativo a questi servizi, ed elencate anche gli eventuali svantaggi. Motivate le risposte.

20.8 Elencate tre vantaggi del collegamento dinamico delle librerie rispetto al collegamento statico; individuate due casi in cui sarebbe preferibile il collegamento statico.

20.9 Confrontate l'uso delle socket e quello della memoria condivisa come meccanismi di comunicazione di dati fra i processi in un singolo calcolatore. Menzionate due tra i vantaggi di ciascun metodo e dite quando l'uno è preferibile all'altro.

20.10 I sistemi UNIX adottavano comunemente tecniche di ottimizzazione relative alla posizione dei dati che tenevano conto della rotazione del disco, ma le versioni moderne, compreso LINUX, ottimizzano solo l'accesso sequenziale ai dati: spiegate perché scelgono questa strada. Dite di quali caratteristiche dei dispositivi si avvantaggia l'accesso sequenziale e spiegate perché l'ottimizzazione rotazionale non è più così utile.

20.11 Il codice sorgente è gratuitamente disponibile tramite la rete Internet o presso i rivenditori di CD-ROM. Individuate tre possibili conseguenze riguardanti la sicurezza del sistema LINUX.

20.14 Note bibliografiche

Il sistema LINUX è un prodotto della rete Internet; di conseguenza, la maggior parte della documentazione disponibile su LINUX si trova in qualche forma in rete. I seguenti siti chiave rimandano alla maggior parte delle informazioni utili esistenti:

- Le *LINUX Cross Reference Pages* all'indirizzo `http://lxr.linux.no/` contengono il codice sorgente aggiornato del nucleo del LINUX, consultabile tramite il Web e con riferimenti incrociati esaustivi.
- *LINUX-HQ* all'indirizzo `http://www.linuxhq.com/` ospita un gran numero di informazioni relative ai *Kernel 2.x*. Questo sito comprende anche collegamenti agli indirizzi della maggior parte delle versioni di distribuzione del sistema LINUX, oltre ad archivi dei principali indirizzari.
- *LINUX Documentation Project* all'indirizzo `http://sunsite.unc.edu/linux/` elenca molti libri sul sistema LINUX disponibili in formato sorgente come parte del *LINUX Documentation Project*. Il sito ospita anche le guide *LINUX How-To*, che contengono una serie di consigli e suggerimenti riguardanti diversi aspetti del sistema LINUX.
- *Kernel Hacker's Guide* è una guida in rete ai meccanismi interni dei nuclei in generale; l'indirizzo di questo sito costantemente in espansione è `http://www.redhat.com:8080/HyperNews/get/khg.html`.

Inoltre esistono molti indirizzari dedicati al sistema LINUX; i più importanti sono tenuti da un gestore che può essere raggiunto all'indirizzo di posta elettronica `majordomo@vger.rutgers.edu`. Per ottenere informazioni su come iscriversi a uno di questi indirizzari inviate a tale indirizzo un messaggio contente la sola parola "help".

Il solo libro che descrive il funzionamento interno del nucleo del LINUX è attualmente [Beck et al. 1998]. Per approfondimenti sul sistema operativo UNIX in generale, [Vahalia 1996] è un buon punto di partenza.

Infine, lo stesso sistema LINUX si può ottenere tramite la rete Internet. I pacchetti di distribuzione completi si possono ottenere dai siti delle aziende produttrici; la comunità LINUX mantiene archivi di componenti del sistema nella rete Internet; gli indirizzi più importanti sono i seguenti:

- `ftp://tsx-11.mit.edu/pub/linux/`
- `ftp://sunsite.unc.edu/pub/linux/`
- `ftp://linux.kernel.org/pub/linux/`

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 21

Windows 2000

Il sistema Microsoft Windows 2000 è un sistema operativo multitasking a 32 bit concepito per le CPU Intel Pentium e successive. È un'evoluzione del sistema Windows NT ed era precedentemente denominato Windows NT 5.0. Gli obiettivi chiave di questo sistema sono l'adattabilità a diverse architetture, la sicurezza, il rispetto dello standard POSIX o IEEE Std. 1003.1, la gestione di più unità d'elaborazione, l'estendibilità, la realizzazione d'interfacce d'utente internazionali, e la compatibilità con le applicazioni per i sistemi MS-DOS e Microsoft Windows. In questo capitolo sono trattati gli obiettivi chiave del sistema, la sua architettura a strati che lo rende così facile da usare, il file system, le reti e l'interfaccia di programmazione

21.1 Storia

Nella metà degli anni Ottanta la Microsoft e l'IBM cooperarono per sviluppare il sistema operativo OS/2, scritto in linguaggio assemblativo per sistemi a singola CPU Intel 80286. Nel 1988 la Microsoft decise d'intraprendere una strada diversa sviluppando un sistema operativo facilmente adattabile dalla 'nuova tecnologia' (NT), che includesse le interfacce per la programmazione delle applicazioni (API) OS/2 e POSIX. Nell'ottobre del 1988 Dave Cutler, progettista del sistema operativo DEC VAX/VMS, fu assunto e incaricato della progettazione e realizzazione di questo nuovo sistema operativo.

Originariamente per il sistema Windows NT si sarebbe dovuta adottare l'API del sistema operativo OS/2 come suo ambiente naturale, ma durante lo sviluppo si scelse l'API a 32 bit dell'ambiente Windows (Win32); questa scelta rifletteva la popolarità dell'ambiente Windows 3.0. Le prime versioni del sistema Windows NT furono Windows NT 3.1 e Windows NT 3.1 Advanced Server (in quel periodo, la più recente versione dell'ambiente Windows a 16 bit era la 3.1). Nella versione 4.0, il sistema Windows NT adottò l'interfaccia d'utente Windows 95 e incorporò il software Web server e il programma di consultazione del Web; inoltre le procedure dell'interfaccia d'utente e il codice per la gra-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

fica furono spostati all'interno del nucleo al fine di migliorare le prestazioni, con l'effetto collaterale di ridurre l'affidabilità del sistema. Sebbene precedenti versioni del sistema Windows NT siano state adattate ad altre architetture, per quel che riguarda il sistema operativo Windows 2000 quest'orientamento non è stato seguito per motivi di mercato. Così l'adattabilità ora si riferisce alle diverse architetture Intel. Il sistema operativo Windows 2000 usa un micronucleo (come il Mach) in questo modo le estensioni si possono realizzare in una parte del sistema operativo senza influire in modo rilevante sulle altre parti. Con l'aggiunta dei servizi per terminali (*terminal services*), il sistema Windows 2000 è un sistema operativo multiutente.

Il sistema operativo Windows 2000 è stato rilasciato nel 2000 e incorpora significativi cambiamenti. I servizi di directory basati su X.500, una migliore gestione dei servizi di rete, la gestione dei dispositivi *plug and play*, un nuovo file system con la gestione della memorizzazione gerarchica, un file system distribuito e la gestione di più unità d'elaborazione e di una maggiore quantità di memoria.

Di questo sistema esistono quattro versioni. La versione Professional è concepita per le stazioni di lavoro e i PC. Le altre tre sono versioni per server: Server, Advanced Server e Datacenter Server. Queste differiscono principalmente nella quantità di memoria e nel numero di unità d'elaborazione gestite. Usano lo stesso nucleo e lo stesso codice del sistema operativo, ma le versioni Windows 2000 Server e Advanced Server sono configurate per applicazioni client-server e possono funzionare come server di applicazioni per reti NetWare e Microsoft. Windows 2000 Datacenter Server gestisce fino a 32 unità d'elaborazione e fino a 64 GB di RAM.

Nel 1996, sono state vendute più licenze di Windows NT Server dell'insieme delle versioni di UNIX. Il codice di base del sistema operativo Windows 2000 è di circa 30 milioni di righe, il codice di base del Windows NT 4.0 è di circa 18 milioni di righe.

21.2 Princìpi di progettazione

Gli obiettivi di progettazione dichiarati dalla Microsoft per il sistema operativo Windows 2000 includono l'estendibilità, l'adattabilità, l'affidabilità, le prestazioni e la realizzazione d'interfacce d'utente internazionali.

L'**estendibilità** si riferisce alla capacità di un sistema operativo di tenere il passo con gli sviluppi della tecnologia informatica. Il sistema Windows 2000 ha una struttura stratificata che facilita eventuali cambiamenti nel corso del tempo. Il suo modulo esecutivo, che è eseguito nel modo protetto o di nucleo, fornisce i servizi di sistema fondamentali; sopra l'esecutivo operano molti sottosistemi server eseguiti nel modo d'utente, fra i quali ci sono i **sottosistemi d'ambiente** che simulano differenti sistemi operativi: in questo modo, un programma scritto per l'MS-DOS, Microsoft Windows o POSIX può essere eseguito dal sistema operativo nell'ambiente appropriato. (Si veda il Paragrafo 21.4 per maggiori informazioni sui sottosistemi d'ambiente.) Grazie alla struttura modulare è possibile aggiungere nuovi sottosistemi d'ambiente senza che ciò abbia ripercussioni sull'esecutivo; inoltre, il sistema Windows 2000 usa driver caricabili all'interno del sistema

di I/O, cosicché si possono aggiungere nuovi file system e nuovi tipi di dispositivi di I/O o di rete mentre il sistema è attivo. Il sistema Windows 2000, come il Mach, adotta un modello client-server e gestisce l'elaborazione distribuita per mezzo di chiamate di procedure remote (RPC) secondo le specifiche dell'Open Software Foundation.

Un sistema operativo è **adattabile** (*portable*) se per poter essere eseguito in un'altra architettura richiede un numero di modifiche relativamente piccolo; il sistema Windows 2000 è progettato per essere adattabile: così come per il sistema operativo UNIX, la maggior parte del sistema è scritta in Linguaggio C o in C++, e tutto il codice dipendente dalla CPU è isolato in una libreria dinamica (*dynamic link library* — DLL) che costituisce uno strato d'astrazione dall'architettura detta HAL (*hardware abstraction layer*). Una DLL è un file associato allo spazio d'indirizzi di un processo in modo che ogni funzione della DLL appaia come parte del processo. Il fatto che i livelli superiori del sistema operativo dipendano dallo HAL piuttosto che dall'architettura sottostante contribuisce all'adattabilità del sistema operativo: lo HAL agisce direttamente sui dispositivi fisici, isolando il resto del sistema operativo dalle differenze fra le piattaforme sulle quali si esegue il sistema.

L'**affidabilità** è la capacità di gestire le condizioni d'errore, compresa la capacità del sistema operativo di proteggere se stesso e i suoi utenti da programmi difettosi o intenzionalmente dannosi. Il sistema operativo Windows 2000 resiste ai difetti e agli attacchi usando la protezione per la memoria virtuale offerta dall'architettura e meccanismi di protezione del sistema operativo per le proprie risorse; esso, inoltre, è dotato di un proprio file system (NTFS) che gestisce automaticamente molti tipi di errori relativi ai file che possono verificarsi come conseguenza di un crollo del sistema. Il governo degli Stati Uniti ha assegnato alla versione 4.0 del sistema Windows NT la classe di sicurezza C2, che indica un moderato livello di protezione dai programmi difettosi e dagli attacchi volontari al sistema. La classificazione governativa del sistema Windows 2000 è in corso di valutazione. Per ulteriori informazioni sulla classificazione della sicurezza, si veda il Paragrafo 19.8.

Il sistema Windows 2000 offre la **compatibilità** al livello sorgente alle applicazioni che seguono lo standard IEEE 1003.1 (POSIX); queste possono quindi essere compilate ed eseguite senza che sia necessario modificare il codice sorgente. Inoltre, può eseguire molti programmi compilati per le architetture Intel *x*86 negli ambienti MS-DOS, Windows a 16 bit, OS/2, Lan Manager e Windows a 32 bit. Usando i già menzionati sottosistemi d'ambiente. Questi sottosistemi sono in grado di gestire una serie di file system, fra i quali ci sono la FAT dell'MS-DOS, lo HPFS dell'OS/2, l'ISO9660 CD, e l'NTFS. La compatibilità al livello del codice eseguibile però non è perfetta: nell'MS-DOS, ad esempio, le applicazioni possono accedere direttamente alle porte di comunicazione, mentre per motivi di sicurezza e affidabilità ciò non è permesso dal sistema Windows 2000.

Il sistema operativo Windows 2000 è concepito per raggiungere buone **prestazioni**. I sottosistemi che lo costituiscono possono comunicare efficientemente grazie a un meccanismo di chiamata di procedura locale (LPC) che permette la trasmissione di messaggi ad alte prestazioni. Eccezion fatta per il nucleo, i thread dei sottosistemi possono essere interrotti dai thread con priorità più alta, in modo che il sistema possa reagire pronta-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

mente agli eventi esterni. Inoltre, il sistema operativo è progettato per l'elaborazione parallela simmetrica: su un calcolatore con più unità d'elaborazione, più thread possono essere eseguiti contemporaneamente. La scalabilità attuale è però limitata rispetto ai sistemi UNIX: mentre il sistema operativo Solaris può gestire sistemi con 64 unità d'elaborazione, alla fine del 2000 il sistema Windows 2000 poteva controllare fino a 32 unità d'elaborazione. Il sistema operativo Windows NT gestiva non più di 8 unità d'elaborazione.

Il sistema Windows 2000 è anche stato concepito per un'**utenza internazionale**. Esso si adatta alle lingue locali grazie all'API per la **gestione delle lingue nazionali** (*national language support* — NLS), che fornisce procedure specializzate per trattare le date, gli orari e le valute in accordo agli usi dei diversi paesi. I confronti fra sequenze di caratteri tengono conto dei diversi alfabeti. Il codice dei caratteri proprio del Windows 2000 è lo UNICODE, i caratteri ANSI sono convertiti in UNICODE (da 8 a 16 bit) prima della manipolazione.

21.3 Componenti del sistema

La struttura del sistema operativo Windows 2000 è un sistema stratificato di moduli (Figura 21.1). Gli strati principali sono quello dell'astrazione dall'architettura fisica, il nucleo e l'esecutivo, operanti in modo protetto, e un'ampia raccolta di sottosistemi eseguiti nel modo d'utente; questi ultimi si dividono in due categorie: i sottosistemi d'ambiente, che simulano sistemi operativi diversi, e i **sottosistemi di protezione**, che forniscono servizi di sicurezza. Uno dei vantaggi principali di questo tipo d'organizzazione è che le interazioni fra i moduli sono semplificate. Il resto di questo paragrafo descrive i diversi strati e sottosistemi.

21.3.1 Strato di astrazione dall'architettura

Lo strato di astrazione dall'architettura (*hardware abstraction layer* — HAL) è uno strato di programmi che nasconde agli strati superiori le differenze presenti nelle architetture fisiche, contribuendo all'adattabilità del sistema operativo. Lo HAL realizza una macchina virtuale usata come interfaccia dal nucleo, dall'esecutivo e dai driver dei dispositivi; un vantaggio di questo metodo è che è sufficiente una sola versione di ogni driver: si può eseguire su tutte le piattaforme senza doverne modificare il codice. Lo HAL fornisce anche funzioni di gestione dell'elaborazione parallela simmetrica. Per motivi legati alle prestazioni, i driver dei dispositivi di I/O (e i driver della grafica) possono accedere direttamente ai controllori dei dispositivi.

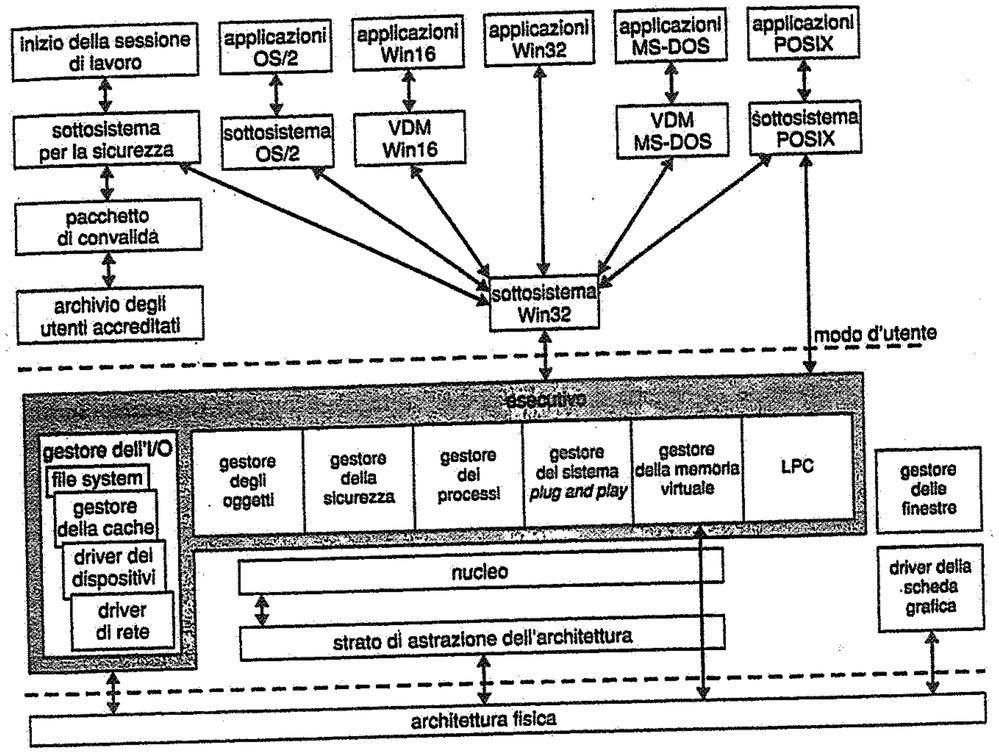


Figura 21.1 Schema a blocchi del sistema operativo Windows 2000.

21.3.2 Nucleo

Il nucleo del sistema operativo rappresenta le fondamenta sulle quali sono costruiti l'esecutivo e i sottosistemi. Il nucleo risiede sempre nella memoria centrale e la sua esecuzione non può mai essere interrotta. Esso ha quattro compiti principali: lo scheduling dei thread, la gestione delle interruzioni e delle eccezioni, la sincronizzazione a basso livello della CPU, e la gestione dei problemi che sorgono a seguito di una caduta di tensione elettrica.

Il nucleo è orientato agli oggetti: un **tipo di oggetto** è un tipo di dato definito dal sistema che ha un insieme di attributi (valori dei dati) e possiede un insieme di metodi (cioè, funzioni o operazioni). Un **oggetto** è semplicemente un'istanza di uno specifico tipo d'oggetto. Il nucleo svolge i suoi compiti usando un insieme d'oggetti i cui attributi memorizzano i dati necessari al nucleo, e i cui metodi eseguono le attività del nucleo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il nucleo fa uso di due serie di oggetti; la prima è costituita dagli **oggetti dispatcher**, i quali controllano la sincronizzazione e la comunicazione all'interno del sistema. Esempi di oggetti di questo tipo sono gli eventi, i semafori, i thread, i temporizzatori, e i cosiddetti *mutant* e *mutex*. Gli oggetti **mutant** realizzano la mutua esclusione nel modo d'utente o di nucleo grazie alla nozione di proprietà; gli oggetti **mutex**, disponibili solo nel modo protetto, realizzano anch'essi la mutua esclusione, garantendo però che il sistema non andrà in stallo. L'**oggetto evento** si usa per registrare il verificarsi di un evento e per sincronizzare l'evento stesso con qualche altra azione. Un **oggetto semaforo** si comporta come un contatore al fine di controllare il numero di thread che accedono a una risorsa. L'**oggetto thread** è l'entità eseguita dal nucleo e associata a un **oggetto processo**. Gli **oggetti temporizzatori** sono usati come orologi, ad esempio per segnalare la necessità d'interrompere operazioni che si prolunghino oltre un certo limite.

La seconda serie di oggetti di nucleo è costituita dagli **oggetti di controllo**, che comprendono le chiamate di procedure asincrone, i segnali d'interruzione, la notificazione dei cali di tensione (*power notify*), lo stato dell'alimentazione (*power status*), i processi, e gli oggetti profilo. Il sistema usa una chiamata di procedura asincrona (APC) per interrompere un thread in esecuzione chiamando una procedura. L'**oggetto interruzione** associa una procedura di servizio di un'interruzione a una fonte di segnali d'interruzione. Il sistema usa l'**oggetto di notificazione di un calo di tensione** per invocare automaticamente una specifica procedura a seguito di un calo di tensione, e l'**oggetto di stato dell'alimentazione** per controllare se si sia effettivamente verificato un calo di tensione. Un oggetto processo rappresenta lo spazio d'indirizzi virtuali e le informazioni di controllo necessarie per eseguire l'insieme di thread associato a un processo. Infine il sistema usa l'**oggetto profilo** per misurare la quantità di tempo impiegata da una parte di codice.

21.3.2.1 Thread e scheduling

Come molti altri moderni sistemi operativi, il sistema Windows 2000 associa le nozioni di processo e thread al codice eseguibile. Il processo possiede uno spazio d'indirizzi nella memoria virtuale e altre informazioni, ad esempio una priorità di base e dei gradi d'affinità relativi a determinate unità d'elaborazione. Ogni processo ha uno o più thread, cioè unità d'esecuzione elaborate dal nucleo; ogni thread possiede un suo stato, che comprende una priorità, dei gradi di affinità relativi alle unità d'elaborazione, e informazioni di contabilizzazione delle risorse.

I sei possibili stati di un thread sono: pronto, in *standby*, in esecuzione, in attesa, in transizione, e terminato. Un thread è **pronto** quando attende d'essere eseguito. Il thread con priorità più alta è posto in *standby* e sarà il prossimo thread a essere eseguito; in un sistema con più unità d'elaborazione c'è un thread in *standby* per ogni unità d'elaborazione. Un thread è **in esecuzione** quando un'unità d'elaborazione lo sta per l'appunto eseguendo, in tal caso rimane in questo stato fino a che non è interrotto da un thread con priorità più alta, finché non termina la sua esecuzione o scade il suo quanto di tempo, o ancora fino a che non esegue una chiamata del sistema bloccante, ad esempio di I/O. Un thread è **in attesa** quando aspetta un segnale, ad esempio relativo al completamento dell'I/O. Un nuovo thread è in stato di **transizione** quando attende le risorse necessarie all'esecuzione. Un thread è **terminato** quando ha completato l'esecuzione.

Il dispatcher adotta uno schema con 32 livelli di priorità per determinare l'ordine d'esecuzione dei thread; le priorità si dividono in due classi: la classe a priorità variabile, che contiene i thread le cui priorità sono comprese fra 0 e 15, e la classe per l'elaborazione in tempo reale, che contiene i thread le cui priorità vanno da 16 a 31. Il dispatcher usa una coda per ogni livello di priorità, e percorre l'insieme delle code partendo dalla priorità più alta fino a che incontra un thread pronto per l'esecuzione. Se un thread ha una particolare affinità con una certa unità d'elaborazione che non è disponibile in quel momento, il dispatcher lo ignora e continua a cercare un thread pronto per l'esecuzione; se nessun thread è pronto, il dispatcher esegue un thread speciale detto **thread d'attesa** (*idle thread*).

Quando scade il suo quanto di tempo, l'esecuzione del thread viene interrotta e, se il thread appartiene alla classe a priorità variabile, si riduce la sua priorità, anche se mai sotto la priorità di base. Il fatto di ridurre la priorità tende a limitare l'impiego di tempo di CPU da parte dei thread con prevalenza d'elaborazione. Quando un thread a priorità variabile abbandona lo stato d'attesa, il dispatcher aumenta la priorità in funzione dell'evento atteso dal thread: un thread che attende I/O dalla tastiera otterrà un notevole aumento di priorità, mentre un thread che attende un'operazione di I/O da un disco otterrà un aumento più moderato. Questa strategia tende a garantire brevi tempi di risposta ai thread interattivi che usano il mouse e le finestre, e permette ai thread con prevalenza di all'I/O di tenere occupati i dispositivi di I/O mentre i thread con prevalenza d'elaborazione possono utilizzare in sottofondo i cicli di CPU disponibili; si tratta di un metodo usato da molti sistemi operativi a partizione del tempo, UNIX compreso. Inoltre, la finestra con cui l'utente sta interagendo riceverà un aumento di priorità in modo da migliorare i tempi di risposta.

Lo scheduling può avvenire quando un thread è pronto o è posto nello stato d'attesa, quando termina, o quando un'applicazione ne cambia la priorità o il grado di affinità con un'unità d'elaborazione. Se un thread per l'elaborazione in tempo reale a più alta priorità diviene pronto durante l'esecuzione di un thread a priorità più bassa, quest'ultimo sarà interrotto; ciò fornisce al thread per l'elaborazione in tempo reale un accesso preferenziale alla CPU quando ne ha bisogno; il sistema Windows 2000, tuttavia, non è un sistema operativo per l'elaborazione in tempo reale stretto perché non garantisce che un thread per l'elaborazione in tempo reale sia eseguito entro limiti di tempo fissati.

21.3.2.2 Segnali di eccezione e d'interruzione

Il nucleo fornisce anche la gestione delle eccezioni e delle interruzioni; definisce diverse eccezioni indipendenti dall'architettura, comprese le violazioni d'accesso alla memoria, il superamento del limite per il massimo intero, il superamento dei limiti massimo e minimo per i numeri rappresentati in virgola mobile, la divisione per zero intera e in virgola mobile, la richiesta d'esecuzione di un'istruzione illegale, l'allineamento scorretto dei dati, la richiesta d'esecuzione di un'istruzione privilegiata, l'errore di lettura di una pagina, il tentativo d'accesso a una pagina di protezione, il superamento della quota di spazio assegnata a un file, la presenza di un punto d'interruzione dell'esecuzione (*breakpoint*) e l'esecuzione a passo singolo per i programmi d'ausilio all'individuazione e correzione degli errori.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Le eccezioni semplici possono essere trattate dal gestore delle eccezioni; le altre sono gestite dal **dispatcher delle eccezioni** del nucleo, il quale annota la causa dell'eccezione per individuarne l'appropriato gestore.

Quando si verifica un'eccezione nel modo protetto, il dispatcher delle eccezioni invoca semplicemente una procedura che ne individua il gestore; nel caso in cui non se ne trovi alcuno avviene un errore di sistema fatale che lascia l'utente davanti al famigerato 'schermo blu della morte', il quale sta a significare l'arresto totale del sistema.

La gestione delle eccezioni è più complessa nel caso dei processi eseguiti nel modo d'utente, perché un sottosistema d'ambiente (ad esempio il POSIX) può specificare una porta di correzione e una porta per le eccezioni per ogni processo da esso creato. Se una porta di correzione è registrata, il dispatcher delle eccezioni trasmette l'eccezione a tale porta; se non si trova una porta di correzione o se essa non è in grado di gestire l'eccezione, il dispatcher tenta di individuare un appropriato gestore dell'eccezione; se non riesce a trovarlo, chiama di nuovo il programma per la correzione degli errori in modo che esso possa rilevare l'errore. Se tale programma non è in esecuzione il dispatcher manda un messaggio alla porta per le eccezioni del processo per dare al sottosistema d'ambiente la possibilità di tradurre l'eccezione: ad esempio l'ambiente POSIX traduce messaggi d'eccezione in segnali POSIX prima di mandarli al thread che ha causato l'eccezione. In ultima analisi, se tutto il resto non ha funzionato il nucleo termina forzatamente il processo contenente il thread che ha causato l'eccezione.

Il dispatcher delle interruzioni del nucleo gestisce le interruzioni chiamando una procedura di servizio delle interruzioni (come in un driver di un dispositivo) o una procedura interna del nucleo. Un segnale d'interruzione è rappresentato da un oggetto interruzione che contiene tutte le informazioni necessarie per gestire l'interruzione, il che rende facile associare procedure di servizio a un segnale d'interruzione senza dover accedere direttamente al dispositivo in questione.

Diverse architetture hanno tipi e numeri d'interruzione diversi; per garantire l'adattabilità, il dispatcher delle interruzioni associa le interruzioni a un insieme convenzionale. Ai segnali d'interruzione è assegnata una priorità e sono serviti secondo questa priorità in ordine decrescente; nel sistema Windows 2000 ci sono 32 livelli d'interruzione (IRQL): otto sono riservati all'uso del nucleo; gli altri ventiquattro rappresentano segnali d'interruzione provenienti dai dispositivi tramite la mediazione dello HAL (sebbene la maggior parte dei sistemi *x*86 usi solo 16 linee). L'elenco delle interruzioni è riportato nella Figura 21.2.

Il nucleo usa una **tabella di dispatch delle interruzioni** per associare a ogni livello delle interruzioni una procedura di servizio; nel caso di un calcolatore con più unità d'elaborazione il nucleo mantiene tabelle delle interruzioni separate per ogni unità d'elaborazione, e l'IRQL di ogni unità d'elaborazione si può regolare in modo indipendente per mascherare determinate interruzioni. Tutte le interruzioni che occorrono a un livello pari o inferiore dell'IRQL di un'unità d'elaborazione sono bloccate fino a che l'IRQL non è abbassato da un thread al livello del nucleo. Il sistema Windows 2000 sfrutta questa caratteristica impiegando le eccezioni per eseguire funzioni di sistema: ad esempio, il nucleo usa eccezioni per avviare il dispatch di un thread, per gestire i temporizzatori, e per eseguire operazioni asincrone.

Livelli di interruzione	Tipi di interruzione
31	controllo di macchina o errore sul bus
30	calo di tensione
29	comunicazione fra unità d'elaborazione (richiesta d'azione a un altra unità d'elaborazione, ad esempio, avvio di un processo o aggiornamento del TLB)
28	orologio
27	profile
3-26	ordinarie interruzioni IRQ del PC
2	dispatch e chiamata di procedura differita (DPC) (nucleo)
1	chiamata di procedura asincrona (APC)
0	passiva

Figura 21.2 Segnali d'interruzione del sistema Windows 2000.

Il nucleo usa le interruzioni di dispatch per controllare i cambi di contesto dei thread. Quando il nucleo è in esecuzione, alza l'IRQL relativo all'unità d'elaborazione appropriata sopra il livello di dispatch. Se determina la richiesta di dispatch di un thread, il nucleo genera un'interruzione di dispatch; tale interruzione è però bloccata finché il nucleo finisce ciò che sta facendo e abbassa l'IRQL; a questo punto l'interruzione di dispatch sceglie un thread da eseguire.

Se decide che qualche funzione di sistema dovrà essere eseguita in un futuro non immediato, il nucleo accoda un oggetto di **chiamata di procedura differita** (*deferred procedure call* — DPC), contenente l'indirizzo della funzione da eseguire, e genera un'interruzione DPC; gli oggetti DPC si eseguono quando l'IRQL dell'unità d'elaborazione raggiunge un livello sufficientemente basso. L'IRQL di un'interruzione DPC è tipicamente più alto di quello dei thread utenti, quindi le DPC interromperanno l'esecuzione dei thread utenti. Per evitare problemi le DPC sono abbastanza semplici: non possono modificare la memoria di un thread; creare, acquisire o attendere oggetti; chiamare i servizi di sistema o generare eccezioni di pagina mancante.

21.3.2.3 Sincronizzazione a basso livello

La terza responsabilità del nucleo consiste nel fornire la sincronizzazione a basso livello. Il meccanismo di **chiamata di procedura asincrona** (*asynchronous procedure call* — APC) è simile alla DPC, ma si usa per scopi più generali: esso permette ai thread di predisporre una chiamata di procedura che si verificherà in futuro, ma in un momento del tutto imprevedibile. Ad esempio, molti servizi di sistema accettano una procedura d'utente come parametro: invece di usare una chiamata del sistema sincrona che bloccherebbe il thread fino al completamento della chiamata del sistema, un thread utente può impartire una chiamata del sistema asincrona, fornire un'APC e continuare così l'elaborazione fino a che non sarà interrotto al momento dell'esecuzione dell'APC, dopo che il servizio di sistema è terminato.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un'APC può essere accodata sia a un thread utente sia a un thread di sistema, anche se un'APC d'utente sarà eseguita solo se il thread ha dichiarato di essere **avvertibile**. Un'APC è più potente di una DPC perché può acquisire e attendere gli oggetti, causare eccezioni di pagina mancante e chiamare servizi di sistema; poiché un'APC è eseguita nello spazio d'indirizzi del thread relativo, l'esecutivo del sistema Windows 2000 usa ampiamente le APC per elaborare l'I/O.

Il sistema operativo Windows 2000 può essere eseguito su macchine con più unità d'elaborazione simmetriche, quindi il nucleo deve impedire che due dei suoi thread modifichino contemporaneamente una struttura di dati condivisa. Il nucleo usa semafori ad attesa attiva (*spinlock*) residenti nella memoria globale per realizzare la mutua esclusione delle unità d'elaborazione; poiché tutte le attività di un'unità d'elaborazione si arrestano nel momento in cui un thread tenta di acquisire un semaforo ad attesa attiva, un thread che possieda un semaforo ad attesa attiva non viene mai interrotto, in modo che possa terminare e liberare il semaforo ad attesa attiva il più velocemente possibile.

21.3.2.4 Ripristino dopo un calo di tensione

La quarta responsabilità del nucleo consiste nel provvedere al ripristino del sistema dopo un calo di tensione. Un segnale d'interruzione da calo di tensione, che ha la seconda priorità più alta, notifica al sistema operativo il verificarsi di un calo di tensione; l'oggetto relativo fornisce ai driver dei dispositivi la possibilità di registrare una procedura che sarà chiamata al momento del ripristino dell'alimentazione per assicurare che i dispositivi siano posti nello stato corretto al riavvio del sistema. Per i sistemi dotati di gruppi di continuità l'oggetto di stato dell'alimentazione è utile per determinare appunto il livello d'energia disponibile. Un driver esamina quest'oggetto prima di eseguire un'operazione critica per accertarsi del fatto che non si sia verificato un calo di tensione, in questo caso alza l'IRQL della sua unità d'elaborazione fino al livello del segnale d'interruzione da calo di tensione, esegue l'operazione e riporta l'IRQL al suo valore precedente. Questo modello di comportamento impedisce che il driver sia interrotto a causa di un calo di tensione durante l'esecuzione dell'operazione critica.

21.3.3 Esecutivo

L'esecutivo del sistema Windows 2000 fornisce una serie di servizi utilizzabili da tutti i sottosistemi d'ambiente. I servizi sono raggruppati come segue: gestione degli oggetti, della memoria virtuale, dei processi, dell'I/O, e servizi relativi alle chiamate di procedura locali e di controllo della sicurezza.

21.3.3.1 Gestore degli oggetti

Essendo un sistema orientato agli oggetti, il Windows 2000 usa oggetti in connessione con tutti i servizi forniti e tutte le entità gestite. Esempi di oggetti sono le directory, i riferimenti simbolici, i semafori, gli eventi, i processi, i thread, le porte e i file. Il compito del gestore degli oggetti è di sovrintendere all'uso di tutti gli oggetti: quando un thread vuole usare un oggetto, invoca il metodo open del gestore degli oggetti per ottenere una

maniglia per l'oggetto desiderato. Le **maniglie** (*handle*) costituiscono un'interfaccia uniforme per tutti i tipi di oggetti: al pari della maniglia di un file, la maniglia di un oggetto è un identificatore unico associato a un processo che conferisce la possibilità di accedere e manipolare una risorsa del sistema.

Poiché il gestore degli oggetti è il solo componente del sistema in grado di generare maniglie per gli oggetti, è anche il luogo naturale dove compiere i controlli di sicurezza. Ad esempio, quando un processo tenta di aprire un oggetto, il gestore controlla i diritti d'accesso del processo; è anche in grado di fissare alcune quote, come la quantità massima di memoria che un processo può riservare.

Il gestore degli oggetti è in grado di tenere traccia dei processi che stanno usando un oggetto: ogni intestazione di un oggetto contiene un contatore del numero di processi che possiedono maniglie dell'oggetto; se l'oggetto è temporaneo, si cancella il suo nome dallo spazio dei nomi quando il contatore raggiunge il valore zero. Poiché il sistema Windows 2000 stesso usa spesso puntatori e non maniglie per accedere agli oggetti, il gestore degli oggetti mantiene anche un contatore dei riferimenti del sistema agli oggetti; quando anche questo contatore vale zero, se l'oggetto è temporaneo, viene cancellato dalla memoria. Gli oggetti permanenti rappresentano entità fisiche, come le unità a disco, quindi non sono cancellati quando i contatori valgono zero.

Gli oggetti si manipolano per mezzo di un insieme standard di metodi: `create`, `open`, `close`, `delete`, `query name`, `parse` e `security`. Per gli ultimi tre è necessaria qualche spiegazione:

- `query name` s'invoca quando un thread possiede la maniglia di un oggetto e vuole conoscere il nome dell'oggetto stesso.
- `parse` è usato dal gestore degli oggetti per individuare un oggetto di cui conosce il nome.
- `security` s'invoca quando un processo cambia o apre la protezione di un oggetto.

L'esecutivo permette di associare un **nome** a ogni oggetto; lo spazio dei nomi è globale, cosicché un processo potrebbe creare un oggetto con un certo nome, e un secondo processo potrebbe poi chiedere una maniglia dell'oggetto, condividendolo quindi col primo processo. Un processo che apra un oggetto con nome può chiedere che la ricerca per nome ignori le differenze fra le lettere maiuscole e minuscole, o ne tenga invece conto.

Un nome può essere temporaneo o permanente; nel secondo caso denota un'entità la cui esistenza è indipendente dal fatto che qualche processo ne stia usufruendo, ad esempio un'unità a disco; nel primo caso, invece, il nome esiste solo fino a che almeno un processo possiede una maniglia dell'oggetto denotato.

Sebbene lo spazio dei nomi non sia direttamente visibile da tutti i punti di una rete, si usa il metodo `parse` del gestore degli oggetti per agevolare l'accesso a un oggetto con nome appartenente a un altro sistema: quando un processo tenta di aprire un oggetto che risiede in un calcolatore remoto, il gestore degli oggetti invoca il metodo `parse`, che, a sua volta, attiva un processo di reindirizzamento di rete per individuare l'oggetto.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

I nomi degli oggetti hanno la stessa struttura dei nomi di file nell'MS-DOS o nello UNIX. Una directory è rappresentata da un **oggetto directory** contenente i nomi di tutti gli oggetti in quella directory. Lo spazio dei nomi degli oggetti può essere esteso con l'aggiunta di **domìni di oggetti**, che sono insiemi di oggetti a sé stanti; alcuni esempi di questi domìni sono i dischi: è facile rendersi conto di come lo spazio dei nomi si espanda quando si aggiunge un dischetto al sistema: il dischetto ha un suo spazio di nomi che s'innesta nello spazio dei nomi esistente.

I file system del sistema UNIX hanno la possibilità di gestire **riferimenti simbolici** in modo che soprannomi multipli, detti anche alias, possano denotare lo stesso file; similmente, il sistema Windows 2000 usa **oggetti di riferimento simbolico**. Uno dei fini per i quali il sistema operativo Windows 2000 sfrutta i riferimenti simbolici è quello di associare i nomi delle unità a disco alle convenzionali lettere delle unità dell'MS-DOS; queste lettere non sono altro che riferimenti simbolici che si possono stabilire secondo le preferenze dell'utente.

Un processo ottiene una maniglia di un oggetto creandolo, aprendone uno esistente, ricevendo il duplicato di una maniglia di un altro processo, o ereditando una maniglia dal **processo genitore**. Tutto ciò è simile ai modi in cui un processo nel sistema operativo UNIX può ottenere un descrittore di file. Le maniglie sono tutte memorizzate nella **tabella degli oggetti** del processo: ogni elemento di questa tabella contiene i diritti d'accesso dell'oggetto e specifica se la maniglia debba essere ereditata dai **processi figli**. Quando un processo termina, il sistema operativo chiude automaticamente tutte le maniglie del processo.

Quando un utente è convalidato dalla procedura d'accesso al sistema, al processo dell'utente si associa un oggetto d'accesso; esso contiene informazioni quali l'identificatore di sicurezza e di gruppo, i privilegi, il gruppo primario, e la lista di controllo degli accessi predefinita: questi attributi determinano i servizi e gli oggetti di cui il processo può usufruire.

Nel sistema Windows 2000 ogni oggetto è protetto da una lista di controllo degli accessi, la quale contiene gli identificatori di sicurezza e l'insieme dei diritti d'accesso concessi a ciascun processo. Quando un processo tenta di accedere a un oggetto, il sistema, al fine di stabilire se l'accesso debba essere consentito, confronta l'identificatore di sicurezza del processo che si trova nell'oggetto d'accesso con la lista di controllo degli accessi relativa all'oggetto in questione. Questo raffronto è eseguito solo al momento dell'apertura di un oggetto, con la conseguenza che i servizi interni del sistema operativo che usano i puntatori e non le maniglie per accedere agli oggetti scavalcano i controlli.

In generale, chi crea un oggetto stabilisce anche la relativa lista di controllo degli accessi; se tale lista non è fornita al momento della creazione, il nuovo oggetto può ereditarne una dall'oggetto creante o può ottenerne una predefinita dall'oggetto d'accesso dell'utente.

Uno dei campi nell'oggetto d'accesso controlla la verifica dell'oggetto. Le operazioni sottoposte a verifica si registrano nel giornale delle verifiche del sistema insieme con un identificatore dell'utente. Il campo di verifica può sorvegliare il giornale delle verifiche al fine di scoprire tentativi d'intrusione nel sistema o d'accesso a oggetti protetti.

21.3.3.2 Gestore della memoria virtuale

La parte dell'esecutivo che si occupa della memoria virtuale (MV per brevità) è il gestore **della memoria virtuale**. Il gestore della MV è concepito assumendo che l'architettura sottostante sia in grado di associare indirizzi virtuali a indirizzi fisici, possieda un meccanismo di paginazione, realizzi coerentemente una cache trasparente nei sistemi con più unità d'elaborazione, e permetta l'associazione di più elementi della tabella delle pagine alla stessa pagina fisica. Il gestore della MV adotta uno schema di gestione basato su pagine della dimensione di 4 KB. Le pagine di dati assegnate a un processo ma non residenti nella memoria fisica sono memorizzate nel **file di paginazione** in un disco. Il gestore della MV usa indirizzi a 32 bit, cosicché ogni processo è dotato di uno spazio d'indirizzi virtuale di 4 GB; i 2 GB superiori sono identici per tutti i processi e sono usati dal sistema operativo nel modo protetto, mentre i 2 GB inferiori sono distinti e accessibili sia dai thread eseguiti nel modo d'utente sia da quelli eseguiti in modo protetto. Si noti che certe configurazioni del sistema Windows 2000 riservano solo 1 GB per gli usi propri del sistema operativo, permettendo a un processo di usare uno spazio d'indirizzi di 3 GB.

Per assegnare memoria il gestore della MV procede in due passi: innanzi tutto *riserva* una parte dello spazio d'indirizzi del processo; quindi *compie* l'assegnazione dello spazio nel file di paginazione. Il sistema operativo può limitare lo spazio del file di paginazione impiegato da un processo imponendo una quota massima d'assegnazione; un processo può rilasciare parti di memoria non più in uso al fine di riguadagnare per scopi utili una parte della sua quota. Poiché anche la memoria è rappresentata da oggetti, quando un primo processo (il genitore) ne crea un altro (il figlio), può mantenere il diritto d'accesso alla memoria virtuale del figlio: è in questo modo che i sottosistemi d'ambiente gestiscono la memoria dei loro processi client. Al fine di non penalizzare eccessivamente le prestazioni, il gestore della MV permette ai processi privilegiati di bloccare un certo numero di pagine virtuali nella memoria fisica, assicurando così il fatto che queste non saranno trasferite nel disco nel file di paginazione.

Sebbene due processi possano condividere memoria ottenendo maniglie di uno stesso oggetto di memoria, ciò potrebbe essere inefficiente, perché l'intero spazio di memoria di un oggetto dovrebbe essere assegnato a entrambi i processi prima che essi possano accedere all'oggetto. Il sistema operativo Windows 2000 fornisce come alternativa un cosiddetto **oggetto sezione** che rappresenta un blocco di memoria condivisa; una volta ottenuta la maniglia di un oggetto sezione, un processo può associare al suo spazio d'indirizzi solo la parte necessaria di memoria: questa parte è detta **finestra** (*view*). Tale meccanismo permette inoltre a un processo di accedere a un oggetto più grande della quota di spazio del file di paginazione assegnata al processo; il sistema stesso, infine, può usare una finestra per esaminare lo spazio d'indirizzi di un oggetto un pezzo per volta.

Un processo può controllare in molti modi l'uso di un oggetto sezione di memoria condivisa. La dimensione massima di una sezione può essere fissata. La memoria ausiliaria per la sezione può sfruttare il file di paginazione di sistema o un file ordinario (un file associato alla memoria). Una sezione può essere *basata*, nel senso che appare allo stes-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

so indirizzo virtuale a ogni processo che vi accede. Infine la protezione delle pagine della sezione si può impostare per la sola lettura, lettura e scrittura, sola esecuzione, pagina di protezione, e copiatura su scrittura. Le ultime due impostazioni di protezione richiedono alcune spiegazioni:

- Una pagina di protezione causa un'eccezione quando si tenta di accedervi; l'eccezione si può usare, fra le altre cose, per controllare se un programma difettoso esegue iterazioni che vanno oltre la dimensione di un vettore.
- Il meccanismo di copiatura su scrittura permette al gestore della MV di risparmiare memoria: quando due processi vogliono copie indipendenti dello stesso oggetto, il gestore pone solo una copia condivisa nella memoria fisica, e attribuisce a quella sezione la proprietà di copiatura su scrittura; se uno dei processi tenta di scrivere dati in una delle pagine caratterizzate da tale proprietà, il gestore fornisce al processo una copia privata della pagina che può poi essere modificata.

La traduzione degli indirizzi virtuali fa uso di molte strutture di dati. Ogni processo possiede una **directory delle pagine** contenente 1024 **elementi di directory delle pagine** (*page directory entry* — PDE) di 4 byte. Di solito questa directory è privata, ma, se l'ambiente lo richiede, può essere condivisa fra processi. Ogni elemento della directory delle pagine punta a una *tabella delle pagine* contenente 1024 **elementi di tabella delle pagine** (*page table entry* — PTE) di 4 byte. Ogni PTE, a sua volta, punta a una **pagina fisica** di 4 KB nella memoria fisica. La dimensione totale di tutte le tabelle delle pagine di un processo è di 4 MB, quindi il gestore della MV trasferisce queste tabelle in un disco quando è necessario; si veda la Figura 21.3 per uno schema di questa struttura.

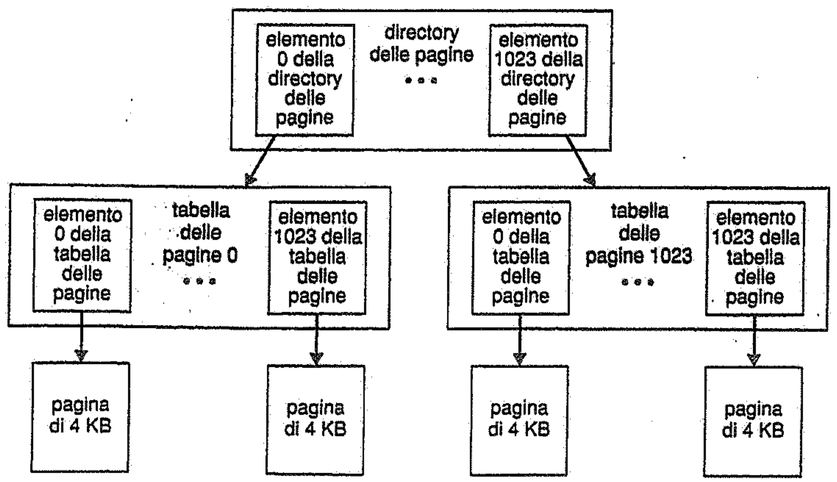


Figura 21.3 Struttura della memoria virtuale.

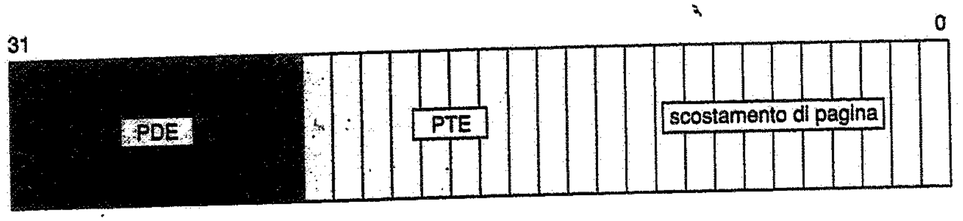


Figura 21.4 Composizione di un indirizzo di memoria virtuale.

Un intero di 10 bit può assumere tutti i valori compresi fra 0 e 1023, può quindi selezionare senza ambiguità un elemento della directory delle pagine o della tabella delle pagine. Questo fatto si utilizza durante la traduzione di un indirizzo virtuale in un indirizzo di memoria fisica di un byte. Un indirizzo di memoria virtuale di 32 bit è composto da tre interi, com'è mostrato nella Figura 21.4: i 10 bit più significativi indicano un elemento della directory delle pagine (PDE) che punta a una tabella delle pagine; l'unità di gestione della memoria (MMU) usa i 10 bit successivi per selezionare un elemento (PTE) di questa tabella delle pagine, che punta a una pagina fisica. I 12 bit meno significativi individuano uno specifico byte della pagina fisica. L'MMU crea un puntatore allo specifico byte nella memoria fisica concatenando 20 bit della PTE con i 12 bit meno significativi dell'indirizzo virtuale. Si vede quindi che 12 bit di una PTE non servono per la traduzione di un indirizzo virtuale; essi invece descrivono la pagina. Un PTE della CPU Pentium riserva 3 bit per gli usi del sistema operativo. I bit restanti specificano se la pagina è stata modificata, ha subìto accessi, si può copiare nella cache, può essere solo letta, va scritta in modo diretto, è in modo protetto, o è valida; sicché descrivono lo stato della pagina nella memoria. Per maggiori informazioni sulle strategie di paginazione si veda il Paragrafo 9.4.

La pagina può essere in uno fra i seguenti sei strati: valida, azzerata, libera, in attesa, modificata, difettosa. Una pagina *valida* è correntemente usata da qualche processo, mentre una pagina *libera* non è puntata da alcun PTE. Una pagina *azzerata* è una pagina libera i cui contenuti sono stati appunto azzerati e che è pronta per l'uso. Una pagina *in attesa* è stata rimossa dall'insieme di lavoro di un processo. Una pagina *modificata* è stata sovrascritta con dati, ma non è stata ancora copiata nel disco. Le pagine in attesa e le pagine modificate sono considerate pagine di transizione. Infine, una pagina *difettosa* è inutilizzabile a causa di un errore fisico.

L'effettiva struttura dei PTE dei file di pagine è rappresentata nella Figura 21.5: 5 bit sono relativi alla protezione della pagina, 20 bit allo scostamento nel file di pagine, 4 bit alla scelta del file di paginazione, e 3 bit allo stato della pagina. Questo PTE di file di pagine apparirebbe alla macchina fisica come una pagina non valida. Poiché il codice eseguibile e i file associati alla memoria possiedono già copie nei dischi, non necessitano di spazio in un file di paginazione. Nel caso in cui una di queste pagine non si trovi nella

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

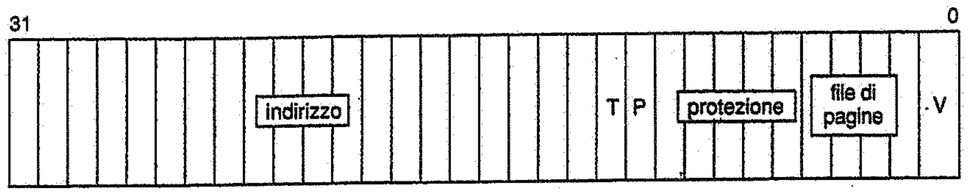


Figura 21.5 PTE dei file di pagine.

memoria fisica, la struttura del PTE è la seguente: il bit più significativo si usa per specificare la protezione della pagina, i 28 bit seguenti s'interpretano come un indice relativo a una struttura di dati del sistema che associa alla pagina un file e uno scostamento all'interno di quel file, i 3 bit meno significativi denotano lo stato della pagina.

Se ogni processo possiede le proprie tabelle delle pagine, è difficile condividere una pagina perché ogni processo avrà il proprio PTE relativo alla pagina fisica: quando una pagina condivisa deve essere trasferita nella memoria fisica, l'indirizzo fisico deve essere memorizzato nel PTE di ciascun processo che condivide quella pagina, e anche i bit di stato e di protezione in tali PTE dovranno essere impostati e aggiornati in modo coerente. Per evitare questi problemi il sistema Windows 2000 fa uso di riferimenti indiretti a un livello: per ogni pagina condivisa il processo possiede un PTE che punta a un **elemento prototipo della tabella delle pagine** anziché a una pagina fisica; questo PTE prototipo contiene l'indirizzo della pagina fisica e i bit di stato e protezione. Così il primo accesso di un processo a una pagina condivisa produce un'eccezione di pagina mancante; gli accessi successivi si eseguono normalmente. Se la pagina è contrassegnata per la sola lettura, il gestore della MV fa una copia su scrittura e il processo non ha più una pagina condivisa. Le pagine condivise non appaiono mai nel file di pagine, ma si trovano nel file system.

Il gestore della MV tiene traccia di tutte le pagine di memoria fisica tramite una **base di dati delle pagine fisiche**; questa contiene un elemento per ogni pagina fisica, il quale punta al PTE che punta alla relativa pagina fisica, in modo che il gestore possa tenere aggiornato lo stato della pagina. Le pagine fisiche sono concatenate in modo da costituire, ad esempio, la lista delle pagine libere e quella delle pagine azzerate.

Quando si verifica un'eccezione di pagina mancante, il gestore della MV trasferisce nella memoria la pagina mancante collocandola nella prima pagina fisica della lista delle pagine libere. Ma non si limita a far ciò, infatti la ricerca ha mostrato che i riferimenti alla memoria di un thread tendono ad avere la proprietà di **località**: quando una pagina è in uso, è probabile che ci saranno nel prossimo futuro riferimenti alle pagine adiacenti (si pensi alle iterazioni su un vettore, o al prelievo di istruzioni sequenziali che costituiscono il codice eseguibile di un thread). Valendosi di questa proprietà, quando trasferisce una pagina nella memoria, il gestore della MV trasferisce anche un certo numero di pagine a essa adiacenti; ciò tende a ridurre il numero totale di assenze di pagine. Per maggiori informazioni sulla località si veda il Paragrafo 10.6.1.

Se non ci sono pagine fisiche disponibili nella lista delle pagine libere, si adotta una strategia di sostituzione FIFO per ogni processo al fine di sottrarre pagine ai processi che ne stiano usando un numero maggiore della dimensione minima del loro insieme di lavoro. Il sistema operativo tiene sotto controllo le assenze di pagine di ogni processo che stia usando un numero di pagine pari alla dimensione minima del suo insieme di lavoro e ne regola la dimensione di conseguenza. In particolare, assegna a ogni processo appena avviato un insieme di lavoro di 30 pagine e controlla periodicamente l'adeguatezza di questa dimensione sottraendo una pagina al processo: se il processo continua l'esecuzione senza generare un'eccezione di pagina mancante per la pagina sottratta, la dimensione dell'insieme di lavoro del processo viene ridotta di 1 e si aggiunge la pagina alla lista delle pagine libere.

21.3.3.3 Gestore dei processi

Il gestore dei processi del sistema Windows 2000 fornisce i servizi necessari alla creazione, l'eliminazione e l'uso dei thread e dei processi. Esso non possiede alcuna informazione sulle relazioni parentali o sulle gerarchie fra i processi; questi dettagli sono lasciati al particolare sottosistema d'ambiente relativo al processo.

Quel che segue è un esempio di come si crea un processo nell'ambiente Win32: quando un'applicazione invoca `CreateProcess`, il sottosistema Win32 riceve un messaggio al quale reagisce chiamando il gestore dei processi al fine di creare un processo; il gestore dei processi, a sua volta, attiva il gestore degli oggetti che crea un oggetto processo e restituisce al Win32 la maniglia dell'oggetto. Il sottosistema Win32 attiva di nuovo il gestore dei processi per creare un thread relativo al processo, e restituisce infine le maniglie al nuovo processo e al nuovo thread.

21.3.3.4 Chiamata di procedura locale

Il sistema operativo impiega le chiamate di procedura locale (*local procedure call* — LPC) per trasferire richieste e risultati fra processi client e server all'interno di uno stesso calcolatore; servono in particolare a richiedere servizi ai vari sottosistemi. Per molti aspetti si tratta di un meccanismo simile alla chiamata di procedura remota (RPC) usata da molti sistemi operativi per l'elaborazione distribuita in una rete; la LPC, tuttavia, è uno strumento ottimizzato in funzione dell'uso all'interno di un unico sistema Windows 2000.

La LPC è un meccanismo per lo scambio dei messaggi. Il processo server pubblica un oggetto visibile globalmente che rappresenta una porta di connessione: quando un client necessita di un servizio da un sottosistema, apre l'oggetto porta di connessione del sottosistema e trasmette una richiesta di connessione alla porta in questione; il server crea un canale e restituisce una maniglia al client. Il canale consiste in una coppia di porte di comunicazione private: una per i messaggi dal client al server, l'altra per quelli dal server al client. I canali di comunicazione forniscono un meccanismo di richiamata che permette al client e al server di accettare richieste anche quando attendono una risposta.

http://www.addtopdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Al momento della creazione di un canale LPC è necessario specificare una fra tre possibili tecniche di scambio dei messaggi:

1. La prima tecnica è adatta a brevi messaggi (fino a 256 byte): in questo caso, si usa la coda dei messaggi della porta come mezzo di memorizzazione intermedio, e si copiano i messaggi da un processo all'altro.
2. La seconda tecnica è adatta a messaggi più lunghi: in questo caso, si crea un oggetto sezione di memoria condivisa per il canale; i messaggi trasmessi attraverso la coda dei messaggi della porta contengono un puntatore alla sezione, oltre a informazioni sulla dimensione dell'oggetto sezione. In questo modo si evita la necessità di copiare lunghi messaggi: il mittente inserisce dati nella sezione condivisa e il ricevente può vederli immediatamente.
3. La terza tecnica di trasferimento dei messaggi, detta **LPC rapida**, è usata da parti del sottosistema Win32 relative alla grafica. Quando un client richiede una connessione di tipo LPC rapida, il server prepara tre oggetti: un thread server specifico per gestire le richieste, un oggetto sezione di 64 KB, e un oggetto detto coppia di eventi. Un *oggetto coppia di eventi* è uno strumento di sincronizzazione usato dal sottosistema Win32 per notificare al server il fatto che il client ha copiato un messaggio, o viceversa; i messaggi LPC sono trasferiti per mezzo della sezione e la sincronizzazione è eseguita dall'oggetto coppia di eventi. La LPC ha diversi vantaggi. L'uso di un oggetto sezione elimina la copiatura dei messaggi, poiché rappresenta una regione di memoria condivisa, e in virtù della sincronizzazione realizzata dall'oggetto coppia di eventi si elimina anche il ritardo aggiuntivo dovuto all'uso dell'oggetto porta per passare messaggi contenenti puntatori e lunghezze. Lo specifico thread server elimina il ritardo dovuto alla necessità di determinare quale thread client stia chiamando il server, poiché c'è un thread server per ogni thread client; inoltre, il nucleo dà priorità a questi thread server specifici durante lo scheduling al fine di migliorare ulteriormente le prestazioni. Il rovescio della medaglia è che l'LPC rapida necessita di più risorse degli altri due metodi, tanto che il sottosistema Win32 la usa solo per la gestione delle finestre e dei dispositivi grafici.

21.3.3.5 Gestore dell'I/O

Il gestore dell'I/O è responsabile della gestione dei file system, della cache, dei driver dei dispositivi e dei driver di rete: tiene traccia dei file system installabili che sono caricati e gestisce le aree di memoria per le richieste di I/O; collabora col gestore della MV per eseguire l'associazione alla memoria dei file di I/O, e controlla il gestore della cache che, a sua volta gestisce i servizi di caching per l'intero sistema di I/O. Il gestore dell'I/O mette a disposizione sia operazioni sincrone sia asincrone, fornisce servizi d'attesa (*timeout*) per i driver e possiede strumenti capaci di mettere in comunicazione due driver.

Il gestore dell'I/O converte le richieste ricevute in una forma standard detta **pacchetto di richiesta di I/O** (*I/O request packet* — IRP) che inoltra al driver appropriato affinché esso possa soddisfare la richiesta; quando l'operazione è terminata, il gestore dell'I/O riceve l'IRP dall'ultimo driver che abbia eseguito un'operazione, e dichiara evasa la richiesta.

In molti sistemi operativi il caching è eseguito dal file system; il sistema Windows 2000 fornisce invece uno strumento di caching centralizzato. Il gestore della cache serve tutti i componenti del sistema sotto il controllo del gestore dell'I/O, in stretta collaborazione con il gestore della MV. La dimensione della cache cambia dinamicamente, secondo la quantità di memoria libera nel sistema. Si ricordi che i 2 GB superiori dello spazio d'indirizzi di un processo costituiscono l'area di sistema, identica per ogni processo: il gestore della MV assegna fino a metà di questo spazio alla cache di sistema. Il gestore della cache associa i file a questo spazio d'indirizzi e impiega i servizi del gestore della MV per trattare l'I/O da file.

La cache è divisa in blocchi di 256 KB: ogni blocco può contenere una finestra su un file (cioè, una regione associata alla memoria). Un blocco della cache è descritto da un **descrittore d'indirizzo virtuale** (*virtual-address control block* — VACB) che contiene l'indirizzo virtuale e lo scostamento relativo al file della finestra, e il numero di processi che stanno usando la finestra; i VACB risiedono in un singolo vettore mantenuto dal gestore della cache.

Il gestore della cache mantiene un vettore di indici di VACB distinto per ogni file aperto: questo vettore ha un elemento per ogni blocco di 256 KB del file, cosicché, ad esempio, un file di 2 MB sarebbe descritto da un vettore di indici di VACB di 8 elementi; ogni elemento di questo vettore punta al VACB di quella finestra, se la regione interessata si trova nella cache, ed è nullo altrimenti.

Quando riceve una richiesta di lettura da parte di un utente, il gestore dell'I/O trasmette un IRP al gestore della cache (sempre che la richiesta non chieda esplicitamente che l'operazione di lettura non usi la cache). Il gestore della cache calcola l'elemento del vettore di indici di VACB del file corrispondente allo scostamento specificato dalla richiesta; l'elemento in questione o è nullo o punta a una finestra nella cache. Nel primo caso, il gestore della cache assegna un blocco di cache insieme col corrispondente elemento del vettore VACB, e associa la finestra al blocco della cache; quindi tenta di copiare i dati dal file associato all'area di memoria per l'I/O (*buffer*) del chiamante. Se ciò riesce l'operazione è completata, altrimenti il fallimento è dovuto a un'assenza di pagina; in questo caso il gestore della MV invia una richiesta di lettura non sottoposta a caching al gestore dell'I/O. Quest'ultimo chiede al driver del dispositivo appropriato di leggere i dati, e li restituisce al gestore della MV che li carica nella cache. Una volta che si trovano nella cache, i dati si possono copiare nell'area di memoria per l'I/O del chiamante, completando così la richiesta di I/O. La Figura 21.6 mostra uno schema di queste operazioni. Quando è possibile — per le operazioni sincrone, con caching, non bloccanti — l'I/O è gestito dal cosiddetto **meccanismo di I/O rapido**. Questo meccanismo copia semplicemente i dati nelle pagine della cache o dalle pagine della cache direttamente e impiega il gestore della cache per compiere ogni I/O necessario.

Un'operazione di lettura al livello del nucleo è simile, ma si può accedere ai dati direttamente dalla cache senza che essi siano copiati in un'area per l'I/O nello spazio d'indirizzi di memoria di un utente. Al fine di leggere metadati relativi al file system, cioè strutture di dati che descrivono il file system, il nucleo usa l'interfaccia di associazione del gestore della cache; per modificarli, il file system usa l'interfaccia di vincolo del gestore della cache: **vincolare** una pagina significa associarla permanentemente a una pagi-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

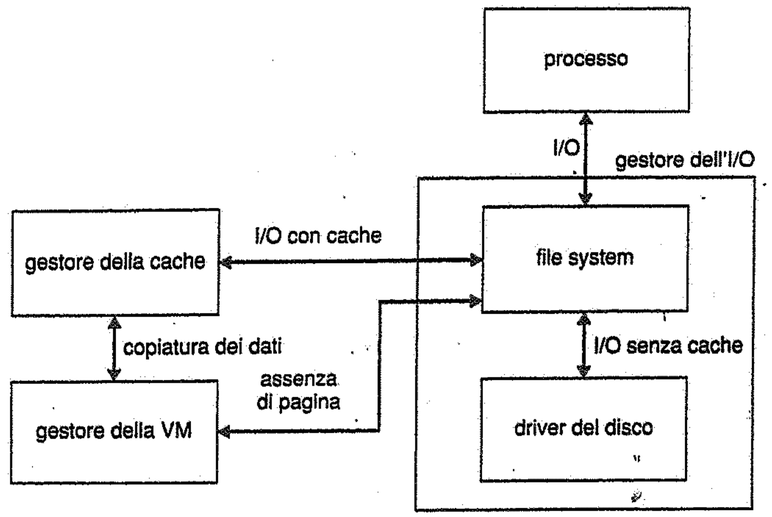


Figura 21.6 I/O con file.

na fisica, in modo che il gestore della MV non possa spostare o trasferire nei dischi la pagina in questione. Una volta che i metadati sono stati aggiornati il file system chiede al gestore della cache di liberare la pagina: poiché la pagina è stata modificata, è contrassegnata come tale, cosicché il gestore della MV la trasferirà nel disco. Si noti che i metadati sono effettivamente memorizzati in file ordinari.

Al fine di migliorare le prestazioni il gestore della cache mantiene un piccolo archivio delle richieste di lettura e tenta di predire le richieste future; se il gestore della cache rileva una forma di regolarità nelle tre richieste precedenti, ad esempio l'accesso sequenziale in avanti o indietro, può trasferire in anticipo dati nella cache prima che l'applicazione sottoponga la prossima richiesta: l'applicazione potrebbe allora trovare nella cache i dati di cui necessita, e non dovrebbe quindi attendere l'esecuzione dell'I/O da disco. Alle funzioni dell'API Win32 `OpenFile` e `CreateFile` è possibile passare l'indicatore (*flag*) `FILE_FLAG_SEQUENTIAL_SCAN`; si tratta di un modo di suggerire al gestore della cache di trasferire anticipatamente nella cache i 192 KB di dati che seguono la richiesta del thread. Poiché normalmente il sistema Windows 2000 esegue le operazioni di I/O a blocchi di 64 KB o 16 pagine, questa lettura anticipata trasferisce una quantità di dati pari a tre volte la dimensione ordinaria.

Il gestore della cache ha anche il compito di comunicare al gestore della MV la necessità di trasferire nei dischi il contenuto della cache; ordinariamente il gestore della cache accumula scritture per 4 o 5 secondi, dopo di che risveglia il thread di scrittura della cache. Quando è invece necessaria la scrittura diretta, un processo può impostare uno specifico indicatore all'apertura del file o chiamare esplicitamente una funzione per il trasferimento nei dischi dei dati della cache.

Un processo che scriva dati molto rapidamente potrebbe anche riempire tutte le pagine libere della cache prima che il thread di scrittura possa trasferirle nei dischi. Lo scrittore della cache evita che ciò accada nel modo seguente: quando la quantità di spazio libero nella cache diviene limitato, il gestore della cache sospende temporaneamente i processi che tentano di scrivere dati e risveglia il thread di scrittura al fine di trasferire pagine nel disco. Se il processo che esegue molte scritture si occupa del reindirizzamento di rete in relazione a un file system di rete, bloccarlo troppo a lungo potrebbe implicare la ritrasmissione di dati lungo la rete, il che costituirebbe uno spreco dell'ampiezza di banda della rete; proprio per evitare un tale spreco per quel che riguarda il reindirizzamento di rete, si può chiedere al gestore della cache di evitare l'accumulo di una grande quantità di scritture nella cache.

Poiché un file system di rete ha bisogno di trasferire dati tra il disco e l'interfaccia di rete, il gestore della cache fornisce anche un'interfaccia DMA per il trasferimento diretto dei dati; ciò permette di evitare la duplicazione dei dati che sarebbe richiesta dall'uso di una memorizzazione di transito intermedia.

21.3.3.6 Gestore della sicurezza

La natura orientata agli oggetti del sistema operativo Windows 2000 permette l'adozione di un meccanismo uniforme per l'autenticazione degli accessi, e per la verifica di ogni entità del sistema. Ogniqualvolta un processo apre un oggetto, il monitor della sicurezza controlla l'oggetto di sicurezza del processo e la sua lista di controllo degli accessi al fine di determinare se il processo ha i diritti necessari.

21.3.3.7 Gestore del sistema *plug and play*

Il sistema operativo usa il gestore *plug and play* per riconoscere eventuali cambiamenti nella configurazione fisica del calcolatore e adattarvi opportunamente il sistema. Affinché il *plug and play* (PnP) possa funzionare, sia il dispositivo sia il driver devono essere compatibili con lo standard PnP. Il gestore PnP riconosce automaticamente i dispositivi installati e mentre il sistema è funzionante rileva eventuali cambiamenti nei dispositivi. Il gestore tiene anche traccia delle risorse impiegate da ciascun dispositivo, insieme con le risorse che potenzialmente potrebbero essere utilizzate e si occupa del caricamento dei driver necessari. Questa gestione delle risorse fisiche (che sono principalmente le interruzioni e gli intervalli di memoria per l'I/O) ha l'obiettivo d'identificare una configurazione che permetta a tutti i dispositivi di funzionare correttamente. Ad esempio, se il dispositivo *B* può usare solo l'interruzione 5 mentre il dispositivo *A* potrebbe usare sia la 5 sia la 7, allora il gestore PnP potrebbe assegnare la 5 a *B* e la 7 ad *A*. Nelle versioni precedenti, l'utente poteva trovarsi nella situazione di dover rimuovere il dispositivo *A* e riconfigurarlo in modo da fargli usare l'interruzione 7, prima di installare il dispositivo *B*. In questo caso, l'utente avrebbe dovuto studiare le risorse di sistema prima di installare nuovi dispositivi e scoprire o ricordare quali dispositivi impiegano determinate risorse fisiche. La proliferazione di dispositivi USB e PCcard rende ancor più utile un sistema per la configurazione dinamica delle risorse.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il gestore PnP opera la riconfigurazione dinamica come segue. Innanzi tutto, ottiene una lista di dispositivi da ciascun driver di bus (ad esempio, PCI, USB). Successivamente carica il driver installato (o, se è necessario, ne installa uno) e invia un comando `add-device` al driver appropriato per ciascun dispositivo. Il gestore PnP trova le assegnazioni ottimali delle risorse e quindi invia un comando `start-device` a ciascun driver insieme all'assegnamento delle risorse per quel dispositivo. Se un dispositivo deve essere riconfigurato il gestore PnP invia un comando `query-stop` che chiede al driver se il dispositivo può essere temporaneamente disabilitato. Se il driver può disabilitare il dispositivo, allora tutte le operazioni sospese vengono completate e non si può avviare nessuna nuova operazione. Successivamente, il gestore invia un comando `stop`, dopodiché può riconfigurare il dispositivo con un altro comando `start-device`. Il gestore PnP prevede altri comandi, come `query-remove`, che si usa quando l'utente sta per rimuovere un dispositivo PCcard e che opera in modo simile a `query-stop`. Il comando `surprise-remove` si usa quando c'è un malfunzionamento di dispositivo, oppure, più comunemente, quando un utente estrae un dispositivo PCcard senza usare prima il programma di utilità delle PCcard per disattivare il dispositivo stesso. Il comando richiede che il driver smetta di usare il dispositivo e rilasci tutte le risorse che sono state assegnate al dispositivo stesso.

21.4 Sottosistemi d'ambiente

I sottosistemi d'ambiente sono processi eseguiti nel modo d'utente, basati sui servizi dell'esecutivo del sistema Windows 2000, che permettono l'esecuzione di programmi sviluppati per altri ambienti (ad esempio Windows a 16 bit, MS-DOS, POSIX e le applicazioni orientate ai caratteri per OS/2 a 16 bit). Ogni sottosistema d'ambiente fornisce un'API o ambiente per le applicazioni.

Il sistema operativo Windows 2000 usa il sottosistema Win32 per avviare ogni processo e più in generale come ambiente operativo principale. Quando si esegue un'applicazione il sottosistema Win32 invoca il gestore della MV per caricare il codice dell'applicazione. Il gestore della memoria restituisce al Win32 informazioni di stato relative al tipo di codice eseguibile di cui si tratta; se esso non è un eseguibile per l'ambiente Win32, quest'ultimo controlla se il sottosistema d'ambiente appropriato è in esecuzione: se così non è, si avvia tale sottosistema come processo nel modo d'utente. Quindi, l'ambiente Win32 crea un processo per eseguire l'applicazione e trasferisce il controllo al sottosistema d'ambiente.

Il sottosistema d'ambiente usa LPC per ottenere i servizi del nucleo necessari al processo; questo fatto contribuisce alla robustezza del sistema operativo, perché i parametri passati a una chiamata del sistema possono essere sottoposti a controlli di correttezza prima che sia chiamata la procedura del nucleo. Il sistema operativo impedisce alle applicazioni di mescolare procedure API appartenenti ad ambienti differenti: un'applicazione Win32, ad esempio, non può chiamare una procedura POSIX.

Poiché ogni sottosistema è realizzato da un distinto processo che si esegue nel modo d'utente, il crollo di uno di essi non ha alcun effetto sugli altri. L'eccezione a questa regola è l'ambiente Win32, il quale gestisce la tastiera, il mouse e il video: se esso fallisce, il sistema è effettivamente compromesso.

L'ambiente Win32 classifica le applicazioni come grafiche o orientate ai caratteri, se un'*applicazione orientata ai caratteri* presume che l'emissione interattiva di dati sia diretta a uno schermo ASCII 80 per 24. L'ambiente Win32 trasforma i dati emessi da un'applicazione orientata ai caratteri in una rappresentazione grafica all'interno di una finestra; si tratta di una trasformazione semplice: ogniqualvolta si chiama una procedura di emissione di dati, il sottosistema d'ambiente invoca una procedura del Win32 per mostrare il testo. Poiché l'ambiente Win32 esegue quest'operazione per tutte le finestre orientate ai caratteri, è in grado di trasferire testo fra le finestre sullo schermo tramite un processo di 'taglia e incolla'. La trasformazione in forma grafica funziona sia per le applicazioni MS-DOS sia per le applicazioni POSIX a riga di comando.

21.4.1 Ambiente MS-DOS

L'ambiente MS-DOS non ha una complessità paragonabile a quella degli altri sottosistemi d'ambiente del sistema operativo Windows 2000. Esso è realizzato da un'applicazione Win32 detta **macchina DOS virtuale** (*virtual DOS machine* — VDM). La VDM è semplicemente un processo utente ed è quindi soggetta ai criteri di scheduling e paginazione come ogni altro thread del sistema Windows 2000. La VDM ha un'**unità d'esecuzione delle istruzioni** che esegue o simula le istruzioni dell'Intel 486; essa fornisce anche procedure che simulano il BIOS dell'MS-DOS e i servizi di gestione delle eccezioni noti come 'int 21', oltre ad avere driver virtuali dei dispositivi per lo schermo, la tastiera e le porte di comunicazione. La VDM è basata sul codice sorgente dell'MS-DOS 5.0 e fornisce alle applicazioni almeno 620 KB di memoria.

L'interprete dei comandi del sistema Windows 2000 è un programma che crea una finestra simile a un ambiente MS-DOS; essa è in grado di eseguire codice a 16 o a 32 bit: quando si esegue un'applicazione MS-DOS, l'interprete dei comandi avvia un processo VDM per eseguire il programma.

Se il sistema operativo Windows 2000 è installato in una macchina basata su una CPU *x*86, le applicazioni grafiche dell'MS-DOS sono eseguite a pieno schermo, mentre le applicazioni orientate ai caratteri sono eseguite a pieno schermo o in una finestra. Se il calcolatore è basato su una CPU diversa, tutte le applicazioni MS-DOS sono eseguite all'interno di una finestra. Le applicazioni per l'MS-DOS che accedono direttamente ai dischi non possono essere eseguite dal sistema operativo Windows 2000, perché esso nega l'accesso diretto ai dischi per proteggere il file system; più in generale, non potranno essere eseguite le applicazioni per l'MS-DOS che accedono direttamente ai dispositivi.

Poiché l'MS-DOS non è un ambiente multitasking, esistono applicazioni che intasano la CPU, ad esempio usando cicli a vuoto per causare ritardi o pause nell'esecuzione. Il meccanismo a priorità del dispatcher del sistema Windows 2000 rileva questi ritardi e riduce automaticamente l'impiego di tempo di CPU, causando così un funzionamento scorretto dell'applicazione interessata.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

21.4.2 Ambiente Windows a 16 bit

L'ambiente d'esecuzione Win16 è fornito da una VDM che incorpora un componente aggiuntivo detto *Windows on Windows* che mette a disposizione le procedure del nucleo dell'ambiente Windows 3.1, e segmenti di codice di riferimento per la gestione delle finestre e le funzioni GDI. I segmenti di codice di riferimento chiamano le appropriate procedure dell'ambiente Win32, convertendo gli indirizzi da 16 a 32 bit. Le applicazioni che impiegano la struttura interna del gestore delle finestre a 16 bit o del GDI potrebbero non funzionare, perché *Windows on Windows* non realizza effettivamente l'API a 16 bit.

Il programma *Windows on Windows* può essere eseguito in modo concorrente con altri processi del sistema Windows 2000, ma per molti aspetti ricorda l'ambiente Windows 3.1: si può eseguire una sola applicazione Win16 alla volta, tutte le applicazioni sono a singolo thread e risiedono nello stesso spazio d'indirizzi, e tutte condividono la stessa coda d'ingresso. Queste peculiarità comportano che un'applicazione che smette di ricevere dati blocca tutte le altre applicazioni Win16, proprio come nel Windows 3.*x*, e che un'applicazione Win16 può danneggiare altre applicazioni Win16 modificando i contenuti del loro spazio d'indirizzi. Usando il comando `start /separate win16application` si possono far coesistere ambienti Win16 multipli.

21.4.3 Ambiente Win32

Il sottosistema principale del sistema operativo Windows 2000 è il Win32: esso esegue le applicazioni Win32 e gestisce tutto l'I/O relativo alla tastiera, al mouse e allo schermo; trattandosi dell'ambiente di controllo, è concepito per essere estremamente robusto. Le caratteristiche che contribuiscono alla sua robustezza sono molte. Diversamente dall'ambiente Win16, ogni processo Win32 ha la propria coda d'ingresso; il gestore delle finestre manda tutti i dati in ingresso del sistema alla coda d'ingresso del processo appropriato, cosicché un processo che fallisca non bloccherà i dati in ingresso agli altri processi. Il nucleo del sistema operativo fornisce anche il multitasking con diritto di prelazione, che permette agli utenti di terminare le applicazioni malfunzionanti o non più necessarie. L'ambiente Win32 convalida inoltre tutti gli oggetti prima di usarli, cosa che permette di prevenire crolli che potrebbero altrimenti verificarsi se un'applicazione tentasse di usare una maniglia sbagliata o non valida. Prima di usare un oggetto, il sottosistema Win32 verifica a quale tipo d'oggetto fa riferimento la maniglia; il conteggio dei riferimenti mantenuto dal gestore degli oggetti impedisce che qualche oggetto sia eliminato mentre è ancora in uso, e ne impedisce l'uso una volta che sia stato effettivamente eliminato.

21.4.4 Sottosistema POSIX

Il sottosistema POSIX è stato concepito per eseguire le applicazioni che seguono lo standard POSIX.1, il quale è basato sul modello del sistema UNIX. Le applicazioni POSIX possono essere avviate dal sottosistema Win32 o da un'altra applicazione POSIX, e usano il server `PSXSS.EXE` del sottosistema POSIX, la libreria di collegamento dinamico `PSXDLL.DLL`, e il gestore della sessione di console POSIX `POSIX.EXE`.

Sebbene lo standard POSIX non stabilisca nulla riguardo alla stampa, le applicazioni POSIX possono usare le stampanti in modo trasparente grazie al meccanismo di reindirizzamento del sistema operativo Windows 2000. Esse hanno inoltre accesso a ogni file system del sistema Windows 2000; l'ambiente POSIX impone controlli d'accesso di tipo UNIX sugli alberi delle directory. Molti servizi offerti dal sottosistema Win32 non sono messi a disposizione dal sottosistema POSIX: fra questi, i file associati allo spazio d'indirizzi di memoria, i servizi di rete, la grafica e lo scambio dinamico dei dati.

21.4.5 Sottosistema OS/2

Sebbene il sistema operativo Windows 2000 fosse originariamente concepito per fornire un robusto ambiente operativo OS/2, il successo dell'ambiente Microsoft Windows ha portato, durante gli stadi iniziali dello sviluppo del sistema Windows 2000, a scegliere Windows come ambiente di base. Perciò, il sistema operativo Windows 2000 fornisce solo pochi strumenti per l'ambiente OS/2: le applicazioni in modo reale dell'OS/2 si possono eseguire su tutte le piattaforme impiegando l'ambiente MS-DOS; le applicazioni dotate sia di codice MS-DOS sia di codice OS/2, si eseguono nell'ambiente OS/2 sempre che esso non sia disattivato.

21.4.6 Sottosistemi d'accesso e di sicurezza

Per poter accedere a un oggetto, un utente deve essere convalidato dal sottosistema d'accesso, e affinché ciò avvenga, è necessario che sia registrato come utente e che fornisca la relativa parola d'ordine.

Il sottosistema di sicurezza genera oggetti d'accesso che rappresentano gli utenti nel sistema; esso adopera un **pacchetto di autenticazione** per eseguire l'autenticazione attraverso le informazioni ottenute dal sottosistema d'accesso o dal server di rete: tipicamente, il pacchetto di convalida si limita a esaminare le informazioni relative agli utenti accreditati, contenute in un archivio locale, per controllare la correttezza della parola d'ordine. Quindi il sottosistema di sicurezza genera l'oggetto d'accesso relativo all'identificatore dell'utente contenente i privilegi appropriati, le quote massime e gli identificatori di gruppo. Ogniqualvolta l'utente tenta di accedere a un oggetto del sistema chiedendone una maniglia, l'oggetto d'accesso è passato al gestore della sicurezza, che controlla i privilegi e le quote. Il pacchetto di autenticazione predefinito per i domìni Windows 2000 è il Kerberos.

21.5 File system

Il file system tradizionale dell'MS-DOS è la FAT (*file allocation table*); il file system FAT a 16 bit ha molti svantaggi, ad esempio la frammentazione interna, la dimensione massima di 2 GB e l'assenza di protezione degli accessi ai file. Il file system FAT a 32 bit ha risolto i problemi della frammentazione interna e della dimensione, ma le sue prestazioni e le sue caratteristiche sono ancora sotto il livello dei moderni file system. Il file system

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

NTFS del sistema operativo Windows 2000 è molto migliore: è stato concepito per comprendere molte funzioni, fra le quali vi sono il recupero dei dati, la sicurezza, la tolleranza ai guasti, i file e i file system di grandi dimensioni, i flussi multipli di dati, i nomi UNICODE e la compressione dei file. Per ragioni di compatibilità il sistema Windows 2000 mette anche a disposizione i file system FAT e HPFS dell'OS/2.

21.5.1 Struttura interna

L'entità fondamentale dell'NTFS è il volume: si crea con l'utilità di amministrazione dei dischi del sistema operativo ed è basato su una partizione logica del disco. Un volume può occupare parte di un disco, un intero disco, o anche più dischi.

L'NTFS non tratta direttamente i singoli settori dei dischi, ma usa unità di assegnazione (*cluster*) costituite da un numero di settori pari a una potenza di 2. La dimensione di un'unità di assegnazione si stabilisce nella fase di formattazione di un file system: la dimensione predefinita è pari alla dimensione di un settore per volumi fino a 512 MB, è pari a 1 KB per volumi fino a 1 GB, a 2 KB per volumi fino a 2 GB, e a 4 KB per volumi più grandi di 2 GB. Si tratta di dimensioni molto minori di quelle delle unità di assegnazione di un file system FAT a 16 bit, quindi si riduce la frammentazione interna: si consideri ad esempio un disco di 1,6 GB con 16.000 file; se si usa un file system FAT a 16 bit, si potrebbero perdere 400 MB in frammentazione interna, perché la dimensione di un'unità di assegnazione è di 32 KB; nell'NTFS, invece, si perderebbero nelle stesse condizioni solo 17 MB.

Come indirizzi relativi al disco l'NTFS usa **numeri di unità di assegnazione logiche** (*logical cluster number* — LCN), che assegna numerando le unità di assegnazione dall'inizio alla fine del disco. Il sistema può quindi calcolare uno scostamento (espresso in byte) relativo al disco fisico moltiplicando l'LCN per la dimensione dell'unità di assegnazione. Un file dell'NTFS non è una semplice sequenza di byte, come nello UNIX o nell'MS-DOS; è invece un oggetto strutturato costituito di **attributi**. Ciascun attributo di un file è una sequenza indipendente di byte che può essere creata, eliminata, letta e scritta. Alcuni attributi sono comuni a tutti i file, ad esempio il nome (o i nomi, se il file ha degli alias), l'ora di creazione, e il descrittore di sicurezza che stabilisce il controllo degli accessi; altri attributi sono specifici per certi tipi di file; i file del sistema Macintosh, ad esempio, hanno due attributi di dati: i cosiddetti *resource fork* e *data fork*. Una directory possiede alcuni attributi che realizzano un indice per i nomi dei file contenuti nella directory. In generale gli attributi si possono aggiungere come è necessario e vi si può accedere impiegando la nomenclatura `nome_del_file:attributo`. La maggior parte degli ordinari file di dati ha un attributo privo di nome che contiene tutti i dati del file. Si noti che, in risposta alle operazioni d'interrogazione dei file, come l'esecuzione del comando `dir`, l'NTFS riporta solo la dimensione dell'attributo privo di nome. Chiaramente certi attributi sono piccoli e altri sono grandi.

Ogni file dell'NTFS è descritto da uno o più elementi contenuti in un vettore memorizzato in un file speciale detto tabella principale dei file (*master file table* — MFT); la dimensione di uno di questi elementi è determinata al momento della creazione del file

system, e varia da 1 KB a 4 KB. Gli attributi di piccole dimensioni sono memorizzati nella MFT stessa, e sono detti **attributi residenti**; gli attributi più grandi, ad esempio la massa dei dati, sono invece **attributi non residenti** e sono memorizzati in una o più **estensioni** contigue (*extent*) nei dischi, e un puntatore a ogni estensione è memorizzato nell'MFT. Se un file è molto piccolo anche l'attributo dei dati si può memorizzare nell'elemento dell'MFT, mentre se un file possiede molti attributi o se è molto frammentato, e richiede quindi molti puntatori per individuare tutti i frammenti, un solo elemento nella MFT potrebbe essere insufficiente. In questo caso il file è descritto da un **elemento di base del file** che punta a elementi aggiuntivi che contengono altri puntatori e attributi.

Ogni file in un volume NTFS possiede un unico identificatore detto **riferimento al file** (*file reference*); si tratta di 64 bit che consistono di un numero del file di 48 bit e di un numero di sequenza di 16 bit. Il numero del file è il numero dell'elemento (cioè l'indice del vettore) nella MFT che descrive il file; il numero di sequenza viene incrementato ogni volta che si riutilizza un elemento della MFT. Ciò permette all'NTFS di eseguire controlli interni di coerenza, ad esempio per rilevare un riferimento a un file cancellato dopo che l'elemento della MFT è stato riusato per un nuovo file.

Lo spazio dei nomi dell'NTFS, come nell'MS-DOS e nello UNIX, è organizzato come una gerarchia di directory. Ogni directory usa una struttura di dati detta **albero B+** per memorizzare un indice dei nomi dei file contenuti nella directory; la scelta di questa struttura di dati è dovuta al fatto che essa elimina il costo della riorganizzazione dell'albero, e al fatto che la lunghezza di ogni percorso dalla radice a una foglia è la stessa. La **radice indice** di una directory contiene il livello più alto di un albero B+; nel caso di una directory di grandi dimensioni, questo livello contiene i puntatori alle estensioni del disco nelle quali è memorizzato il resto dell'albero. Ogni elemento della directory contiene il nome del file e il suo riferimento, così come una copia dell'ora dell'ultimo aggiornamento e della dimensione del file presa dagli attributi residenti nella MFT; il motivo per cui le copie di queste informazioni sono memorizzate nella directory è che in questo modo è più efficiente elencarne i contenuti: tutti i nomi, le dimensioni e gli orari sono disponibili all'interno della directory stessa, cosicché non c'è bisogno di recuperare questi attributi dagli elementi della MFT per ognuno dei file.

I metadati di ogni volume dell'NTFS sono tutti memorizzati in alcuni file, il primo dei quali è proprio la MFT. Il secondo, usato durante la procedura di recupero dei dati a seguito del danneggiamento della MFT, contiene una copia dei primi 16 elementi della MFT. Anche alcuni file seguenti sono speciali e si chiamano file di registrazione (*log file*), file del volume, tabella di definizione degli attributi, directory radice, file della mappa di bit, file d'avviamento e file delle unità di assegnazione difettose. Nel *file di registrazione* (Paragrafo 21.5.2) si annotano tutti gli aggiornamenti apportati ai metadati del file system. Il **file del volume** contiene il nome del volume, la versione dell'NTFS che lo ha formattato, e un bit che indica se è possibile che il volume sia stato alterato e necessiti quindi di controlli di coerenza. La **tabella di definizione degli attributi** indica quali tipi di attributi sono usati nel volume e quali operazioni si possono eseguire su di essi. La directory radice è il livello più alto della gerarchia del file system. Il **file della mappa di bit** in-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dica quali unità di assegnazione di un volume sono assegnate ai file, e quali invece sono libere. Il **file d'avviamento** contiene il codice d'avviamento del sistema operativo e deve risiedere a un indirizzo particolare del disco in modo da poter essere facilmente trovato dal semplice caricatore d'avviamento residente nella ROM; il file d'avviamento contiene anche l'indirizzo fisico della MFT. Il **file delle unità di assegnazione difettose**, infine, tiene traccia delle aree difettose nel volume; l'NTFS si serve di tali annotazioni per il ripristino dei dati a seguito di un errore.

21.5.2 Ripristino

In molti semplici file system un calo di tensione al momento sbagliato può danneggiare le strutture di dati del file system così gravemente da compromettere un intero volume. Molte versioni del sistema operativo UNIX memorizzano nei dischi metadati ridondanti, e tentano il ripristino dei dati a seguito di un crollo del sistema usando il programma `fsck` per controllare tutte le strutture di dati del file system e ripristinarne forzatamente uno stato coerente; si tratta di una procedura che spesso comporta l'eliminazione dei file danneggiati e il rilascio delle unità di assegnazione su cui erano stati scritti dati degli utenti ma che non erano stati correttamente registrati nelle strutture di metadati del file system: è un processo che può essere lento e può portare a significative perdite di dati.

Per conferire robustezza al file system l'NTFS adotta un orientamento differente: tutti gli aggiornamenti delle strutture di dati del file system sono eseguiti all'interno di transazioni. Prima che una struttura di dati sia modificata, la transazione annota il giornale delle modifiche con le informazioni necessarie per ripetere o annullare l'operazione; dopo che la struttura di dati è stata cambiata, la transazione scrive un'annotazione di conferma nel giornale delle modifiche per indicare il successo della transazione. A seguito di un crollo, il sistema è in grado di riportare le strutture di dati del file system a uno stato coerente elaborando le informazioni contenute nel giornale delle modifiche, prima ripetendo le operazioni confermate, poi annullando le operazioni che non erano state confermate prima del crollo. Periodicamente (di solito ogni 5 secondi) si scrive un elemento di controllo nel giornale delle modifiche: il sistema non fa uso degli elementi che precedono l'elemento di controllo al fine di ripristinare i dati dopo un crollo; tali elementi si possono quindi eliminare in modo che il file che contiene il giornale delle modifiche non cresca eccessivamente. La prima volta che si accede a un volume dopo l'avviamento del sistema, l'NTFS esegue automaticamente la procedura di ripristino.

Questa strategia non garantisce l'integrità di tutti i contenuti dei file degli utenti dopo un crollo; assicura solo che le strutture di dati del file system (cioè i file di metadati) siano integre e riflettano uno stato coerente precedente al crollo. Sarebbe possibile estendere il metodo delle transazioni ai file degli utenti, ma solo al prezzo di grossi svantaggi dal punto di vista delle prestazioni del file system.

Il giornale delle modifiche è memorizzato nel terzo file di metadati all'inizio del volume; viene creato nella fase di formattazione del file system e ha una dimensione massima fissata. È costituito da due parti: l'**area di annotazione**, che è una coda circolare di elementi, e l'**area di riavvio**, che contiene informazioni contestuali quali il punto dell'area di annotazione dal quale l'NTFS dovrebbe cominciare la lettura durante una procedura di ripristino. In effetti l'area di riavvio contiene due copie di queste informazioni, in modo che il ripristino sia possibile se una copia è stata danneggiata durante il crollo. La funzione di annotazione è fornita dal servizio dei file di registrazione del sistema operativo; oltre a scrivere gli elementi nel giornale delle modifiche e a eseguire operazioni di ripristino, questo servizio tiene traccia dello spazio libero nel file del giornale delle modifiche: quando esso si riduce eccessivamente il servizio dei file di registrazione accoda le transazioni inevase, e l'NTFS sospende tutte le nuove operazioni di I/O. Una volta che tutte le operazioni in corso sono state completate, l'NTFS attiva il gestore della cache per trasferire tutti i dati nei dischi, reimposta il giornale delle modifiche e, infine, esegue le transazioni in coda.

21.5.3 Sicurezza

La sicurezza di un volume dell'NTFS deriva dal modello a oggetti del sistema Windows 2000. Ogni oggetto file possiede un attributo descrittore della sicurezza memorizzato nel proprio elemento all'interno della MFT; quest'attributo contiene il contrassegno d'accesso del proprietario del file e la lista di controllo degli accessi che determina i privilegi d'accesso concessi a ogni utente che goda del diritto d'accesso al file.

21.5.4 Gestione dei volumi e tolleranza ai guasti

Il driver del disco a prova di guasti del sistema Windows 2000 si chiama `FtDisk`: una volta installato mette a disposizione molti modi di combinare diverse unità a disco in un unico volume logico, in modo da migliorare le prestazioni, la capacità di memorizzazione e l'affidabilità.

Uno dei possibili modi di combinare dischi diversi è di concatenarli logicamente in modo da formare un solo volume logico, com'è illustrato dalla Figura 21.7. Questo volume logico si chiama **insieme di volumi** e consiste al massimo di 32 partizioni fisiche. Un insieme di volumi che contenga un volume dell'NTFS può essere esteso senza che ciò danneggi i dati già memorizzati all'interno del file system: i metadati del file della mappa di bit del volume dell'NTFS sono semplicemente modificati al fine di includere lo spazio che si desidera aggiungere. L'NTFS continua a usare lo stesso meccanismo LCN che usa per un unico disco fisico, e il driver `FtDisk` fornisce le conversioni degli scostamenti riferiti a un volume logico in scostamenti riferiti a uno specifico disco.

Un altro modo di combinare più partizioni fisiche è di intercalare i loro blocchi in modo circolare per formare un cosiddetto **insieme di sezioni** (*stripe set*), com'è mostrato nella Figura 21.8; si tratta di un metodo noto anche come RAID di livello 0 o **sezionamento del disco**. Il driver `FtDisk` usa sezioni di 64 KB: i primi 64 KB del volume logico sono memorizzati nella prima partizione fisica, i secondi 64 KB nella seconda partizione, e così via fino a che ogni partizione abbia contribuito con 64 KB di spazio; a questo punto il processo di assegnazione riprende dal primo disco, il quale contribuisce con un secondo blocco di 64 KB. Un insieme di sezioni costituisce un solo ampio volume logico, ma l'effettiva configurazione fisica può migliorare l'ampiezza di banda dell'I/O, perché

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

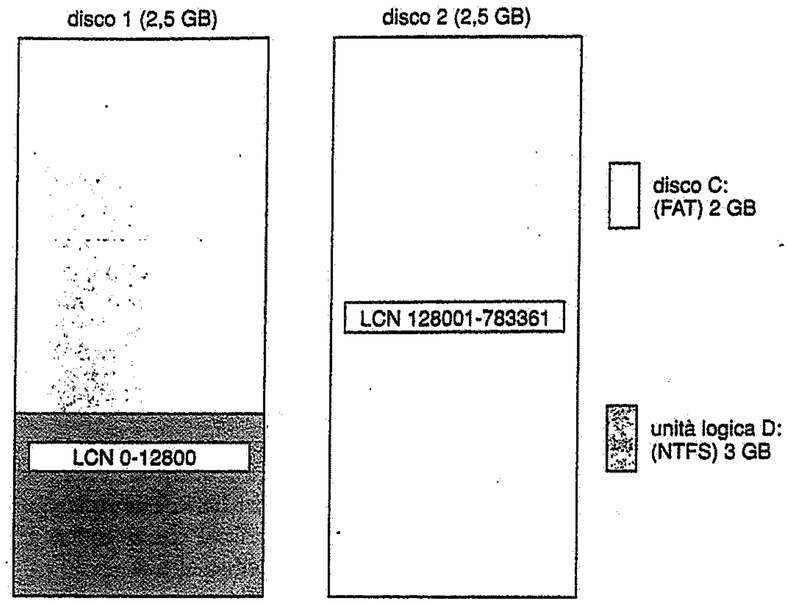


Figura 21.7 Insieme di volumi in due dischi.

per trasferimenti ingenti tutti i dischi possono trasferire dati in parallelo. Per ulteriori informazioni sulle tecniche RAID si veda il Paragrafo 14.5.

Una variante di quest'idea è l'**insieme di sezioni con parità**, illustrato nella Figura 21.9; questo metodo è anche noto come RAID di livello 5. Se l'insieme di sezioni coinvolge otto dischi, ogni sette sezioni di dati, in sette dischi separati, vi sarà una sezione di

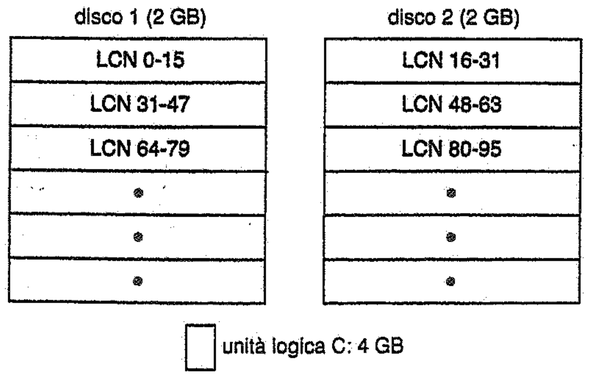


Figura 21.8 Insieme di sezioni su due unità.

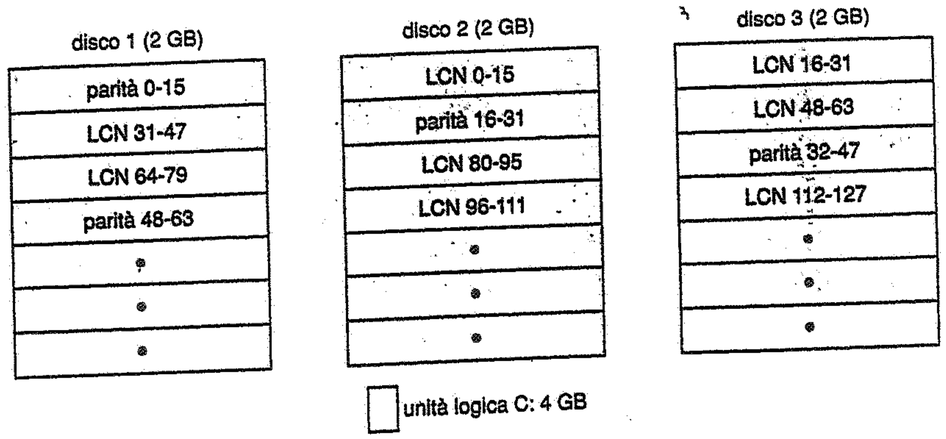


Figura 21.9 Insieme di sezioni con parità su tre unità.

parità nell'ottavo disco; essa contiene l'*or esclusivo* per byte delle sezioni di dati: se una qualunque fra le otto sezioni viene distrutta il sistema è in grado di ricostruirne i dati ricalcolando l'*or esclusivo* delle sette rimanenti. Questa caratteristica riduce di molto la probabilità di perdere dati. Si noti che la modifica di una sezione richiede l'aggiornamento della sezione di parità: sette scritture in parallelo su sette diverse sezioni richiederebbero l'aggiornamento di sette sezioni di parità. Se le sezioni di parità risiedessero in un unico disco, quel disco dovrebbe sostenere un carico di lavoro pari a sette volte il carico dei dischi contenenti dati. Per evitare una tale strozzatura per le prestazioni le sezioni di parità sono distribuite su tutti i dischi, ad esempio ancora in modo circolare. Per costruire un insieme di sezioni con parità sono necessarie almeno tre partizioni di pari dimensione in tre dischi distinti.

Un metodo ancora più robusto è la **copiatura speculare** o RAID di livello 1, rappresentato nella Figura 21.10. Un **insieme speculare** è costituito di due partizioni di pari dimensione contenenti gli stessi dati e poste in due dischi diversi. Quando un'applicazione scrive dati in un insieme speculare, `FtDisk` scrive i dati in entrambe le partizioni; se una partizione cessa di funzionare correttamente, `FtDisk` può ricorrere alla copia dei dati residente nell'altro disco. La copiatura speculare può anche migliorare le prestazioni, perché le richieste di lettura si possono ripartire fra i due componenti di un insieme speculare sottoponendo ognuno di essi alla metà del carico di lavoro. Per cautelarsi dall'eventualità di un guasto a un controllore si possono collegare i due dischi di un insieme speculare a due controllori diversi: tale organizzazione è detta **insieme duplice**.

Per gestire il malfunzionamento dei settori `FtDisk` si serve di una funzione propria di molte unità a disco detta **accantonamento di settori**, e l'NTFS esegue la cosiddetta riassociazione delle unità di assegnazione. Per quel che riguarda l'accantonamento di settori, quando si formatta un disco, l'unità crea una corrispondenza fra i numeri dei blocchi logici e i settori fisici funzionanti del disco; essa accantona alcuni settori lasciandoli voluta-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

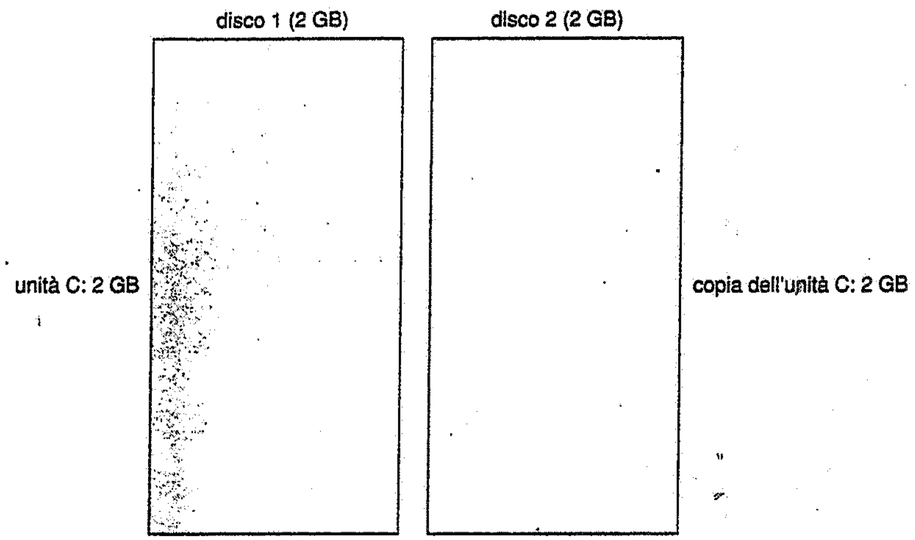


Figura 21.10 Insieme speculare su due unità.

mente inutilizzati, in modo che se un settore diviene difettoso, `FtDisk` può chiedere all'unità di sostituirlo con uno di essi. La **riassociazione delle unità di assegnazione** è una procedura messa in atto dal file system: se un blocco del disco diventa difettoso, l'NTFS lo sostituisce con un blocco libero differente cambiando i puntatori interessati nella MFT; l'NTFS inoltre annota che il blocco malfunzionante non dovrà più essere assegnato.

L'usuale conseguenza del malfunzionamento di un blocco è la perdita di dati; la riassociazione delle unità di assegnazione o l'accantonamento di settori, però, si possono combinare con i volumi tolleranti i guasti — ad esempio gli insiemi di sezioni — per evitare che in tale circostanza si perdano dati. Se una lettura non riesce, il sistema ricostruisce i dati mancanti leggendo l'immagine speculare o calcolando la parità nel caso di un insieme di sezioni con parità; i dati così ricostruiti si memorizzano in una nuova locazione del disco ottenuta grazie alla riassociazione delle unità di assegnazione o all'accantonamento di settori.

21.5.5 Compressione

L'NTFS è in grado di eseguire la compressione dei dati di un singolo file o di tutti i file di dati di un'intera directory. Per comprimere un file, divide i dati in **unità di compressione**, cioè blocchi di 16 unità di assegnazione contigue. Al momento della scrittura di ciascuna unità di compressione si applica un algoritmo di compressione dei dati; se il risultato occupa meno di 16 unità di assegnazione, si scrive nei dischi la versione compressa. Durante la lettura l'NTFS è in grado di determinare se i dati sono stati compressi, perché in questo caso la lunghezza dell'unità di compressione memorizzata è inferiore alle 16 unità di

assegnazione; per migliorare le prestazioni durante la lettura di unità di compressione contigue, l'NTFS legge e decomprime in anticipo rispetto alle richieste dell'applicazione.

Nel caso di file radi o file che contengono prevalentemente zeri, l'NTFS adotta un'altra tecnica per risparmiare spazio: le unità di assegnazione che contengono solo zeri non sono realmente assegnate o memorizzate nei dischi, ma si lasciano dei buchi nella sequenza dei numeri virtuali delle unità di assegnazione memorizzate nell'elemento della MFT relativo al file. Se nella lettura di un file trova un buco nella sequenza in questione, l'NTFS riempie semplicemente di zeri la parte interessata dell'area di memoria per l'I/O del chiamante. Questa tecnica è adottata anche nel sistema operativo UNIX.

21.5.6 Punti d'interpretazione

I punti d'interpretazione (*reparse point*) sono una nuova funzione del file system che genera appropriati codici quando si compie un accesso a determinate locazioni del file system. Un insieme di dati (*reparse data*) associato a ogni punto d'interpretazione indica al gestore di I/O quali azioni intraprendere.

I punti di montaggio costituiscono un'altra funzione aggiunta al sistema Windows 2000 e si possono considerare una forma di punto d'interpretazione. Diversamente dai sistemi UNIX, le versioni precedenti del sistema Windows non fornivano alcun modo per unire logicamente più partizioni. A ogni partizione si assegnava una lettera d'identificazione dell'unità, diversa da quella d'ogni altra partizione. Tra le altre cose, quest'organizzazione comportava che, qualora un file system avesse raggiunto il massimo della propria capacità, sarebbe stato necessario modificare la struttura delle directory per aggiungere più spazio. I punti di montaggio, in questo caso, permetterebbero di creare un nuovo volume in un'altra unità, spostare i vecchi dati nel nuovo volume e quindi montarlo nella locazione originaria. I dati sarebbero ancora utilizzabili dai programmi installati, poiché dal loro punto di vista sarebbero esattamente nella stessa locazione del file system. Il punto di montaggio è realizzato come un punto d'interpretazione con il vero nome del volume contenuto fra i dati associati al punto d'interpretazione.

Anche la funzione dei **servizi di memorizzazione remota** impiega i punti d'interpretazione. Quando si sposta un file in mezzi di memorizzazione non in linea, i dati originali del file sono sostituiti con un punto d'interpretazione che contiene le informazioni sulla locazione del file. Quando sarà richiesto l'accesso al file, s'individua il file, lo si carica e si sostituisce il punto d'interpretazione con i dati del file. Per maggiori informazioni sulla memoria terziaria, si veda il Paragrafo 14.8.1.

21.6 Servizi di rete

I servizi di rete del sistema Windows 2000 operano sia secondo il modello client-server sia secondo il modello paritetico (*peer-to-peer*); il sistema operativo fornisce anche strumenti per la gestione della rete. I componenti del sottosistema di rete permettono la trasmissione dei dati, la comunicazione fra processi, la condivisione dei file attraverso la rete e la possibilità di stampare impiegando stampanti remote.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Nella descrizione dei servizi di rete si fa riferimento a due interfacce di rete interne, dette NDIS (*network device interface specification*) e TDI (*transport driver interface*). L'interfaccia NDIS fu sviluppata nel 1989 dalla Microsoft e dalla 3Com al fine di separare gli adattatori di rete dai protocolli di trasporto in modo che gli uni potessero essere modificati senza che ciò avesse effetti sugli altri. L'NDIS costituisce l'interfaccia fra lo strato di collegamento dei dati e lo strato di controllo dell'accesso al mezzo e permette a molti protocolli di funzionare alla presenza di diversi adattatori di rete. Analogamente, la TDI è l'interfaccia fra lo strato di trasporto (strato 4) e lo strato di sessione (strato 5) del modello OSI: essa permette a ogni componente dello strato di sessione di adoperare ogni meccanismo di trasporto disponibile. (Necessità simili hanno portato al cosiddetto meccanismo *stream* dello UNIX.) L'interfaccia TDI può gestire sia il trasporto privo di connessione sia quello basato sulla connessione, ed è dotata di funzioni per la trasmissione di tutti i tipi di dati.

21.6.1 Protocolli

Nel sistema Windows 2000 i protocolli di trasporto sono realizzati come driver che si possono caricare e scaricare dinamicamente, anche se in pratica il sistema deve di solito essere riavviato a seguito di una di queste modifiche. Il sistema Windows 2000 dispone di diversi protocolli di rete.

Il protocollo SMB (*server message-block*) fu introdotto per la prima volta con l'MS-DOS 3.1, e si usa per trasmettere richieste di I/O lungo la rete. Esso tratta quattro tipi di messaggi. I messaggi `Session control` sono comandi per l'instaurazione e la chiusura della connessione di un oggetto di reindirizzamento con una risorsa condivisa del server. I messaggi `File` si usano nel reindirizzamento per accedere a file del server. Il sistema usa i messaggi `Printer` per trasmettere dati a una coda di stampa remota e per ricevere informazioni sullo stato della stampa, mentre i messaggi `Message` si usano per comunicare con un altro punto della rete.

Il NetBIOS (*network basic input/output system*) è un'interfaccia di astrazione per i dispositivi di rete analoga all'interfaccia BIOS concepita per i PC dotati del sistema operativo MS-DOS. NetBIOS fu sviluppata nei primi anni Ottanta, ed è diventata un'interfaccia standard per la programmazione di rete; è usata per stabilire nomi logici sulla rete e connessioni logiche o **sessioni** tra due nomi logici della rete, e per realizzare la trasmissione affidabile dei dati nel contesto di una sessione per mezzo di richieste NetBIOS o SMB.

Il NetBEUI (*NetBIOS extended user interface*) fu introdotto nel 1985 dalla IBM come protocollo di rete semplice ed efficiente in grado di funzionare con al più 254 macchine. Si tratta del protocollo predefinito per le trasmissioni paritetiche del sistema Windows 95 e per i servizi di rete dell'ambiente Windows for Workgroup: quando il sistema operativo Windows 2000 deve condividere risorse per mezzo di questi due tipi di rete, usa il protocollo NetBEUI. Alcuni limiti di questo protocollo sono l'assenza di tecniche d'instradamento e l'uso del nome di un calcolatore come suo indirizzo.

La pila di protocolli TCP/IP che si usa nella rete Internet è diventata lo standard *de facto* per la comunicazione in rete, ed è conseguentemente gestita da molti sistemi. Il sistema operativo Windows 2000 usa i protocolli TCP/IP per mettere in comunicazione

un'ampia gamma di sistemi operativi e di piattaforme. Il pacchetto TCP/IP del sistema Windows 2000 comprende un protocollo per la gestione di rete SNMP (*simple network-management protocol*), il protocollo DHCP (*dynamic host-configuration protocol*) per la configurazione dinamica dei calcolatori che si connettono alla rete, il servizio WINS (*Windows Internet name service*) di risoluzione dei nomi, e inoltre gestisce il NetBIOS.

Il protocollo PPTP (*point-to-point tunneling protocol*) è un protocollo messo a disposizione dal sistema Windows 2000 per la comunicazione fra moduli server per l'accesso remoto residenti in calcolatori server Windows 2000 e altri sistemi client connessi alla rete Internet; i server possono cifrare i dati trasmessi, e gestiscono reti virtuali multiprotocollo nella rete Internet.

I protocolli NetWare della Novell (servizio IPX datagram sullo strato di trasporto SPX) sono largamente usati per le reti locali di PC. Il protocollo NWLink del sistema Windows 2000 connette le reti NetBIOS e NetWare; insieme con un servizio di reindirizzamento (come il Client Service della Microsoft per Netware o il Netware Client della Novell per il sistema Windows 2000), questo protocollo permette a un client Windows 2000 di collegarsi a un server NetWare.

Il protocollo DLC (*data link control*) consente l'accesso ai mainframe IBM e alle stampanti HP direttamente collegate alla rete, ma non ha altri usi nei sistemi Windows 2000.

Il **protocollo AppleTalk** è stato concepito dalla Apple come una connessione a basso costo per consentire ai calcolatori Macintosh di condividere file. I sistemi Windows 2000 possono condividere file e stampanti con i calcolatori Macintosh per mezzo del protocollo AppleTalk se nel server di rete Windows 2000 sono stati attivati i servizi per il sistema Macintosh.

21.6.2 Elaborazione distribuita

Anche se non è un sistema operativo distribuito, il sistema Windows 2000 è comunque in grado di gestire applicazioni distribuite. Alcuni fra i meccanismi che contribuiscono a questo scopo sono il NetBIOS, le pipe con nome e le caselle postali (*mailslot*), le socket windows, le RPC e lo scambio dinamico dei dati in una rete (*network dynamic data exchange* — NetDDE).

Le applicazioni NetBIOS possono comunicare attraverso la rete grazie ai protocolli NetBEUI, NWLink o TCP/IP.

Le **pipe con nome** sono un meccanismo di trasmissione dei messaggi basato sulla connessione e furono originariamente sviluppate come interfaccia ad alto livello per le connessioni di rete NetBIOS. Un processo può anche usare una pipe con nome per comunicare con un altro processo nello stesso calcolatore. Poiché si accede alle pipe con nome tramite il file system, i controlli di sicurezza usati per i file sono applicabili anche alle pipe con nome.

Il nome di una pipe con nome segue una convenzione detta UNC (*uniform naming convention*). Un nome UNC si presenta in modo simile a un tipico nome di file remoto; il suo formato è `\\nome_del_server\nome_delle_condivisioni\x\y\z`, dove `nome_del_server` identifica un server della rete; `nome_delle_condivisioni` speci-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

fica le risorse rese disponibili agli utenti della rete, ad esempio directory, file, pipe con nome e stampanti, e la parte terminale \x\y\z è un ordinario nome di percorso di un file.

Le **caselle postali** sono un meccanismo di trasmissione dei messaggi privo di connessione; esso non è affidabile poiché un messaggio spedito a una casella postale può perdersi prima di arrivare a destinazione. Le caselle postali sono usate da applicazioni che comunicano per diffusione come gli strumenti di ricerca di componenti in una rete; esse sono anche usate dal servizio Computer Browser del sistema operativo.

Le socket dell'API del sistema Windows 2000, dette **Winsock**, sono un'interfaccia dello strato di sessione in gran parte compatibile con le socket dello UNIX, ma con estensioni per il sistema operativo Windows 2000. Esse forniscono un'interfaccia uniforme a molti protocolli di trasporto che possono avere diverse strategie d'indirizzamento, in modo che ogni applicazione Winsock possa essere eseguita usando un qualunque insieme di protocolli che segua le convenzioni Winsock.

Una RPC è un meccanismo di comunicazione client-server che permette a un'applicazione residente in una certa macchina di eseguire una chiamata di procedura relativa a codice residente in un'altra macchina. Quando il client invoca una procedura remota, il sistema delle RPC chiama un segmento di codice di riferimento (*stub*) che impacca i suoi argomenti in un messaggio e li invia tramite la rete a un determinato processo server, dopo di che si blocca. Nel frattempo, il server estrae gli argomenti dal messaggio, chiama la procedura, impacca i risultati in un messaggio, e li spedisce al segmento di codice di riferimento del client; quest'ultimo riprende l'elaborazione, riceve il messaggio, estrae i risultati della RPC e li restituisce al chiamante.

La funzione RPC del sistema operativo Windows 2000 aderisce al diffuso ambiente RPC standard per l'elaborazione distribuita, cosicché i programmi che usano la RPC di questo sistema sono facilmente adattabili ad altri sistemi. Lo standard RPC è dettagliato: nasconde molte differenze esistenti fra le architetture dei calcolatori, ad esempio la dimensione dei numeri binari e l'ordine dei bit e dei byte nelle parole, specificando formati convenzionali dei dati per i messaggi RPC.

Il sistema Windows 2000 può trasmettere messaggi RPC tramite NetBIOS, Winsock in reti TCP/IP, o pipe con nome in reti LanManager. Lo strumento LPC discusso in precedenza è simile all'RPC, eccetto per il fatto che nel caso dell'LPC la comunicazione avviene fra processi in esecuzione nello stesso calcolatore.

La scrittura del codice necessario all'impaccamento degli argomenti, alla trasmissione degli argomenti in formato standard, alla loro estrazione e all'esecuzione della procedura remota, all'impaccamento e alla trasmissione dei risultati, e infine alla loro estrazione, non è solo un'attività tediosa, ma spesso anche una fonte d'errori. Fortunatamente molto del codice in questione si può generare automaticamente sulla base di una semplice descrizione degli argomenti e dei risultati.

Il sistema Windows 2000 fornisce il linguaggio di definizione *Microsoft Interface Definition Language* per descrivere nomi, argomenti e risultati delle procedure remote. Il compilatore di questo linguaggio genera file di intestazione che dichiarano i segmenti di codice di riferimento delle procedure remote e i tipi di dati degli argomenti e dei valori di ritorno; esso genera inoltre il codice sorgente per i segmenti di codice di riferimento

usati dai client, e per le procedure di estrazione e servizio usate dai server. Quando si esegue il collegamento dell'applicazione, i segmenti di codice di riferimento s'incorporano nel codice dell'applicazione, e quando l'applicazione esegue il segmento di codice di riferimento per l'RPC il codice generato si occupa di tutto il necessario.

Il meccanismo per la comunicazione fra processi COM è stato sviluppato per l'ambiente Windows. Gli oggetti COM offrono una ben definita interfaccia per manipolare i dati negli oggetti. Il sistema Windows 2000 ha un'estensione detta DCOM che si può usare in una rete, col meccanismo RPC, per fornire un metodo trasparente di sviluppo di applicazioni distribuite.

21.6.3 Reindirizzamento e server

Nel sistema operativo Windows 2000 un'applicazione può usare l'API di I/O per accedere ai file di un'altro calcolatore come se essi fossero locali, purché in quest'ultimo sia in esecuzione un server MS-NET come nel caso dello stesso Windows 2000 o del Windows for Workgroup. Un **oggetto di reindirizzamento** è un oggetto relativo al client che trasmette richieste di I/O ai file remoti, le quali sono poi soddisfatte da un server. Per motivi legati alle prestazioni e alla sicurezza, il reindirizzamento e i server sono eseguiti nel modo protetto. Più in dettaglio, l'accesso a un file remoto avviene come segue:

- l'applicazione chiama il gestore dell'I/O per richiedere l'apertura di un file, fornendo un nome nel formato standard UNC;
- il gestore dell'I/O costruisce un pacchetto di richiesta di I/O (IRP) com'è descritto nel Paragrafo 21.3.3.5;
- il gestore dell'I/O rileva che la richiesta si riferisce a un file remoto, e chiama un driver detto MUP (*multiple universal-naming-convention provider*);
- il MUP invia in modo asincrono l'IRP a tutti gli oggetti di reindirizzamento attivi;
- un oggetto di reindirizzamento capace di soddisfare la richiesta risponde al MUP; per evitare di porre in futuro la stessa domanda a tutti gli oggetti di reindirizzamento, il MUP usa una cache per annotare l'identità dell'oggetto di reindirizzamento capace di trattare questo file;
- l'oggetto di reindirizzamento trasmette la richiesta lungo la rete al sistema remoto;
- i driver di rete del sistema remoto ricevono la richiesta e la passano al driver server;
- il driver server passa la richiesta al driver appropriato del file system locale;
- l'appropriato driver del dispositivo è chiamato per accedere ai dati;
- i risultati sono restituiti al driver server, che li rispedisce all'oggetto di reindirizzamento richiedente, il quale li restituisce all'applicazione chiamante tramite il gestore dell'I/O.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Un processo simile avviene nel caso delle applicazioni che usano l'API di rete Win32 anziché i servizi UNC, in questo caso invece di un MUP si usa un modulo detto instradatore multiplo.

Per motivi legati all'adattabilità il reindirizzamento e i server usano le API TDI per il trasporto di rete; le richieste stesse sono espresse per mezzo di un protocollo di più alto livello, che di solito è il protocollo SMB menzionato nel Paragrafo 21.6.1. L'elenco degli oggetti di reindirizzamento è contenuto in una base di dati del sistema detta registro.

21.6.4 Domìni

Per molti ambienti di rete esistono gruppi naturali di utenti, ad esempio gli studenti di un laboratorio scolastico o gli impiegati di un'azienda. Molto spesso si vuole che tutti i membri di un gruppo possano accedere a risorse condivise messe a disposizione dai calcolatori del gruppo. Per gestire i diritti d'accesso globale all'interno di un tale gruppo, il sistema Windows 2000 fa uso del concetto di dominio. In precedenza, questi domìni non avevano alcuna relazione con il DNS (*domain name system*) che trasforma nomi di calcolatori collegati alla rete Internet in indirizzi IP; ora invece sono strettamente correlati. In particolare, un dominio Windows 2000 è un gruppo di stazioni di lavoro e server con sistema operativo Windows 2000 che condividono i criteri di sicurezza e hanno una base di dati degli utenti comune. Poiché il sistema Windows 2000 ora impiega per l'autenticazione e la fidatezza il protocollo Kerberos, un dominio Windows 2000 è quello che nel Kerberos si chiama *realm* (reame). Nel sistema Windows NT s'impiegavano controllori di dominio principali e controllori di dominio di riserva, mentre ora tutti i server in un dominio sono controllori di dominio. Inoltre, le versioni precedenti richiedevano l'uso di relazioni di fiducia (*trust*) unidirezionali tra i domìni. Nel sistema Windows 2000 si segue un orientamento gerarchico basato sul DNS, e si usano relazioni di fiducia transitive che si possono trasmettersi in entrambe le direzioni nella gerarchia. Questo metodo riduce il numero di relazioni di fiducia richieste da $n \times (n-1)$ a $O(n)$. Le stazioni di lavoro nel dominio considerano fidato il controllore di dominio, cioè confidano nel fatto che esso dia informazioni corrette sui diritti d'accesso di ciascun utente (attraverso il contrassegno d'accesso dell'utente). Ogni utente si riserva comunque la possibilità di limitare l'accesso alle sue stazioni di lavoro, indipendentemente dalle disposizioni del controllore il dominio.

Poiché un'azienda potrebbe avere molti reparti e una scuola molte classi, è spesso necessario poter gestire diversi domìni all'interno di una singola organizzazione; un **albero di dominio** è una gerarchia di nomi contigui del DNS. Ad esempio, la radice dell'albero potrebbe essere `bell-labs.com`, con i due nodi figli `research.bell-labs.com` e `pez.bell-labs.com` (i domìni `research` e `pez`). Una **foresta** è un insieme di nomi non contigui; ad esempio una foresta potrebbe essere composta degli alberi `bell-labs.com` e `lucent.com`. Una foresta potrebbe anche essere composta di un unico albero di dominio.

Le relazioni tra i domìni che determinano quali calcolatori si possono considerare fidati, dette anche relazioni di fiducia, possono essere di tre tipi: unidirezionali, transitive o di collegamento. Le varie versioni del sistema Windows NT permettevano soltanto le relazioni di fiducia unidirezionali. Una **relazione di fiducia unidirezionale** (*one-way trust*) dice, ad esempio, che un dominio *A* deve fidarsi di un dominio *B*; tuttavia *B* non deve fidarsi di *A*, a meno che non sia stata stabilita un'altra relazione che lo specifichi. Secondo una **relazione di fiducia transitiva** (*transitive trust*) se *A* si può fidare di *B* e *B* si può fidare di *C*, allora *A*, *B* e *C* si possono fidare reciprocamente poiché le relazioni di fiducia sono anche simmetriche. La relazioni di fiducia transitive sono automaticamente abilitate per i nuovi domìni di un albero, e si possono stabilire solo tra i domìni in una foresta. Il terzo tipo, la **relazione di fiducia di collegamento** (*cross-link trust*), è utile per diminuire il traffico dovuto al processo di autenticazione. Si supponga che i domìni *A* e *B* siano nodi foglia e che gli utenti in *A* usino spesso le risorse in *B*. Se si adopera una relazione di fiducia transitiva, le richieste di autenticazione devono attraversare l'albero fino a raggiungere il predecessore comune dei due nodi foglia; ma se tra *A* e *B* esiste una relazione di fiducia di collegamento, allora le richieste di autenticazione verrebbero inviate direttamente all'altro nodo.

21.6.5 Risoluzione dei nomi nelle reti TCP/IP

In una rete IP la **risoluzione dei nomi** è il processo che traduce il nome di un calcolatore in un indirizzo IP (`www.bell-labs.com`, ad esempio, diventa `135.104.1.14`). Il sistema Windows 2000 fornisce diversi metodi di risoluzione dei nomi, compreso il WINS (*windows internet name service*), la risoluzione per diffusione, il sistema DNS (*domain name system*) e i file `Hosts` e `Lmhosts`. La maggior parte di questi metodi è adottata da molti altri sistemi operativi, per tale ragione ci si limiterà a descrivere il servizio WINS.

Questo metodo è basato sul fatto che due o più server WINS mantengono una base di dati dinamica contenente corrispondenze fra nomi e indirizzi IP, e mettono inoltre a disposizione un client per sottoporre richieste ai server. Si usano almeno due server per evitare che il malfunzionamento di un server blocchi il servizio WINS, oltre che per distribuire il carico di lavoro necessario per la risoluzione dei nomi.

Il servizio WINS si basa sul protocollo DHCP (*dynamic host-configuration protocol*), che aggiorna automaticamente gli indirizzi della base di dati del WINS, senza richiedere l'intervento dell'utente o dell'amministratore. Quando si avvia, un client DHCP diffonde un messaggio `discover`; ogni server DHCP che riceva il messaggio risponde con un messaggio `offer` contenente un indirizzo IP e informazioni di configurazione per il client. Il client sceglie una delle possibili configurazioni e trasmette un messaggio `request` al server DHCP prescelto; quest'ultimo ritrasmette l'indirizzo IP e le informazioni di configurazione precedentemente inviate, oltre a una licenza relativa all'indirizzo. La licenza conferisce al client il diritto di usare l'indirizzo IP per un certo periodo di tempo; allo scadere della metà di questo periodo, il client tenterà di rinnovare la licenza: se il rinnovo non venisse concesso, il client potrebbe soltanto richiedere una nuova licenza.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

21.7 Interfaccia per il programmatore

L'API Win32 è l'interfaccia fondamentale ai servizi del sistema operativo Windows 2000. Questo paragrafo descrive cinque aspetti chiave dell'API Win32: l'accesso agli oggetti del nucleo, la condivisione degli oggetti fra i processi, la gestione dei processi, la comunicazione fra i processi e la gestione della memoria.

21.7.1 Accesso agli oggetti del nucleo

Il nucleo del sistema operativo Windows 2000 mette a disposizione dei programmi d'applicazione molti servizi: i programmi usufruiscono di questi servizi manipolando oggetti del nucleo. Un processo accede a un oggetto del nucleo di nome *XXX* chiamando la funzione `Create`*XXX* per ottenere una maniglia di *XXX*; questa maniglia è unica per il processo. Se la funzione `Create` non va a buon fine riporta il valore `0` o una costante specifica detta `INVALID_HANDLE_VALUE`, secondo l'oggetto che si sta aprendo. Un processo può chiudere una maniglia chiamando la funzione `CloseHandle`, e il sistema può eliminare l'oggetto se il contatore che totalizza il numero dei processi che usano l'oggetto arriva a zero.

Il sistema Windows 2000 fornisce ai processi tre modi di condividere un oggetto. Il primo consiste nel fatto che un processo figlio può ereditare una maniglia dell'oggetto: quando invoca la funzione `Create`*XXX*, il genitore fornisce una struttura `SECURITIES_ATTRIBUTES` il cui campo `bInheritHandle` ha valore `TRUE`; questo campo crea una maniglia ereditabile. A questo punto si può creare il processo figlio, passando il valore `TRUE` all'argomento `bInheritHandle` della funzione `CreateProcess`. Nella Figura 21.11 è mostrato un esempio di codice che crea la maniglia di un semaforo ereditabile dal processo figlio.

```
...
SECURITY_ATTRIBUTES sa;
sa.nlength = sizeof(sa);
sa.lpSecurityDescriptor = NULL;
sa.bInheritHandle = TRUE;
Handle a_semaphore = CreateSemaphore(&sa,1,1,NULL);
char command_line[132];
ostrstream ostring(command_line,sizeof(command_line));
ostring << a_semaphore << ends;
CreateProcess("another_process.exe",command_line,
    NULL,NULL,TRUE, ... );
...
```

Figura 21.11 Codice che permette a un figlio di condividere un oggetto ereditandone la maniglia.


```
// processo A
...
Handle a_semaphore = CreateSemaphore(NULL,1,1,"MySEM1");
...
// processo B
...
Handle b_semaphore = OpenSemaphore(SEMAPHORE_ALL_ACCESS,
                     FALSE, "MySEM1");
...
```

Figura 21.12 Codice per la condivisione di un oggetto per ricerca del nome.

Nell'ipotesi che il processo figlio sappia quali maniglie sono condivise, la condivisione degli oggetti permette di realizzare la comunicazione fra genitore e figlio; nell'esempio della Figura 21.11 il processo figlio otterrebbe il valore della maniglia dal primo argomento della riga di comando, e potrebbe quindi condividere il semaforo col processo genitore.

Il secondo metodo di condivisione degli oggetti assume che l'oggetto abbia ricevuto un nome dal processo che lo ha creato, cosicché un altro processo può richiedere l'apertura dell'oggetto riferendosi al suo nome. Questa tecnica ha due svantaggi: il primo è che il sistema operativo Windows 2000 non fornisce un modo di controllare se un oggetto con un certo nome già esiste; il secondo è che lo spazio dei nomi degli oggetti è globale, senza alcuna distinzione relativa ai tipi di oggetti. Due applicazioni potrebbero ad esempio creare un oggetto denominato *pipe*, mentre ciò che si voleva erano due oggetti distinti e forse anche diversi.

Il vantaggio degli oggetti con nome è che processi non correlati possono facilmente condividerli: un primo processo chiamerebbe una delle funzioni `Create`*XXX*, e fornirebbe un nome tramite il parametro `lpszName`; un secondo processo potrebbe ottenere una maniglia di questo oggetto chiamando `Open`*XXX* (o `Create`*XXX*) con lo stesso nome, com'è mostrato dall'esempio nella Figura 21.12.

Il terzo modo di condividere oggetti si basa sulla funzione `DuplicateHandle`; esso richiede qualche altro metodo di comunicazione fra processi per trasmettere la maniglia duplicata. Se un processo possiede una maniglia con un certo valore, un altro processo può ottenere una maniglia dello stesso oggetto, e pertanto condividerlo; un esempio di questo metodo è mostrato nella Figura 21.13.

21.7.2 Gestione dei processi

Nel sistema Windows 2000 un processo è un'istanza in esecuzione di un'applicazione, e un thread è un'unità di codice che può essere sottoposta a scheduling dal sistema operativo: un processo contiene uno o più thread. Un processo si avvia quando qualche altro processo invoca la funzione `CreateProcess`, la quale carica le librerie dinamiche usate

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
...
// il processo A vuole fornire al processo B l'accesso a un semaforo
// processo A
Handle a_semaphore = CreateSemaphore(NULL,1,1,NULL);
// invia il valore del semaforo al processo B
// usando un messaggio o la condivisione della memoria
...
// processo B
Handle process_a = OpenProcess(PROCESS_ALL_ACCESS,FALSE,
                   process_id_of_A);
Handel b_semaphore;
DuplicateHandle(process_a,a_semaphore,
   GetCurrentProcess(),&b_semaphore,0,FALSE,
   DUPLICATE_SAME_ACCESS);
// usa b_semaphore per accedere al semaforo
...
```

Figura 21.13 Codice per la condivisione di un oggetto tramite il passaggio di una maniglia.

dal processo e crea un **thread principale**; tramite la funzione `CreateThread` si possono creare thread aggiuntivi: ogni thread possiede la sua pila, la cui dimensione predefinita è di 1 MB se non è altrimenti specificato da un argomento di `CreateThread`. A causa del fatto che alcune funzioni di fase d'esecuzione del Linguaggio C mantengono informazioni di stato in variabili statiche quali errno, un'applicazione multithread deve cautelarsi da accessi non sincronizzati: la funzione `beginthreadex` fornisce la sincronizzazione appropriata.

Le librerie di collegamento dinamico e i file eseguibili caricati nello spazio d'indirizzi di un processo sono identificati da una **maniglia di istanza**, il cui valore è in effetti l'indirizzo virtuale a partire dal quale si carica il file. Un'applicazione ottiene la maniglia di un modulo che risiede nel suo spazio d'indirizzi passando il nome del modulo alla funzione `GetModuleHandle`: se come nome si passa il valore `NULL`, si ottiene come risposta l'indirizzo di base del processo. I 64 KB inferiori dello spazio d'indirizzi non sono usati, cosicché un processo che tentasse di accedere alla locazione indicata da un puntatore `NULL` commetterebbe una violazione d'accesso.

Le priorità nell'ambiente Win32 sono basate sul modello di scheduling del sistema Windows 2000, ma non tutti i valori di priorità possono essere scelti. L'ambiente Win32 usa quattro classi di priorità: `IDLE_PRIORITY_CLASS` (livello di priorità 4), `NORMAL_PRIORITY_CLASS` (livello di priorità 8), `HIGH_PRIORITY_CLASS` (livello di priorità 13) e `REALTIME_PRIORITY_CLASS` (livello 24). Ordinariamente i processi appartengono alla classe `NORMAL_PRIORITY_CLASS`, sempre che il genitore del processo non fosse di classe `IDLE_PRIORITY_CLASS`, oppure non fosse stata specificata un'altra classe al momento della chiamata alla funzione `CreateProcess`. La classe di priorità di un processo si può modificare tramite la funzione `SetPriorityClass`, o passando un argomento al comando `START`; il comando `START /REALTIME cbserver.exe`, ad esempio, eseguirebbe il programma `cbserver` assegnandolo alla classe di priorità `REALTIME_PRIORITY_CLASS`. Si noti che solo gli utenti che godono del privilegio che consente d'incrementare le priorità di scheduling possono assegnare un processo alla classe `REALTIME_PRIORITY_CLASS`; gli amministratori e gli appartenenti al gruppo *Power users* godono automaticamente di questo privilegio.

Quando un utente esegue un programma interattivo il sistema deve cercare di fornire prestazioni particolarmente buone per il processo interessato: è per questo motivo che il sistema operativo Windows 2000 ha una regola di scheduling speciale per i processi di classe `NORMAL_PRIORITY_CLASS`. Il sistema operativo distingue tra il processo in primo piano correntemente selezionato sullo schermo, e i processi in sottofondo correntemente non selezionati. Quando un processo si sposta in primo piano il sistema operativo incrementa il suo quanto di tempo di un certo fattore — tipicamente 3. (Questo fattore può essere cambiato dall'opzione sulle prestazioni della sezione di sistema del pannello di controllo.) Questo aumento ha l'effetto di concedere al processo in primo piano un tempo d'esecuzione tre volte più lungo prima che il processo subisca la prelazione dovuta alla partizione del tempo.

La priorità iniziale di un thread è determinata dalla sua classe, ma si può modificare tramite la funzione `SetThreadPriority`; essa accetta un argomento che specifica una priorità relativa alla priorità di base della classe del thread:

- `THREAD_PRIORITY_LOWEST`: base – 2
- `THREAD_PRIORITY_BELOW_NORMAL`: base – 1
- `THREAD_PRIORITY_NORMAL`: base + 0
- `THREAD_PRIORITY_ABOVE_NORMAL`: base + 1
- `THREAD_PRIORITY_HIGHEST`: base + 2

Per regolare la priorità si usano altri due valori convenzionali. Il nucleo ha due classi di priorità (Paragrafo 21.3.2): la classe per le elaborazioni in tempo reale (livelli 16-31) e la classe a priorità variabile (livelli 0-15); `THREAD_PRIORITY_IDLE` rappresenta il livello di priorità 16 per i thread d'elaborazione in tempo reale, e il livello di priorità 1 per i thread a priorità variabile. `THREAD_PRIORITY_TIME_CRITICAL` rappresenta il livello di priorità 31 per i thread d'elaborazione in tempo reale, e il livello 15 per i thread a priorità variabile.

Come si descrive nel Paragrafo 21.3.2, il nucleo regola dinamicamente la priorità di un thread in funzione del fatto che si tratti di un thread con prevalenza di I/O o con prevalenza d'elaborazione; l'API Win32 fornisce un modo di disattivare questo processo di regolazione tramite le funzioni `SetProcessPriorityBoost` e `SetThreadPriorityBoost`.

Un thread si può creare in uno stato detto **stato sospeso**; in questo caso non sarà eseguito fino a che un altro thread non lo renda idoneo all'esecuzione tramite la funzione `ResumeThread`; la funzione `SuspendThread` compie l'operazione inversa. Queste fun-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

zioni mantengono un contatore in modo che se il thread è sospeso due volte deve essere richiamato due volte tramite la funzione `ResumeThread` prima di poter essere eseguito.

Per sincronizzare l'accesso concorrente dei thread agli oggetti condivisi, il nucleo fornisce oggetti di sincronizzazione come i semafori e i mutex; inoltre, i thread si possono sincronizzare usando le funzioni `WaitForSingleObject` o `WaitForMultipleObjects`. Un altro metodo di sincronizzazione dell'API Win32 è la sezione critica: si tratta di una sezione sincronizzata di codice eseguibile da un solo thread alla volta. Un thread istituisce una sezione critica invocando la funzione `InitializeCriticalSection`; prima di entrare nella sezione critica l'applicazione deve invocare la funzione `EnterCriticalSection`, e prima di lasciarla deve invocare la funzione `LeaveCriticalSection`. Queste due funzioni garantiscono che nella circostanza in cui diversi thread tentino di accedere in modo concorrente alla sezione critica, l'accesso sarà concesso a un solo thread, mentre gli altri dovranno attendere all'interno della funzione `EnterCriticalSection`. Questo metodo è un po' più rapido di quello basato sugli oggetti di sincronizzazione del nucleo.

Una **fibra** (*fiber*) è codice d'utente sottoposto a un regime di scheduling definito dall'utente: così come un processo può essere costituito di diversi thread, esso può essere costituita di diverse fibre. Una differenza sostanziale fra i thread e le fibre è che mentre i thread si possono eseguire in modo concorrente, le fibre si possono eseguire solo una alla volta, anche se sono disponibili più unità d'elaborazione. Questo strumento è fornito dal sistema Windows 2000 al fine di facilitare l'adattabilità delle applicazioni scritte per un modello d'esecuzione a fibre del sistema UNIX.

Il sistema crea una fibra chiamando una tra le due funzioni `ConvertThreadToFiber` o `CreateFiber`; la differenza più importante fra queste due funzioni è che la seconda non avvia l'esecuzione della fibra appena creata: per cominciare l'esecuzione l'applicazione deve chiamare la funzione `SwitchToFiber`; per terminare l'esecuzione di una fibra l'applicazione deve chiamare la funzione `DeleteFiber`.

21.7.3 Comunicazione fra processi

Uno dei modi in cui le applicazioni per l'ambiente Win32 possono realizzare la comunicazione fra processi è basato sulla condivisione degli oggetti del nucleo; un altro metodo, particolarmente diffuso per le applicazioni con interfaccia d'utente grafica, è basato sullo scambio di messaggi.

Un thread può spedire un messaggio a un altro thread o a una finestra chiamando una fra le funzioni `PostMessage`, `PostThreadMessage`, `SendMessage`, `SendThreadMessage` e `SendMessageCallback`. La differenza fra le funzioni *Post* e quelle *Send* è che le prime sono asincrone: restituiscono immediatamente il controllo in modo che il thread chiamante non sappia esattamente quando il messaggio giungerà a destinazione; le funzioni *Send*, invece, sono sincrone e bloccano quindi il chiamante fino a quando il messaggio non sia stato recapitato.

Un thread può trasmettere dati insieme con un messaggio; in questo caso i dati devono essere copiati, perché processi diversi hanno spazi d'indirizzi distinti. Il sistema copia i dati chiamando `SendMessage` per trasmettere un messaggio di tipo `WM_COPYDATA`

con una struttura di dati `COPYDATASTRUCT` che contiene la lunghezza dell'indirizzo dei dati da trasferire; quando si trasmette il messaggio, il sistema operativo copia i dati in una nuova regione della memoria, e ne fornisce l'indirizzo virtuale al processo ricevente.

A differenza di ciò che avviene nell'ambiente Windows a 16 bit, ogni thread dell'ambiente Win32 ha la propria coda d'ingresso dalla quale riceve messaggi (tutti i dati in ingresso si ricevono tramite messaggi). Si tratta di un'architettura più affidabile rispetto alla condivisione della coda d'ingresso dell'ambiente Windows a 16 bit, perché con le code distinte un'applicazione bloccata non impedisce la ricezione dei dati per le altre applicazioni. Se un'applicazione per l'ambiente Win32 non impiega la funzione `GetMessage` per trattare gli eventi nella sua coda d'ingresso, la coda si riempirà, e dopo circa 5 secondi il sistema contrassegnerà l'applicazione come 'Non rispondente'.

21.7.4 Gestione della memoria

L'API Win32 mette a disposizione delle applicazioni molti modi d'uso della memoria: la memoria virtuale, i file associati allo spazio d'indirizzi di memoria, gli *heap*, e la memoria locale dei thread.

Per prenotare o eseguire l'assegnazione di una regione della memoria virtuale, un'applicazione invoca la funzione `VirtualAlloc`, e per eseguire le operazioni inverse invoca `VirtualFree`. Queste funzioni permettono all'applicazione di specificare l'indirizzo virtuale a partire dal quale si deve assegnare la memoria; esse operano con multipli della dimensione della pagina di memoria, e richiedono che l'indirizzo iniziale specificato sia maggiore di 0x10000. Alcuni esempi di queste funzioni sono riportati dalla Figura 21.14.

```
...
// riserva 16 MB all'estremo superiore del nostro spazio di indirizzi
void *buf = VirtualAlloc(0,0x1000000,MEM_RESERVE | MEM_TOP_DOWN,
            PAGE_READWRITE);
// effettua l'assegnazione degli 8 MB superiori dello spazio riservato
VirtualAlloc(buf + 0x800000,0x800000 MEM_COMMIT,PAGE_READWRITE);
// fa qualcosa
// annulla l'assegnazione
VirtualFree(buf + 0x800000,0x800000,MEM_DECOMMIT);
// rilascia tutto lo spazio assegnato
VirtualFree(buf,0,MEM_RELEASE);
...
```

Figura 21.14 Frammenti di codice per l'assegnazione della memoria virtuale.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.


```
...
// apre il file o lo crea nel caso in cui non esista
HANDLE hfile = CreateFile("somefile",GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,NULL,OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,NULL);
// crea uno spazio di 8 MB per l'associazione del file
HANDLE hmap = CreateFileMapping(hfile,PAGE_READWRITE;SEC_COMMIT,
    0,0x800000,"SHM_1");
// ottiene una finestra sullo spazio associato
void *buf = MapViewOfFile(hmap,FILE_MAP_ALL_ACCESS,0,0,0x800000);
// fa qualcosa
// annulla l'assegnazione del file
UnmapViewOfFile(buf);
CloseHandle(hmap)
CloseHandle(hfile);
...
```

Figura 21.15 Frammenti di codice per l'associazione allo spazio d'indirizzi di memoria di un file.

Un processo può fissare nella memoria fisica alcune sue pagine chiamando la funzione `VirtualLock`; il massimo numero fissabile di pagine per processo è 30, sempre che il processo non chiami prima la funzione `SetProcessWorkingSetSize` per aumentare la dimensione minima dell'insieme di lavoro.

Un'applicazione può anche fare uso della memoria associando un file al proprio spazio d'indirizzi; si tratta anche di un metodo conveniente per la condivisione di memoria fra due processi: entrambi i processi associano lo stesso file alla loro memoria virtuale. L'associazione di un file è un procedimento a più fasi, come dimostra l'esempio della Figura 21.15.

Se un processo desidera associare uno spazio d'indirizzi al solo scopo di condividere memoria con un altro processo, la presenza di un file è superflua: il processo può invocare la funzione `CreateFileMapping` specificando come maniglia del file il valore 0xffffffff, oltre a una certa dimensione; l'oggetto risultante si può condividere per ereditarietà, per nome o per duplicazione.

Il terzo modo in cui un'applicazione può usare la memoria è rappresentato dallo heap. Uno *heap* nell'ambiente Win32 è semplicemente una regione riservata di uno spazio d'indirizzi. Un processo del Win32 è dotato al momento della sua creazione di uno *heap predefinito* di 1 MB; poiché molte funzioni dell'ambiente Win32 usano lo heap predefinito, è necessario sincronizzare gli accessi allo heap per evitare che le sue strutture di dati siano danneggiate da accessi concorrenti di thread diversi. L'ambiente Win32 mette a disposizione le seguenti funzioni per la gestione e l'assegnazione di uno heap: `HeapCreate`, `HeapAlloc`, `HeapRealloc`, `HeapSize`, `HeapFree` e `HeapDestroy`; l'API Win32

fornisce anche le funzioni `HeapLock` e `HeapUnlock` che garantiscono a un thread l'accesso esclusivo a uno heap; a differenza di `VirtualLock`, queste funzioni eseguono solo la sincronizzazione, e non fissano pagine nella memoria fisica.

Il quarto modo in cui un'applicazione può usare la memoria è rappresentato dal meccanismo di memorizzazione dei thread locali. Di solito le funzioni che fanno uso di dati statici o globali non operano correttamente in un ambiente multithread: la funzione di fase d'esecuzione `strtok` del Linguaggio C, ad esempio, impiega una variabile statica per tenere traccia della posizione corrente durante la scansione di una sequenza di caratteri; affinché due thread concorrenti possano eseguire correttamente `strtok`, devono tenere distinte le variabili che mantengono la posizione corrente. Il meccanismo di assegnazione della memoria locale a un thread fornisce spazio di memoria globale thread per thread; esso può essere messo in atto sia tramite metodi dinamici sia tramite metodi statici. Il metodo dinamico è illustrato nella Figura 21.16.

Per usare una variabile statica locale a un thread, in modo da assicurare che ogni thread ne possieda una copia privata, un'applicazione dovrebbe dichiarare la variabile come segue:

```
__declspec(thread) DWORD cur_pos = 0;
```

21.8 Sommario

Il sistema operativo Windows 2000 è stato concepito dalla Microsoft come sistema operativo adattabile ed estendibile, capace di trarre vantaggio dalle architetture recenti e di mettere a frutto nuove tecniche; esso gestisce l'elaborazione parallela simmetrica e mette a disposizione diversi ambienti operativi. L'uso degli oggetti del nucleo per fornire i servizi fondamentali e la gestione del modello di elaborazione client-server permettono a questo sistema operativo di offrire un'ampia gamma di ambienti d'applicazione: il sistema operativo Windows 2000 ad esempio è in grado di eseguire programmi compila-

```
// riserva spazio per una variabile
DWORD var_index = TlsAlloc();
// gli assegna un valore
TlsSetValue(var_index,10);
// lo rilegge
int var = TlsGetValue(var_index);
// rilascia l'indice
TlsFree(var_index);
```

Figura 21.16 Codice per l'assegnazione dinamica di memoria privata a un thread.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

ti per l'MS-DOS, Win16, Windows 95, Windows 2000 e POSIX. Offre la gestione della memoria virtuale, servizi integrati di caching e lo scheduling con diritto di prelazione. Adotta un modello di sicurezza più robusto di quelli dei precedenti sistemi operativi della Microsoft, e comprende caratteristiche internazionali. È eseguibile su un'ampia varietà di calcolatori; in questo modo gli utenti possono scegliere e aggiornare i propri calcolatori, e i dispositivi che li compongono, in funzione delle loro disponibilità finanziarie e delle prestazioni richieste senza dover modificare le applicazioni eseguite.

21.9 Esercizi

21.1 Spiegate alcune ragioni per le quali lo spostamento dal modo d'utente al modo protetto del codice del sistema operativo Windows NT relativo alla grafica può ridurre l'affidabilità del sistema. Dite quali sono gli originari obiettivi di progettazione del sistema Windows NT violati da tale riduzione.

21.2 Il gestore della MV del sistema operativo Windows 2000 adotta un processo a due stadi per assegnare la memoria. Identificate alcuni aspetti dell'utilità di questo metodo.

21.3 Discutete alcuni vantaggi e svantaggi della particolare struttura della tabella delle pagine del sistema operativo Windows 2000.

21.4 Determinate il massimo numero di eccezioni di pagina mancante che si possono verificare rispettivamente durante l'accesso a un indirizzo virtuale e a un indirizzo virtuale condiviso. Descrivete il meccanismo fornito dall'architettura della maggior parte delle CPU per diminuire queste quantità.

21.5 Discutete lo scopo di un elemento prototipo della tabella delle pagine.

21.6 Elencate i passi che il gestore della cache deve eseguire per copiare i dati dalla cache e nella cache.

21.7 Discutete i problemi principali che comporta l'esecuzione di applicazioni per l'ambiente Windows a 16 bit su una VDM, e descrivete le soluzioni offerte dal sistema operativo Windows 2000 per ciascuno di questi problemi evidenziando almeno uno svantaggio di ciascuna soluzione.

21.8 Discutete i cambiamenti che sarebbe necessario apportare al sistema operativo Windows 2000 affinché possa eseguire processi dotati di uno spazio d'indirizzi a 64 bit.

21.9 Il sistema operativo Windows 2000 possiede un gestore centralizzato della cache; discutetene vantaggi e svantaggi.

21.10 Il sistema operativo Windows 2000 adotta un sistema per l'I/O orientato ai pacchetti; discutete i pro e i contro di questo metodo.

21.11 Dite quali meccanismi del sistema operativo Windows 2000 si potrebbero adoperare per accedere a una base di dati di 1 terabyte residente nella memoria centrale.

21.10 Note bibliografiche

[Solomon e Russinovich 2000] offre un'introduzione al sistema operativo Windows 2000 e si addentra notevolmente nei dettagli dei meccanismi e dei componenti interni del sistema. [Tate 2000] è un buon testo di consultazione per l'uso del sistema Windows 2000. La serie di sei volumi *Microsoft Windows 2000 Server Resource Kit* [Microsoft 2000b] è molto utile per l'uso, la configurazione e la diffusione dell'installazione di tale sistema operativo. Il periodico *Microsoft Developer Network Library* [Microsoft 2000a] ha cadenza quadrimestrale, ed è ricco di informazioni sul sistema operativo Windows 2000 e su altri prodotti della Microsoft. [Iseminger 2000] è un buon testo di consultazione sull'Active Directory. [Richter 1997] discute nei dettagli la stesura di programmi che si servono dell'API Win32. Una buona trattazione degli alberi B+ è contenuta in [Silberschatz et al. 2001].

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Capitolo 22

Prospettiva storica

Nel Capitolo 1 è presentata una breve rassegna storica dello sviluppo dei sistemi operativi. Tale rassegna è priva di dettagli, poiché i concetti fondamentali dei sistemi operativi (scheduling della CPU, gestione della memoria, processi e così via) sono trattati nei capitoli seguenti. Una volta chiariti i concetti di base, è possibile esaminare come questi stessi concetti siano stati applicati in più sistemi operativi vecchi e molto influenti. Alcuni di essi, come il sistema XDS-940 o il sistema THE, sono stati pezzi unici, altri, come l'OS/360, hanno avuto una grande diffusione. L'ordine di presentazione è stato scelto in modo da evidenziare le analogie e le differenze tra i sistemi, quindi non è strettamente cronologico o ordinato per importanza. Chiunque si occupi di sistemi operativi dovrebbe avere confidenza con tutti questi sistemi.

Nella descrizione sono inclusi riferimenti a ulteriori letture. I documenti scritti dai progettisti dei sistemi sono importanti sia per il contenuto tecnico sia per lo stile e il gusto.

22.1 Primi sistemi

I primi calcolatori erano macchine molto grandi che si gestivano da una console dalla quale il programmatore, che era anche l'operatore del sistema di calcolo, poteva scrivere e gestire un programma. Il programma veniva dapprima caricato manualmente nella memoria impiegando gli interruttori del pannello frontale, impartendo un'istruzione alla volta, oppure inserendo un nastro di carta o un pacco di schede perforate; quindi si premevano i pulsanti appropriati per impostare l'indirizzo iniziale e per avviare l'esecuzione. Durante l'esecuzione del programma, il programmatore o operatore poteva controllarne l'esecuzione per mezzo di spie situate sulla console. Se si riscontravano errori, il programmatore poteva fermare il programma, esaminare il contenuto della memoria e dei registri e mettere a punto il programma direttamente dalla console. L'esito veniva stampato, oppure trascritto perforando appositi nastri o schede di carta per essere stampato successivamente.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Col passare del tempo furono sviluppati altri programmi, e altre macchine e dispositivi. I lettori di schede, le stampanti e i nastri magnetici divennero d'uso comune. Per facilitare la programmazione furono progettati assemblatori, caricatori e collegatori, e create librerie di funzioni comuni che si potevano copiare in un programma senza dover essere riscritte, consentendo il riutilizzo di codice già scritto.

Le procedure che eseguivano I/O assunsero un'importanza particolare. Ogni nuovo dispositivo di I/O aveva le sue caratteristiche che richiedevano un'attenta programmazione. Per ogni dispositivo di I/O si scriveva una procedura speciale, chiamata driver di dispositivo, capace di interagire con le memorie di transito, gli indicatori, i registri, i bit di controllo e i bit di stato di ogni specifico dispositivo. Un compito semplice, come la lettura di un carattere da un lettore di nastri, poteva implicare complesse sequenze di operazioni specifiche del dispositivo. Invece di dover scrivere ogni volta il codice necessario, s'impiegava semplicemente il driver di dispositivo della libreria.

In seguito apparvero i compilatori per i linguaggi FORTRAN, COBOL e altri linguaggi, che resero ancora più semplice la programmazione, ma più complesso il funzionamento del calcolatore. Per preparare l'esecuzione di un programma scritto in FORTRAN, il programmatore doveva prima caricare nel calcolatore il compilatore del FORTRAN, che normalmente si teneva in un nastro magnetico; quindi era necessario montare il nastro giusto nell'unità a nastri. Il programma veniva letto dal lettore di schede e scritto in un altro nastro. Il compilatore del FORTRAN produceva in uscita un programma in linguaggio assemblativo che, a sua volta, doveva essere assemblato; quest'operazione richiedeva il montaggio di un altro nastro in cui si trovava l'assemblatore. Il risultato prodotto dall'assemblatore si doveva collegare alle procedure della libreria. Infine, si poteva leggere ed eseguire la forma binaria del programma. Si effettuava il caricamento nella memoria, la correzione e la messa a punto dalla console e infine l'esecuzione.

Il tempo di preparazione necessario per eseguire un lavoro d'elaborazione poteva essere molto elevato; ciascun lavoro consisteva di molti passi distinti:

1. il caricamento del nastro col compilatore del linguaggio FORTRAN;
2. l'esecuzione del compilatore;
3. la rimozione del nastro con il compilatore;
4. il caricamento del nastro dell'assemblatore;
5. l'esecuzione dell'assemblatore;
6. la rimozione del nastro dell'assemblatore;
7. il caricamento del programma oggetto;
8. l'esecuzione del programma oggetto.

Se durante una di queste fasi si riscontrava un errore, il programmatore o l'operatore doveva ricominciare tutto da capo. Ogni fase poteva implicare il caricamento e la rimozione di nastri magnetici, nastri di carta e schede perforate.

Il tempo necessario per la preparazione di un lavoro era un vero problema: durante il montaggio dei nastri e l'attività del programmatore alla console la CPU restava inattiva. Si ricordi che i primi calcolatori disponibili erano molto pochi e quei pochi erano molto costosi; un calcolatore poteva costare milioni di dollari, anche senza considerare i costi del lavoro dei programmatori, della corrente elettrica, del raffreddamento e così via; il tempo d'uso di un calcolatore assumeva quindi un valore molto elevato e per ammortizzare i costi d'investimento era necessario un uso intenso.

Per ottimizzare la produttività di un calcolatore si trovò una soluzione su due fronti. Innanzi tutto si ricorreva a un operatore professionista, e il programmatore quindi non doveva più lavorare alla macchina. Di conseguenza, non appena si terminava un lavoro, l'operatore poteva iniziare il lavoro successivo; poiché l'operatore aveva più esperienza nel montaggio dei nastri di quanta ne avesse un programmatore, anche il tempo di preparazione si ridusse. Il programmatore doveva fornire le schede o i nastri necessari insieme con una breve descrizione dei modi d'esecuzione del programma. Naturalmente, l'operatore non poteva mettere a punto un programma non corretto dalla console, giacché che non era in grado di capire il programma stesso. Perciò, quando si presentavano errori nel programma, si stampava un'immagine dell'intero contenuto della memoria e dei registri e il programmatore doveva fare le correzioni basandosi sulle informazione così ottenute. In questo modo l'operatore poteva continuare immediatamente col lavoro successivo, ma al programmatore restava un compito di correzione assai più difficile.

Un secondo aspetto era che i lavori che avevano requisiti simili venivano raggruppati in lotti (*batch*) ed eseguiti nel calcolatore come un unico gruppo per ridurre il tempo di preparazione. Si supponga ad esempio che l'operatore ricevesse un lavoro scritto in FORTRAN, uno scritto in COBOL e uno scritto ancora in FORTRAN. Per eseguirli in quell'ordine avrebbe dovuto predisporre la macchina prima per il FORTRAN, quindi per il COBOL e poi di nuovo per il FORTRAN, tutto ciò richiedeva il caricamento dei nastri compilatori, e così via. Se, invece, eseguiva i due lavori scritti in FORTRAN come un unico lotto, poteva predisporre il calcolatore una volta sola per il linguaggio FORTRAN, risparmiando tempo.

Ma c'erano ancora alcuni problemi. Ad esempio, l'operatore doveva stabilire, osservando la console, se un programma era terminato in maniera normale o anormale e *perché* ciò era accaduto, quindi, se era necessario, doveva stampare l'immagine del contenuto della memoria, caricare il lavoro successivo nel lettore di schede o nel lettore del nastro e infine riavviare il calcolatore. Durante il passaggio da un lavoro a un altro, la CPU restava inattiva. Per eliminare anche questo tempo morto, fu sviluppato il cosiddetto **sequenzializzatore automatico dei lavori d'elaborazione**; questa tecnica permise la creazione dei primi rudimentali sistemi operativi. Fu creato un piccolo programma, chiamato **monitor residente** per il trasferimento automatico del controllo dell'elaborazione da un lavoro a quello successivo (Figura 22.1), che si trovava sempre, cioè risiedeva, nella memoria.

Al momento dell'accensione del calcolatore, s'invocava il monitor residente che trasferiva il controllo a un programma. Terminato il programma, il controllo tornava al monitor residente che lo passava al programma successivo; in questo modo il monitor residente sequenzializzava automaticamente i programmi e i lavori d'elaborazione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

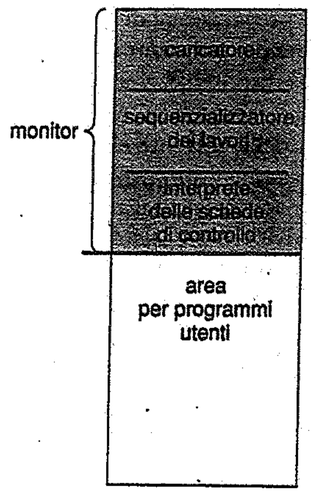


Figura 22.1 Configurazione della memoria per un monitor residente.

Ci si può domandare come faceva il monitor residente a sapere quale programma doveva eseguire. In precedenza all'operatore si consegnava una breve descrizione dei programmi da eseguire e dei relativi dati. Per passare queste informazioni direttamente al monitor furono adottate le **schede di controllo**. L'idea era semplice. Oltre al programma o ai dati, il programmatore inseriva le schede di controllo, che contenevano le direttive che indicavano al monitor residente quale programma doveva eseguire. Ad esempio, un normale programma utente avrebbe potuto richiedere l'esecuzione di uno di questi tre programmi: compilatore del linguaggio FORTRAN (FTN), assemblatore (ASM), oppure programma d'utente (RUN). Per ciascuno di questi programmi si può impiegare una scheda di controllo diversa:

$FTN — esegue il compilatore del linguaggio FORTRAN;
$ASM — esegue l'assemblatore;
$RUN — esegue il programma d'utente.

Queste schede indicavano al monitor residente quali erano i programmi da eseguire. Per definire i limiti di ogni lavoro si potevano adoperare altre due schede di controllo:

$JOB — prima scheda di un lavoro;
$END — ultima scheda di un lavoro.

Queste due schede avrebbero potuto essere utili per contabilizzare le risorse di macchina impiegate dal programmatore. Per definire il nome del lavoro, il codice d'addebito e così via si potevano usare opportuni parametri. Per altre funzioni, come la richiesta di caricamento o rimozione di un nastro da parte dell'operatore, si potevano definire altre schede di controllo.

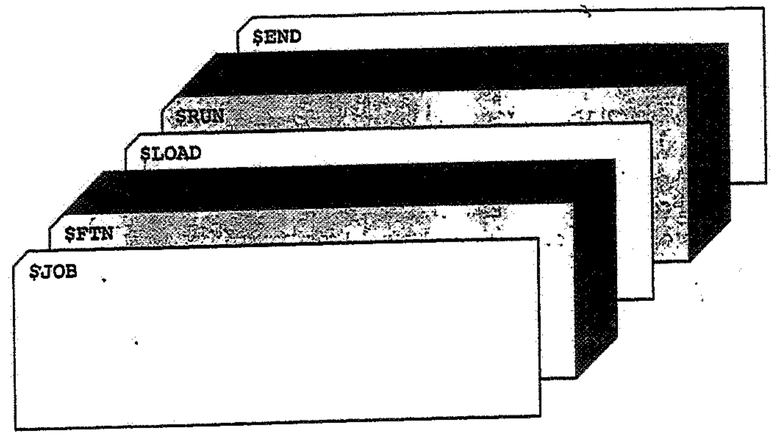


Figura 22.2 Pacco di schede per un sistema a lotti.

Le schede di controllo dovevano essere distinte dalle schede dei dati o del programma. Tale distinzione si otteneva per mezzo di un carattere o di un simbolo speciale riportato sulla scheda stessa. Parecchi sistemi identificavano una scheda di controllo riportando nella prima colonna il carattere $. Altri usano un simbolo diverso: il Job Control Language (JCL) della IBM, ad esempio, usava due barre (//) inserite nelle prime due colonne. Nella Figura 22.2 è riportato un esempio di preparazione del pacco di schede per un sistema a lotti.

Un monitor residente era quindi composto da diversi elementi chiaramente identificabili:

- **L'interprete delle schede di controllo**, responsabile della lettura e dell'esecuzione delle istruzioni contenute sulle schede stesse.
- **Il caricatore**, invocato dall'interprete delle schede di controllo per caricare nella memoria opportuni programmi di sistema e programmi d'applicazione.
- **I driver dei dispositivi**, usati sia dall'interprete delle schede di controllo sia dal caricatore per l'esecuzione delle operazioni di I/O. Spesso i programmi di sistema e quelli d'applicazione si servivano di questi stessi driver di dispositivi, che garantivano un funzionamento corretto e un risparmio di spazio nella memoria e di tempo di programmazione.

Questi sistemi a lotti lavoravano abbastanza bene. Il monitor residente garantiva la sequenzializzazione automatica dei lavori, così com'era indicata dalle schede di controllo. Quando una scheda di controllo indicava che si doveva eseguire un programma, il monitor caricava nella memoria il programma e gli trasferiva il controllo. Terminato il pro-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

gramma, il controllo tornava al monitor, il quale leggeva la scheda di controllo successiva, caricava il nuovo programma e così via. Si ripeteva il ciclo fino a che tutte le schede di controllo del lavoro in corso erano state interpretate. Quindi, il monitor continuava automaticamente a lavorare, passando al lavoro successivo.

Il passaggio ai sistemi a lotti con sequenzializzatore automatico dei lavori si è verificato quando ormai era diventato indispensabile migliorare il rendimento. Il problema, abbastanza semplice, consisteva nella lentezza dell'intervento umano; che veniva pertanto sostituito da programmi di un sistema operativo: il sequenzializzatore automatico dei lavori, infatti, eliminava la perdita di tempo derivante dalla preparazione e dalla sequenzializzazione dei lavori da parte dell'operatore.

Com'è evidenziato nel Paragrafo 1.2.1, anche con questo sistema, la CPU spesso rimaneva inattiva. Il problema a questo punto riguardava i dispositivi meccanici di I/O, che sono intrinsecamente più lenti dei dispositivi elettronici. I tempi caratteristici di una CPU, anche se lenta, sono misurabili in microsecondi. Quindi, mentre questa eseguiva migliaia di istruzioni al secondo, un lettore di schede veloce, invece, poteva leggere 1200 schede al minuto, vale a dire 20 al secondo. Di conseguenza, la differenza di velocità tra la CPU e i suoi dispositivi di I/O poteva essere di tre o più ordini di grandezza. Col tempo, i perfezionamenti tecnologici hanno portato allo sviluppo di dispositivi di I/O più veloci, ma contemporaneamente era aumentata anche la velocità delle CPU, quindi il problema non solo non si era risolto, ma si era aggravato.

Una soluzione diffusa prevedeva la sostituzione dei lettori di schede e delle stampanti con unità a nastro magnetico. Alla fine degli anni Cinquanta e all'inizio degli anni Sessanta la maggior parte dei sistemi di calcolo era costituita da sistemi a lotti che utilizzavano come dispositivi di lettura i lettori di schede e come dispositivi di scrittura le stampanti o i perforatori di schede. Tuttavia, anziché far leggere le schede direttamente dalla CPU, si eseguiva un primo passaggio di copiatura delle schede in nastro magnetico. Quindi, quando un nastro era sufficientemente pieno, lo si rimuoveva e lo si portava al calcolatore. Quando una scheda veniva richiesta da un programma, si leggeva dal nastro il settore corrispondente. Analogamente, i risultati dell'elaborazione venivano inviati al nastro e il contenuto del nastro stesso veniva passato solo in una fase successiva alla stampante. I lettori di schede e le stampanti anziché dal calcolatore centrale erano gestiti fuori linea (Figura 22.3).

Il vantaggio principale dato da questo metodo consisteva nel fatto che il calcolatore principale non dipendeva più dalla velocità dei lettori di schede e delle stampanti, ma da quella delle unità a nastro magnetico, che, comunque, erano molto più veloci dei dispositivi di I/O precedenti. L'uso del nastro magnetico come I/O poteva essere applicato con qualsiasi sistema di registrazione, si trattasse di lettori di schede, perforatori, nastri o stampanti.

Il guadagno effettivo ottenibile derivava dalla possibilità di impiegare per una stessa CPU più sistemi lettore-nastro e nastro-stampante. Se la CPU è in grado di elaborare i dati in ingresso con una velocità doppia rispetto a quella con cui il lettore legge le schede, allora due lettori contemporaneamente attivi possono produrre una quantità di nastro sufficiente per tenere costantemente occupata la CPU. D'altra parte, ora il ritardo

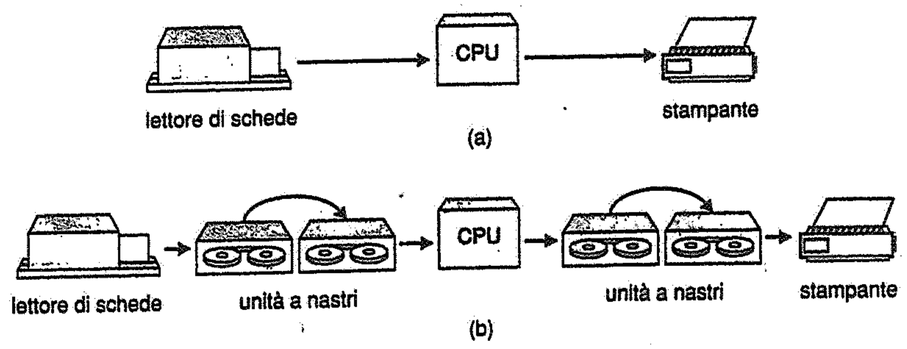


Figura 22.3 Funzionamento dei dispositivi di I/O: (a) in linea; (b) fuori linea.

maggiore si riscontra nell'esecuzione di un particolare lavoro d'elaborazione; prima occorre eseguirne la lettura nel nastro, poi bisogna aspettare che il nastro sia riempito completamente con altri lavori. Quando il nastro è pieno deve essere riavvolto, rimosso, portato manualmente al calcolatore e infine montato su un'unità a nastri libera. Naturalmente, tutto questo è ragionevole per i sistemi a lotti. Molti lavori simili si possono raggruppare in lotti in un nastro, prima di portare quest'ultimo al calcolatore.

Per qualche tempo i lavori continuarono a essere preparati con le procedure fuori linea, ma ben presto nella maggior parte dei sistemi queste procedure furono sostituite; la maggior diffusione di sistemi con unità disco permise di perfezionare le operazioni fuori linea. Il problema nelle unità a nastro era dovuto al fatto che il lettore di schede non poteva scrivere a un'estremità del nastro mentre la CPU leggeva dall'altra estremità. Prima di poter essere riavvolto e letto, il nastro doveva essere completamente scritto, giacché si tratta di un **dispositivo ad accesso sequenziale**. I sistemi a disco hanno eliminato questo problema. La testina si sposta da una zona all'altra del disco, e può passare rapidamente dalla zona usata dal lettore per memorizzare il contenuto di nuove schede alla posizione richiesta dalla CPU per leggere la scheda successiva, caratterizzandosi quindi come **dispositivo ad accesso diretto**.

In un sistema a dischi le schede vengono trasferite direttamente dal lettore al disco, e la posizione delle schede viene registrata in una tabella del sistema operativo. Durante l'esecuzione di un lavoro d'elaborazione, il sistema operativo soddisfa le richieste di lettura delle schede leggendo dal disco i dati corrispondenti alle schede richieste. Analogamente, quando il lavoro richiede alla stampante di stampare una riga, quella riga viene copiata nella memoria del sistema e scritta nel disco. I risultati vengono effettivamente stampati quando il lavoro è completato. Questo tipo di gestione asincrona dell'elaborazione e dell'I/O (Figura 22.4) — il cosiddetto *spooling* (*simultaneous peripheral operation on line*) — adopera fondamentalmente il disco come un enorme mezzo di memorizzazione transitoria, per leggere in anticipo il maggior numero possibile di dati dai dispositivi d'ingresso e per memorizzare i dati in uscita fino a che i dispositivi di emissione siano in grado di accettarli.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

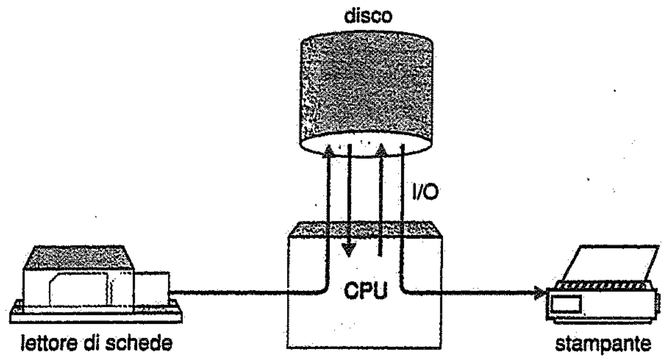


Figura 22.4 Gestione asincrona dell'elaborazione e dell'I/O (*spooling*).

La gestione asincrona dell'elaborazione e dell'I/O si usa anche per elaborare dati in postazioni remote. La CPU invia i dati a una stampante remota su linee di comunicazione, oppure accetta in ingresso un intero lavoro proveniente da un lettore di schede remoto. Le operazioni remote si compiono alla loro propria velocità, senza l'intervento della CPU; la CPU deve semplicemente essere informata del completamento delle operazioni in modo da intervenire sul lotto successivo di dati.

Con la gestione asincrona dell'elaborazione e dell'I/O si sovrappone l'I/O di un lavoro con la computazione di altri lavori; anche in un sistema semplice è infatti possibile leggere i dati in ingresso di un lavoro mentre si stampano i risultati di un altro lavoro, contemporaneamente si può eseguire un terzo lavoro (o più lavori), leggendo le sue 'schede' dal disco e 'stampando' i risultati ancora nel disco.

La gestione asincrona dell'elaborazione e dell'I/O ha un effetto positivo sulle prestazioni del sistema: impiegando poco spazio del disco e alcune tabelle, la computazione di un lavoro può essere sovrapposta all'I/O di altri lavori; quindi permette alla CPU e ai dispositivi di I/O di lavorare a velocità maggiori.

La gestione asincrona dell'elaborazione e dell'I/O conduce in modo naturale alla multiprogrammazione, che è il fondamento di tutti i moderni sistemi operativi.

22.2 Atlas

Il sistema operativo Atlas [Kilburn et al. 1961], [Howarth et al. 1961] è stato progettato alla University of Manchester in Inghilterra fra la fine degli anni Cinquanta e l'inizio degli anni Sessanta. Molte sue caratteristiche di base, che a quel tempo erano originali, sono diventate componenti standard dei sistemi operativi moderni. I driver dei dispositivi costituivano la parte principale del sistema. Inoltre, le chiamate del sistema erano state aggiunte attraverso un insieme di istruzioni speciali, chiamate *extra codes*.

L'Atlas era un sistema operativo a lotti con gestione asincrona dell'elaborazione e dell'I/O. Ciò permetteva al sistema di eseguire lo scheduling dei lavori d'elaborazione secondo la disponibilità dei dispositivi periferici, come unità a nastri magnetici, lettori di nastri di carta, perforatori di nastri di carta, stampanti, lettori di schede o perforatori di schede.

La caratteristica più notevole di questo sistema operativo era la sua gestione della memoria. A quel tempo la **memoria a nuclei magnetici** era ancora nuova e costosa. In molti calcolatori, come l'IBM 650, si usava un tamburo per la memoria centrale. Anche nel sistema Atlas si usava un tamburo per la memoria centrale, ma come cache per il tamburo si adoperava una piccola quantità di memoria a nuclei magnetici. Per trasferire automaticamente le informazioni tra la memoria a nuclei magnetici e il tamburo si usava la paginazione su richiesta.

Il sistema operativo Atlas era eseguito su un calcolatore British con parole di 48 bit. Gli indirizzi erano di 24 bit, ma erano codificati in decimali, il che permetteva di accedere solo a un milione di parole. A quel tempo, comunque, era uno spazio d'indirizzi estremamente grande. La memoria fisica era costituita di un tamburo di 98 K parole e di 16 K parole di memoria a nuclei. La memoria era divisa in pagine di 512 parole, quindi forniva 32 pagine fisiche di memoria. Una memoria associativa di 32 registri realizzava l'associazione dagli indirizzi virtuali agli indirizzi fisici.

Se si verificava un'assenza di pagina, si richiamava un algoritmo di sostituzione delle pagine. Si teneva sempre vuota una pagina fisica della memoria, perciò un trasferimento dal tamburo si poteva avviare immediatamente. L'algoritmo di sostituzione delle pagine tentava di predire il comportamento futuro dell'accesso alla memoria, basandosi sul comportamento passato. S'impostava un bit di riferimento per ciascuna pagina fisica a ogni accesso a tale pagina. I bit di riferimento venivano letti nella memoria ogni 1024 istruzioni. Gli ultimi 32 valori di questi bit si usavano per definire il tempo trascorso dal più recente riferimento (t_1) e l'intervallo di tempo trascorso tra gli ultimi due riferimenti (t_2). Si sceglievano le pagine per la sostituzione nel seguente ordine:

1. qualsiasi pagina con $t_1 > t_2$, si ritiene che questa pagina non sia più in uso;
2. se $t_1 \leq t_2$ per tutte le pagine, allora si sostituisce la pagina dove la differenza $t_2 - t_1$ è maggiore.

L'algoritmo di sostituzione delle pagine presuppone che i programmi accedano alla memoria in modo ciclico. Se il tempo trascorso tra gli ultimi due riferimenti è t_2, il riferimento successivo è atteso dopo t_2 unità di tempo. Se non si verifica alcun riferimento ($t_1 > t_2$), si suppone che la pagina non sia più utilizzata e quindi viene sostituita. Se tutte le pagine sono ancora in uso, si sostituisce la pagina che non sarà più richiesta per il periodo di tempo più lungo. Si suppone che il riferimento successivo avvenga entro $t_2 - t_1$ unità di tempo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

22.3 XDS-940

Il sistema operativo XDS-940 [Lichtenberger e Pirtle 1965] è stato progettato alla University of California di Berkeley. Come il sistema Atlas, impiegava la paginazione per la gestione della memoria; ma diversamente dall'Atlas l'XDS-940 era un sistema a partizione del tempo.

La paginazione si usava solo per la rilocazione e non per la paginazione su richiesta. La memoria virtuale di ogni processo d'utente era di sole 16 K parole, mentre la memoria fisica conteneva 64 K parole, e le pagine erano ciascuna di 2 K parole. La tabella delle pagine era conservata in registri. Poiché la memoria fisica era più grande della memoria virtuale di un processo d'utente, parecchi processi potevano essere contemporaneamente nella memoria. Il numero degli utenti poteva essere aumentato condividendo le pagine, quando queste ultime contenevano codice rientrante di sola lettura. I processi erano conservati in un tamburo e venivano caricati nella memoria e da essa scaricati secondo le necessità.

Il sistema XDS-940 fu costruito a partire da un XDS-930 modificato. Le modifiche erano rappresentative di quelle apportate a un calcolatore di base per permettere di scrivere adeguatamente un sistema operativo: fu aggiunto il duplice modo di funzionamento (di sistema e d'utente); alcune istruzioni, come quelle di I/O e d'arresto furono definite come privilegiate. Il tentativo di eseguire un'istruzione privilegiata nel modo d'utente avrebbe causato l'emissione di un segnale di eccezione al sistema operativo.

All'insieme di istruzioni da eseguire nel modo d'utente fu aggiunta un'istruzione di chiamata del sistema. Quest'istruzione si usava per creare nuove risorse, come file, permettendo al sistema operativo di gestire le risorse fisiche. I file, ad esempio, si registravano nel tamburo in blocchi di 256 parole. Per gestire i blocchi di tamburo liberi si usava una mappa di bit. Ogni file aveva un blocco indice con puntatore ai blocchi di dati effettivi. I blocchi indice erano concatenati tra loro.

Il sistema XDS-940 forniva anche chiamate del sistema per permettere ai processi di creare, avviare, sospendere ed eliminare sottoprocessi. Un utente programmatore poteva costruire un sistema di processi. Processi separati potevano condividere la memoria per scopi di comunicazione e sincronizzazione. La creazione di processi definiva una struttura ad albero, dove un processo costituiva la radice e i suoi sottoprocessi i nodi sottostanti nell'albero. Ognuno dei sottoprocessi poteva, a sua volta, creare altri sottoprocessi.

22.4 THE

Il sistema operativo THE ([Dijkstra 1968], [McKeag e Wilson 1976]) è stato progettato alla Technische Hogeschool di Eindhoven in Olanda. Si tratta di un sistema a lotti che funzionava su un calcolatore olandese, l'EL X8, con 32 K parole di 27 bit. Il sistema si fece notare soprattutto per il suo progetto pulito, in particolare per la sua struttura a strati e il suo uso di un insieme di processi concorrenti sincronizzati tramite semafori.

Tuttavia, diversamente dal sistema XDS-940, l'insieme dei processi del sistema THE era statico. Il sistema operativo stesso era stato progettato come un insieme di processi cooperanti. Inoltre, erano stati creati cinque processi utenti, i quali servivano come agenti attivi per la compilazione, l'esecuzione e la stampa di programmi utenti. Una volta terminato un lavoro, il processo ritornava alla coda d'ingresso per selezionare un altro lavoro.

Si usava un algoritmo di scheduling della CPU con priorità. Le priorità si ricalcolavano ogni 2 secondi ed erano inversamente proporzionali al tempo di CPU usato recentemente, vale a dire negli ultimi 8-10 secondi. Questo schema offriva una priorità più alta ai processi con prevalenza di I/O e ai processi nuovi.

La gestione della memoria era limitata dall'assenza di adeguate funzioni dell'architettura fisica sottostante. Tuttavia, poiché il sistema era limitato e i programmi utenti si potevano scrivere solo in Algol, s'impiegava uno schema di paginazione gestito dal sistema operativo. Il compilatore Algol generava automaticamente chiamate alle procedure di sistema, le quali assicuravano che le informazioni richieste si trovassero nella memoria, liberandone delle parti, se fosse stato necessario, in modo da consentire l'avvicendamento. La memoria ausiliaria era un tamburo di 512 K parole. Si adoperavano pagine di 512 parole, con un criterio di sostituzione delle pagine LRU.

Un altro punto importante del sistema THE era il controllo delle situazioni di stallo. Per evitare le situazioni di stallo si ricorreva all'algoritmo del banchiere.

Il sistema Venus [Liskov 1972] è strettamente correlato al sistema THE. Anche il progetto del sistema Venus era basato su una struttura a strati, che impiegava semafori per sincronizzare i processi. I livelli inferiori del progetto, però, erano realizzati in microcodice, sicché il sistema era molto più veloce. Per la gestione della memoria si usava uno schema di memoria paginata e segmentata; inoltre il sistema è stato progettato come sistema a partizione del tempo, anziché come sistema a lotti.

22.5 RC 4000

Il sistema RC 4000, come il sistema THE, si fece notare soprattutto per i concetti applicati nella progettazione. È stato progettato per il calcolatore danese RC 4000 dalla Regnecentralen, e in particolare da P. Brinch Hansen ([Brinch Hansen 1970], [Brinch Hansen 1973]). L'obiettivo non era quello di progettare un sistema a lotti, oppure un sistema a partizione del tempo, o qualsiasi altro sistema specifico, ma quello di creare il nucleo di un sistema operativo, sul quale fosse possibile costruire un sistema operativo completo. Così, la struttura del sistema fu concepita a strati, e furono forniti solo i livelli inferiori, cioè il nucleo.

Il nucleo gestiva un insieme di processi concorrenti. Lo scheduler della CPU seguiva un criterio RR. Anche se i processi potevano condividere la memoria, il meccanismo principale di comunicazione e sincronizzazione era il sistema di messaggi fornito dal nucleo. I processi potevano comunicare tra loro scambiandosi messaggi di dimensione fis-

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sa, 8 parole di lunghezza. Tutti i messaggi si memorizzavano in apposite sezioni della memoria (*buffer*) che facevano parte di un gruppo comune di sezioni. Quando una di queste sezioni di memoria per i messaggi non era più necessaria, veniva restituita al gruppo comune.

A ogni processo era associata una coda di messaggi, che conteneva tutti i messaggi inviati a quel processo che non erano ancora stati ricevuti. I messaggi venivano rimossi dalla coda in ordine FIFO. Il sistema disponeva di quattro operazioni primitive, che erano eseguite in modo atomico:

- `send-message(in ricevente, in messaggio, out buffer);`
- `wait-message(out mittente, out messaggio, out buffer);`
- `send-answer(out risultato, in messaggio, in buffer);`
- `wait-answer(out risultato, out messaggio, in buffer).`

Le ultime due operazioni permettevano ai processi di scambiarsi più messaggi alla volta.

Queste primitive richiedevano che un processo servisse la sua coda di messaggi in ordine FIFO, e che si bloccasse mentre altri processi stavano gestendo i suoi messaggi. Per rimuovere tali limitazioni, i progettisti fornirono altre due primitive di comunicazione, che permettevano a un processo di attendere l'arrivo del messaggio successivo oppure di rispondere e servire la sua coda in qualunque ordine:

- `wait-event(in buffer_precedente, out buffer_successivo, out risultato);`
- `get-event(out buffer).`

Anche i dispositivi di I/O erano trattati come processi. I driver dei dispositivi erano composti di codice che convertiva i segnali d'interruzione e i registri dei dispositivi in messaggi; così un processo scriveva in un terminale inviandogli un messaggio. Il driver del dispositivo riceveva il messaggio ed emetteva il carattere al terminale. L'immissione di un carattere interrompeva il sistema e determinava l'attivazione del driver del dispositivo, il quale a sua volta creava un messaggio contenente il carattere immesso e lo inviava al processo in attesa.

22.6 CTSS

Il sistema CTSS (*compatible time-sharing system*), [Corbato et al. 1962], è stato progettato al MIT come sistema sperimentale a partizione del tempo, e realizzato su un calcolatore IBM 7090. Gestiva fino a 32 utenti interattivi, i quali disponevano di un insieme di comandi interattivi, che permetteva loro di manipolare i file e di compilare ed eseguire programmi interagendo col sistema tramite un terminale.

L'IBM 7090 aveva una memoria di 32 K parole di 36 bit. Il monitor impiegava 5 K parole, lasciando le altre 27 K parole agli utenti. Le immagini della memoria d'utente

s'avvicendavano tra la memoria e un tamburo rapido. Per lo scheduling della CPU s'impiegava un algoritmo a code multiple con retroazione. Il quanto di tempo per il livello i era di $2 \times i$ unità di tempo. Se un programma non terminava la propria sequenza di operazioni di CPU entro un quanto di tempo, veniva abbassato al livello successivo della coda, e otteneva un quanto di tempo doppio. Il programma che si trovava al livello massimo, con il quanto più corto, veniva eseguito per primo. Il livello iniziale di un programma era stabilito dalla sua dimensione in modo che il quanto di tempo fosse lungo almeno come il tempo d'avvicendamento.

Il CTSS ha avuto un notevole successo ed era in uso fino al 1972. Benché limitato, esso è riuscito a dimostrare che la partizione del tempo era un modo d'elaborazione conveniente e pratico. Un risultato ottenuto dal CTSS è stato quello di favorire lo sviluppo dei sistemi a partizione del tempo, tra i quali quello del MULTICS.

22.7 MULTICS

Il sistema operativo MULTICS è stato progettato al MIT ([Corbato e Vyssotsky 1965], [Organick 1972]) come un'estensione naturale del CTSS. Il CTSS e altri tra i primi sistemi a partizione del tempo hanno avuto un tale successo che è stato subito sentito il desiderio di procedere rapidamente verso sistemi più grandi e perfezionati. Quando sono diventati disponibili calcolatori più grandi, i progettisti del CTSS hanno avviato lo sviluppo di uno strumento d'elaborazione a partizione del tempo. Il servizio di calcolo era fornito come l'energia elettrica; si potevano collegare, attraverso linee telefoniche, grandi sistemi di calcolo a terminali di uffici e case di tutta una città. Il sistema operativo era un sistema a partizione del tempo continuamente in funzione con un vasto file system di programmi e dati condivisi.

Il MULTICS è stato progettato da un gruppo proveniente dal MIT, il GE (che più tardi ha venduto il proprio dipartimento di informatica alla Honeywell), e dai Bell Laboratories (che uscirono dal progetto nel 1969). L'architettura del calcolatore GE 635 di base fu modificata in quella di un nuovo calcolatore chiamato GE 645, soprattutto per consentire la segmentazione paginata della memoria.

Un indirizzo virtuale era composto di un numero di segmento di 18 bit e di uno scostamento di una parola di 16 bit. I segmenti venivano quindi suddivisi in pagine di 1 K parole, per le quali si usava l'algoritmo di sostituzione con seconda chance.

Lo spazio d'indirizzi virtuali segmentato è stato unito al file system; ogni segmento era un file e ai segmenti si faceva riferimento tramite il nome del file corrispondente. Lo stesso file system era una struttura ad albero a più livelli, che permetteva agli utenti di creare le proprie strutture di directory. Come il CTSS, il MULTICS impiegava uno scheduling della CPU a code multiple con retroazione. La protezione era garantita da una lista di controllo degli accessi associata a ogni file e da un insieme di anelli di protezione per i processi in esecuzione. Il sistema, che è stato scritto quasi completamente in PL/1, comprendeva circa 300.000 righe di codice. È stato esteso a un sistema dotato di più unità d'elaborazione che permetteva di sospendere dal servizio una CPU per motivi di manutenzione mentre il sistema continuava la sua esecuzione.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

22.8 OS/360

La più lunga linea di sviluppo dei sistemi operativi è indubbiamente quella dei calcolatori IBM. I primi calcolatori IBM, come l'IBM 7090 e l'IBM 7094, sono i primi esempi dello sviluppo delle procedure di I/O comuni, seguite da un monitor residente, istruzioni privilegiate, protezione della memoria ed elaborazione a lotti semplice. Questi sistemi sono stati sviluppati separatamente, spesso da ogni sito in modo indipendente. Il risultato è stato che l'IBM si è trovata di fronte a molti calcolatori diversi, con linguaggi diversi e programmi di sistema diversi.

L'IBM/360 fu progettato per modificare tale situazione. Il suo progetto prevedeva una famiglia di calcolatori che si estendeva su una gamma completa che andava da piccole macchine commerciali fino a grandi macchine scientifiche. Per questi sistemi sarebbe stato necessario un solo insieme di programmi; tutti questi sistemi impiegavano lo stesso sistema operativo: l'OS/360, [Mealy et al. 1966]. Quest'orientamento doveva ridurre i problemi di manutenzione dell'IBM, e permettere agli utenti di spostare liberamente programmi e applicazioni tra i sistemi IBM.

Sfortunatamente con l'OS/360 s'è tentato di fare tutto per tutti, col risultato di non riuscire a svolgere particolarmente bene nessun compito. Il file system comprendeva un campo di tipi che definiva il tipo di ciascun file, ed erano definiti diversi tipi di file per elementi (*record*) a lunghezza fissa o a lunghezza variabile, così come per i file a blocchi o non a blocchi. Si usava l'allocazione contigua, quindi l'utente doveva predire la dimensione di ogni file impiegato per contenere i risultati delle elaborazioni. Il linguaggio Job Control Language (JCL) prevedeva parametri per ogni possibile opzione, diventando incomprensibile all'utente medio.

Le procedure di gestione della memoria erano ostacolate dall'architettura. Sebbene si adoperasse un modo d'indirizzamento con registro di base, il programma poteva accedere al registro base e modificarlo, perciò la CPU generava indirizzi assoluti. Questa situazione ha impedito la rilocazione dinamica; l'associazione degli indirizzi del programma agli indirizzi della memoria fisica si eseguiva nella fase di caricamento. Di questo sistema operativo sono state prodotte due versioni: l'OS/MFT impiegava regioni fisse, mentre l'OS/MVT impiegava regioni variabili.

Il sistema è stato scritto in linguaggio assemblativo da migliaia di programmatori, quindi le righe del codice risultante erano milioni. Lo stesso sistema operativo richiedeva grandi quantità di memoria per il proprio codice e le proprie tabelle. L'esecuzione del solo sistema operativo spesso richiedeva metà dei cicli totali della CPU. Con gli anni sono uscite nuove versioni mediante le quali sono state aggiunte nuove caratteristiche e sono stati corretti errori. Tuttavia, la correzione di un errore spesso ne causava un altro in una parte remota del sistema, perciò il numero degli errori noti del sistema era pressoché costante.

Col passaggio all'architettura dell'IBM 370, all'OS/360 è stata aggiunta la memoria virtuale. L'architettura sottostante forniva una memoria virtuale con segmentazione paginata. Le nuove versioni del sistema operativo impiegavano quest'architettura in modi diversi. L'OS/VS1 creava un grande spazio d'indirizzi virtuali e in quella memoria virtuale eseguiva l'OS/MFT. Quindi lo stesso sistema operativo era paginato, così come i programmi utenti. L'OS/VS2 Release 1 eseguiva l'OS/MVT nella memoria virtuale. Infine, l'OS/VS2 Release 2, che ora si chiama MVS, forniva a ogni utente la sua memoria virtuale.

L'MVS è ancora fondamentalmente un sistema operativo a lotti. Il sistema CTSS veniva eseguito su un IBM 7094, ma il MIT decise che lo spazio d'indirizzi del 360, il successore IBM del 7094, era troppo piccolo per il MULTICS, perciò cambiò fornitore. L'IBM decise allora di creare il proprio sistema a partizione del tempo, il TSS/360, [Lett e Konigsford 1968]. Come il MULTICS, il TSS/360 era considerato come uno strumento d'elaborazione a partizione del tempo. L'architettura di base del 360 fu modificata nel modello 67, per fornire la memoria virtuale. Parecchi enti acquistarono il 360/67 in vista del TSS/360.

Il TSS/360 subì però alcuni ritardi, perciò furono sviluppati altri sistemi a partizione del tempo che servivano come sistemi temporanei finché non fosse stato disponibile il TSS/360. All'OS/360 è stata aggiunta un'opzione per disporre della partizione del tempo (TSO). Il Cambridge Scientific Center dell'IBM ha sviluppato il CMS come sistema per utente singolo e il CP/67 per fornire una macchina virtuale su cui farlo funzionare, [Meyer e Seawright 1970], [Parmelee et al. 1972].

Quando finalmente fu disponibile il TSS/360, si dimostrò un fallimento. Era troppo grande e troppo lento. Nessun ente cambiò il proprio sistema temporaneo con il TSS/360. Oggi, la partizione del tempo su sistemi IBM si trova diffusamente con l'opzione TSO sul sistema MVS o anche col CMS sul CP/67 (ribattezzato VM).

È lecito domandarsi quali fossero i problemi principali del TSS/360 e del MULTICS. Una parte dei problemi consisteva nel fatto che, pur trattandosi di sistemi progrediti, erano troppo grandi e troppo complessi per essere capiti. Un altro problema derivava dal presupposto che un calcolatore grande e remoto avrebbe reso disponibile una certa potenza di calcolo grazie alla partizione del tempo. Ora sembra che la maggior parte delle attività d'elaborazione sia svolta da piccoli calcolatori individuali, i PC, e non da grandi sistemi remoti a partizione del tempo che tentano di fare tutto per tutti.

22.9 Mach

Le origini del sistema Mach risalgono al sistema operativo Accent sviluppato alla Carnegie Mellon University (CMU), [Rashid e Robertson 1981]. Il sistema e la filosofia di comunicazione del Mach sono derivati dal sistema Accent, sebbene molte altre parti significative del sistema (ad esempio, il sistema di memoria virtuale, la gestione di task e thread) furono sviluppate *ex novo* ([Rashid 1986], [Tevanian et al. 1989] e [Accetta et al. 1986]). Lo scheduler è descritto nei dettagli in [Tevanian et al. 1987a] e [Black 1990]. Una prima versione della memoria condivisa e del sistema di associazione della memoria è presentata in [Tevanian et al. 1987b].

http://www.adulpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Il sistema operativo Mach è stato progettato tenendo presenti i seguenti tre obiettivi critici:

1. emulare lo UNIX 4.3BSD in modo che i file eseguibili per il sistema UNIX si potessero eseguire anche nel sistema Mach;
2. essere un sistema operativo moderno che gestisse molti modelli di memoria, e l'elaborazione parallela e distribuita;
3. avere un nucleo che fosse più semplice e più facile da modificare di quello del 4.3BSD.

Lo sviluppo del sistema Mach ha seguito un percorso che parte dal sistema BSD UNIX: Il codice fu sviluppato inizialmente all'interno del nucleo del 4.2BSD, sostituendo i componenti BSD con i componenti Mach appena questi venivano completati; i componenti BSD furono aggiornati alla versione 4.3BSD non appena questa divenne disponibile. Nel 1986 i sottosistemi di memoria virtuale e di comunicazione erano funzionanti per la famiglia di calcolatori DEC VAX, comprendente versioni dotate di più unità d'elaborazione. Dopo breve tempo seguirono le versioni per le stazioni di lavoro IBM RT/PC e SUN 3. Nel 1987 furono completate sia le versioni Encore Multimax e Sequent Balance con più unità d'elaborazione, comprendenti la gestione di task e thread, sia le prime versioni ufficiali, Versione 0 e Versione 1.

Dalla Versione 2 il sistema Mach è compatibile con i corrispondenti sistemi BSD, poiché comprende nel nucleo gran parte dello stesso codice BSD. Le nuove caratteristiche e funzioni del sistema Mach fanno sì che il nucleo di queste versioni sia più grande dei corrispondenti nuclei BSD. Il Mach 3 trasferisce il codice BSD all'esterno del nucleo, lasciando un micronucleo assai più piccolo. Questo sistema incorpora nel nucleo unicamente le funzioni essenziali; tutto il codice specifico dello UNIX è stato estratto per essere eseguito come server nel modo d'utente. L'esclusione dal nucleo del codice specifico dello UNIX consente di sostituire il BSD con un altro sistema operativo o di eseguire simultaneamente sopra il micronucleo più interfacce di sistemi operativi. Tali interfacce, oltre che per il BSD sono state sviluppate per l'MS-DOS, per il sistema operativo Macintosh e per l'OSF/1. Questo metodo ha delle similitudini col concetto di macchina virtuale, anche se in questo caso la macchina virtuale è definita da un livello logico (l'interfaccia del nucleo del Mach), anziché da una macchina fisica. Per quel che riguarda la versione 3.0, il sistema Mach diviene disponibile per molti sistemi diversi, tra cui i calcolatori con singola CPU della SUN Microsystems, Intel, IBM e DEC e i sistemi con più unità d'elaborazione DEC, Sequent e Encore.

Il sistema Mach è stato spinto all'attenzione dell'industria quando, nel 1989, la Open Software Foundation (OSF) ha annunciato che intendeva impiegarlo come base per il suo nuovo sistema operativo, l'OSF/1. L'edizione iniziale dell'OSF/1 apparve un anno dopo per competere con lo UNIX System V, versione 4, il sistema operativo scelto dai membri della UNIX International (UI). L'OSF vanta tra i suoi membri società tecnologicamente importanti come la IBM, DEC e HP. L'OSF ha nel frattempo cambiato orientamento e solo lo UNIX DEC è basato sul nucleo del Mach.

Tra l'altro il Mach 2.5 è anche alla base del sistema operativo della stazione di lavoro NeXT, nata dalla mente di Steve Jobs, noto per essere stato uno dei fondatori della Apple Computer.

Diversamente dallo UNIX, che era stato sviluppato senza considerare la multielaborazione, il sistema operativo Mach offre un completa gestione della multielaborazione. Si tratta di una gestione molto flessibile, che varia dai sistemi con memoria condivisa tra le unità d'elaborazione ai sistemi senza memoria condivisa. Il sistema Mach usa i processi leggeri, nella forma di thread d'esecuzione multipli all'interno di un task (o spazio d'indirizzi), per gestire la multielaborazione e l'elaborazione parallela. Il suo esteso uso dei messaggi come unico metodo di comunicazione garantisce che i meccanismi di protezione siano completi ed efficienti. Integrando i messaggi col sistema della memoria virtuale, il sistema Mach assicura che i messaggi siano gestiti in maniera efficiente. Infine, poiché il sistema di gestione della memoria virtuale impiega i messaggi per comunicare con i processi che gestiscono la memoria ausiliaria, il sistema operativo Mach consente una grande flessibilità nella progettazione e nella realizzazione dei task di gestione degli oggetti di memoria. Offrendo chiamate del sistema di basso livello, o primitive, con le quali si possono costruire funzioni complesse, il sistema Mach riduce le dimensioni del nucleo permettendo l'emulazione dei sistemi operativi al livello d'utente in modo molto simile ai sistemi a macchine virtuali della IBM.

L'edizione precedente di questo testo comprendeva un intero capitolo sul sistema operativo Mach. Tale capitolo è disponibile all'indirizzo `http://www.bell-labs.com/topic/books/os-book/Mach.ps`.

22.10 Altri sistemi

Naturalmente esistono altri sistemi operativi, la maggior parte dei quali ha caratteristiche interessanti. Il sistema operativo MCP, per la famiglia di calcolatori Burroughs, [McKeag e Wilson 1976], è stato il primo a essere scritto in un linguaggio di programmazione di sistema. Impiegava anche la segmentazione e più unità d'elaborazione. Anche il sistema operativo SCOPE per il CDC 6600, [McKeag e Wilson 1976], era un sistema con più unità d'elaborazione. Il coordinamento e la sincronizzazione dei processi multipli erano stati progettati sorprendentemente bene. Il Tenex, [Bobrow et al. 1972], è stato uno tra i primi sistemi con paginazione su richiesta per il PDP-10, e ha avuto una forte influenza su successivi sistemi partizione del tempo, come il TOPS-20 per il DEC-20. Il sistema operativo VMS per il VAX è basato sul sistema operativo RSX per il PDP-11. Il CP/M era il sistema operativo più diffuso per microcalcolatori a 8 bit; l'MS-DOS era il sistema più diffuso per microcalcolatori a 16 bit. Le interfacce grafiche, o GUI, sono ormai molto diffuse poiché rendono più semplice l'utilizzo dei calcolatori; i sistemi operativi Macintosh e Microsoft Windows sono i più noti in questo campo.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Bibliografia

[Abbot 1984] C. Abbot, "Intervention Schedules for Real-Time Programming", *IEEE Transactions on Software Engineering*, Volume SE-10, No. 3 (Maggio 1984), pagg. 268-274.

[Accetta et al. 1986] M. Accetta, R. Baron, W. Bolosky, D. B. Golub, R. Rashid, A. T. jr e M. Young, "Mach: A New Kernel Foundation for Unix Development", *Proceedings of the Summer 1986 USENIX Conference*, pagg. 93-112.

[Agrawal e Abbadi 1991] D. P. Agrawal e A. E. Abbadi, "An Efficient and Fault-Tolerant Solution of Distributed Mutual Exclusion", *ACM Transactions on Computer Systems*, Volume 9, No. 1 (Febbraio 1991), pagg. 1-20.

[Ahituv et al. 1987] N. Ahituv, Y. Lapid e S. Neumann, "Processing Encrypted Data", *Communications of the ACM*, Volume 30, No. 9 (1987), pagg. 777-780.

[Ahmed 2000] I. Ahmed, "Cluster Computing: A Glance at Recent Events", *IEEE Concurrency*, Volume 8, No. 1 (2000).

[Akl 1983] S. G. Akl, "Digital Signatures: A Tutorial Survey", *Computer*, Volume 16, No. 2 (Febbraio 1983), pagg. 15-24.

[Akyurek e Salem 1993] S. Akyurek e K. Salem, "Adaptive Block Rearrangement", *Proceedings of the International Conference on Data Engineering* (Aprile 1993), pagg. 182-189.

[Alt 1993] H. Alt, "Removable Media in Solaris", *Proceedings of the Winter 1993 USENIX Conference* (Gennaio 1993), pagg. 281-287.

[Anderson et al. 1989] T. E. Anderson, E. D. Lazowska e H. M. Levy, "The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors", *IEEE Transactions on Computers*, Volume 38, No. 12 (Dicembre 1989), pagg. 1631-1644.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Anderson et al. 1991] T. E. Anderson, B. N. Bershad, E. D. Lazowska e H. M. Levy, "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism", *Proceedings of the ACM Symposium on Operating Systems Principles* (Ottobre 1991), pagg. 95-109.

[Anyanwu e Marshall 1986] J. A. Anyanwu e L. F. Marshall, "A Crash Resistant UNIX File System", *Software—Practice and Experience*, Volume 16 (Febbraio 1986), pagg. 107-118.

[Apple 1987] *Apple Technical Introduction to the Macintosh Family*, Addison-Wesley, Reading, MA (1987).

[Apple 1991] *Inside Macintosh*, Volume VI, Addison-Wesley, Reading, MA (1991).

[Artsy 1989] Y. Artsy, "Designing a Process Migration Facility: The Charlotte Experience", *Computer*, Volume 22, No. 9 (Settembre 1989), pagg. 47-56.

[Asthana e Finkelstein 1995] P. Asthana e B. Finkelstein, "Superdense Optical Storage", *IEEE Spectrum*, Volume 32, No. 8 (Agosto 1995), pagg. 25-31.

[Axelsson 1999] S. Axelsson, "The Base-Rate Fallacy and Its Implications for Intrusion Detection" *Proceedings of the ACM Conference on Computer and Communications Security* (1999), pagg. 1-7.

[Babaoglu e Joy 1981] O. Babaoglu e W. Joy, "Converting A Swap-Based System to Do Paging in an Architecture Lacking Page-Referenced Bits", *Proceedings of the Eighth ACM Symposium on Operating System Principles* (Dicembre 1969), pagg. 78-86.

[Bach 1987] M. J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall (1987).

[Back et al. 2000] G. Back, P. Tullman, L. Stoller, W.C. Hsieh e J. Lepreau, "Techniques for the Design of Java Operating Systems", *Techniques for the Design of Java Operating Systems* (2000).

[Balkovich et al. 1985] E. Balkovich, S. R. Lerman e R. P. Parmelee, "Computing in Higher Education: The Athena Experience", *Communications of the ACM*, Volume 28, No.11 (Novembre 1985), pagg. 1214-1224.

[Bar 2000] M. Bar, *Linux Internals*, McGraw-Hill (2000).

[Barrera 1991] J. S. Barrera, "A Fast Mach Network IPC Implementation", *Proceedings of the USENIX Mach Symposium* (1991), pagg. 1-12.

[Bayer et al. 1978] R. Bayer, R. M. Graham e G. Seegmuller (Editors), *Operating Systems—An Advanced Course*, Springer Verlag, Berlin (1978).

[Beck et al. 1998] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus e D. Verworner, *Linux Kernel Internals, Second Edition*, Addison Wesley (1998).

[Belady 1966] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Systems Journal*, Volume 5, No. 2 (1966), pagg. 78-101.

[Belady et al. 1969] L. A. Belady, R. A. Nelson e G. S. Shedler, "An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine", *Communications of the ACM*, Volume 12, No. 6 (Giugno 1969), pagg. 349-353.

[Ben-Ari 1990] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, Prentice-Hall (1990).

[Benjamin 1990] C. D. Benjamin, "The Role of Optical Storage Technology for NASA", *Proceedings, Storage and Retrieval Systems and Applications* (Febbraio 1990), pagg. 10-17.

[Bernstein e Goodman 1980] P. A. Bernstein e N. Goodman, "Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems", *Proceedings of the International Conference on Very Large Data Bases* (1980), pagg. 285-300.

[Bernstein et al. 1987] A. Bernstein, V. Hadzilacos e N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).

[Bershad e Pinkerton 1988] B. N. Bershad e C. B. Pinkerton, "Watchdogs: Extending the Unix File System", *Proceedings of the Winter 1988 USENIX Conference* (Febbraio 1988).

[Bershad et al. 1990] B. N. Bershad, T. E. Anderson, E. D. Lazowska e H. M. Levy, "Lightweight Remote Procedure Call", *ACM Transactions on Computer Systems*, Volume 8, No. 1 (Febbraio 1990), pagg. 37-55.

[Beveridge e Wiener 1997] J. Beveridge e R. Wiener, *Multithreading Applications in in Win32*, Addison Wesley (1997).

[Birman e Joseph 1987] K. Birman e T. Joseph, "Reliable Cummunication in the Presence of Failures", *ACM Transactions on Computer Systems*, Volume 5, No. 1 (Febbraio 1987).

[Birrell e Nelson 1984] A. D. Birrell e B. J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, Volume 2, No. 1 (Febbraio 1984), pagg. 39-59.

[Black 1990] D. L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System", *IEEE Computer* (Maggio 1990), pagg. 35-43.

[Blair et al. 1985] G. S. Blair, J. R. Malone e J. A. Mariani, "A Critique of UNIX", *Software—Practice and Experience*, Volume 15, No. 6 (Dicembre 1985), pagg. 1125-1139.

[Bobrow et al. 1972] D. G. Bobrow, J. D. Burchfiel, D. L. Murphy e R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10", *Communications of the ACM*, Volume 15, No. 3 (Marzo 1972).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Bourne 1978] S. R. Bourne, "The UNIX Shell", *Bell System Technical Journal*, Volume 57, No. 6 (Luglio-Agosto 1978), pagg. 1971-1990.

[Bourne 1983] S. R. Bourne, *The UNIX System*, Addison-Wesley (1983); disponibile anche nella traduzione italiana a cura di Addison-Wesley Italia.

[Bovet e Cesati 2001] D. P. Bovet e M. Cesati, *Understanding the Linux Kernel*, O'Reilly & Associates (2001).

[Brain 1996] M. Brain, *Win32 System Services*, Second Edition, Prentice-Hall (1996).

[Brereton 1986] O. P. Brereton, "Management of Replicated Files in a UNIX Environment", *Software—Practice and Experience*, Volume 16 (Agosto 1986), pagg. 771-780.

[Brinch-Hansen 1970] P. Brinch-Hansen, "The Nucleus of a Multiprogramming System", *Communications of the ACM*, Volume 13, No. 4 (Aprile 1970), pagg. 238-241 e 250.

[Brinch-Hansen 1972] P. Brinch-Hansen, "Structured Multiprogramming", *Communications of the ACM*, Volume 15, No. 7 (Luglio 1972), pagg. 574-578.

[Brinch-Hansen 1973] P. Brinch-Hansen, *Operating System Principles*, Prentice-Hall (1973).

[Brownbridge et al. 1982] D. R. Brownbridge, L. F. Marshall e B. Randell, "The Newcastle Connection or UNIXes of the World Unite!" *Software—Practice and Experience*, Volume 12, No. 12 (Dicembre 1982), pagg. 1147-1162.

[Burns 1978] J. E. Burns, "Mutual Exclusion with Linear Waiting Using Binary Shared Variables", *SIGACT News*, Volume 10, No. 2 (1978), pagg. 42-47.

[Buyya 1999] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, Prentice Hall (1999).

[Callaghan 2000] B. Callaghan, *NFS Illustrated*, Addison-Wesley (2000).

[Carr e Hennessy 1981] W. R. Carr e J. L. Hennessy, "WSClock—A Simple and Effective Algorithm for Virtual Memory Management", *Proceedings on the ACM Symposium on Operating System Principles* (Dicembre 1981), pagg. 87-95.

[Carvalho e Roucariol 1983] O. S. Carvalho e G. Roucariol, "On Mutual Exclusion in Computer Networks", *Communications of the ACM* Volume 26, No. 2 (1983), pagg. 146-147.

[Cerf e Cain 1983] V. G. Cerf ed E. Cain, "The DoD Internet Architecture Model", *Computer Networks*, Volume 7, No. 5 (Ottobre 1983), pagg. 307-318.

[Chang 1980] E. Chang, "N-Philosophers: An Exercise in Distributed Control", *Computer Networks*, Volume 4, No. 2 (Aprile 1980), pagg. 71-76.

[Chang e Mergen 1988] A. Chang e M. F. Mergen, "801 Storage: Architecture and Programming", *ACM Transactions on Computer Systems*, Volume 6, No. 1 (Febbraio 1988), pagg. 28-50.

[Chen et al. 1994] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz e D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage", *ACM Computing Surveys*, Volume 26, No. 2 (Giugno 1994), pagg. 145-185.

[Cheriton e Zwaenepoel 1983] D. R. Cheriton e W. Z. Zwaenepoel, "The Distributed V Kernel and Its Performance for Diskless Workstations", *Proceedings of the ACM Symposium on Operating Systems Principles* (Ottobre 1983), pagg. 129-140.

[Cheung e Loong 1995] W. H. Cheung e A. H. S. Loong "Exploring Issues of Operating Systems Structuring: From Microkernel to Extensible Systems", *Operating Systems Review*, Volume 29 (1995), pagg. 4-16.

[Chi 1982] C. S. Chi, "Advances in Computer Mass Storage Technology", *Computer*, Volume 15, No. 5 (Maggio 1982), pagg. 60-74.

[Coffman e Denning 1973] E. G. Coffman e P. J. Denning, *Operating Systems Theory*, Prentice-Hall (1973).

[Coffman e Kleinrock 1968] E. C. Coffman e L. Kleinrock, "Feedback Queuing Models for Time-Shared Systems", *Journal of the ACM*, Volume 15, No. 4 (Ottobre 1968), pagg. 549-576.

[Coffman et al. 1971] E. G. Coffman, M. J. Elphick e A. Shoshani, "System Deadlocks", *Computing Surveys*, Volume 3, No. 2 (Giugno 1971), pagg. 67-78.

[Cohen e Jefferson 1975] E. S. Cohen e D. Jefferson, "Protection in the Hydra Operating System", *Proceedings of the Fifth Symposium on Operating System Principles* (Novembre 1975), pagg. 141-160.

[Comer 1999] D. Comer, *Internetworking with TCP/IP*, Volume II, Third Edition, Prentice-Hall (1999).

[Comer 2000] D. Comer, *Internetworking with TCP/IP*, Volume I, Fourth Edition, Prentice-Hall (2000); disponibile anche nella traduzione italiana a cura di Pearson Education Italia.

[Comer e Stevens 1991] D. Comer e D. L. Stevens, *Internetworking with TCP/IP*, Volume II, Prentice-Hall (1991).

[Comer e Stevens 1993] D. Comer e D. L. Stevens, *Internetworking with TCP/IP Principles*, Volume III, Prentice-Hall (1993).

[Corbato e Vyssotsky 1965] F. J. Corbato e V. A. Vyssotsky, "Introduction and Overview of the MULTICS System", *Proceedings of the AFIPS Fall Joint Computer Conference* (1965), pagg. 185-196.

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Corbato et al. 1962] F. J. Corbato, M. Merwin-Daggett e R. C. Daley, "An Experimental Time-Sharing System", *Proceedings of the AFIPS Fall Joint Computer Conference* (1962), pagg. 335-344.

[Courtois et al. 1971] P. J. Courtois, F. Heymans e D. L. Parnas, "Concurrent Control with 'Readers' and 'Writers'", *Communications of the ACM*, Volume 14, No. 10 (Ottobre 1971), pagg. 667-668.

[Custer 1994] H. Custer, *Inside the Windows NT File System*, Microsoft Press, (1994).

[Davcev e Burkhard 1985] D. Davcev e W. A. Burkhard, "Consistency and Recovery Control for Replicated Files", *Proceedings of the ACM Symposium on Operating Systems Principles*, Volume 19, Number 5 (Dicembre 1985), pagg. 87-96.

[Davies 1983] D. W. Davies, "Applying the RSA Digital Signature to Electronic Mail", *Computer*, Volume 16, No. (Febbraio 1983), pagg. 55-62.

[deBruijn 1967] N. G. deBruijn, "Additional Comments on a Problem in Concurrent Programming and Control", *Communications of the ACM*, Volume 10, No. 3 (Marzo 1967), pagg. 137-138.

[Deitel 1990] H. M. Deitel, *An Introduction to Operating Systems*, Second Edition, Addison-Wesley (1990).

[Deitel e Kogan 1992] H. M. Deitel e M. S. Kogan, *The Design of OS/2*, Addison-Wesley (1992).

[Denning 1968] P. J. Denning, "The Working Set Model for Program Behavior", *Communications of the ACM*, Volume 11, No. 5 (Maggio 1968), pagg. 323-333.

[Denning 1980] P. J. Denning, "Working Sets Past and Present", *IEEE Transactions on Software Engineering*, Volume SE-6, No. 1 (Gennaio 1980), pagg. 64-84.

[Denning 1982] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley (1982).

[Denning 1983] D. E. Denning, "Protecting Public Keys and Signature Keys", *IEEE Computer*, Volume 16, No. 2 (Febbraio 1983), pagg. 27-35.

[Denning 1984] D. E. Denning, "Digital Signatures with RSA and Other PublicKey Cryptosystems", *Communications of the ACM*, Volume 27, No. 4 (Aprile 1984), pagg. 388-392.

[Dennis 1965] J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems", *Journal of the ACM*, Volume 12, No. 4 (Ottobre 1965), pagg. 589-602.

[Dennis e Horn 1966] J. B. Dennis ed E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations", *Communications of the ACM*, Volume 9, Number 3 (Marzo 1966), pagg. 143-155.

[Diffie e Hellman 1976] W. Diffie e M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Volume 22, No. 6 (Novembre 1976), pagg. 644-654.

[Diffie e Hellman 1979] W. Diffie e M. E. Hellman, "Privacy and Authentication", *Proceedings of the IEEE*, Volume 67, No. 3 (Marzo 1979), pagg. 397-427.

[Dijkstra 1965a] E. W. Dijkstra, "Cooperating Sequential Processes", *Technical Report EWD-123*, Technological University, Eindhoven, Olanda (1965), pagg. 43-112.

[Dijkstra 1965b] E. W. Dijkstra, "Solution of a Problem in Concurrent Programming Control", *Communications of the ACM*, Volume 8, No. 9 (Settembre 1965), pag. 569.

[Dijkstra 1968] E. W. Dijkstra, "The Structure of the THE Multiprogramming System", *Communications of the ACM*, Volume 11, No. 5 (Maggio 1968), pagg. 341-346.

[Dijkstra 1971] E. W. Dijkstra, "Hierarchical Ordering of Sequential Processes", *Acta Informatica*, Volume 1, No. 2 (1971), pagg. 115-138.

[DoD 1985] *Trusted Computer System Evaluation Criteria*, Department of Defense (1985).

[Dougan et al. 1999] C. Dougan, P. Mackerras e V. Yodaiken, "Optimizing the Idle Task and Other MMU Tricks", *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (1999).

[Douglis et al. 1991] F. Douglis, M. F. Kaashoek e A. S. Tanenbaum, "A Comparison of Two Distributed Systems: Amoeba and Sprite", *Computing Systems*, Volume 4 (1991).

[Draves et al. 1991] R. P. Draves, B. N. Bershad, R. F. Rashid e R. W. Dean, "Using Continuations to Implement Thread Management and Communication in Operating Systems", *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles* (Ottobre 1991), pagg. 122-136.

[Eaahart 1986] S. V. Earhart, editor, *UNIX Programmer's Manual*, Holt, Rinehart and Winston (1986).

[Eastlake 1999] D. Eastlake, "Domain Name System Security Extensions", *Network Working Group, Request for Comments: 2535* (1999).

[Eisenberg e McGuire 1972] M. A. Eisenberg e M. R. McGuire, "Further Comments on Dijkstra's Concurrent Programming, Control Problem", *Communications of the ACM*, Volume 15, No. 11 (Novembre 1972), pag. 999.

[Ekanadham e Bernstein 1979] K. Ekanadham e A. J. Bernstein, "Conditional Capabilities", *IEEE Transactions on Software Engineering*, Volume SE-5, No. 5 (Settembre 1979), pagg. 458-464.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Engelschall 2000] R. Engelschall, "Portable Multithreading: The Signal Stack Trick For User-Space Thread Creation", *Proceedings of the 2000 USENIX Annual Technical Conference* (2000).

[Eswaran et al. 1976] K. P. Eswaran, J. N. Gray, R. A. Lorie e I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", *Communications of the ACM*, Volume 19, No. 11 (Novembre 1976), pagg. 624-633.

[Eykholt et al. 1992] J. R. Eykholt, S. R. Kleiman, S. Barton, S. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks e D. Williams, "Beyond Multiprocessing: Multithreading the SunOS Kernel", *Proceedings of the Summer 1992 USENIX Conference* (Giugno 1992), pagg. 11-18.

[Farley 1998] J. Farley, *Java Distributed Computing*, O'Reilly & Associates (1998).

[Farrow 1986a] R. Farrow, "Security for Superusers, or How to Break the UNIX System", *UNIX World* (Maggio 1986), pagg. 65-70.

[Farrow 1986b] R. Farrow, "Security Issues and Strategies for Users", *UNIX World* (Aprile 1986), pagg. 65-71.

[Feitelson e Rudolph 1990] D. Feitelson e L. Rudolph, "Mapping and Scheduling in a Shared Parallel Environment Using Distributed Hierarchical Control", *Proceedings of the International Conference on Parallel Processing* (Agosto 1990).

[Filipski e Hanko 1986] A. Filipski e J. Hanko, "Making UNIX Secure", *Byte* (Aprile 1986), pagg. 113-128.

[Finkel 1988] R. A. Finkel, *Operating Systems Vade Mecum*, Second Edition, Prentice-Hall (1988).

[Folk e Zoellick 1987] M. J. Folk e B. Zoellick, *File Structures*, Addison-Wesley (1987).

[Forrest et al. 1996] S. Forrest, S. A. Hofmeyr e T. A. Longstaff, "A Sense of Self for UNIX Processes", *Proceedings of the IEEE Symposium on Security and Privacy* (1996), pagg. 120-128.

[Fortier 1989] P. J. Fortier, *Handbook of LAN Technology*, McGraw-Hill (1989).

[FreeBSD 1999] FreeBSD, *FreeBSD* Handbook, The FreeBSD Documentation Project (1999).

[Freedman 1983] D. H. Freedman, "Searching for Denser Disks", *Infosystems* (Settembre 1983), pag. 56.

[Fujitani 1984] B. Fujitani, "Laser Optical Disk: The Coming Revolution in OnLine Storage", *Communications of the ACM*, Volume 27, No. 6 (Giugno 1984), pagg. 546-554.

[Gait 1988] J. Gait, "The Optical File Cabinet: A Random Access File System for Write-On Optical Disks", *Computer*, Volume 21, No. 6 (Giugno 1988).

[Ganapathy e Schimmel 1998] N. Ganapathy e C. Schimmel, "General Purpose Operating system Support for Multiple Page Sizes", *Proceedings of the USENIX Technical Conference* (1998).

[Garcia-Molina 1982] H. Garcia-Molina, "Elections in Distributed Computing Systems", *IEEE Transactions on Computers*, Volume C-31, No. 1 (Gennaio 1982).

[Garfinkel e Spafford 1991] S. Garfinkel e G. Spafford, *Practical UNIX Security*, O'Reilly & Associates (1991).

[Gifford 1982] D. K. Gifford, "Cryptographic Sealing for Information Secrecy and Authentication", *Communications of the ACM*, Volume 25, No. 4 (Aprile 1982), pagg. 274-286.

[Golden e Pechura 1986] D. Golden e M. Pechura, "The Structure of Microcomputer File Systems", *Communications of the ACM*, Volume 29, No. 3 (Marzo 1986), pagg. 222-230.

[Goldman 1989] P. Goldman, "Mac VM Revealed", *Byte* (Settembre 1989).

[Gong et al. 1997] L. Gong, M. Mueller, H. Prafullchandra e R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", *Proceedings of the USENIX Symposium on Internet technologies and Systems* (1997).

[Gosling et al. 1996] J. Gosling, B. Boy e G. Steele, *The Java Language Specification*, Addison-Wesley (1996).

[Grampp e Morris 1984] F. T. Grampp e R. H. Morris, "UNIX Operating System Security", *AT&T Bell Laboratories Technical Journal*, Volume 63 (Ottobre 1984), pagg. 1649-1672.

[Gray 1978] J. N. Gray, "Notes on Data Base Operating Systems", in [Bayer et al. 1978], pagg. 393-481.

[Gray 1981] J. N. Gray, "The Transaction Concept: Virtues and Limitations", *Proceedings of the International Conference on Very Large Data Bases* (1981), pagg. 144-154.

[Gray 1997] J. N. Gray, Interprocess Communications in UNIX, Prentice Hall (1997).

[Gray et al. 1981] J. N. Gray, P. R. McJones e M. Blasgen, "The Recovery Manager of the System R Database Manager", *ACM Computing Surveys*, Volume 13, No. 2 (Giugno 1981), pagg. 223-242.

[Grosshans 1986] D. Grosshans, *File Systems Design and Implementation*, Prentice-Hall (1986).

[Gupta e Franklin 1978] R. K. Gupta e M. A. Franklin, "Working Set and Page Fault Frequency Replacement Algoritms: A Performance Comparison", *IEEE Transactions on Computers*, Volume C-27, No. 8 (Agosto 1978), pagg. 706-712.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Habermann 1969] A. N. Habermann, "Prevention of System Deadlocks", *Communications of the ACM*, Volume 12, No. 7 (Luglio 1969), pagg. 373-377 e 385.

[Hagmann 1989] R. Hagmann, "Comments on Workstation Operating Systems and Virtual Memory", *Proceedings of the Second Workshop on Workstation Operating Systems* (Settembre 1989).

[Haldar e Subramanian 1991] S. Haldar e D. Subramanian, "Fairness in Processor Scheduling in Time Sharing Systems", *Operating Systems Review* (Gennaio 1991).

[Halsall 1992] F. Halsall, *Data Communications, Computer Networks, and Open Systems*, Addison-Wesley (1992); disponibile nella traduzione italiana della quarta edizione, a cura di Addison-Wesley Italia.

[Han e Ghosh 1998] K. Han e S. Ghosh, "A Comparative Analysis of Virtual Versus Physical Process-Migration Strategies for Distributed Modeling and Simulation of Mobile Computing Networks" *Wireless Networks*, Volume 4, No. 5 (1998) pagg. 365-378.

[Hansen e Atkins 1993] S.E. Hansen e E. T. Atkins, "Automated System Monitoring and Notification With Swatch", *Proceedings of the USENIX Systems Administration Conference* (1993).

[Harchol-Balter e Downey 1997] M. Harchol-Balter e A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing" *ACM Transactions on Computer Systems*, Volume 15, No. 3 (1997), pagg. 253-285.

[Harish e Owens 1999] V. C. Harish e B. Owens, "Dynamic Load Balancing DNS", *Linux Journal*, Volume 1999 No. 64 (1999).

[Harker et al. 1981] J. M. Harker, D. W. Brede, R. E. Pattison, G. R. Santana e L. G. Taft, "A Quarter Century of Disk File Innovation", *IBM Journal of Research and Development*, Volume 25, No. 5 (Settembre 1981), pagg. 677-689.

[Harrison et al. 1976] M. A. Harrison, W. L. Ruzzo e J. D. Ullman, "Protection in Operating Systems", *Communications of the ACM*, Volume 19, No. 8 (Agosto 1976), pagg. 461-471.

[Hartley 1998] S. Hartley, *Concurrent Programming: The Java Programming Language*, Oxford University Press (1998).

[Havender 1968] J. W. Havender, "Avoiding Deadlock in Multitasking Systems", *IBM Systems Journal*, Volume 7, No. 2 (1968), pagg. 74-84.

[Hecht et al. 1988] M. S. Hecht, A. Johri, R. Aditham e T. J. Wei, "Experience Adding C2 Security Features to UNIX", *Proceedings of the Summer 1988 USENIX Conference* (Giugno 1988), pagg. 133-146.

[Hendricks e Hartmann 1979] E. C. Hendricks e T. C. Hartmann, "Evolution of a Virtual Machine Subsystem", *IBM Systems Journal*, Volume 18, No. 1 (1979), pagg. 111-142.

[Hennessy e Patterson 1996] J. L. Hennessy e D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers (1996).

[Henry 1984] G. Henry, "The Fair Share Scheduler", *AT&T Bell Laboratories Technical Journal* (Ottobre 1984).

[Hoagland 1985] A. S. Hoagland, "Information Storage Technology—A Look at the Future", *Computer*, Volume 18, No. 7 (Luglio 1985), pagg. 60-68.

[Hoare 1972] C. A. R. Hoare, "Towards a Theory of Parallel Programming", in [Hoare e Perrott 1972], pagg. 61-71.

[Hoare 1974] C. A. R. Hoare, "Monitors: An Operating System Structuring Concept", *Communications of the ACM*, Volume 17, No. 10 (Ottobre 1974), pagg. 549-557;

[Hoare e Perrott 1972] C. A. R. Hoare e R. H. Perrott, editors, *Operating Systems Techniques*, Academic Press (1972).

[Holt 1971] R. C. Holt, "Comments on Prevention of System Deadlocks", *Communications of the ACM*, Volume 14, No. 1 (Gennaio 1971), pagg. 36-38.

[Holt 1972] R. C. Holt, "Some Deadlock Properties of Computer Systems", *Computing Surveys*, Volume 4, No. 3 (Settembre 1972), pagg. 179-196.

[Holt 1983] R. C. Holt, *Concurrent Euclid, the Unix System, and Tunis*, Addison-Wesley (1983).

[Hong et al. 1989] J. Hong, X. Tan e D. Towsley, "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System", *IEEE Transactions on Computers*, Volume 38, No. 12 (Dicembre 1989), pagg. 1736-1744.

[Horstmann e Cornell 1998] C. Horstmann e G. Cornell, *Core Java 1.2, Volume I: Fundamentals*, Second Edition, Prentice-Hall (1998).

[Howard et al. 1988] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan e R. N. Sidebotham, "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Volume 6, No. 1 (Febbraio 1988), pagg. 55-81.

[Howarth et al. 1961] D. J. Howarth, R. B. Payne e F. H. Sumner, "The Manchester University Atlas Operating System, Part II: User's Description", *Computer Journal*, Volume 4, Number 3 (Ottobre 1961), pagg. 226-229.

[Hsiao et al. 1979] D. K. Hsiao, D. S. Kerr e S. E. Madnick, *Computer Security*, Academic Press (1979).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Hyman 1985] D. Hyman, *The Columbus Chicken Statute, and More Bonehead Legislation*, S. Greene Press (1985).

[Iacobucci 1988] E. Iacobucci, *OS/2 Programmer's Guide*, Osborne McGraw-Hill (1988).

[IBM 1983] *Technical Reference*, IBM Corporation (1983).

[Iliffe e Jodeit 1962] J. K. Iliffe e J. G. Jodeit, "A Dynamic Storage Allocation System", *Computer Journal*, Volume 5, No. 3 (Ottobre 1962), pagg. 200-209.

[Intel 1985a] *iAPX 286 Programmer's Reference Manual*, Intel Corporation (1985).

[Intel 1985b] *iAPX 86/88, 186/188 User's Manual Programmer's Reference*, Intel Corporation (1985).

[Intel 1986] *iAPX 386 Programmer's Reference Manual*, Intel Corporation (1986).

[Intel 1989] *i486 Microprocessor*, Intel Corporation (1989).

[Intel 1990] *i486 Microprocessor Programmer's Reference Manual*, Intel Corporation (1990).

[Intel 1993] *Pentium Processor User's Manual, Volume 3: Architecture and Programming Manual*, Intel Corporation (1993).

[Iseminger 2000] D. Iseminger, *Active Directory Services for Microsoft Windows 2000. Technical Reference*. Microsoft Press (2000).

[Jacob e Mudge 1997] B. Jacob e T. Mudge, "Software-Managed Address Translation", *Proceedings of the Third International Symposium on High performance computer Architecture and Implementation* (1997).

[Jacob e Mudge 1998a] B. Jacob e T. Mudge, "Virtual Memory in Contemporary Microprocessors", *IEEE Micro Magazine*, Volume 18, (1998), pagg. 60-75.

[Jacob e Mudge 1998b] B. Jacob e T. Mudge, "Virtual Memory: Issues of Implementation", *IEEE Computer Magazine*, Volume 31, (1998), pagg. 33-43.

[Jensen et al. 1985] E. D. Jensen, C. D. Locke e H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of the IEEE Real-Time Systems Symposium* (Dicembre 1985), pagg. 112-122.

[Jones e Lin 1996] R. Jones e R. Lin, *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*, John Wiley and Sons (1996).

[Jones e Liskov 1978] A. K. Jones e B. H. Liskov, "A Language Extension for Expressing Constraints on Data Access", *Communications of the ACM*, Volume 21, No. 5 (Maggio 1978), pagg. 358-367.

[Jones e Schwarz 1980] A. K. Jones e P. Schwarz, "Experience Using Multiprocessor Systems—A Status Report", *Computing Surveys*, Volume 12, No. 2 (Giugno 1980), pagg. 121-165.

[Katz et al. 1989] R. H. Katz, G. A. Gibson e D. A. Patterson, "Disk System Architectures for High Performance Computing", *Proceedings of the IEEE*, Volume 77, No. 12 (Dicembre1989).

[Kay e Lauder 1988] J. Kay e P. Lauder, "A Fair Share Scheduler", *Communications of the ACM*, Volume 31, No. 1 (Gennaio 1988), pagg. 44-45.

[Kenah et al. 1988] L. J. Kenah, R. E. Goldenberg e S. F. Bate, *VAX/VMS Internals and Data Structures*, Digital Press (1988).

[Kenville 1982] R. F. Kenville, "Optical Disk Data Storage", *Computer*, Volume 15, Number 7 (Luglio 1982), pagg. 21-26.

[Kernighan e Pike 1984] B. W. Kernighan e R. Pike, *The UNIX Programming Environment*, Prentice-Hall (1984).

[Kernighan e Ritchie 1988] B. W. Kernighan e D. M. Ritchie, *The C Programming Language*, Second Edition, Prentice-Hall (1988).

[Kessels 1977] J. L. W. Kessels, "An Alternative to Event Queues for Synchronization in Monitors", *Communications of the ACM*, Volume 20, No. 7 (Luglio 1977), pagg. 500-503.

[Khanna et al. 1992] S. Khanna, M. Sebree e J. Zonlowsky, "Realtime Scheduling in SunOS 5.0", *Proceedings of the Winter USENIX Conference* (1992), pagg. 375-390.

[Kieburtz e Silberschatz 1978] R. B. Kieburtz e A. Silberschatz, "Capability Managers", *IEEE Transactions on Software Engineering*, Volume SE-4, No. 6 (Novembre 1978), pagg. 467-477.

[Kieburtz e Silberschatz 1983] R. B. Kieburtz e A. Silberschatz, "Access Right Expressions", *ACM Transactions on Programming Languages and Systems*, Volume 5, No. 1(Gennaio 1983), pagg. 78-96.

[Kilburn et al. 1961] T. Kilburn, D. J. Howarth, R. B. Payne e F. H. Sumner, "The Manchester University Atlas Operating System, Part I: Internal Organization", *Computer Journal*, Volume 4, No. 3 (Ottobre 1961), pagg. 222-225.

[Kim e Spafford 1993] G. H. Kim e E. H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker", *Technical Report, Purdue University* (1993).

[King 1990] R. P. King, "Disk Arm Movement in Anticipation of Future Requests", *ACM Transactions on Computer Systems*, Volume 8, Number 3 (Agosto 1990), pagg. 214-229.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Kleiman et al. 1996] S. Kleiman, D. Shah e B. Smaalders, *Programming with Threads*, Sunsoft Press (1996).

[Kleinrock 1975] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*, Wiley-Interscience (1975).

[Knapp 1987] E. Knapp, "Deadlock Detection in Distributed Databases", *Computing Surveys*, Volume 19, No. 4 (Dicembre 1987), pagg. 303-328.

[Knuth 1973] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Second Edition, Addison-Wesley (1973).

[Koch 1987] P. D. L. Koch, "Disk File Allocation Based on the Buddy System", *ACM Transactions on Computer Systems*, Volume 5, No. 4 (Novembre 1987), pagg. 352-370.

[Kogan e Rawson 1988] M. S. Kogan e F. L. Rawson, "The Design of Operating System/2", *IBM Systems Journal*, Volume 27, No. 2 (1988), pagg. 90-104.

[Korn 1983] D. Korn, "KSH, A Shell Programming Language", *Proceedings of the Summer USENIX Conference* (Luglio 1983), pagg. 191-202.

[Kosaraju 1973] S. Kosaraju, "Limitations of Dijkstra's Semaphore Primitives and Petri Nets", *Operating Systems Review*, Volume 7, No. 4 (Ottobre 1973), pagg. 122-126.

[Kramer 1988] S. M. Kramer, "Retaining SUID Programs in a Secure UNIX", *Proceedings of the Summer USENIX Conference* (Giugno 1988), pagg. 107-118.

[Kurose e Ross 2001] J. Kurose e K. Ross, *Computer Networking – A Top-Down Approach Featuring The Internet*, Addison-Wesley (2001).

[Lamport 1974] L. Lamport, "A New Solution of Dijkstra's Concurrent Programming Problem", *Communications of the ACM*, Volume 17, No. 8 (Agosto 1974), pagg. 453-455.

[Lamport 1976] L. Lamport, "Synchronization of Independent Processes", *Acta Informatica*, Volume 7, No. 1 (1976), pagg. 15-34.

[Lamport 1977] L. Lamport, "Concurrent Reading and Writing", *Communications of the ACM*, Volume 20, No. 11 (Novembre 1977), pagg. 806-811.

[Lamport 1978a] L. Lamport, "The Implementation of Reliable Distributed Multiprocess Systems", *Computer Networks*, Volume 2, No. 2 (Aprile 1978), pagg. 95-114.

[Lamport 1978b] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, Volume 21, No. 7 (Luglio 1978), pagg. 558-565.

[Lamport 1981] L. Lamport, "Password Authentication with Insecure Communications", *Communications of the ACM*, Volume 24, No. 11 (Novembre 1981), pagg. 770-772.

[Lamport 1986] L. Lamport, "The Mutual Exclusion Problem", *Journal of the ACM*, Volume 33, No. 2 (1986), pagg. 313-348.

[Lamport 1991] L. Lamport, "The Mutual Exclusion Problem Has Been Solved", *Communications of the ACM*, Volume 34, No. 1 (Gennaio 1991), pag. 110.

[Lamport et al. 1982] L. Lamport, R. Shostak e M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, Volume 4, No. 3 (Luglio 1982), pagg. 382-401.

[Lampson 1968] B. W. Lampson, "A Scheduling Philosophy for Multiprocessing Systems", *Communications of the ACM*, Volume 11, No. 5 (Maggio 1968), pagg. 347-360.

[Lampson 1969] B. W. Lampson, "Dynamic Protection Structures", *Proceedings of the AFIPS Fall Joint Computer Conference* (1969), pagg. 27-38.

[Lampson 1971] B. W. Lampson, "Protection", *Proceedings of the Fifth Annual Princeton Conference on Information Science Systems* (1971), pagg. 437-443; stampato in "Operating System Review", Volume 8, No. 1 (Gennaio 1974), pagg. 18-24.

[Lampson 1973] B. W. Lampson, "A Note on the Confinement Problem", *Communications of the ACM*, Volume 10, No. 16 (Ottobre 1973), pagg. 613-615.

[Lampson e Sturgis 1976] B. Lampson e H. Sturgis, "Crash Recovery in a Distributed Data Storage System", *Technical Report*, Xerox Research Center (1976).

[Landwehr 1981] C. E. Landwehr, "Formal Models of Computer Security", *Computing Surveys*, Volume 13, No. 3 (Settembre 1981), pagg. 247-278.

[Lann 1977] G. L. Lann, "Distributed Systems–Toward a Formal Approach", *Proceedings of the IFIP Congress* (1977), pagg. 155-160.

[Larson e Kajla 1984] P. Larson e A. Kajla, "File Organization: Implementation of a Method Guaranteeing Retrieval in One Access", *Communications of the ACM*, Volume 27, No. 7 (Luglio 1984), pagg. 670-677.

[Lazowska et al. 1984] L. D. Lazowska, J. Zahorjan, G. S. Graham e K. C. Sevcik, *Quantitative System Performance*, Prentice-Hall (1984).

[Lea 2000] D. Lea, *Concurrent Programming in Java, Second Edition* Addison-Wesley (2000).

[Leffler et al. 1989] S. J. Leffler, M. K. McKusick, M. J. Karels e J. S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley (1989).

[Lehmann 1987] F. Lehmann, "Computer Break-Ins", *Communications of the ACM*, Volume 30, No. 7 (Luglio 1987), pagg. 584-585.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Lempel 1979] A. Lempel, "Cryptology in Transition", *Computing Surveys*, Volume 11, No. 4 (Dicembre 1979), pagg. 286-303.

[Lett e Konigsford 1968] A. L. Lett e W. L. Konigsford, "TSS/360: A Time-Shared Operating System", *Proceedings of the AFIPS Fall Joint Computer Conference* (1968), pagg. 15-28.

[Letwin 1988] G. Letwin, *Inside OS/2*, Microsoft Press (1988).

[Leutenegger e Vernon 1990] S. Leutenegger e M. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Policies", *Proceedings of the Conference on Measurement and Modeling of Computer Systems* (Maggio 1990).

[Levin et al. 1975] R. Levin, E. S. Cohen, W. M. Corwin, F. J. Pollack e W. A. Wulf, "Policy/Mechanism Separation in Hydra", *Proceedings of the Fifth ACM Symposium on Operating System Principles* (1975), pagg. 132-140.

[Levy e Lipman 1982] H. M. Levy e P. H. Lipman, "Virtual Memory Management in the VAX/VMS Operating System", *Computer*, Volume 15, No. 3 (Marzo 1982), pagg. 35-41.

[Lewis e Berg 1988] B. Lewis e D. Berg, *Multithreaded programming with Pthreads*, Sun Microsystems Press (1988).

[Lichtenberger e Pirtle 1965] W. W. Lichtenberger e M. W. Pirtle, "A Facility for Experimentation in Man-Machine Interaction", *Proceedings of the AFIPS Fall Joint Computer Conference* (1965), pagg. 589-598.

[Lindholm e Yellin 1998] T. Lindholm e F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley (1998).

[Ling et al. 2000] Y. Ling, T. Mullen e X. Lin, *Concurrent Programming in Java, Second Edition* Addison-Wesley (2000).

[Lipner 1975] S. Lipner, "A Comment on the Confinement Problem", *Operating System Review*, Volume 9, No. 5 (Novembre 1975), pagg. 192-196.

[Lipton 1974] R. Lipton, "On Synchronization Primitive Systems", PhD. Thesis, Carnegie-Mellon University (1974).

[Liskov 1972] B. H. Liskov, "The Design of the Venus Operating System", *Communications of the ACM*, Volume 15, No. 3 (Marzo 1972), pagg. 144-149.

[Litzkow et al. 1988] M. J. Litzkow, M. Livny e M. W. Mutka, "Condor—A Hunter of Idle Workstations", *Proceedings of the Eighth IEEE International Conference on Distributed Computing Systems* (1988), pagg. 104-111.

[Liu e Layland 1973] C. L. Liu e J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, Volume 20, No. 1 (Gennaio 1973), pagg. 46-61.

[Lobel 1986] J. Lobel, *Foiling the System Breakers: Computer Security and Access Control*, McGraw-Hill (1986).

[Loucks e Sauer 1987] L. K. Loucks e C. H. Sauer, "Advanced Interactive Executive (AIX) Operating System Overview", *IBM Systems Journal*, Volume 26, No. 4 (1987), pagg. 326-345.

[MacKinnon 1979] R. A. MacKinnon, "The Changing Virtual Machine Environment: Interfaces to Real Hardware, Virtual Hardware, and Other Virtual Machines", *IBM Systems Journal*, Volume 18, No. 1 (1979), pagg. 18-46.

[Maekawa 1985] M. Maekawa, "A Square Root Algorithm for Mutual Exclusion in Decentralized Systems", *ACM Transactions on Computer Systems*, Volume 3, No. 2 (Maggio 1985), pagg. 145-159.

[Maher et al. 1994] C. Maher, J. S. Goldick, C. Kerby e B. Zumach, "The Integration of Distributed File Systems and Mass Storage Systems", *Proceedings of the Thirteenth IEEE Symposium on Mass Storage Systems* (Giugno 1994), pagg. 27-31.

[Marsh et al. 1991] B. D. Marsh, M. L. Scott, T. J. LeBlanc ed E. P. Markatos, "First-Class User-Level Threads", *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, (Ottobre 1991), pagg. 110-121.

[Massalin e Pu 1989] H. Massalin e C. Pu, "Threads and Input/Output in the Synthesis Kernel", *Proceedings of the 12th Symposium on Operating Systems Principles* (Dicembre 1989), pagg. 191-200.

[Mattson et al. 1970] R. L. Mattson, J. Gecsei, D. R. Slutz e I. L. Traiger, "Evaluation Techniques for Storage Hierarchies", *IBM Systems Journal*, Volume 9, No. 2 (1970), pagg. 78-117.

[Mauro e McDougall 2001] J. Mauro e R. McDougall, *Solaris Internals: Core Kernel Architecture*, Prentice-Hall (2001).

[McGraw e Andrews 1979] J. R. McGraw e G. R. Andrews, "Access Control in Parallel Programs", *IEEE Transactions on Software Engineering*, Volume SE-5, No. 1 (Gennaio 1979), pagg. 1-9.

[McKeag e Wilson 1976] R. M. McKeag e R. Wilson, *Studies in Operating Systems*, Academic Press, London (1976).

[McKeon 1985] B. McKeon, "An Algorithm for Disk Caching with Limited Memory", *Byte*, Volume 10, Number 9 (Settembre 1985), pagg. 129-138.

[McKusick e Karels 1988] M. K. McKusick e K. J. Karels, "Design of a General Purpose Memory Allocator for the 4.3BSD UNIX Kernel", *Proceedings of the Summer USENIX Conference* (Giugno 1988), pagg. 295-304.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[McKusick et al. 1984] M. K. McKusick, W. N. Joy, S. J. Leffler e R. S. Fabry, "A Fast File System for UNIX", *ACM Transactions on Computer Systems*, Volume 2, No. 3 (Agosto 1984), pagg. 181-197.

[McKusick et al. 1986] M. K. McKusick, K. Bostic e M. J. Karels, *The Design and Implementation of the 4.4 BSD UNIX Operating System*, John Wiley and Sons (1996).

[McVoy e Kleiman 1991] L. W. McVoy e S. R. Kleiman, "Extent-like Performance from a UNIX File System", *Proceedings of the Winter USENIX Conference* (Gennaio 1991), pagg. 33-44.

[Mealy et al. 1966] G. H. Mealy, B. I. Witt e W. A. Clark, "The Functional Structure of OS/360", *IBM Systems Journal*, Volume 5, No. 1 (1966).

[Menasce e Muntz 1979] D. Menasce e R. R. Muntz, "Locking and Deadlock Detection in Distributed Data Bases", *IEEE Transactions on Software Engineering*, Volume SE-5, No. 3 (Maggio 1979), pagg. 195-202.

[Meyer e Downing 1997] J. Meyer e T. Downing, *Java Virtual Machine*, O'Reilly and Associates (1997).

[Meyer e Seawright 1970] R. A. Meyer e L. H. Seawright, "A Virtual Machine Time-Sharing System", *IBM Systems Journal*, Volume 9, No. 3 (1970), pagg. 199-218.

[Microsoft 1986] *Microsoft MS-DOS User's Reference and Microsoft MS-DOS Programmer's Reference*, Microsoft Press (1986).

[Microsoft 1989] *Microsoft Operating System/2 Programmer's Reference*, 3 volumi, Microsoft Press (1989).

[Microsoft 1991] *Microsoft MS-DOS User's Guide and Reference*, Microsoft Press (1991).

[Microsoft 1996] *Microsoft Windows NT Workstation resource KIT*, Microsoft Press (1996).

[Microsoft 2000a] *Microsoft Developer Network Development Library*, Microsoft Press (2000).

[Microsoft 2000b] *Microsoft Windows 2000 Server Resource Kit*, Microsoft Press (2000).

[Milenkovic 1987] M. Milenkovic, *Operating Systems: Concepts and Design*, McGraw-Hill (1987).

[Miller e Katz 1993] E. L. Miller e R. H. Katz, "An Analysis of File Migration in a UNIX Supercomputing Environment", *Proceedings of the Winter USENIX Conference* (Gennaio 1993), pagg. 421-434.

[Milo et al. 2000] D. Milo, F. Douglis, Y. Paindaveine, R. Wheeler e S. Zhou, "Process Migration", *ACM Computing Survey*, Volume 32, No. 3 (2000), pagg. 241-299.

[Mockapetris 1987] P. Mockapetris, "Domain Names – Concepts and Facilities" *Network Working Group, Request for Comments: 1034* (1987).

[Mohan e Lindsay 1983] C. Mohan e B. Lindsay, "Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions", *Proceedings of the ACM Symposium on Principles of Database Systems* (1983).

[Morris 1973] J. H. Morris, "Protection in Programming Languages", *Communications of the ACM*, Volume 16, No. 1 (Gennaio 1973), pagg. 15-21.

[Morris e Thompson 1979] R. Morris e K. Thompson, "Password Security: A Case History", *Communications of the ACM*, Volume 22, No. 11 (Novembre 1979), pagg. 594-597.

[Morris et al. 1986] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal e F. D. Smith, "Andrew: A Distributed Personal Computing Environment", *Communications of the ACM*, Volume 29, No. 3 (Marzo 1986), pagg. 184-201.

[Morshedian 1986] D. Morshedian, "How to Fight Password Pirates", *Computer*, Volume 19, No. 1 (Gennaio 1986).

[Motorola 1989a] *MC68000 Family Reference*, Second Edition, Prentice-Hall (1989).

[Motorola 1989b] *MC68030 Enhanced 32-Bit Microprocessor User's Manual*, Second Edition, Prentice-Hall (1989).

[Motorola 1993] *PowerPC 601 RISC Microprocessor User's Manual*, Motorola Inc. (1993).

[Mullender 1993] S. Mullender (Editor), *Distributed Systems*, Second Edition, ACM Press (1993).

[Mullender et al. 1990] S. J. Mullender, G. Van Rossum, A. S. Tanenbaum, R. Van Renesse e H. Van Staveren, "Amoeba: A Distributed-Operating System for the 1990s", *IEEE Computer*, Volume 23, No. 5 (Maggio 1990), pagg. 44-53.

[Murray 1998] J. Murray, *Inside Microsoft Windows CE*, Microsoft Press (1998).

[Mutka e Livny 1987] M. W. Mutka e M. Livny, "Scheduling Remote Processor Capacity in a Workstation-Processor Bank Network", *Proceedings of the IEEE International Conference on Distributed Computing Systems* (1987).

[Needham e Walker 1977] R. M. Needham e R. D. H. Walker, "The Cambridge CAP Computer and its Protection System", *Proceedings of the Sixth Symposium on Operating System Principles* (Novembre 1977), pagg. 1-10.

[Nelson et al. 1988] M. Nelson, B. Welch e J. K. Ousterhout, "Caching in the Sprite Network File System", *ACM Transactions on Computer Systems*, Volume 6, No. 1 (Febbraio 1988), pagg. 134-154.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Nichols 1987] D. A. Nichols, "Using Idle Workstations in a Shared Computing Environment", *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles* (Ottobre 1987), pagg. 5-12.

[Niemeyer e Peck 1997] P. Niemeyer e J. Peck, *Exploring Java, Second Edition*, O'Reilly & Associates (1997).

[Norton 1986] P. Norton, *Inside the IBM PC*, edizione riveduta e ampliata, Brady Books (1986).

[Norton e Wilton 1988] P. Norton e R. Wilton, *The New Peter Norton Programmer's Guide to the IBM PC & PS/2*, Microsoft Press (1988).

[Nutt 1999] G. Nutt, *Operating Systems: A Modern Perspective, Second Edition*, Addison-Wesley (1999).

[Oaks e Wong 1999] S. Oaks e H. Wong, *Java Threads, Second Edition*, O'Reilly & Assoiates (1999).

[Obermarck 1982] R. Obermarck, "Distributed Deadlock Detection Algorithm", *ACM Transactions on Database Systems*, Volume 7, No. 2 (Giugno 1982), pagg. 187-208.

[O'Leary e Kitts 1985] B. T. O'Leary e D. L. Kitts, "Optical Device for a Mass Storage System", *Computer*, Volume 18, No. 7 (1985).

[Olsen e Kenley 1989] R. P. Olsen e G. Kenley, "Virtual Optical Disks Solve the On-Line Storage Crunch", *Computer Design*, Volume 28, No. 1 (Gennaio 1989), pagg. 93-96.

[Organick 1972] E. I. Organick, *The Multics System: An Examination of its Structure*, MIT Press (1972).

[Ousterhout et al. 1985] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer e J. G. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", *Proceedings of the ACM Symposium on Operating Systems Principles* (Dicembre 1985), pagg. 15-24.

[Ousterhout et al. 1988] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson e B. B. Welch, "The Sprite Network-Operating System", *IEEE Computer*, Volume 21, No. 2 (Febbraio 1988), pagg. 23-36.

[Parmelee et al. 1972] R. P. Parmelee, T. I. Peterson, C. C. Tillman e D. Hatfield, "Virtual Storage and Virtual Machine Concepts", *IBM Systems Journal*, Volume 11, No. 2 (1972), pagg. 99-130.

[Parnas 1975] D. L. Parnas, "On a Solution to the Cigarette Smokers' Problem Without Conditional Statements", *Communications of the ACM*, Volume 18, No. 3 (Marzo 1975), pagg. 181-183.

[Patil 1971] S. Patil, "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes", *Technical Report*, MIT (1971).

[Patterson et al. 1988] D. A. Patterson, G. Gibson e R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (1988).

[Peacock 1992] J. K. Peacock, "File System Multithreading in System V Release 4 MP", *Proceedings of the Summer 1992 USENIX Conference* (Giugno 1992), pagg. 19-29.

[Pease et al. 1980] M. Pease, R. Shostak e L. Lamport, "Reaching Agreement in the Presence of Faults", *Journal of the ACM*, Volume 27, No. 2 (Aprile 1980), pagg. 228-234.

[Pechura e Schoeffler 1983] M. A. Pechura e J. D. Schoeffler, "Estimating File Access Time of Floppy Disks", *Communications of the ACM*, Volume 26, No. 10 (Ottobre 1983), pagg. 754-763.

[Peterson 1981] G. L. Peterson, "Myths About the Mutual Exclusion Problem", *Information Processing Letters*, Volume 12, No. 3 (Giugno 1981).

[Pfleeger 1989] C. Pfleeger, *Security in Computing*, Prentice-Hall (1989).

[Pham e Garg 1996] T. Pham e P. Garg, *Multithreaded Programming with Windows NT*, Prentice-Hall (1996).

[Popek 1974] G. J. Popek, "Protection Structures", *Computer*, Volume 7, Number 6 (Giugno 1974), pagg. 22-33.

[Popek e Walker 1985] G. Popek e B. Walker (Editors), *The LOCUS Distributed System Architecture*, MIT Press (1985).

[Prieve e Fabry 1976] B. G. Prieve e R. S. Fabry, "VMIN—An Optimal Variable Space Page-Replacement Algorithm", *Communications of the ACM*, Volume 19, Number 5 (Maggio 1976), pagg. 295-297.

[Psaltis e Mok 1995] D. Psaltis e F. Mok, "Holographic Memories", *Scientific American*, Volume 273, No. 5 (Novembre 1995), pagg. 70-76.

[Purdin et al. 1987] T. D. M. Purdin, R. D. Schlichting e G. R. Andrews, "A File Replication Facility for Berkeley UNIX", *Software—Practice and Experience*, Volume 17 (Dicembre 1987), pagg. 923-940.

[Quarterman 1990] J. S. Quarterman, *The Matrix Computer Networks and Conferencing Systems Worldwide*, Digital Press (1990).

[Quarterman e Hoskins 1986] J. S. Quarterman e H. C. Hoskins, "Notable Computer Networks", *Communications of the ACM*, Volume 29, No. 10 (Ottobre 1986), pagg. 932-971.

[Quinlan 1991] S. Quinlan, "A Cached WORM File System", *Software—Practice and Experience*, Volume 21, No. 12 (Dicembre 1991), pagg. 1289-1299.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Rago 1993] S. Rago, *UNIX System V Network Programming*, Addison-Wesley (1993).

[Rashid 1986] R. F. Rashid, "From RIG to Accent to Mach: The Evolution of a Network-Operating System", *Proceedings of the ACM/IEEE Computer Society, Fall Joint Computer Conference* (1986).

[Rashid e Robertson 1981] R. Rashid e G. Robertson, "Accent: A Communication Oriented Network-Operating System Kernel", *Proceedings of the Eighth Symposium on Operating System Principles* (Dicembre 1981).

[Raynal 1986] M. Raynal, *Algorithms for Mutual Exclusion*, MIT Press (1986).

[Raynal 1991] M. Raynal, "A Simple Taxonomy for Distributed Mutual Exclusion Algorithms", *Operating Systems Review*, Volume 25 (Aprile 1991), pagg. 47-50.

[Redell e Fabry 1974] D. D. Redell e R. S. Fabry, "Selective Revocation of Capabilities", *Proceedings of the IRIA International Workshop on Protection in Operating Systems* (1974), pagg. 197-210.

[Reed 1983] D. P. Reed, "Implementing Atomic Actions on Decentralized Data", *ACM Transactions on Computer Systems*, Volume 1 (Febbraio 1983), pagg. 3-23.

[Reed e Kanodia 1979] D. P. Reed e R. K. Kanodia, "Synchronization with Eventcounts and Sequences", Communications of the ACM, Volume 22, No. 2 (Febbraio 1979), pagg. 115-123.

[Reid 1987] B. Reid, "Reflections on Some Recent Widespread Computer Break-Ins", *Communications of the ACM*, Volume 30, No. 2 (Febbraio 1987), pagg. 103-105.

[Rhodes e McKeehan 1999] N. Rhodes e J. McKeehan, *Understanding The Linux Kernel*, O'Reilly & Associates (1999).

[Ricart e Agrawala 1981] G. Ricart e A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks", *Communications of the ACM*, Volume 24, No. 1 (1981) pagg. 9-17.

[Richards 1990] A. E. Richards, "A File System Approach for Integrating Removable Media Devices and Jukeboxes", *Optical Information Systems*, Volume 10, No. 5 (Settembre 1990), pagg. 270-274.

[Richter 1997] J. Richter, *Advanced Windows*, Third Edition, Microsoft Press (1997).

[Ritchie e Thompson 1974] D. M. Ritchie e K. Thompson, "The UNIX Time-Sharing System", *Communications of the ACM*, Volume 17, No. 7 (Luglio 1974), pagg. 365-375; una versione successiva apparve in *Bell System Technical Journal*, Volume 57, No. 6 (Luglio-Agosto 1978), pagg. 1905-1929.

[Rivest et al. 1978] R. L. Rivest, A. Shamir e L. Adleman, "On Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, Volume 21, No. 2 (Febbraio 1978), pagg. 120-126.

[Rosenkrantz et al. 1978] D. J. Rosenkrantz, R. E. Stearns e P. M. Lewis, II, "System Level Concurrency Control for Distributed Database Systems", *ACM Transactions on Database Systems*, Volume 3, No. 2 (Giugno 1978), pagg. 178-198.

[Ruemmler e Wilkes 1991] C. Ruemmler e J. Wilkes, "Disk Shuffling", *Technical Report*, Hewlett-Packard Laboratories (Ottobre 1991).

[Ruemmler e Wilkes 1993] C. Ruemmler e J. Wilkes, "Unix Disk Access Patterns", *Proceedings of the Winter USENIX Conference* (Gennaio 1993), pagg. 405-420.

[Ruemmler e Wilkes 1994] C. Ruemmler e J. Wilkes, "An Introduction to Disk Drive Modeling", *IEEE Computer*, Volume 27, No. 3 (Marzo 1994), pagg. 17-29.

[Ruschitzka e Fabry 1977] M. Ruschizka e R. S. Fabry, "A Unifying Approach to Scheduling", *Communications of the ACM*, Volume 20, No. 7 (Luglio 1977), pagg. 469-477.

[Rushby 1981] J. M. Rushby, "Design and Verification of Secure Systems", *Proceedings of the ACM Symposium on Operating System Principles* (Dicembre 1981), pagg. 12-21.

[Rushby e Randell 1983] J. Rushby e B. Randell, "A Distributed Secure System", *Computer*, Volume 16, No. 7 (Luglio 1983), pagg. 55-67.

[Russell e G. T. Gangemi 1991] D. Russell e G. T. Gangemi, Sr., *Computer Security Basics*, O'Reilly & Associates (1991).

[Saltzer e Schroeder 1975] J. H. Saltzer e M. D. Schroeder, "The Protection of Information in Computer Systems", *Proceedings of the IEEE*, Volume 63, No. 9 (Settembre 1975), pagg. 1278-1308.

[Sandberg 1987] R. Sandberg, *The Sun Network File System: Design, Implementation, and Experience*, Sun Microsystems (1987).

[Sandberg et al. 1985] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh e B. Lyon, "Design and Implementation of the Sun Network Filesystem", *Proceedings of the Summer USENIX Conference* (Giugno 1985), pagg. 119-130.

[Sargent e Shoemaker 1995] M. Sargent e R. Shoemaker, *The Personal Computer from the Inside Out*, Third Edition, Addison-Wesley (1995).

[Sarisky 1983] L. Sarisky, "Will Removable Hard Disks Replace the Floppy?", *Byte* (Marzo 1983), pagg. 110-117.

[Satyanarayanan 1989] M. Satyanarayanan, "Integrating Security in a Large Distributed System", *ACM Transactions on Computer Systems*, Volume 7 (Agosto 1989), pagg. 247-280.

[Satyanarayanan 1990] M. Saryanarayanan, "Scalable, Secure, and Highly Available Distributed File Access", *Computer*, Volume 23, No. 5 (Maggio 1990), pagg. 9-21.

[Sauer e Chandy 1981] C. H. Sauer e K. M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall (1981).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Schell 1983] R. R. Schell, "A Security Kernel for a Multiprocessor Microcomputer", *Computer*, Volume 16, No. 7 (Luglio 1983), pagg. 47-53.

[Schindler e Gregory 1999] J. Schindler e G. Gregory, "Automated Disk Drive Characterization", *technical Report*, Carnegie-Mellon University (1999).

[Schlichting e Schneider 1982] R. D. Schlichting e F. B. Schneider, "Understanding and Using Asynchronous Message Passing Primitives", *Proceedings of the Symposium on Principles of Distributed Computing* (Agosto 1982), pagg. 141-147.

[Schneider 1982] F. B. Schneider, "Synchronization in Distributed Programs", *ACM Transactions on Programming Languages and Systems*, Volume 4, No. 2 (Aprile 1982), pagg. 125-148.

[Schrage 1967] L. E. Schrage, "The Queue M/G/I with Feedback to Lower Priority Queues", *Management Science*, Volume 13 (1967), pagg. 466-474.

[Schroeder et al. 1985] M. D. Schroeder, D. K. Gifford e R. M. Needham, "A Caching File System for a Programmer's Workstation", *Proceedings of the ACM Symposium on Operating Systems Principles* (Dicembre 1985), pagg. 25-32.

[Schultz 1988] B. Schultz, "VM: The Crossroads of Operating Systems", *Datamation*, Volume 34, Number 14 (Luglio 1988), pagg. 79-84.

[Schwartz e Weissman 1967] J. I. Schwartz e C. Weissman, "The SDC Time-Sharing System Revisited", *Proceedings of the ACM National Meeting* (Agosto 1967), pagg. 263-271.

[Schwartz et al. 1964] J. I. Schwartz, E. G. Coffman e C. Weissman, "A General Purpose Time-Sharing System", *Proceedings of the AFIPS Spring Joint Computer Conference* (1964), pagg. 397-411.

[Seely 1989] D. Seely, "Password Cracking: A Game of Wits", *Communications of the ACM*, Volume 32, No. 6 (Giugno 1989), pagg. 700-704.

[Seltzer et al. 1990] M. Seltzer, P. chen e J. Ousterhout, "Disk Scheduling Revisited", *Proceedings of the Winter USENIX Conference* (1990) pagg. 313-323.

[Shrivastava e Panzieri 1982] S. K. Shrivastava e F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism", *IEEE Transactions on Computers*, Volume C-31, No. 7 (Luglio 1982), pagg. 692-697.

[Silberschatz et al. 2001] A. Silberschatz, H. F. Korth e S. Sudarshan, *Database System Concepts*, Fourth Edition, McGraw-Hill (2001).

[Silverman 1983] J. M. Silverman, "Reflections on the Verification of the Security of an Operating System Kernel", *Proceedings of the ACM Symposium on Operating Systems Principles*, Volume 17, No. 5 (Ottobre 1983), pagg. 143-154.

[Silvers 2000] C. Silvers, "UBC: An Efficient Unified I/O and Memory Caching Subsystem for NetBSD", *USENIX Annual Technical Conference – FREENIX Track* (2000).

[Simmons 1979] G. J. Simmons, "Symmetric and Asymmetric Encryption", *Computing Surveys*, Volume 11, No. 4 (Dicembre 1979), pagg. 304-330.

[Sincerbox 1994] G. T. Sincerbox (Editor), *Selected Papers on Holographic Storage*, Number MS 95, SPIE Milestone Series, Optical Engineering Press (1994).

[Singhal 1989] M. Singhal, "Deadlock Detection in Distributed Systems", *IEEE Computer*, Volume 22, No. 11 (Novembre 1989), pagg. 37-48.

[Smith 1982] A. J. Smith, "Cache Memories", *ACM Computing Surveys*, Volume 14, No. 3 (Settembre 1982), pagg. 473-530.

[Smith 1985] A. J. Smith, "Disk Cache-Miss Ratio Analysis and Design Considerations", *ACM Transactions on Computer Systems*, Volume 3, No. 3 (Agosto 1985), pagg. 161-203.

[Solomon 1998] D. A. Solomon, *Inside Windows NT, Second Edition*, Microsoft Press (1998).

[Solomon e Russinovich 2000] D. A. Solomon E M. E. Russinovich, *Inside Microsoft Windows 2000, Third Edition*, Microsoft Press (2000).

[Spafford 1989] E. H. Spafford, "The Internet Worm: Crisis and Aftermath", *Communications of the ACM*, Volume 32, No. 6 (Giugno 1989), pagg. 678-687.

[Spector e Schwarz 1983] A. Z. Spector e P. M. Schwarz, "Transactions: A Construct for Reliable Distributed Computing", *ACM SIGOPS Operating Systems Review*, Volume 17, No. 2 (1983), pagg. 18-35.

[Stallings 2000a] W. Stallings, *Local and Metropolitan Area Networks*, Prentice-Hall 2000.

[Stallings 2000b] W. Stallings, *Operating Systems*, Fourth Edition, Prentice-Hall (2000).

[Stankovic 1982] J. S. Stankovic, "Software Communication Mechanisms: Procedure Calls Versus Messages", *Computer*, Volume 15, No. 4 (Aprile 1982).

[Staunstrup 1982] J. Staunstrup, "Message Passing Communication versus Procedure Call Communication", *Software—Practice and Experience*, Volume 12, No. 3 (Marzo 1982), pagg. 223-234.

[Stevens 1992] R. Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley (1992).

[Stevens 1994] R. Stevens, *TCP/IP Illustrated Volume 1: The Protocols*, Addison-Wesley (1994).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Stevens 1995] R. Stevens, *TCP/IP Illustrated Volume 2: The Implementation*, Addison-Wesley (1995).

[Stevens 1997] W. R. Stevens, *UNIX Network Programming – Volume I*, Prentice-Hall (1997).

[Stevens 1998] W. R. Stevens, *UNIX Network Programming – Volume II*, Prentice-Hall (1998).

[Strachey 1959] C. Strachey, "Time Sharing in Large Fast Computers", *Proceedings of the International Conference on Information Processing* (Giugno 1959), pagg. 336-341.

[Su 1982] Z. Su, "A Distributed System for Internet Name Service", *Network Working Group, Request for Comments: 830* (1982).

[Sun 1990] *Network Programming Guide*, Sun Microsystems (1990)

[Sun 1995] *Solaris Multithreaded Programming Guide*, Sunsoft Press (1995)

[Svobodova 1976] L. Svobodova, *Computer Performance Measurement and Evaluation*, Elsevier North-Holland (1976).

[Svobodova 1984] L. Svobodova, "File Servers for Network-Based Distributed Systems", *ACM Computing Surveys*, Volume 16, No. 4 (Dicembre 1984), pagg. 353-398.

[Talluri et al. 1990] M. Talluri, M. D. Hill e Y. A. Khalidi, "A New Page Table for 64-bit Address Spaces", *Proceedings of the ACM Symposium on operating Systems Principles* (1995).

[Tanenbaum 1990] A. S. Tanenbaum, *Structured Computer Organization*, Third Edition, Prentice-Hall (1990).

[Tanenbaum 1996] A. S. Tanenbaum, *Computer Networks*, Third Edition, Prentice-Hall (1996).

[Tanenbaum 2001] A. S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall (2001).

[Tanenbaum e Woodhull 1997] A. S. Tanenbaum e A. S. Woodhull, *Operating System Design and Implementation*, Second Edition, Prentice-Hall (1997).

[Tanenbaum et al. 1990] A. S. Tanenbaum, R. Van Renesse, H. Van Staveren, G. J. Sharp, S. J. Mullender, J. Jansen e G. Van Rossum, "Experiences with the Amoeba Distributed-Operating System", *Communications of the ACM*, Volume 33, No. 12 (Dicembre 1990), pagg. 46-63.

[Tate 2000] S. Tate, *Windows 2000 Essential Reference*, New Riders (2000).

[Tay e Ananda 1990] B. H. Tay e A. L. Ananda, "A Survey of Remote Procedure Calls", *Operating Systems Review*, Volume 24, No. 3 (Luglio 1990), pagg. 68-79.

[Teorey e Pinkerton 1972] T. J. Teorey e T. B. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies", *Communications of the ACM*, Volume 15, No. 3 (Marzo 1972), pagg. 177-184.

[Tevanian et al. 1987a] A. Tevanian, Jr., R. F. Rashid, D. B. Golub, D. L. Black, E. Cooper e M. W. Young, "Mach Threads and the Unix Kernel: The Battle for Control", *Proceedings of the Summer USENIX Conference* (Luglio 1987).

[Tevanian et al. 1987b] A. Tevanian, Jr., R. F. Rashid, M. W. Young, D. B. Golub, M. R. Thompson, W. Bolosky e R. Sanzi, "A UNIX Interface for Shared Memory and Memory Mapped Files Under Mach", *Technical Report*, Carnegie-Mellon University, (Luglio 1987).

[Tevanian et al. 1989] A. Tevanian, Jr. e B. Smith, "Mach: The Model for Future Unix", *Byte* (Novembre 1989).

[Thompson 1978] K. Thompson, "UNIX Implementation", *The Bell System Technical Journal*, Volume 57, No. 6, Part 2 (Luglio-Agosto 1978), pagg. 1931-1946.

[Toigo 2000] J. Toigo, "Avoiding A Data Crunch", *Scientific American*, Volume 282, No. 5 (2000), pagg. 58-74.

[Traiger et al. 1982] I. L. Traiger, J. N. Gray, C. A. Galtieri e B. G. Lindsay, "Transactions and Consistency in Distributed Database Management Systems", *ACM Transactions on Database Systems*, Volume 7, No. 3 (Settembre 1982), pagg. 323-342.

[Tucker e Gupta 1989] A. Tucker e A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors", *Proceedings of the ACM Symposium on Operating System Principles* (Dicembre 1989).

[Vahalia 1996] U. Vahalia, *Unix Internals: The New Frontiers*, Prentice-Hall (1996).

[Vee e Hsu 2000] V. Vee e W. Hsu, "Locality-Preserving Load-Balancing Mechanisms for Synchronous Simulations on Shared-Memory Multiprocessors", *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation* (2000), pagg. 131-138.

[Venners 1998] B. Venners, *Inside the Java Virtual Machine*, McGraw-Hill (1998).

[Vuillemin, 1978] A. Vuillemin, "A Data Structure for Manipulating Priority Queues", *Communications of the ACM*, Volume 21, No. 4 (Aprile 1978), pagg. 309-315.

[Wah 1984] B. W. Wah, "File Placement on Distributed Computer Systems", *Computer*, Volume 17, No. 1 (Gennaio 1984), pagg. 23-32.

[Waldo 1988] J. Waldo, "OO Systems", *IEEE Concurrency*, Volume 16, No. 3 (1988).

[Wallach et al. 1987] D. S. Wallach, D. Balfanz, D. Dean e E. W. Felten, "Extensible Security Architectures for Java", *Proceedings of the ACM Symposium on Operating Systems Principles* (1987).

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

[Williams 2001] R. Williams, *Computer Systems Architecture – A Networking Approach*, Addison-Wesley (2001).

[Wood e Kochan 1985] P. Wood e S. Kochan, *UNIX System Security*, Hayden (1985).

[Woodside 1986] C. Woodside, "Controllability of Computer Performance Tradeoffs Obtained Using Controlled-Share Queue Schedulers", *IEEE Transactions on Software Engineering*, Volume SE-12, No. 10 (Ottobre 1986), pagg. 1041-1048.

[Worthington et al. 1995] B. L. Worthington, G. R. Ganger, Y. N. Patt e J. Wilkes, "On-Line Extraction of SCSI Disk Drive Parameters", *Proceedings of the 1995 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems* (Maggio 1995), pagg. 146-156.

[Wulf 1969] W. A. Wulf, "Performance Monitors for Multiprogramming Systems", *Proceedings of the Second ACM Symposium on Operating System Principles* (Ottobre 1969), pagg. 175-181.

[Wulf et al. 19[illegible]] W. A. Wulf, R. Levin e S. P. Harbison, *Hydra/C.mmp: An Experimental Computer System*, McGraw-Hill (1981).

[Yeong et al. 1995] W. Yeong, T. Howes e S. Kille, "Lightweight Directory Access Protocol", *Network Working Group, Request for comments: 1777* (1995).

[Zabatta e Young 1998] F. Zabatta e K. Young, "A Thread Performance Comparison: Windows NT and Solaris on a Symmetric Multiprocessor", *Proceedings of the 2nd USENIX Windows NT Symposium*(1998).

[Zahorjan e McCann 1990] J. Zahorjan e C. McCann, "Processor Scheduling in Shared Memory Multiprocessors", *Proceedings of the Conference on Measurement and Modeling of Computer Systems* (Maggio 1990).

[Zhao 1989] W. Zhao (Editor), *Special Issue on Real-Time Operating Systems*, ACM (1989).

Riconoscimenti

Figura 3.9: da Iaccobucci, *OS/2 Programmer's Guide*, ©1988, McGraw-Hill, Inc., New York. Figura 1.7, p. 20. Ristampata con il permesso dell'editore.

Figura 6.8: da Khanna, Sebree, Zolnowsky, "Realtime Scheduling in SunOS 5.0", *Proceedings of Winter USENIX*, gennaio 1992, San Francisco, California. Adattata con il permesso degli autori.

Figura 6.10: adattata con il permesso di Sun Microsystems, Inc.

Figura 9.21: da *80386 Programmer's Reference Manual*, Figura 5-12, p. 5-12. Ristampata con il permesso di Intel Corporation, Copyright /Intel Corporation 1986.

Figura 10.16: da *IBM Systems Journal*, Vol. 10, No. 3, ©1971, International Business Machines Corporation. Ristampata con il permesso di IBM Corporation.

Figura 12.9: da Leffler, McKusick, Karels, Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, ©1989 by Addison-Wesley Publishing Co., Inc., Reading, Massachusetts. Figura 7.6, p. 196. Ristampata con il permesso dell'editore.

Figura 13.4: da *Pentium Processor User's Manual: Architecture and Programming Manual*, Volume 3, ©1993. Ristampata con il permesso di Intel Corporation.

Figure 15.5, 15.6, e 15.8: da Halsall, *Data Communications, Computer Networks, and Open Systems, Third Edition*, ©1992, Addison-Wesley Publishing Co., Inc., Reading, Massachusetts. Figura 1.9, p. 14, Figura 1.10, p. 15, e Figura 1.11, p. 18. Ristampata con il permesso dell'editore.

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Paragrafi dei Capp. 7 e 17 da Silberschatz, Korth, *Database System Concepts, Third Edition*, ©1997, McGraw-Hill, Inc., New York, New York. Paragrafi 13.5, p. 451-454, 14.1.1, p. 471-742, 14.1.3, p. 476-479, 14.2, p. 482-485, 15.2.1, p. 512-513, 15.4, p.517-518, 15.4.3, p. 523-524, 18.7, p. 613-617, 18.8, p. 617-622. Ristampati con il permesso dell'editore.

Figura A.1: da Quarterman, Wilhelm, *UNIX, POSIX and Open Systems: The Open Standards Puzzle*, ©1993, Addison-Wesley Publishing Co., Inc. Reading, Massachusetts. Figura 2.1, p. 31. Ristampata con il permesso dell'editore.

Le informazioni cronologiche sono state assemblate da diverse fonti, tra cui: "The History of Electronic Computing", compilata ed editata da Marc Rettig, Association for Computing Machinery, Inc. (ACM), New York, New York, e Shedroff, Hutto, Fromm, "Understanding Computers", ©1992, Vivid Publishing, distribuito da SYBEX, San Francisco, California.

Indice analitico

Ethernet 10BaseT; 50, 569

Ethernet 100BaseT; 50, 569

2PC (*two-phase commit*), protocollo; 622

50 per cento, regola del; 293

A

a basso livello (*raw*)
- disco
  - partizione; 518
  - strutture che utilizzano; 430
- I/O; 486
  - caratteristiche e utilizzo; 518
- memorizzazione, gestione dei nastri come; 539

a cascata
- annullamenti, strategie di eliminazione, nella gestione delle situazioni di stallo in sistemi distribuiti, 630
- montaggi, nell'NFS; 459
- terminazione, definizione; 111

a code multiple
- scheduling a code; 169
- scheduling con retroazione a code; 171

a doppino intrecciato
- cavi utilizzati nelle LAN, 569

a due fasi
- protocollo d'accesso bloccante; 238
- protocollo di conferma; 622, 630

a lotti, batch
- file, nell'MS-DOS; 388
- nei primi sistemi; 821
- sistema a lotti; 7
  - nello scheduling; 106

abilitazione/i
- definizione; 661
- di dati, nel sistema Cambridge CAP basato sulle abilitazioni; 667
- liste, per domìni, nella realizzazione della matrice d'accesso; 661
- logica
  - nei meccanismi di protezione basata sul linguaggio; 670
  - nel sistema Cambridge CAP basato sulle abilitazioni; 667

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

possesso della, come permesso d'accesso; 661
problematiche di revoca dei diritti d'accesso; 664
sistemi basati su; 665

abort/terminazione
di processi; 111, 270
transazione, definizione; 232

accesso
agli oggetti del nucleo, nell'API Win32; 808
ai dati, come motivazione di processi di migrazione; 565
ai file remoti; 599
alla rete, gestione dello strato di trasporto, in modelli di rete; 580
componenti del tempo di; 510
controllo
come attributo dei file; 384
dipendente dall'identità dell'utente; 417
in sistemi dinamici, monitor; 228
nel LINUX; 763
uso di parole d'ordine per; 419
diretto; 435, 480
diretto, o sequenziale, come aspetto di dispositivo; 484
diritti di
come informazione associata a un file aperto; 387
definizione; 651
revoca dei; 663
file
associazione alla memoria di; 338
meccanismi di protezione; 416
meccanismo LDAP; 413
problemi relativi ad utenti multipli; 409
tipi di; 416
intervallo di, per permettere il cambio di dominio nel MULTICS; 655
latenza di, memoria terziaria; 542
liste di, per oggetti, nella realizzazione della matrice d'accesso; 661
matrice di; 656, 660
metodi
accesso diretto; 392
accesso indicizzato; 393
accesso sequenziale; 391
ISAM (metodo ad accesso sequenziale indicizzato); 394
per files; 391
sequenziale; 436, 484

accesso bloccante, bloccaggio
a due fasi; 238
condiviso; 238, 625
gestione del protocollo sbilanciato, nei sistemi distribuiti; 628
del nucleo, supporto nel Solaris; 230
di lettura e scrittura, utilizzo nel Solaris 2; 228
esclusivo; 238, 625, 628
pagine, uso del bit di vincolo; 372
pagine vincolate alla memoria, problematiche e strategie; 371
protocolli di
nei sistemi distribuiti; 625
nelle transazioni atomiche concorrenti; 238

accesso bloccante esclusivo; 238, 625, 628

ACL (*access control list*)
vedi anche accesso; file system; sicurezza
come meccanismo di controllo nell'accesso ai file condivisi; 417
come meccanismo di protezione dell'AFS; 609

active directory; 412

adattabilità
come funzione del micronucleo del Windows 2000; 770
come obiettivo di progettazione per il Windows 2000; 771

AES (*advanced encryption standard*); 707

affidabilità
caricamento dinamico di classi non fidate, in Java; 672
classificazione della sicurezza dei sistemi di calcolo; 710
collegamento a una rete; 695
come obiettivo di progettazione per Windows 2000; 771
come vantaggio dei sitemi distribuiti; 561
come vantaggio per i sistemi a più unità d'elaborazione; 13
del file system; 416, 434
delle procedure, in Hydra; 666
delle risorse condivise, protezione delle; 649
nei sistemi RAID; 524
relazioni di fiducia nei domìni del Windows 2000; 807

AFS (*Andrew File system*)
caratteristiche del; 607
come esempio di file system distribuito; 607
copiatura nelle cache di file nel; 610
realizzazione; 611
semantica della coerenza; 415
spazi di nomi del; 608
spazio dei nomi condivisi del; 609
uso di cache da parte dei clienti nel; 607

agevolezza d'uso
come obiettivo di progettazione di un sistema operativo; 3

albero
di processi, (figura); 109
directory con struttura ad; 399
reti strutturate ad, come topologia di sistemi operativi distribuiti; 568

algoritmo con bit supplementari di riferimento; 350

algoritmo/i
assegnazione
contigua della memoria; 290
contigua, file system; 436
indicizzata, file systems; 441
assegnazione dei blocchi di memoria; 356
alternative nella sostituzione delle pagine; 357
assegnazione proporzionale; 356
assegnazione uniforme; 356
buddy-heap; 740
con bit supplementari di riferimento; 350
del banchiere; 262, 630
del fornaio; 202
dello spaccone; 639
di autenticazione; 705
di cifratura; 706
AES; 707
DES; 706
nelle parole d'ordine di protezione; 683
di controllo della concorrenza; 236
di elezione; 638
di scheduling; 160, 176
a code multiple; 169
FCFS; 160
per priorità; 165
RR; 167
SJF; 162
di scheduling del disco; 511, 513, 514, 515
di verifica della sicurezza; 263
grafo di assegnazione delle risorse; 261
memory gestione
best fit; 292
first fit; 292
worst-fit; 292
mutua esclusione; 618, 619, 621
orologio; 351
orologio con due lancette; 365

http://www.adobepdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

per evitare le situazioni di stallo; 259, 261, 262, 263, 264
per rilevare delle situazioni di stallo; 266, 269
scelta; 450
scheduling a code multiple con retroazione; 171
scheduling per sistemi con più unità d'elaborazione; 173
soluzione al problema della sezione critica; 199, 202
sostituzione delle pagine; 357
a pila; 349
basata su conteggio; 353
con bit supplementari di riferimento; 350
con memorizzazione transitoria delle pagine; 353
con seconda chance; 351
con seconda chance migliorato; 351
FIFO; 344
lettura anticipata; 453
LRU; 347, 350
ottimale; 346
rilascio indietro; 453
sostituzione delle pagine basata su conteggio; 353
tabella delle pagine a due livelli; 304

ambienti
con più unità d'elaborazione, problematiche di architetture di sincronizzazione; 204
d'elaborazione; 22
basata sul Web; 22
integrati; 23
tradizionale; 22
di processo, nel LINUX; 732
distribuito/i
DCE; 607
in IPC; 114
WWW (Web); 566
multiutente, operazioni di open e close file in; 387
paginato; 302, 310
sottosistemi d'ambiente del Windows 2000; 790
thread, TEB; 148

ampiezza di banda
del disco, definizione; 510
effettiva, memoria terziaria; 542
sostenuta, memoria terziaria; 542

amplificazione dei diritti nel sistema Hydra; 666

anello
algoritmo ad; 640
reti ad; 568, 569
struttura ad; 621, 654

anomalia di Balady
algoritmo FIFO di sostituzione delle pagine e; 346
assenza negli algoritmi a pila; 349
assenza nella sostituzione LRU; 349

anonimo
file d'accesso; 411
FTP; 563

APC (*asynchronous procedure call*)
definizione; 141
uso nel Windows 2000; 777

aperti
file, strutture di informazioni nella memoria per; 428
open-source; 607, 719
tabella dei file; 386, 494

API (*application programmer interface*)
nel Windows 2000; 808

applet
come forma di migrazione dei processi; 566

applicazione
interfaccia di I/O per le; 483
interfaccia per il programmatore, nel Windows 2000; 808
interfaccia per le, dispositivi per la memoria terziaria; 539
programmi di; 75
strato di, in modelli di rete; 580

architettura a micronucleo
come innovazione del sistema operativo Mach; 834
struttura del sistema operativo; 81
utilizzo nel Windows 2000; 770

archiviazione
creazione di copie di riserva; 455

area d'avvicendamento o di scambio
collocazione; 521
come meccanismo d'ausilio alla paginazione su richiesta; 332
definizione; 332
disco privo di struttura logica per; 430
gestione; 520
esempio, nello UNIX; 522
gestione ed uso generale nella paginazione su richiesta; 336
uso; 521

ARP (*Address Resolution Protocol*)
gestione del pacchetto; 588

Arpanet
come prima WAN; 49, 569

ASID (*address space identifiers*)
definizione; 300

asimmetrica/i
batteria; 17
multielaborazione
vedi anche multielaborazione simmetrica
definizione; 13, 174

asincrono/a
vedi anche sincronizzazione
cancellazione, definizione; 139
chiamata di procedura; 141, 777
chiamate del sistema; 489
eventi, gestiti dalle architetture delle interruzioni; 480
I/O
aspetto delle chiamate del sistema; 488
definizione; 31
utilizzo di STREAMS; 500
messaggi, definizione; 117
scritture, definizione; 453
sincroni o, come aspetto di dispositivo; 484

assegnazione
blocchi di memoria liberi, prima e dopo; (figura); 298
come algoritmo di file system; 436
concatenata, file system; 438
contigua
dei blocchi di memoria; 354
dei file, influenza sulla scelta dell'algoritmo di scheduling del; 516
della memoria; 290
della memoria; 291
globale e locale; 357
indicizzata, algoritmo, file system; 441
risorse; 227
come servizio di un sistema operativo; 64

assegnazione contigua
come algoritmo di file system; 436
con problema di frammentazione esterna; 436
della memoria; 290

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

assemblatore
a due passi, sovrapposizione di sezioni per, (figura); 286

associazione
degli indirizzi; 280

assoluto
codice; 280
nomi di percorso; 401

ATA (*advanced technology attachment*)
definizione; 39

ATA I/O, architettura per il bus; 532

Atlas, sistema operativo; 826

atomicità
come requisito di semaforo; 210
nei sistemi distribuiti; 621
uso della registrazione con scrittura anticipata; 233

atomico
architetture di controllo e modifica, per processi di sincronizzazione; 204
transazioni; 231

attacchi per rifiuto del servizio
come minaccia alla sicurezza del sistema; 693

attacco basato sull'alterazione della pila; 687

attesa
circolare; 251, 258
possesso e; 251, 256
stato del processo, definizione; 100
tempo di; 159

attesa attiva (*busy waiting*); 208, 476
nelle batterie asimmetriche; 17

attesa indefinita (*starvation*)
definizione; 166, 211
per evitare situazioni di; 618, 619, 620
stallo e; 210, 257, 631
strategia di recupero di situazioni di stallo esercitando la prelazione sulle risorse; 271

attesa limitata
mutua esclusione; 206 (figura); 206
requisito di soluzione della sezione critica; 198

attività di paginazione degenere (*thrashing*); 358
cause della; 358
modello dll'insieme di lavoro come soluzione alla; 360
modello PFF come soluzione alla; 363

attributi
dei file; 384
di directory, effetti sulla condivisione dei file; 409

autenticazione
vedi anche sicurezza
degli utenti; 681
nel LINUX; 762
nel Windows 2000, Kerberos; 793
nell'FTP; 563
nelle CIFS; 412
problematiche di identificazione dei client; 411
problematiche nel NIS; 412
secondo due fattoti, sistema di parole d'ordine monouso; 685
tramite protocollo Kerberos, nell'AFS; 607
utilizzo della crittografia per; 705

autenticazione secondo due fattori
nei sistemi a parole d'ordine monouso; 685

automatica/o
archivio, come memorizzazione quasi in linea; 538
jukebox, gestione e utilizzo; 541
memorizzazione transitoria, definizione; 118
regolazione dell'insieme di lavoro; realizzazione della memoria virtuale

automontaggio
nell'NFS; 597

avviamento
blocco di; 518
come blocco di controllo dell'avviamento per l'UFS; 428
blocco di controllo, come struttura di dati nei dischi; 428
disco di, come partizione del disco; 518
partizioni nei dischi; 431

avvicendamento dei processi; 286
vedi anche memoria, gestione della;
definizione; 107, 286
di due processi, con disco come memoria ausiliare, (figura); 287
nel LINUX; 744

azzeramento su richiesta (*zero-fill-on-demand*)
uso della copiatura su scrittura; 337

B

back door, classe delle chiamate di sistema
dispositivo d'accesso a basso livello con; 485

banchiere, algoritmo del
nei sistemi distribuiti; 630
per evitare le situazioni di stallo; 262

barriera di sicurezza (*firewall*)
come meccanismo di sicurezza di rete; 695
gestione, nel LINUX; 761

basi di dati
tecniche, uso all'interno dei sistemi operativi; 231
uso di dischi privi di struttura logica; 430

batterie di sistemi (*cluster*)
asimmetriche; 17
blocchi contigui in gruppi; 440, 449
come tecnica di configurazione dei sistemi distribuiti; 586
definizione; 17
nell'AFS; 608
riassociazione, nel file system di Windows 2000; 800
simmetriche; 17
tabelle di pagine a gruppi, definizione; 307

BeOS, sistema operativo
gestore dei thread al livello del nucleo; 136

best-fit, algoritmo di assegnazione; 292, 436

big-endian
definizione; 125

http://www.addtpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

bilanciamento del carico
vedi anche condivisione
come motivazione della migrazione dei processi; 565
definizione; 23
nella gestione dell'area d'avvicendamento; 521
per migliorare l'efficienza dell'I/O; 503

binari
semafori; 211

bit
organizzazione con bit di parità intercalati; RAID di livello 3; 528
sezionamento al livello dei, parallelismo nei sistemi RAID; 525
vettore di, realizzazione elenco dei blocchi liberi con; 446

bit busy
utilizzato dal controllore dell'I/O; 475

bit di modifica
utilizzo nell'algoritmo di sostituzione delle pagine; 341

bit di modifica (*dirty bit*)
utilizzo nell'algoritmo di sostituzione delle pagine; 341

bloccante
I/O, aspetto delle chiamate di sistema; 488
indefinita, definizione; 166
invio, definizione; 117
processi, per evitare l'attesa attiva;209
ricezione, definizione; 117

blocchi di memoria, pagine fisiche (*frames*)
vedi anche memoria, gestione della
algoritmi di assegnazione dei; 343, 356
assegnazione dei; 354
come messaggi di lunghezza costante nelle reti di comunicazioni; 576
definizione; 294
gestione dello strato di collegamento dati, nel modello di rete ISO; 579
liberi, prima e dopo l'assegnazione, (figura); 298
numero mininimo da assegnare; 355
tabella, definizione; 298

blocco/i
criteri di assegnazione dell'*ext2fs*, (figura); 752
d'avviamento; 518
del disco, dimensione; 425
difettosi; 519
dimensione dei, problematiche di copiatura dati nelle cache in un DFS; 600
ETHREAD, definizione; 148
FCB, per il mantenimento delle strutture di file; 426
indice, 441; 516
KTHREAD, definizione; 148
logici, come unità di trasferimento; 509
numero relativo, nel metodo ad accesso diretto; 392
organizzazione con blocchi di parità intercalati, RAID di livello 4; 528
organizzazione con blocchi intercalati a parità distribuita, RAID di livello 5; 529
sezionamento al livello dei, in sistemi RAID; 509
TEB, definizione; 148

blocco/i indice
definizione; 441
posizione dei, effetto sulla scelta dell'algoritmo di scheduling del disco; 515

BlueTooth, dispositivi; 15

booting
definizione; 92

Brinch Hansen, Per
soluzione applicabile alle definizioni di monitor; 226

broadcasting, diffusione
nel trasferimento di un pacchetto in Ethernet; 588

BSD UNIX, sistema operativo
4.2, utilizzo con il sistema operativo Mach; 834

buddy-heap, algoritmo
gestione della memoria fisica, nel LINUX; 740

Burroughs
sistema operativo MCP; 835

bus
definizione; 472
di espansione, definizione; 472
di un PC, tipica struttura di un, (figura); 473
I/O, definizione; 39
multiaccesso, uso nelle LAN; 569
PCI, definizione; 472
schede di I/O dette bus-mastering, utilizzo di componenti DMA nelle; 481
struttura (figura); 473

C

C-LOOK (*circular-LOOK*)
algoritmo di scheduling del disco; 515

C-SCAN (*circular SCAN*)
algoritmo di scheduling del disco; 514

C-threads
come thread al livello d'utente; 136

cache; 41, 492
block buffer, nel LINUX; 755
buffer unificata; 450, 451
coerenza; 42, 602
criteri di aggiornamento delle, 601
locazione delle; 600
per il meccanismo del servizio remoto; 599
come requisito di una FAT; 441
confronto con il servizio remoto, in file system distribuiti; 603
copiatura di file nelle, nell'AFS; 610
definizione; 37, 492
delle pagine; 450, 451
disco
differenze con un disco RAM; 453
locazioni di cache per, (figura); 454
strategie di ottimizzazione della; 450
double, definizione; 451
gestione della; 41
gestite dal Venus; 612
in file system distribuiti
invecchiamento, nel trasferimento di un pacchetto in Ethernet; 588
memoria a nuclei magnetici, nel sistema operativo Atlas, 827
per le strutture del file system tenute nella memoria; 430
su dischi da parte dei client, 601; 607
uso delle memorie di transito e uso delle; 492
utilizzo nella risoluzione del nome in indirizzo di Internet; 573

cachefs, file system
gestione di cache da parte dei client nel Solaris, 601

CAL
gestione delle abilitazioni; 664
strategie di protezione, 663

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

calcolo
accelerazione
come motivo di migrazione dei processi; 565
come ragione di processi cooperanti; 112
come vantaggio dei sistemi distribuiti; 560
migrazione delle, in sistemi operativi distribuiti; 564

Caldera
versone commerciale di LINUX; 723

cambio di contesto, 107
definizione; 49, 107
(figura); 102
impatto sulle prestazioni; 501
per evitare, come vantaggio del semaforo spinlock; 208

cambio di contesto, commutazione (*switching*)
definizione; 49
di circuito, come strategia di connessione; 576
di messaggio, come strategia di connessione; 576
di pacchetto, come strategia di connessione; 576
dominio
in MULTICS; 654
in UNIX; 653
nei meccanismi di protezione; 651
nella matrice d'accesso; 656

Cambridge CAP, sistema
esempio di sistema di protezione basato sulle abilitazioni; 667

Cambridge Digital Communication Ring
utilizzo negli intervalli di messaggi; 578

canale di comunicazione, collegamento
assegnazione; 438, 439
collegamenti effettivi, gestione del contatore dei riferimenti; 405
come tramite nello scambio di messaggi; 115
condivisione di file o directory con; 402
dinamico
vedi anche librerie condivise nella gestione di memoria e; 283
definizione; 283
nel LINUX; 748
lista, gestione dello spazio libero con; 447
risolvere il collegamento; 403
schema, gestione del blocco indice; 443
simbolico, gestione della cancellazione di un file; 404
statico; 283, 748

canali di I/O
aumento delle prestazioni con; 503

cancellazione
come problematica di file condiviso; 404
del TLB, definizione; 300
di directory, criteri di gestione; 401
di file
cicli come problematiche di prestazione nella; 406
come operazione di directory; 396
realizzazione; 386

caricamento
dei programmi utenti, nel LINUX; 746
dinamico; 283
vedi anche memoria, gestione della
definizione; 283
ed esecuzione, nei programmi di sistema; 75
nei primi sistemi; 823

cassette postali
definizione, 116
nel Windows 2000; 804

CAV (*constant angular velocity*); 510

cavalli di Troia
come minaccia alla sicurezza dei programmi; 686

CD (*collision detection*)
tecnica per evitare collisioni nelle reti di comunicazione; 577

CD-R, dischi
come memoria terziaria, 537

CD-ROM, dischi
come memoria terziaria, 537
formato di file system; 427

certificati
nel SSL; 707

checksum
Ethernet packet use; 567

chiamata di procedura
asincrona, utilizzo nel Windows 2000; 777
differita, utilizzo nel Windows 2000; 777

chiamate del sistema, 65
come interruzione; 28
comunicazione; 73
con unità a disco, informazioni contenute nella; 511
controllo dei processi; 68
controllo delle, strategia per il rilevamento di anomalie; 702
definizione; 479
gestione dei dispositivi; 72
gestione dei file; 72
gestione delle informazioni; 73
interfaccia basata sulle, come strato di realizzazione del file system; 433
meccanismo di interruzione utilizzato da; 479
memoria condivisa; 73
modello a scambio di messaggi; 73
tipi di, (tabella); 68

chiave/i
gestione delle abilitzioni; 664
meccanismo chiave-serratura, come meccanismo di protezione; 662
privata, nell'autenticazione della firma digitale; 706
uso nella crittografia; 705

cicli
delle fasi di CPU e di I/O; 155
di I/O basato sulle interruzioni, (figura); 477
falsi, come guida al rilevamento delle situazioni di stallo; 634
problemi in directory con struttura a grafo aciclico e generale; 405
sottrazione di, definizione; 481

cifratura
utilizzo di parole d'ordine; 683
vedi anche protezione; algoritmi, di cifratura; 706
AES; 707
DES; 706
simmetrica; 706

CIFS (*Common Internet File System*)
accesso alla memoria secondaria connessa alla rete tramite; 533
come servizio di nominazione distribuita; 412

cilindro
definizione; 38

circolare
attesa; 251, 258
coda, 351

client
cache gestita dal, nel protocollo NFS del Solaris; 601
cache gestita dal, nell'AFS; 607

http://www.edulpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

comunicazione nei sistemi client-server; 121
controllo di validità dei dati copiati iniziata dal; 602
definizione; 593
identificazione del, problemi di autenticazione; 411
mobilità dei, nell'AFS; 608
modello server; 411
privi di dischi, in file system distribuiti; 596
server; 15, 83
utilizzo in sistemi distribuiti; 559

CLV (*constant linear velocity*); 510

CMS (*Cambridge Monitor System*)
relazione con il CP/67; 833

coda con capacità zero
definizione; 118

coda di file, gestione asincrona dell'elaborazione e dell'I/O (*spooling*); 492, 826

coda/e
analisi delle reti, come metodo di valutazione degli algoritmi di scheduling; 179
criterio d'accodamento, nella realizzazione di semafori; 210
d'ingresso, definizione; 280
dei processi pronti
definizione; 104, 288
(figura); 104
del dispositivo; 104; 490
di messaggi, come interfaccia dei dispositivi di rete; 487
di processi, definizione; 104
di richieste relativa a un'unità a disco, come oggetto dello scheduling del disco; 511
di scheduling; 104
diagramma di accodamento
come rappresentazione dello scheduling dei processi; 105
con scheduling a medio termine, (figura); 107
(figura); 105
dispositivi I/O, (figura); 104
scheduling a code multiple; 169
scheduling a code multiple con retroazione; 171
strutture a, tornelli; 229
utilizzo nel meccanismo di STREAMS; 499

codice
assoluto, generazione del; 280
rientrante, definizione; 309
rilocabile, generazione del; 280

codice di rilevazione
utilizzato dal protocollo SCSI per la gestione degli errori; 494

codice rientrante
definizione; 309

codice rilocabile
generazione di; 280

coerenza; 42
dei dati, riferita alla gestione delle basi di dati; 231
della cache
definizione; 599
in file system distribuiti; 602
nell'AFS; 607
semantica della
dei file condivisi immutabili; 415
file system Andrew; 415
importanza nella condivisione dei file; 414
in schemi di replicazione dei file; 606
nell'AFS; 610
verifica, come strategia di ripristino di file system; 454

collegamenti simbolici
gestione nella cancellazione di un file, 404

collegamento a margherita (*daisy chain*)
definizione; 472

collisione/i
nella tabella hash; 435
rilevamento di; 577

COM
nel Windows 2000; 805

command, registro
bit command-ready nel, utilizzato dal controllore dell'I/O; 475

command-ready, bit
utilizzato dal controllore dell'I/O; 475

commit
protocollo di conferma a due fasi; 622
transazione, definizione; 232

commutazione di circuito; 576

commutazione di messaggio
come strategia di connessione; 576

compattazione
definizione; 293
della memoria contigua, nella gestione dello spazio su dischetto; 437

compilazione
associazione degli indirizzi nella; 280

compressione
dei dati, nel file system di Windows 2000; 800

comunicazione, trasmissione dei messaggi
nel sistema operativo RC 4000; 829
nel Windows 2000; 803
priva di connessione; 576

comunicazione/i
vedi anche distributo/i, IPC; reti
canale di; 115
chiamate del sistema per; 73
come servizio di un sistema operativo; 64
come vantaggio dei sistemi distribuiti; 561
costo della, come criterio di configurazione di una rete; 566
elaboratori di; 51, 570
fra calcolatori e suoi costi (figura); 502
in sistemi client-server; 121
inaffidabile, in sistemi distribuiti, gestione; 642
protocolli di; 578
reti
contesa; 577
definizione; 61
problematiche di progettazione; 572
risoluzione dei nomi; 572
strategie d'instradamento; 574
strategie di connessione; 576
strategie riguardanti i pacchetti; 576
utilizzo nei sistemi distribuiti; 559
sessioni di; 576
tra processi; 110, 108, 109

concentratore di terminali
per migliorare le prestazioni; 503

concorrente/i, concorrenza
vedi anche sincronizzazione
aumento del tasso di, per migliorare l'efficienza dell'I/O; 503
controllo della
algoritmi di, assicurazione di serializzabilità; 236
in distributed system; 605
problemi di sincronizzazione; 212
eventi, definizione; 616
processi, cooperazione tra; 112
processi, nel sistema operativo RC 4000; 829
sequenza d'esecuzione serializzabile, (figura); 237
transazioni atomiche; 235

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Concurrent C, linguaggio
nei monitor; 223

condivisione del carico
vedi anche bilanciamento del carico
definizione; 174, 560

condiviso/i, condivisione
vedi anche bilanciamento del carico
come criterio di valutazione degli algoritmi di gestione della memoria; 321
dati; 112, 338
definizione; 284
della CPU, visto come algoritmo di scheduling circolare; 168
delle risorse; 560, 649
di codice, in un ambiente paginato, (figura): 310
di directory, grafo aciclico strutturato con; 402
di file; 409
indipendente dalla locazione, come funzione di AFS; 607
problema dei lettori e degli scrittori; 213
problematiche dgli utenti multipli; 409
(figura); 315
in sistemi con memoria segmentata; 314
librerie
vedi anche memoria, gestione della
collegamento dinamico e; 283
memoria
regioni di, nel LINUX; 759
soluzione del problema dei produttori e dei consumatoricon memoria limitata tramite; 195
pagine; 309
richieste d'accesso bloccante
gestite dal protocollo sbilanciato, nel controllo della concorrenza nei sistemi distribuiti; 628
nel controllo della concorrenza nei sistemi distribuiti; 625
nelle transazioni atomiche concorrenti; 238
spazio dei nomi; 608, 609

conflitto
di operazioni, nelle sequenze di transazioni atomiche; 236
fase di, della latenza di dispatch; 176
risoluzione, meccansimo fornito dal sistema LINUX; 730
sequenze in conflitto serializzabile; 237

connessione
vedi anche reti
sockets orientate alla, nel Java; 122
strategie di, come problematica di progettazione di reti di comunicazione; 576

contabilizzazione delle risorse
come servizio di un sistema operativo; 65
informazioni di, come componente PCB; 103

contesa
come problematica di progettazione di reti di comunicazione; 577

contesto; 148

contrassegno/i
definizione; 621
metodo con passaggio di, per fornire la mutua esclusione; 621
passaggio di; 577

controllo
degli accessi
di operazioni; 416
dipendente dall'identità dell'utente; 417
diritti `control`, rappresentazione della matrice d'accesso con; 659
della concorrenza, nei sistemi distribuiti; 625
delle chiamate del sistema, come strategia di rilevamento di anomalie; 702
istruzioni di; 63
processo, chiamate di sistema per; 68
programma di, sistema operativo visto come; 5
registro `control`; 475

controllo di flusso
disposizione delle code; 499

controllore/i
dei dischi, definizione; 39
del DMA, operazioni di trasferimento; 481
di macchina, definizione; 39

convenienza
come ragione della cooperazione tra processi; 112

coordinatore
delle transazioni, nei sistemi distribuiti; 622
elezione del; 638
guasto del, gestione del protocollo di conferma a due fasi; 624

copia primaria
nel controllo della concorrenza nei sitemi distribuiti; 628

copiatura
dei dati nella memoria, riduzione del numero di, per migliorare l'efficienza dell'I/O; 503
diritti d'accesso `copy`, rappresentazione della matrice d'accesso con; 658
semantica delle; 491
su scrittura, creazione dei processi con; 336

copiatura speculare
in strutture RAID; 524
nel file system del Windows 2000; 799
RAID di livello 1; 526

copiatura speculare (*shadowing*)
nei sistemi RAID; 524

copie di riserva (*backup*)
automatico, gestione e utilizzo; 541
come strategia di recupero dei dati; 455

CP/67 operating system
partizione di tempo in; 833

CP/M, sistema operativo; 835

CPU (*central processing unit*)
vedi anche dispositivo/i; I/O; memoria; processore/i
ciclicità delle fasi di elaborazione; 155
commutazione tra i processi, (figura); 112
condivione della CPU nell'algoritmo di scheduling circolare; 168
di comunicazione, controllo dei collegamenti per le comunicazioni di una WAN; 570
multiple, algoritmi di scheduling; 173
processo con prevalenza d'elaborazione; 106
protezione della; 48
registri; 101
scheduler; 157
definizione; 106
scheduling; 155
(capitolo); 155
utilizzo, come criterio di scheduling; 159

Craftworks
versone commerciale del LINUX; 723

creazione
dei thread; 150
di un file; 385, 396, 429
di un processo; 108

crediti, algoritmo basato sui; 188

criteri
d'uso di una risorsa, rafforzamento; 650
di aggiornamento delle cache
scrittura differita; 601
scrittura diretta; 601
scrittura su chiusura; 602
distinzione dai meccanismi; 89
distinzione dai meccanismi di protezione; 650, 656, 669
Java 2, strategie di protezione; 672
matrice d'accesso, specifiche di; 656
sicurezza, TCB; 710

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

criteri di valutazione
vedi:
adattabilità
affidabilità
convenienza
disponibilità
efficienza
estendibilità
modularità
prestazione/i
robustezza
scalabilità
sicurezza

criterio d'avvicendamento pigro (*lazy swapper*); 328

criterio di scrittura differita
aggiornamento delle cache; 601

crittografia; 704
vedi anche sicurezza
uso della; 709

crollo della testina
definizione; 39

CSMA (*carrier sense with multiple access*)
come tecnica di gestione della contesa tra reti di comunicazione; 577

CTSS (*Compatible Time-Sharing System*), sistema operativo; 830

Cutler, Dave
come progettista di VMS e Windows NT; 769

D

datagramma/i
come messaggio di lunghezza costante nelle reti di comunicazione; 576
protocollo per, utilizzo nella migrazione delle computazioni; 565
trasmissione TCP; 580
trasmissione UDP; 580

dati
vedi anche file; file system; I/O
accesso ai; 565
coerenza, come correlazione nella gestione delle basi di dati; 231
compressione, nel file system Windows 2000; 800
condivisione di; 112, 338
definizione; 142
migrazione dei, nei sistemi operativi distribuiti; 564
modalità di trasferimento, come aspetto di dispositivo; 483
senza replicazione, controllo della concorrenza in sistemi distribuiti; 626
sezionamento dei, parallelismo nei sistemi RAID; 525
sezione di; 100
specifici dei thread; 142
strato di collegamento, modello di rete ISO; 579

DCE (*distributed computing environment*)
di cui fa parte il DFS Transarc; 607

Debian
versone del LINUX; 723

DEC
vedi Digital Equipment Corporation

DEC VMS
vedi sistema operativo VMS

degradazione controllata
definizione; 13

densità superficiale
definizione; 538

DES (*data encryption standard*); 706

descrittore (di file)
vedi anche file system
come componente di informazioni di stato; 605
come puntatore nella tabella dei file aperti per ciascun processo; 430

DFS (*distributed file system*)
vedi anche rete/i
(capitolo); 593
come meccanismo di file remoto condiviso; 410
definizione; 594
File System Andrew come esempio di; 607
nominazione; 595
nominazione trasparente, tecniche di realizzazione; 598
unità componente; 594
uso della cache in un; 599

di scorta (*hot spare*)
vedi anche disco
utilizzo nei sistemi RAID; 531

differita
cancellazione, definizione; 139
chiamata di procedura, uso nel Windows 2000; 777

Digital Equipement Corporation
TOPS-20 nei minicalcolatori della, *vedi* TOPS-20, sistema operativo; 542
UNIX, *vedi* Tru64 UNIX, sistema operativo

dinamico/a
associazione alla memoria; 290
associazione tra un processo e un dominio; 652, 657, 663
caricamento; 283
collegamento
vedi anche memoria, librerie condivise e gestione del; 283
definizione; 283
nel LINUX; 748
instradamento; 574
meccanismo per il controllo degli accessi; 671
problema di assegnazione della memoria; 292, 436
regolazione dei diritti, nell'Hydra; 666
rilocazione, uso di un registro di rilocazione, (figura); 283

directory
vedi anche file; file system
a due livelli; 397
a singolo livello; 397
con struttura a grafo aciclico; 402
con struttura a grafo generale; 392, 405
con struttura ad albero; 399
condivisa, strutturazione con grafo aciclico; 402
contenuti della; 395
operazioni sulla; 395
posizione delle, effetto sulla scelta dell'algoritmo di scheduling del disco; 516
protezione delle, problematiche di controllo degli accessi; 419
realizzazione; 434
struttura delle; 394, 428

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

directory corrente
definizione; 400

diretto
accesso
blocco indice come requisito per; 441
nel metodo di assegnazione contigua; 436
per file; 392
svantaggi dell'assegnazione concatenata; 439
accesso alla memoria, *vedi* DMA
blocchi, utilizzo nell'assegnazione di schema combinato; 443
comunicazione, come realizzazione del sistema di scambio di messaggi; 115

diritti
amplificazione dei, in Hydra, 666
ausiliari, in Hydra, 665
d'accesso; 651, 663
regolazione dinamica dei, in Hydra; 666

dischetto (*floppy disk*)
come memoria terziaria; 536
definizione; 39
utilizzo dell'assegnazione contigua con; 437

dischi magneto-ottici; 536

dischi ottici
come memorizzazione terziaria; 537

disco
vedi anche dispositivo/i; I/O
a cambio di fase, come memoria terziaria; 537
a lettura e scrittura, come memoria terziaria; 537
a sola lettura, come memoria terziaria; 537
andamento dei costi, (figura); 546
architettura di paginazione su richiesta; 332
blocco d'avviamento; 518
braccio del, definizione; 38
cache
differenze con un disco RAM; 453
locazioni, (figure); 454
strategie di ottimizzazione; 450
come mezzo di memorizzazione dei file, vantaggi; 425
connessione dei; 532
controllori; 39, 496
copiatura speculare, nel file system Windows 2000; 799
di basso livello; 518
di sistema, partizione del programma d'avviamento come caratteristica del; 518
dischi WORM, come memoria terziaria; 537
efficienza; 448
esiti d'insuccesso, requisiti di ripristino della memoria stabile; 535
floppy; 39, 536
formattazione del; 517, 518
gestione; 61, 517
gestione dei blocchi difettosi nei; 519
gestione dello spazio libero; 446, 447
memoria del file, problematiche di struttura e frammentazione; 391
memoria secondaria connessa alla macchina; 532
memoria secondaria connessa alla rete; 533
ottici, come memoria terziaria; 537
prestazioni; 448
RAM, definizione; 40
rimovibile; 39, 536, 542
scheduling del; 61, 510
schema funzionale, (figura); 38
settori del, creazione su di un disco magnetico; 517
sezionamento del, nel file system Windows 2000; 797

spazio
metodi di assegnazione; 435
uso efficiente dello; 448
struttura a blocchi logici dei; 509
struttura dei; 509
strutture, per file system; 464
suddivisione in partizioni; 517
tecniche di strutturazione fisica; 510

disco RAM
definizione; 40

dispatch
latenza di; 159, 175
tabella, come strutura di dati dell'I/O; 494

dispatcher; 159
definizione; 185
oggetti, nel Windows 2000; 230

disponibilità
come criterio di selezione di topologie di rete; 566
come obiettivo di progettazione delle batterie di sistemi; 17
incremento tramite la replicazione dei file, nei file system distribuiti; 606

dispositivi d'elaborazione integrati, 23

dispositivi fisici, architettura (*hardware*)
vedi anche CPU; dispositivi; I/O
adatta
per la sostituzione LRU delle pagine, 350
aspetti dei dispositivi; 483
come componente di un sistema di calcolo; 4
di paginazione; 295, 299, 301
di paginazione su richiesta; 330
di protezione; 43
di segmentazione; 312
(figura); 313
di sincronizzazione; 203
dipendenza dal cambio di contesto; 107
e dispositivi di I/O; 472, 496
interfaccia fornita dai driver dei dispositivi; 472
nei primi sistemi; 820
oggetti, definizione; 650
per registri di rilocazione e di limite, (figura); 291
protezione gestita da, vantaggi; 670
realizzazione delle primitive tramite, per migliorare l'efficienza dell'I/O; 503

dispositivo/i
vedi anche I/O
a blocchi, nel LINUX; 755
a caratteri, nel LINUX; 757
code
definizione; 104
(figura); 104
riorganizzate dallo scheduler dell'I/O; 490
di controllo, come componente del sistema di calcolo; 27
differenza di velocità tra, memorizzazione transitoria come meccanismo per la gestione della; 490
driver di
caricare su richiesta i; 497
come componente del livello di controllo I/O di un file system; 426
come meccanismo di incapsulazione; 472
definizione; 34
interfaccia, vantaggi della stratificazione dei programmi nel; 483
vantaggio nell'utilizzo di STREAMS per i; 501
gestione dei, chiamate del sistema per; 72
nomi dei, associazione dei nomi dei file ai; 496
per la memorizzazione terziaria; 536
dischetti; 536
dischi a lettura e scrittura; 537
dischi a sola lettura; 537

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

dischi magneto-ottici; 536
dischi ottici; 537
dischi WORM; 537
disco a cambio di fase; 537
gestiti da un sistema operativo; 539
interfaccia per le applicazioni; 539
successione delle realizzazioni dei servizi di I/O, (figura); 504
tabella di stato, (figura); 33
uso esclusivo dei, per flussi di dati intercalati; 492
velocità di trasferimento dei, a confronto (figura); 491

distribuito/i
vedi anche rete/i; accesso, diritti di, revoca; 664
ambienti
DCE; 607
in IPC; 114
WWW (Web); 566
coordinazione, (capitolo); 615
elaborazione, nel Windows 2000; 803
file system, *vedi* DFS (distributed file systems)
sistemi; 15
atomicità nei; 621
(capitolo); 559
definizione; 16, 61, 559
gestione delle sistuazioni di stallo nei; 630
meccanismi di coordinazione, (capitolo); 615
mutua esclusione nei; 618
ordinamento degli eventi nei; 615
problematiche di robustezza; 583
problemi di progettazione; 585
protocolli d'accesso bloccante nei; 625
schemi di controllo della concorrenza nei; 625
strutture di, definizione; 593
vantaggi dei; 560
sistemi informativi, uso di nei file system remoti; 412
sistemi operativi; 564
migrazione dei dati nei; 564
migrazione dei processi nei; 565
migrazione delle computazioni nei; 564
topologie di rete; 566

DLL (*dynamic link library*)
definizione; 148
nel Windows 2000; 771

DLM (*distributed lock manager*)
definizione; 18

DMA (*direct memory access*); 480
operazioni di trasferimento; 481
passi di un trasferimento (figura); 482
struttura; 33

DMZ (*demilitarised zone*)
come meccanismo di sicurezza della rete; 695

DNS (*domain name system*)
come servizio di nominazione distribuito per Internet; 412
ricerche del; 574
risoluzione dei nomi nel; 572

documenti ipertestuali
programmi di consultazione del Web; 16
protocollo HTTP, utilizzo sul WWW; 62

domìnio/i
cambio di
definizione nella matrice d'accesso del; 657
nei maccanismi di protezione; 652
nel MULTICS; 655
nello UNIX; 653
come meccanismi di autenticazione; 412
definizione; 651
di protezione; 650, 651
struttura dei; 651
liste delle abilitazioni per, nella realizzazione della matrice d'accesso; 661
name server, *vedi* DNS
nel Windows 2000; 806
rappresentazione della matrice d'accesso con; 656

doppia
caching, definizione; 451
funzione del doppio click, meccanismo; 402
memorizzazione transitoria, definizione; 490

DPC (*deferred procedure call*)
uso inWindows 2000; 777

DRAM (*dynamic random-access memory*)
come struttura della memoria; 35
tendenze dei prezzi della, (figura); 545

driver
vedi anche dispositivi; I/O
dispositivo; 34, 426
LINUX, registrazione dei; 729

duplice modo di funzionamento
meccanismo di protezione; 44

DVD, dischi
come memoria terziaria; 537

DVD-R, dischi
come memoria terziaria; 537

DVMA (*direct virtual memory access*); 481
vedi anche memoria virtuale

E

ECC (*error-correcting code*)
codici Reed-Solomon; 529
creazione ed aggiornamento dell'; 517
RAID di livello 2; 526

eccezione
definizione; 478
evento segnalato da; 28
gestione, *vedi anche* interruzione
nel Windows 2000; 775

effettivo
ampiezza di banda, memoria terziaria; 542
tempo d'accesso alla memoria, definizione; 301
tempo d'accesso, per una memoria con paginazione su richiesta; 334

effetto convoglio
definizione; 162

efficienza
come obiettivo di progettazione di un sistema operativo; 3
come vantaggio dello schema di I/O asincrono; 33
d'uso dei dischi; 448
di partizione; 448
del file system, con algoritmi di assegnazione e di gestione delle directory; 434
dell'I/O, metodi per migliorare la; 503
nei meccanismi di protezione; 670

EIDE (*enhanced integrated drive electronics*)
definizione; 39

elaborazione dei tracciati di verifica
come strategia per il rilevamento delle intrusioni; 699

elaborazione in tempo reale
nel LINUX, 740
gestita dal Solaris; 228
gestita dal Windows 2000; 230
nelle problematiche di memoria virtuale; 373
scheduling per; 174
sistemi; 17, 18

elezione
algoritmi di; 638

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

elezione (con passaggio di contrassegno); 578, 621

ELF (*Executable and Linking Format*), formato binario
vedi anche file system
uso nel LINUX; 746

ereditarietà (protocollo di ereditarietà delle priorità)
definizione; 176
utilizzo nei tornelli del sistema operativo Solaris 2; 230

errori
di sincronizzazione, nell'utilizzo dei semafori; 217
gestione degli, nei sottosistemi per l'I/O; 493
interfacce, rilevazione di, come ruolo della protezione; 650
rilevamento, come servizio di un sistema operativo; 64

esecutivo
come componente del sistema Windows 2000; 778

esecuzione
associazione degli indirizzi, nella fase di; 280
caricamento e, nei programmi di sistema; 75

estendibilità
come obiettivo di progettazione per il Windows 2000; 770

estensione
del nome del file, nell'incapsulazione del tipo di file; 388

estensione (*extent*)
del file, definizione; 438

Ethernet
vedi anche rete/i
10BaseT; 50
100BaseT; 50
come schema per costruire LAN; 49, 569
pacchetto di trasferimento TCP/IP; 588
pacchetto, (figura); 589
reti con passaggio di contrassegno confrontate con; 578

ETHREAD (*Executive thread block*)
definizione; 148

eventi
asincroni, gestiti dai segnali d'interruzione; 480
concorrenti, definizione; 616
gestione, *vedi anche interruzione*
nel sistema operativo Windows 2000; 230
ordinamento degli, nei sistemi distribuiti; 615
produzione di segnali d'interruzione, proprietà degli; 478
totale ordinamento degli, nei sistemi distribuiti; 615
uso di una marca temporale, negli algoritmi di mutua esclusione; 619

evitare
le situazioni di stallo, definizione; 255

***ext2fs* (*second extended file system*)**
nel LINUX; 750

F

fase
di crescita, del protocollo d'accesso bloccante a due fasi; 238
di riduzione, del protocollo d'accesso bloccante a due fasi; 238
disco a cambio di, nella memoria terziaria; 537

FAT (*file-allocation table*)
vedi anche file system
come metodo di assegnazione dei file system; 440
gestione della lista dei blocchi liberi con; 448

FC (*Fibre Channel*), architettura seriale; 532

FC-AL (*Fibre Channel-arbitrated loop*)
come variante all'FC; 532

FCB (*file control block*)
vedi anche file system
(figura); 429
mantenimento delle strutture di file tramite; 426

FCFS (*first-come, first-served*)
algoritmo di scheduling; 160
algoritmo di scheduling del disco; 511
per evitare situazioni d'attesa indefinita, nella mutua esclusione; 618

fiber
libreria, definizione, 147
ottiche, utilizzo nelle LAN; 569
thread e processi confrontati con, nel Windows 2000; 812

FIFO (*first-in, first-out*)
algoritmo di sostituzione delle pagine; 344
algoritmo di sostituzione delle pagine con seconda chance; 351
code di messaggi, nel sistema operativo Mach; 118

file
aperti; 428
apertura di; 386
associazione alla memoria dei; 338
(figura); 339
attributi dei; 384, 409
cancellazione di; 386, 396, 406
concetto di; 383
condivisione di; 409
indipendente dalla locazione; 607
problematiche degli utenti multipli relative alla; 409
contatore dei file aperti; 387
copiatura nelle cache di; 610
creazione di; 385, 396
definizione; 383
delle sessioni, semantica della consistenza nei; 415
descrittori di; 430, 605
diritti d'accesso ai, come informazione associata ad un file aperto; 387
elencazione di, come operazione di directory; 396
gestione dei; 59-60, 72, 74
identificatori di; 598, 609
lettura di; 385
maniglie di; 430, 434, 460
metodi d'accesso ai; 391
accesso diretto; 392
accesso indicizzato; 393
accesso sequenziale; 391
ISAM (metodo ad accesso sequenziale indicizzato); 394
metodo per l'assegnazione dei; 516
migrazione dei, nei file system distribuiti; 595
modifica dei, nei programmi di sistema; 74
montaggio; 458
nel nastro, gestiti dal sistema operativo; 539
nomi dei
estensione, nell'incapsulazione dei tipi di; 388
nei mezzi removibili; 540
problematiche degli utenti multipli; 409
trasformazione da indirizzi hardware a; 496
operazioni sui; 385
accesso diretto; 392
accesso sequenziale; 392
nell'AFS; 610
organizzazione di; 399, 426
protezione di; 409
remoti; 562, 599

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

replicazione dei; 595, 606
ricerca
cicli; 406
come operazione di directory; 395
problematiche di nominazione; 399
ridenominazione di; 396
riposizionamento in; 385
scrittura di; 385
servizio di, con informazioni di stato; 604
sistema server di; 16
struttura dei; 389
interna; 391
struttura di directory; 394
tabella dei file aperti per ciascun processo; 428
tipi di; 388, 389
trasferimento di; 580
troncamento di; 385

file condivisi immutabili
semantica dei; 415

file system
vedi anche AFS; CIFS; DFS; ext2fx; file; I/O; NFS; NTFS file system; file system proc; Tripwire; UFS;VFS
accesso, tipi di; 416
affidabilità; 416
ampliamento della memoria terziaria tramite; 541
attraversamento del, come operazioni su una directory; 396
con annotazione delle modifiche; 456
creazione su disco di un; 517
distribuiti, (capitolo); 593
gestione; 64
interazioni con la cache delle pagine e i driver dei dischi; 452
interfaccia, (capitolo); 383
logico; 426
meccanismi di protezione 416
metodi di assegnazione; 435
assegnazione concatenata; 438
assegnazione contigua; 436
montaggio; 406
organizzazione, (figura); 396
prestazioni; 434, 450
problematiche delle unità nastro con; 540
realizzazione; 427
(capitolo); 425
remoti; 410, 411
ripristino; 454
stratificato; 434
(figura); 427
interfaccia basata sulle chiamate del sistema; 433
virtuale; 433
struttura; 425
strutture del file system che si mantengono nella memoria, (figura); 431
strutture di dati; 427
Tripwire, come strumento per il rilevamento di anomalie; 700
validità; 407
virtuale; 432
come strato di realizzazione; 433
nel LINUX; 749
NFS integrato nel sistema operativo tramite; 462
schema, (figura); 433

file system Minix; 720

file system proc
nel LINUX; 752

firma digitale
vedi anche protezione; sicurezza
come algoritmo di autenticazione; 705
utilizzo nella JVM; 672

first-fit, algoritmo; 292
per l'assegnazione della memoria; 436

fisico/a
formattazione, dei dischi; 517
indirizzo, definizione; 282
memoria
e memoria logica; 11
gestione della, nel LINUX; 740
modello di paginazione di, (figura); 296
sicurezza; 680
spazi di indirizzi logici e; 282
strato, modello di rete ISO; 579

formattazione
del disco; 517

formattazione di basso livello
dei dischi; 517
scelta di dimensione dei blocchi logici; 509

fornaio, algoritmo del
come soluzione alla sezione critica per più processi; 202

frammentazione
vedi anche memoria, gestione della
assegnazione indicizzata come soluzione alla; 441
come criterio di valutazione degli algoritmi di gestione della memoria; 320
come problema di assegnazione contigua della memoria; 293
esterna; 292, 436
interna, definizione; 293
nei sistemi segmentati; 316
nel file su disco; 391

frammentazione esterna
come problematica dell'associazione contigua; 436
definizione; 292

frammentazione interna; 293

Free Software Foundation
progetto GNU; 722, 724

frequenza di scansione delle pagine (*scanrate*)
nella realizzazione della memoria virtuale nel Solaris 2; 365

FTP (*file transfer protocol*)
vedi anche file system
anonimo; 563
automatico, nella migrazione dei dati; 564
come meccanismo di condivisione di file remoti; 410
innovazioni legate al WWW; 62
utilizzo per il trasferimento di file in modo remoto; 563

funzione di associazione, caricamento
dei programmi nella memoria virtuale, nel LINUX; 746
di nome di file alla locazione, nei file system distribuiti; 598
tra oggetti logici e fisici, nomi visti come; 595

fuori linea
elaborazione, nei primi sistemi; 824

G

Gantt, schema di
per illustrare risultati dell'algoritmo di scheduling; 161

garbage collection
vedi anche memoria, gestione della
uso nella JVM; 87
utilizzo nella cancellazione di file; 406

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

gateway
utilizzo nelle reti di comunicazione; 575

GDT (*global descriptor table*)
definizione; 318

generazione (fork)
dei sottoprocessi; 110

gerarchia, gerarchico
delle memorie, 40
gestione; 541
paginazione; 303

gestione
dei dischi; 61, 517
dei dispositivi, chiamate del sistema per; 72
dei file; 59-60
chiamate del sistema per; 72
operazioni dei programmi di sistema; 74
dei processi; 57-58
del sistema di I/O; 60
della cache; 41
della memoria centrale; 59

Gigabit Ethernet
utilizzo in una rete di memoria secondaria; 534

giornale delle modifiche, registrazione
come strategia per il rilevamento delle intrusioni; 699
con scrittura anticipata; 233
definizione; 233
ripristino basato sulla registrazione delle modifiche; 233, 456

GPL (*General Public License*)
nel LINUX; 724

grado di multiprogrammazione
definizione; 106

grafo d'attesa
globale, (figura); 633, 634
locale
aggiornato, (figura); 637
(figura); 632, 635
utilizzato dall'algoritmo di rilevamento di situazioni di stallo; 267

grafo/i
d'attesa, utlizzo nel rilevamento delle situazioni di stallo; 266
di assegnazione delle risorse; 252, 261
directory con struttura a grafo aciclico; 402
directory con struttura a grafo generale; 405

gruppi
di pagine libere, utilizzo nella copiatura su scrittura; 337
di thread; 142
definizione; 142

gruppo
come classe di controllo degli accessi; 417
identificatori di, utilizzo nella gestione della condivisione di file; 410

guasto
di un disco, gestione del sistema Raid; 524
gestione del protocollo di conferma a due fasi; 622
del coordinatore; 624
della rete; 625
di un sito partecipante nel; 623
malfunzionamenti; 413, 534
rilevamento, nei sistemi distribuiti; 583
ripristino dopo
nei sistemi distribuiti; 584
nel Windows 2000; 778, 796

H

HAL (*hardware-abstraction layer*)
nel Windows 2000; 771, 772

handspread
realizzazione della memoria virtuale nel sistema operativo Solaris 2; 365

hard
collegamenti effettivi, gestione del contatore dei riferimenti; 405
sistemi in tempo reale stretto, definizione; 174

heap
vedi anche memoria
area d'avvicendamento utilizzata per; 336

High Sierra, formato
come formato di file system nei CD-ROM; 427

Hoare, C. A. R.
definizione dei monitor; 223, 226

HSM (*hierarchical storage management*); 541

HTTP (*hypertext transfer protocol*); 62

Hydra
esempio di sistema basato su abilitazioni; 665

I

I/O (input/output)
vedi anche dispositivo/i; file;
a basso livello; 486
caratteristiche e utilizzo; 518
accesso diretto alla memoria; 34, 480
architetture e dispositivi di; 472
asincrono, definizione; 31
associato alla memoria; 36
associato alla memoria, disposto dal controllore di dispositivo; 474
bloccante, meccanismo di chiamata del sistema; 488
bus di, definizione; 39
canale di, per incrementare le prestazioni; 503
ciclicità delle fasi di elaborazione; 155
ciclo di, basato sulle interruzioni, (figura); 477
coda del dispositivo; 104
con una buffer cache unificata, (figura); 452
controllo del, come livello del file system; 426
di un nucleo, struttura relativa a, (figura); 484
dispositivi per, caratteristiche, (tabella); 485
efficienza del, principi per il miglioramento; 503
gestione degli errori; 493
gestore dell', nel Windows 2000; 786
impatto sulle prestazioni; 501
informazioni sullo stato; 103
interfaccia di, per le applicazioni; 483
interrogazione ciclica; 475
interruzioni; 31, 476
nastri; 540
nei primi sistemi; 820
nel LINUX; 754
non bloccante, meccanismo di chiamata del sistema; 488
operazioni di, come servizio di un sistema operativo; 63
porte di, utilizzo nella memoria secondaria connessa alla macchina; 532
processo con prevalenza di; 106
programmato; 37
protezione; 45
registri; 475
relativo a un file, uso dell'associazione alla memoria; 338
scheduling; 489
senza una buffer cache unificata, (figura); 451
sincrono, definizione; 31

http://www.abbyy.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

sottosistema di; 60, 471, 495
sottosistema per la I/O del nucleo; 489
stato in, strutture di dati nel nucleo; 494
struttura; 30
struttura dei dati del nucleo per la gestione del, nello UNIX, (figura); 495
svolgimento di una richiesta di; 497, 498
trasferimenti di; 473
trasformazione in operazione dei dispositivi; 496
vincolo di; 371

Ibis, sistema operativo
replicazione dei file nel; 606

IBM
7090, caratteristiche del; 832
7094; caratteristiche del; 832
sistemi operativi, Vedi
sistema operativo CMS
sistema operativo CP/67
sistema operativo MFT
sistema operativo MVT
sistema operativo OS/360
sistema operativo TSO
sistema operativo TSS
sistema operativo VM

IDE, architettura per il bus di I/O; 532

identificatore/i, identificazione
come elemento di nominazione dei sistemi nella rete; 572
del client, problematiche di autenticazione; 411
del file
come attributo; 384
indipendenti dalla locazione, utilizzo nella nominazione trasparente; 598
nel LINUX; 763
nell'AFS; 609
dell'utente
controllo degli accessi dipendente da; 417
relativi ai gruppi; 410
dello spazio di indirizzi(ASID), definizione; 300
di processi, nei sistemi di scambio di messaggi; 115
processo; 110, 732

idle thread
definizione; 185

IDS (*intrusion detection systems*); 697

illimitata
capacità di coda, definizione; 118
memoria, problema del produttore/consumatore con; 113

impaccamento
in strutture dei file; 391

imposizione basata sul compilatore; 669

impronta della mano
nei meccanismi di sicurezza; 685

impronte digitali
nei meccanismi di sicurezza; 685

inaffidabile
comunicazione, nei sistemi distribuiti, gestione; 642
comunicazione priva di connessione; 576
processi, nei sistemi distribuiti, gestione; 642

incapsulazione
nell'interfaccia di I/O per le applicazioni; 483
sicurezza dei tipi in Java; 674

indice a più livelli, multiplo/a
delle partizioni, definizione; 291
metodo di accesso bloccante, nel controllo della concorrenza nei sistemi distribuiti; 626, 627
per la gestione dello spazio del blocco indice; 443

indicizzato
accesso; 393
ISAM; 394
assegnazione, algoritmo, file systems; 441

indiretto
blocchi, utilizzo nello schema combinato d'assegnazione; 443
comunicazione, come realizzazione del sistema di scambio di messaggi; 116
gestione delle abilitazioni; 664

indirizzo
vedi anche memoria
associazione degli; 280
d'Internet, risoluzione del nome in; 573
fisico, definizione; 282
logico, definizione; 282
procedura di gestione delle interruzioni; 478
spazio; 195, 300
traduzione, 304, 319
virtuale, definizione; 282

Infiniband
utilizzo in una rete di memoria secondaria; 534

informazione/i
condivisione di, come ragione della cooperazione tra processi; 112
di stato, nei programmi di sistema; 74
gestione, chiamate del sistema per; 73
propagazione delle, limitazione della; 660
servizi di nominazione distribuiti, utilizzo nel file system remoto; 412

inodes
assegnazione preventiva, prestazioni; 448
come FCG nell'UFS, 428
dello UNIX, (figura); 444
posizione; 430
utilizzo del contatore dei riferimenti; 404

instradamento
dinamico; 574
fisso; 574
protocollo di; 575
strategie di, come problematica di progettazione delle reti di comunicazione; 574
tabella di; 574
virtuale; 574

instradatori
utilizzo nelle reti di comunicazione; 575
utilizzo nelle reti regionali di Internet; 570
utilizzo nelle WAN; 51
utilizzo nello strato di rete, nel modello di rete; 580

intercalazione
nel disco, nel file system del Windows 2000; 797

interconnessioni FDDI basate su fibre ottiche
utilizzo nelle LAN; 569

interfaccia intermacchina
definizione; 593

interfaccia/i
di I/O per le applicazioni; 483
errori, rilevazione di, come meccanismo di protezione; 650
per il programmatore, nel Windows 2000; 808
per l'accesso ai dispositivi, driver dei dispositivi come; 472
per le applicazioni, dispositivi di memoria terziaria; 539

Internet
vedi anche rete/i
indirizzo, risoluzione del nome; 573
relazione con Arpanet; 569

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

utilizzo del DNS in; 412
WAN, collegamenti per le comunicazioni nella; 570

interprete dei comandi
come componente del sistema operativo; 63
finalità progettuali; 75

interprete delle schede di controllo; 823

interrogazione ciclica (*polling*); 475
definizione; 476

interruzione/i; 476
controllore delle; 477
definizione; 476
di I/O; 31
evento segnalato da; 28
gestione della sincronizzazione del nucleo, nel LINUX; 736
gestione delle interruzioni, definizione; 476
guidato dalle, definizione; 37
impatto sulle prestazioni; 501
linea di richiesta della, definizione; 476
livelli di priorità delle; 478
livelli di protezione delle, nel LINUX; 738
mascherabili, definizione; 478
nel Windows 2000; 775
non mascherabili, definizione; 478
riduzione delle, per migliorare l'efficienza dell'I/O; 503
vettore delle
definizione; 30, 478
del processore Intel Pentium, (figura); 479

intervalli di messaggi
come strategia di gestione della contesa di rete; 578

intrusione
caratteristiche; 696-697
rilevamento; 696-697

invecchiamento
definizione; 167
degli elementi della cache, nel trasferimento di un pacchetto in Ethernet; 588

inversione (delle priorità)
prevenzione nel Solaris 2; 230
term description; 176

IP (*Internet Protocol*)
accesso alla memoria connessa alla rete tramite; 533
pila di protocolli di rete; 580
versione PPP che funziona con connessioni realizzate tramite modem; 571

IPC (**sistema di comunicazione tra processi); 114**
vedi anche processi
nel LINUX; 758
nel Windows 2000; 812
uso del vettore nel problema produttore/consumatore; 113

IPSec
sicurezza al livello di rete; 709

IRQL (*interrupt request level*)
nel Windows 2000; 776

ISAM (*indexed sequential access method*); 394

ISO (*International Standards Organization*)
modello OSI, implementazione nel Windows 2000; 802
strati dei protocolli di comunicazione, 579 (figura); 581
struttura del messaggio di rete, (figura); 582

J

Java, linguaggio
applet, come forma di migrazione dei processi; 566
protezione; 672
thread nel; 149
uso della macchina virtuale; 87

JCL (*Job Control Language*)
nei primi sistemi; 823
nell'OS/360; 832

journaling
file system annotati, strategie di ripristino; 456

jukebox
automatico, gestione e utilizzo; 541
latenza d'accesso; 543

JVM (*Java Virtual Machine*); 87
relazioni con la macchina virtuale, thread; 151

K

Kansas, legge dello stato del
esempio di situazione di stallo; 249

Kerberos, protocollo
vedi anche protezione; autenticazione sicura, nell'AFS; 607
utilizzo nel Windows 2000; 793

Kerr, effetto
utilizzo nel disco magneto ottico; 536

KTHREAD (*Kernel thread block*)
definizione; 148

L

LAN (**rete locale); 569**
vedi anche Internet; rete/i; WAN (rete geografica)
accesso alla memoria secondaria connessa alla rete tramite; 533
definizione; 15
struttura; 50

latenza
d'accesso, memorizzazione terziaria; 542
di dispatch; 159, 175
di rotazione, definizione; 38, 510
effetto sullo scheduling del disco; 516

lavori d'elaborazione (*job*)
vedi anche processo/i
come sinonimo di processo; 100
scheduler; 8, 106
sequenzializzatore dei, nei primi sistemi; 821
uso tradizionale del termine; 99

LDAP (*lightweight directory access protocol*); 413

LDT (*local descriptor table*)
definizione; 317

lettura
anticipata, algoritmo di sostituzione delle pagine; 453
di un file; 385
dischi a lettura e scrittura, nella memorizzazione terziaria; 537
dischi a sola lettura, nella memorizzazione terziaria; 537
e scrittura, come aspetto di dispositivi; 484

LFU (*least frequently used*), **algoritmo**
sostituzione delle pagine basata su conteggio, 353

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

librerie
come componente del sistema LINUX; 725
condivise
vedi anche memoria, gestione della
collegamento dinamico e; 283
definizione, 283

linguaggi di programmazione
vedi anche Concurrent C, linguaggio; Java, linguaggio
ambienti d'ausilio, nei programmi di sistema; 74
protezione basata su; 668

LINUX, sistema operativo
autenticazione; 762
avvicendamento dei processi; 744
(capitolo); 719
caricamento ed esecuzione; 746
comunicazione fra processi; 758
confronto con UNIX; 719
controllo degli accessi; 763
driver; 729
file system; 748
gestione della memoria; 740
I/O; 754
licenze; 724
memoria virtuale; 742, 745
modello `fork/exec` dei processi; 731
moduli del nucleo; 727
multielaborazione simmetrica; 740
nucleo; 720, 720
pacchetti di distribuzione; 720, 723
paginazione; 744
principi di progettazione; 724
processi; 187, 731, 734, 738
risoluzione dei conflitti; 730
sicurezza; 762
sincronizzazione del nucleo; 735
sistema; 720, 722
strutture di rete; 759
thread nel; 148, 734

lista, elenco
abilitazioni, per domìni, nella realizzazione della matrice d'accesso; 661
d'accesso, per oggetti, nella realizzazione della matrice d'accesso; 661
elenco dei file, come operazione su directory; 396
lista concatenata, gestione dello spazio libero con; 447

Little, formula di
utilizzo nelle analisi delle reti di code; 180

little-endian
definizione; 126

locale
assegnazione, assegnazione globale e; 357
sostituzione delle pagine, come soluzione alla degenerazione dell'attività di paginazione; 359
spazio di nomi, nell'AFS; 608

località dei riferimenti
effetto sulle prestazione della paginazione su richiesta; 331

locazione
come attributo di file; 384
come requisito nella replicazione dei file; 606
del file nel disco, come informazione associata ad un file aperto; 387
delle cache dei DFS; 600
indipendente dalla
condivisione dei, come funzione dell'AFS; 607

indipendenza dalla
differenze tra la trasparenza di locazione statica e; 596
nei file system distribuiti; 595
trasparenza di, nei file system distribuiti; 595
volume, nell'AFS; 609

Locus, sistema operativo
replicazione dei file nel; 606

logico/a
blocco/i, come unità di trasferimento; 509
clock, nella realizzazione della relazione verificato-prima; 617
e spazi di indirizzi fisici; 282
elementi, utilizzo nell'accesso diretto; 392
file system, come strato del file system; 426
formattazione, dei dischi; 517
memoria; 11, 296
spazio degli indirizzi; 282

LOOK
algoritmo di scheduling del disco; 515

look-aside buffer
translation (TLB), definizione; 300

love bug, virus; 693

LPC (*local procedure call*)
nel Windows 2000; 120, 785
per raggiungere buone prestazioni; 771

LRU (*least recently used*), algoritmo
sostituzione delle pagine meno recenti con; 347
sostituzione delle pagine per approssimazione a; 350
utilizzato dal Venus per limitare le dimensioni delle cache; 612

LWP (*lightweight process*)
definizione; 133

M

MAC (*Medium Access Control*), indirizzo
utilizzo nella gestione di un pacchetto; 588

MAC (*message authentication code*)
come algoritmo di autenticazione; 705

macchina, calcolatore
adattatore, definizione; 473
controllori, definizione; 39
memoria secondaria connessa a; 532
nome di; 73, 572
utilizzo nei sistemi distribuiti; 559

Mach, sistema operativo; 833
C-threads del, come thread a livello d'utente; 136
come esempio di sistema operativo basato sullo scambio di messaggi; 118
orientamento a micronucleo; 81
protezione basata sulle abilitazioni; 662

Macintosh, sistema operativo
gestione della memoria virtuale del, uso dell'algoritmo con seconda chance migliorato; 352
gestione di tipi di file; 389
gestore di strutture di file; 390

magic number
gestione di tipi di file nel sistema operativo UNIX; 389

magnetico
disco; 37
come memoria secondaria; 36
nastro, come memorizzazione secondaria; 39

MAN (rete metropolitana)
definizione; 15

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

maniglia/e
di file; 430, 460
gestore degli oggetti nel Windows 2000; 779

marcatura temporale
nel controllo della concorrenza nei sistemi distribuiti; 628
nella realizzazione della relazione verificato-prima; 616
protocolli basati sulla, nelle transazioni atomiche concorrenti; 239
schema a ordinamento di; 630
uso nella prevenzione delle situazioni di stallo, nei sistemi distribuiti; 631

mascherabile, interruzione; 478

matchmaker
vedi anche rendezvous
definizione; 126

MCP, sistema operativo; 835

meccanismi
confronto con i criteri
nella progettazione del sistema operativo; 89
nelle strategie di protezione; 650, 656
di protezione, 662, 672
offerti dal compilatore
alle strategie di protezione; 669-670

meccanismo di generazione (*spawning*)
di processi; 110
worm, minaccia alle prestazioni del sistema; 688

media esponenziale
definizione; 163

memoria
vedi anche memoria ausiliaria, protocollo di accesso alla memoria,
a nuclei magnetici, utilizzo nel sistema operativo Atlas; 827
a semiconduttore; 40
accesso diretto alla; 480
ad accesso diretto; 35
applicazione dinamica; 290
assegnazione della; 290, 291
associazione dei file alla; 338, 486
(capitolo); 509
caricamento dei programmi nella; 746
centrale; 36
come componente di un sistema di calcolo; 28
con paginazione su richiesta; 334
condivisa
modello, chiamate del sistema per; 73
problema produttore/consumatore; 195
regioni di, nel LINUX; 759
uso della, nel problema produttore/consumatore; 113
configurazione, per un monitor residente, nei primi sistemi; 822
dinamica ad accesso diretto; 35
fisica; 11, 296, 740
frammentazione; 293
gerarchia; 40
gerarchica, gestione della; 541
gestione della; 59, 103
(capitolo); 279
nel LINUX; 740
nel Windows 2000; 813
nell'OS/360; 832
I/O associato alla; 36, 458
logica; 11, 296
memoria secondaria connessa alla rete vista come; 533
MEMS, come tecnologia futura; 538
non volatile; 40, 232
olografica, tecnologie future; 538
operazioni; 279
organizzazione con ECC, RAID di livello 2; 526
paginazione; 294
pagine residenti nella; 328

pagine vincolate alla; 371
problema di assegnazione dinamica della; 292
problema di assegnazione dinamica della; 292, 436
protezione della; 46, 290
realizzazione della; 534
registro dell'indirizzo di; 282
secondaria; 36
gestione della; 61
secondaria connessa alla macchina; 532
secondaria connessa alla rete; 533
stabili, utilizzo nelle transazioni atomiche; 233
struttura della, 35
strutture del file system residenti nella; 428, 430
(figura); 431
strutture, per file system; 428
tempo effettivo d'accesso alla; 301
terziaria
affidabilità; 544
costi; 544
dispositivi per; 536
prestazioni; 542
strutture per; 536
velocità; 542
unità di gestione (MMU), definizione; 282
virtuale
(capitolo); 325
definizione; 10, 325, 326
di rete; 599
(figura); 327
nel LINUX; 742, 745
nel sistema operativo XDS-940; 828
nel Windows 2000; 781, 813
volatile; 40, 232

memoria ausiliaria
vedi anche disco; memoria; memoria secondaria e terziaria
definizione; 286

memoria limitata
problema del produttore/consumatore con; 113, 212
regioni critiche in; 219
soluzione tramite memoria condivisa; 195

memoria secondaria e terziaria
(capitolo); 509

memoria stabile
definizione; 534
realizzazione; 534
utilizzo in transazioni atomiche; 233

memoria terziaria
vedi anche dischetto; nastri
costi; 544
dispositivi per; 536
prestazioni; 542
strutture per; 536
tecnologia futura; 538
velocità; 542

memorie non volatili
definizione; 40
utilizzo nella transazione atomica; 232

memorie volatili
definizione; 40

memorizzazione olografica
tecnologie future; 538

memorizzazione quasi in linea (*near-line storage*)
come archivio automatico; 538

memorizzazione transitoria (*buffering*)
definizione; 490
nella comunicazione tra processi; 118
uso della cache e uso della; 492

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

MEMS (*micro-electronic mechanical systems*)
tecnologie future; 538

metadati
come informazioni necessarie alla gestione dei dischi; 413

metadati (file system); 426, 457

metodo totalmente distribuito
per il rilevamento delle situazioni di stallo, nei sistemi distribuiti; 636
per la realizzazione della mutua esclusione, in un sistema distribuito; 619

mezzi
rimovibili
affidabilità; 544
come caratteristica della memoria terziaria, 536
costi; 544
prestazioni; 542
problematiche di nominazione dei file; 540
velocità; 542

MFD (*master file directory*)
vedi anche file system
come componente della directory a due livelli; 397

MFT (*master file table*)
vedi anche file system
come blocco di controllo delle partizioni per il NTFS; 428
strutture di directory contenute all'interno della, nell'NTFS; 428

MFT (nell'OS/360)
metodo di partizione nell'assegnazione della memoria; 291

MFU (*most frequently used*), **algoritmo; 353**

Michelangelo, virus; 692

microrete; 15

Microsoft Windows NT
vedi Windows 2000, sistema operativo; Windows NT, sistema operativo

migrazione
dei dati, nei sistemi operativi distribuiti; 564
dei file nei file system distribuiti, 595
dei processi nei sistemi operativi distribuiti; 565
delle computazioni, nei sistemi operativi distribuiti; 564
delle funzioni, nella storia dei sistemi operativi; 21

minidischi
vedi anche partizione
nel sistema operativo VM; 84

MIPS, architetture
strategia di gestione del TLB; 369

MMU (*memory-management unit*)
definizione; 282

mobilità
degli utenti, nei problemi di progettazione dei sistemi distribuiti; 585
dei client, nell'AFS; 608

modelli
client-server; 411
deterministici, come metodo di valutazione analitica; 177
di comunicazione, chiamate del sistema per; 73
di programmazione multithread; 136

modelli deterministici
come metodo di valutazione analitica; 177

modello di programmazione multithread da uno a uno; 137

modello dell'insieme di lavoro; 360, 363, 364

modello di programmazione multithread da molti a molti
nella JVM; 151

modello di programmazione multithread da molti a uno
nella JVM; 151

modello fork/exec dei processi
nella gestione dei processi nel LINUX; 731

modem
definizione; 52

modi di funzionamento
bit di modo; 44
come meccanismo di protezione; 44
d'utente; 44
del supervisore; 44
modo monitor; 44
privilegiato; 44

modifica
dei file, nei programmi di sistema, 75

modo del supervisore
nel duplice modo di funzionamento; 44

modularità
come ragione della cooperazione tra processi; 112
come vantaggio nell'utilizzo di STREAM, 501
come vantaggio per il metodo stratificato; 79

molteplicità degli utenti
problema relativo alla condivisione dei file; 409

monitor
con variabili `condition`, (figura), 224
modo, in duplice modo di funzionamento; 44
per l'assegnazione di una singola risorsa, (figura); 227
realizzazione, uso di semafori, 225
residenti; 43, 821
schema, (figura); 223
sintassi di, (figura); 222

monitor residente
definizione; 43
nei primi sistemi; 821

montaggio
dei file, 458
della partizione radice; 432
di un file system; 406
partizioni e; 430
punto di; 406, 801
tabella di; 432, 496

Morris, Robert Tappan
programma worm; 688

MS-DOS, sistema operativo
come sistema operativo Intel a 16-bit; 835
come sottosistema d'ambiente del Windows 2000; 792
configurazione del disco nell', (figura); 519
file; 388, 390
struttura; 77

MULTICS, sistema operativo; 831
gestione delle abilitazioni; 664
meccanismo di protezione; 654
relazione con il CTSS; 831
strategie di protezione; 663

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

multielaborazione simmetrica; 13, 740

multiprogrammazione
di partizioni multiple; 291
grado di, definizione; 106
in relazione con la gestione asincrona dell'elaborazione e dell'I/O; 826
sistemi; 9

multithread, modelli di programmazione
da molti a uno; 137
da uno a uno; 137

multithread, programmazione
vedi anche concorrenza; thread
definizione; 133
modelli di; 136
nel Solaris 2; 228
nucleo del Windows 2000, 230

MUP (*multiple universal-naming convention provider*)
nel Windows 2000; 805

mutex (semaforo)
vedi anche semaforo/i; sincronizzazione/sincrono
adattivo, utilizzo nel Solaris 2; 229
definizione; 207
nella soluzione del problema dei lettori e degli scrittori; 214
nella soluzione del problema dei produttori e consumatori con memoria limitata; 212
realizzazione di esclusione con semafori, (figura); 208
utilizzo di oggetti dispatcher nel Windows 2000; 230
utilizzo nella regione critica; 220

mutua esclusione
attesa limitata, uso dell'istruzione `TestAndSet`; 206
come condizione necessaria per una sistuazione di stallo; 251
come requisito della soluzione del problema della sezione critica; 197
(figura); 205
in un sistema distribuito; 618
metodo centralizzato; 618
metodo con passaggio di contrassegno; 621
metodo totalmente distribuito; 619
mutex, relazioni con; 207
realizzazione con l'istruzione TestAndSet; 204
realizzazione di un semaforo; 208, 225

mutuamente sospette/i
classi, soluzione nel Java al problema delle; 672
sottosistemi, soluzione di Hydra al problema dei; 666

MVS, sistema operativo (OS/360)
memoria virtuale nel; 833

MVT, nel sistema operativo (OS/360)
metodo di assegnazione della memoria a partizioni nel OS/360; 291

N

NAS (*network-attached storage*); 533

nastro/i; 540
danno dei costi, (figura); 546
come memorizzazione terziaria, 537
file nel, gestiti dal sistema operativo; 539
latenza d'accesso dei; 542
magnetici, come memorizzazione secondaria; 39
trace tape, utilizzo nella simulazione; 180

NDIS (*network device interface specification*); 802

negoziazione
come meccanismo di rilevamento dei guasti, nei sistemi distribuiti; 583
definizione; 475
tra il DMA e il controllore del dispositivo; 481

nel sistema Windows NT; 364

NetBEUI (*NetBIOS Extended User Interface*)
SMB, condivisione dei file di rete; 564

NetBIOS (*network basic input/output system*)
nel Windows 2000; 802

NetWare, protocolli della Novell; 803

NeXT, sistema operativo
utilizzo del nucleo Mach nel; 835

NFS (*network file system*); 458
accesso alla memoria secondaria connessa alla rete tramite; 533
architettura del, schema, (figura); 463
come server senza informazioni di stato; 604
funzionamento remoto; 464
innovazioni legate al WWW; 62
metodi di autenticazione, 411
migrazioni di dati nel; 564
montaggio dei file nel; 458
protocollo; 461
protocollo di montaggio; 461
schemi di nominazione; 597
traduzione dei nomi di percorso; 462
uso di cache nella memoria da parte del client, nel Solaris; 601

NIS (*network information service*)
come servizio di nominazione distribuita; 412

NLS (*national language support*)
nel Windows 2000; 772

nome/i/ nominazione
del file
come attributo di; 384
incapsulazione del tipo di file nel; 388
nei mezzi rimovibili; 540
problematiche degli utenti multipli; 409
della transazione, come campo del giornale delle modifiche; 233
di percorso
definizione; 398
nomi di percorso relativi confrontati con quelli assoluti; 401
traduzione NSF dei; 462
DNS
ricerche, 574
risoluzione dei nomi nel; 573
metodo WINS, risoluzione dei nomi nelle reti TCP/IP tramite; 807
nei file system distribuiti; 595
nel sistema di scambio di messaggi; 115
nella convenzione UNC, nel Windows 2000; 803
pipe con, nel Windows 2000; 803
problema delle collisioni dei, in directory a singolo livello; 397
problematiche di progettazione delle reti di comunicazione; 572
risoluzione dei
gestione della pila di protocolli TCP/IP; 587
nelle reti TCP/IT; 807
problematiche di progettazione di reti di comunicazione; 572
schemi; 597, 598
server dei, nel DNS; 573
servizi, distribuiti, utilizzo nel file system remoto; 412

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

spazio di
condivisi, nell'AFS; 608, 609
locali, nell'AFS; 608
nell'AFS; 607
unico, nome di percorso come meccanismo per, nella directory a due livelli; 398

nomi di percorso
assoluti, nomi di pecorso relativi confrontati con; 401
con chiamata del protocollo NSF, traduzione dei; 462
definizione; 398
relativi, nomi di pecorso assoluti confrontati con; 401

non bloccante, asincrono
invio, definizione; 117
messaggi, definizione; 117
ricezione, definizione; 117

non in linea
memorizzazione, confrontata con la memorizzazione quasi in linea; 538

non mascherabile, interruzione; 478

NSFnet
come rete regionale di Internet; 570

NTFS, file system
come file system del Windows 2000; 793
compressione dei dati; 800
del Windows 2000, aspetti di affidabilità, 771
gestione dei volumi; 797
punti d'interpretazione; 801
ripristino nel; 796
sicurezza nel; 797
tolleranza ai guasti; 797

nucleo
accesso agli oggetti del, nell'API Win32; 808
come componente di sistema del Windows 2000; 494
controllo del flusso, utilizzo di meccanismi delle interruzioni nel; 480
I/O del, architettura stratificata, (figura); 484
LINUX; 720
come componente del sistema; 725
moduli; 727
sincronizzazione, nel LINUX; 735
sottosistema per l'I/O del; 489
strutture di dati del; 494

numero di vnode
come struttura di rappresentazione dei file del VFS; 434
nell'AFS; 609

NVRAM (*non-volatile RAM*)
utilizzo nel RAID di livello 3; 528
utilizzo nella memoria stabile; 535

O

ONC+ (*Open Network Computing*)
NFS, componente del; 458
schema di nominazione nell'NFS; 597

ordine degli accessi (*race condition*); 197

orfani, rilevamento ed eliminazione
nel servizio di file con informazioni di stato; 605

orologio con due lancette, algoritmo
realizzazione della memoria virtuale nel Solaris 2; 365

orologio/i
caratteristiche e meccanismi delle chiamate del sistema per; 487
logico, nella realizzazione della relazione *verificato-prima*; 617

OS/2, sistema operativo
come sottosistema d'ambiente del Windows 2000; 793

OS/360, sistema operativo; 832

OS/MFT, sistema operativo; 832

OS/MVT, sistema operativo; 832

OSF (*Open Software Foundation*)
DCE, di dui fa parte il DFS Transarc; 607
utilizzo del Mach 2.5; 834

OSI (*Open Systems Interconnection*), **modello**
nel Windows 2000; 802

ottimale, ottimizzazione
algoritmo di sostituzione delle pagine; 346
come caratteristica di algoritmo di scheduling; 163
prestazioni, file system; 450
problematiche di assegnazione di spazio del disco; 444

P

P, operazione per accedere ad un semaforo; 207

pacchetto
come messaggi di lunghezza costante nelle reti di comunicazioni; 576
commutazione di, come strategia di connessione; 576
definizione; 129
Ethernet, (figura); 589
trasferimento di, (esempio); 587

pagina/e
ambiente, condivisione di codice in, (figura); 310
frequenza della, influenza sull'esecuzione della paginazione su richiesta; 335
gestione della sincronizzazione del nucleo, nel LINUX; 736
gestione nella memoria virtuale dell'eccezione di; 329
modello PFF (page-fault frequency),come soluzione della degenerazione di paginazione; 363
tempo di servizio della, operazioni principali; 335
cache delle; 450, 451
condivise; 309
dimensione delle, problematiche nella scelta; 367
memorizzazione transitoria delle; 353
numero di, definizione; 294
paginazione con priorità, realizzazione della memoria virtuale nel Solaris 2; 365
protezione; 302
scostamento di, definizione; 294
sostituzione delle; 339
algoritmi di; 451, 452
algoritmi di assegnazione globale e locale dei blocchi di memoria; 357
algoritmo ottimale di; 346
basata su conteggio, algoritmo di; 353
con bit supplementari di riferimento, algoritmo di; 350
con memorizzazione transitoria, algoritmo di; 353
con seconda chance, algoritmo di; 351
con seconda chance migliorato, algoritmo di; 351
FIFO, algoritmo di; 344
LRU; 347, 350
necessità di, (figura); 341
strategia di base; 340
vincolate alla memoria, problematiche e strategie; 371

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

paginazione; 294
con TLB, (figura); 301
algoritmo pageout, realizzazione della memoria virtuale nel Solaris 2; 365
architettura di; 299
(figura); 295
definizione; 294
esempio, (figura); 297
gerarchica; 303
memoria, trasferimento nello spazio di un disco, (figura); 329
memoria virtuale, utilizzo del meccanismo delle interruzioni per la gestione della; 479
metodo di base; 294
modello di, di memoria logica e fisica, (figura); 296
nel LINUX; 744
nel sistema operativo Tenex; 835
nel sistema operativo XDS-940; 828
paginatore, definizione; 328
realizzazione della memoria virtuale nella forma della; 327
segmentazione con; 317, 317
strategia di prepaginazione; 366
su richiesta; 328
concetti fondamentali; 328
impatto sullo scheduling del disco; 516
problematiche nella prestazione della; 334
sostituzione delle pagine in relazione con; 343
uso dell'area di avvicendamento nella; 521

paginazione su richiesta; 328
concetti fondamentali; 328
impatto sullo scheduling del disco; 516
problematiche di prestazioni; 334
realizzazione della memoria virtuale tramite, 327
sostituzione delle pagine in relazione alla; 343

PAM (*pluggable authentication modules*)
nel LINUX; 763

parità
nei RAID; 528, 529

parola d'ordine
cifratura; 683
come meccanismo di autenticazione degli utenti; 681
come meccanismo di controllo degli accessi; 419
monouso; 684
sicurezza mediante; 681

partizione
blocco di controllo della, come struttura di file system nei dischi; 428
come strutture del disco; 394
definizione; 291
dei dischi; 417
montaggio e; 430
radice, come struttura nel disco; 432
settore d'avviamento della, come blocco di controllo dell'avviamento per il NTFS; 428
tabella delle, come struttura di informazioni di gestione di file system nella memoria; 428

PCB (*process control block*); 101
contesto del processo registrato in; 108
(figura); 102
uso nell'implementazione di un semaforo; 210

PDA (*personal digital assistant*)
come sistema palmare; 20

Pentium, architettura
strategia di gestione del TLB nel; 369
tabella delle pagine ad associazione diretta nel; 305

personalità
nel LINUX, identità dei processi; 732

PFF (*page-fault frequency*), **modello; 363**

piatti
definizione; 37

pila
algoritmi a; 349
area d'avvicendamento utilizzata per; 336
come componente di processo; 100
definizione; 100
dei protocolli di rete
OSI, (figura); 581
TCP/IP; 580
TCP/IP, (figura); 582
ispezione della, nel linguaggio Java 2; 673
overflow, attacco basato sull'alterazione della pila; 687
realizzazione dell'algoritmo LRU con; 349

PIN (*personal identification number*)
nei meccanismi di parole d'ordine monouso; 684-685

PIO (*programmed I/O*)
definizione; 37, 480

pipe
half-duplex, come interfaccia dei dispositivi di rete; 487
meccanismo per il passaggio di dati, nel LINUX; 758
utilizzo nel meccanismo STREAMS; 499

Plug-and-Play (*PnP*), gestore del sistema
nel Windows 2000; 789

porta/e
definizione; 37, 116, 472
di I/O
accesso alla memoria secondaria connessa alla macchina dalle; 532
indirizzi delle porte dei dispositivi, (figura); 474
indirizzo di, utilizzo nell'UDP; 581
nella RPC; 125

POSIX.1, standard
supporto del Windows 2000; 791

POSIX.1b, standard
implementazione nel LINUX; 738

POSIX.1c, standard
implementazione nel LINUX; 735

POSIX, standard
come sottosistema d'ambiente del Windows 2000; 776, 792
Pthreads; 143
come thread al livello d'utente; 136

possesso e attesa (*hold and wait*)
come condizione necessaria per le situazioni di stallo; 251
nella prevenzione delle situazioni di stallo; 256

PowerPC, architettura
strategia di gestione dei TLB; 369

PPP (*point-to-point protocol*)
come protocollo delle WAN; 571

PPTP (*point-to-point tunneling protocol*)
nel Windows 2000; 803

prelazione, diritto di prelazione
algoritmo SJF, definizione; 164
scheduling; 157
definizione; 158
su risorse, come strategia di gestione delle situazioni di stallo; 271
utilizzo nell'algoritmo di scheduling RR; 168

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

prenotazione
delle risorse, definizione; 175
uso esclusivo dei dispositivi, code di file come tecnica per flussi di dati intercalati; 492

prepaginazione
come strategia di paginazione; 366

presentazione, strato di; 580

prestazione/i
vedi anche costi; affidabilità
come criterio di valutazione degli algoritmi di gestione della memoria, 320
come obiettivo di progettazione per il Windows; 770
come ragione della cooperazione tra processi; 112
come vantaggio di un servizio con informazioni di stato; 604
controllo della concorrenza dei sistemi distribuiti, problematiche del coordinatore singolo; 626
dei file system distribuiti, criteri di valutazione; 594
del file system; 434, 450
dell'algoritmo FIFO di sostituzione delle pagine; 344
della memoria terziaria; 542
della memoria virtuale, problematiche di attività di paginazione degenere; 358
della paginazione su richiesta; 334
delle reti di comunicazione, effetto sulla contesa; 577
effetto dell'algoritmo di mutua esclusione totalmente distribuito sulle; 619
effetto dell'I/O sulle; 501
effetto sulla copiatura di file nelle cache; 610
impatto sul cambio di contesto; 108
influsso positivo della replicazione di file nei file system distribuiti; 606
miglioramento tramite sistemi RAID; 525
MULTICS, meccanismi di protezione, impatto su; 656
problematiche di assegnazione dei dispositivi d'accesso; 444
problematiche di, nei primi sistemi; 820
problematiche di, nelle directory con struttura a grafo generale; 405
procedure di bloccaggio del nucleo, nel Solaris 2; 230
ripristino basato sulla registrazione delle modifiche, effetto sulle; 233

prevenire
le situazioni di stallo, definizione; 255
le situazioni di stallo, nei sistemi distribuiti; 630

primi sistemi
caricatore; 823
compilatori; 820
configurazione della memoria; 822
dispositivi; 820
I/O; 820
monitor residente; 821
programmazione; 820
programmi; 820

principio del privilegio minimo
definizione; 651
meccanismi di protezione; 652
nel MULTICS; 655

priorità
algoritmo di sostituzione per, come tentativo di soluzione all'attività di paginazione degenere; 359
inversione delle; 176, 230

livelli di priorità delle interruzioni; 478
numbero di; 227
negli algoritmi di elezione; 639
paginazione con, come realizzazione della memoria virtuale del Solaris 2; 365
protocollo di ereditarietà delle; 176, 230
scheduling per; 165

priva di connessione
comunicazione; 576
socket, nel Java; 122

privi di dischi
client, nei file system distribuiti; 596

privilegiate/o
istruzioni, in duplice modo di funzionamento; 44
modo, in duplice modo di funzionamento; 44

problema dei cinque filosofi; 215
vedi anche sincronizzazione
per evitare situazioni di stallo, uso dei monitor per; 223

problema dei generali bizantini
in sistemi distribuiti; 641

problema dei lettori e degli scrittori (figura); 214

problema del produttore/consumatore
come paradigma per processi cooperanti; 112
questioni sulla sincronizzazione dei processi; 195

problema della reclusione; 660

procedura di autoverifica
nel LINUX; 730

processi demoni
definizione; 73
utilizzo nei meccanismi di protezione; 653

processo/i
vedi anche IPC; lavori d'elaborazione
nel LINUX; 721; 758
nel Windows 2000; 812
albero di, (figura); 109
avviamento, con paginazione su richiesta; 328
blocco di controllo di un processo; 101
(capitolo); 99
coda d'ingresso dei; 280
combinazione di; 106
componenti di; 100
comunicazione tra; 113, 114
concetto di; 99
contesto; 733
commutazione, (figura); 102
definizione; 107
controllo, chiamate di sistema per; 68
cooperanti; 112, 195
creazione di; 108, 336
definizione; 10, 57, 99, 731
difettosi, nei sistemi distribuiti, gestione; 642
differenze con i programmi; 100
elementi dell'identità di; 732
figlio; 108
terminazione da parte del genitore; 111
genitore; 108
gestione dei; 57-58, 785, 809
identificatore di; 110
in primo piano, nel Windows 2000; 187
utilizzo di scheduling a code multiple nei; 169
in sottofondo, nel Windows 2000; 187
nello scheduling a code multiple; 169
in tempo reale, problematiche della memoria virtuale; 373
leggero, definizione; 133
migrazione dei, nei sistemi operativi distribuiti; 565
nei domìni di protezione; 651
nel LINUX, gestione dei; 731

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nome di; 73
operazioni sui; 108
produttore
problema dei produttori e consumatori con memoria limitata, (figura); 213
scheduling; 103
diagramma di accodamento; 105
modelli di; 182
nel LINUX; 738
sincronizzazione dei
(capitolo); 195
semafori; 207
soluzione per più processi al problema della sezione critica; 202
stato del; 100, 101
strutture di; 198, 199, 200, 201, 203
tabella dei file aperti per ciascun processo; 428
terminazione di; 110, 270
threads e; 734

produttività
come criteri di scheduling; 159
come vantaggio per i sistemi con più unità d'elaborazione; 12
definizione; 159

progettazione
di reti di comunicazione; 572
problemi di, nei sistemi distribuiti; 585
sistema operativo
distinzione tra meccanismi e criteri; 89
e realizzazione; 88
LINUX; 724
scopi; 88
struttura; 77
Windows 2000; 770

progetto GNU
General Public License, nel LINUX; 724
relazioni col LINUX; 722

programma d'avviamento, 92
avviamento del sistema; 27
memorizzazione su disco del; 518
scopo; 92

programma swatch
rilevamento delle intrusioni tramite, 700

programma/i
contatore
come componente di un blocco di controllo di un processo; 101
come componente di un processo; 100
definizione; 100
uso nella gestione dei processi; 58
dal punto di vista utente, (figura); 311
di sistema; 74
differenze con i processi; 100
esecuzione di, come servizio di un sistema operativo; 63
minacce alla sicurezza; 686
struttura dei, effetto sulla realizzazione della memoria virtuale; 370

programmi per le chat
come esempio di IPC; 114

pronto
coda dei processi pronti
definizione, 104, 288
(figura); 104
stato del processo, definizione; 100

propagazione
dei diritti di accesso, limitazione; 660
delle informazioni, limitazione; 660

protezione
vedi anche autenticazione
abilitazione; 661, 665
architetture di; 43
duplice modalità di funzionamento; 44
basata sul linguaggio; 668
(capitolo); 649

come attributo del file; 384
come criterio di valutazione degli algoritmi di gestione della memoria; 321
come servizio di un sistema operativo; 65
definizione; 62
dei file, problema relativo agli utenti multipli; 409
del file system; 416
dell'I/O; 45
della CPU; 48
della memoria; 46, 290
delle risorse condivise; 649
domìni; 650
imposizione basata sul compilatore; 669
in un ambiente paginato; 302
meccanismo chiave-serratura; 662
nel linguaggio Java 2; 672
nell'AFS; 609
nella segmentazione; 314
nucleo di protezione e compilatore; 669
scopi della; 649
sicurezza; 649, 679
sistema di, come componente di un sistema operativo; 62
sottosistemi mutuamente sospetti, soluzione di Hydra al problema dei; 666

protocollo/i;
AppleTalk, nel Windows 2000; 803
ARP, utilizzo nella gestione di un pacchetto; 588
basati sulla marcatura temporale, in transazioni atomiche concorrenti; 239
crittografico; 707
d'accesso bloccante
a due fasi; 238
in transazioni atomiche concorrenti; 238
nei sistemi distribuiti; 625
d'instradamento; 575
dell'operazione di montaggio; 461
di comunicazione; 578
di conferma a due fasi
nei sistemi distribuiti; 622
nello schema a ordinamento di marche temporali; 630
di ereditarietà delle priorità; 176, 230
di maggioranza, nel controllo della concorrenza nei sistemi distribuiti; 627
di rete
nel Windows 2000; 802
supporto di LINUX; 760
di trasporto; 581
DLC
nel Windows 2000; 803
FTP; 562
IP
accesso alla memoria secondaria connessa alla rete tramite; 533
nel Windows 2000; 802
pila di protocolli di rete; 580
PPP come versione che funziona con connessioni realizzate tramite modem; 571
LDAP; 413
NetWare della Novell, nel Windows 2000; 803
NFS; 461
migrazione di dati nel; 564
per datagrammi, utilizzo nella migrazione delle computazioni; 565
per l'accesso alla memoria secondaria, memoria connessa alla rete vista come; 533
sbilanciato, nel controllo della concorrenza nei sistemi distribuiti; 627
SCSI
caratteristiche dell'architettura a bus; 532
codice di rilevazione del; 494
definizione; 473
SMB, condivisione dei file in una rete; 564
SSL; 707
TCP; 580, 802
UDP; 565, 580
Virtue, nell'AFS; 608

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

PTBR (*page-table base register*)
definizione; 299

Pthreads; 143
come thread al livello d'utente; 136

PTLR (*page-table length register*)
definizione; 302

puntatore
come problema di file condiviso, 404

puntatore alla posizione corrente del file
in lettura e scrittura di files; 385

puntatori
al file, come informazione associata a un file aperto; 387
come collegamenti di file system; 404
dimensioni dei, considerazioni sull'uso efficiente di un disco; 449

punti d'interpretazione
nel file system di Windows 2000; 801

punti di verifica
definizione; 235
uso dei, nel ripristino basato sulla registrazione delle modifiche; 234

Q

QNX, sistema operativo
orientamento a micronucleo; 82

quanto
di tempo, definizione; 49

R

raggruppamento
come modifica dell'elenco dei blocchi liberi; 448

RAID (*redundant arrays of independent disks*); 526, 527, 528, 529
disco privo di struttura logica come contenitore per; 430
livelli, compromessi tra costi e prestazioni; 526
nel ripristino di malfunzionamenti, utilizzo di; 413
scelta di un livello, criteri di; 531
strutture e utilizzi; 524

RAM (*random-access memory*); 35, 453

rampino d'arrembaggio (*grappling hook*)
uso nei worm; 689

RC 4000, sistema operativo; 829

realizzazione
del file system; 427
(capitolo); 425
dell'algoritmo LRU; 348
della relazione verificato-prima; 617
delle directory; 434
di scheduling, algoritmi; 181
di strutture di nominazione nei file system distribuiti; 598
macchine virtuali; 84

Red Hat
versione commerciale del LINUX; 723

Reed-Solomon, codici
utilizzo nel RAID di livello 6; 529

regione critica
vedi anche sincronizzazione
come soluzione a errori di sincronizzazione; 217
definizione; 218
realizzazione, (figura); 221

registrazione con scrittura anticipata; 233

registri
d'istruzione, architettura di von Neumann; 35
di base, protezione della memoria; 47
di limite; 47, 290
di lunghezza della tabella delle pagine (PTLR), definizione; 302
di rilocazione; 282, 290

registro d'istruzione; 35

registro di base
protezione della memoria; 47

reindirizzamento
nel Windows 2000; 805

relativo
accesso, *vedi* accesso diretto
nomi di percorso, confrontati con i nomi di percorso assoluti; 401
numero di blocco, per i metodi di accesso diretto; 392
velocità, esecuzione di processo a; 198

relazione verificato-prima
nell'ordinamento degli eventi; 616
realizzazione; 617

remoto
accesso ai file; 599
file system; 410, 411, 412, 413
funzionamento, NFS; 464
servizio
confronto con l'uso delle cache, nei file system distribuiti; 603
meccanismo del, accesso ai file remoti tramite; 599
sessioni di lavoro; 562
strato di applicazione per la gestione delle, modello di rete OSI; 580
trasferimento di file; 562
unità d'elaborazione, utilizzo nei sistemi distribuiti; 559

rendezvous
vedi anche matchmaker
definizione; 117

replicazione
dei file,
nei DFS trasparenti; 595
nei file system distribuiti; 606
per la sola lettura, nell'AFS; 610

rete locale
vedi LAN

rete parzialmente connessa
come topologia di sistemi operativi distribuiti; 568

rete totalmente connessa; 567

rete/i, interconnessione
a stella, come topologia di sistemi operativi distribuiti; 568
ad anello, come topologia di sistemi operativi distribuiti; 568
come componente di un sistema operativo; 61
comunicazioni fra calcolatori, (figura); 502
definizione; 15
di comunicazione
progettazione; 572
utilizzo nei sistemi distribuiti; 559
dispositivi di, caratteristiche, meccanismo delle chiamate del sistema per; 487

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

esempio di comunicazione in; 587
LAN; 50
memoria secondaria connessa alla; 533
nel Windows 2000; 801
parzialmente connessa, come topologia di sistemi operativi distribuiti; 568
protocolli di; 501, 802
sicurezza, 680
sistema operativo di; 562
definizione; 16
strati, modello di rete ISO; 579
struttura; 50, 759
strutturata ad albero, come topologia di sistemi operativi distribuiti; 568
strutture, sistemi distribuiti, (capitolo); 559
tipi di; 568
topologie; 566, 567
totalmente connessa, come topologia di sistemi operativi distribuiti; 568
traffico di, impatto sulle prestazioni; 501
WAN; 51

revoca dei diritti d'accesso; 663

ricerca
di un file
cicli di, come problematica di prestazione per; 406
come operazione su directory; 395
problematiche di nominazione; 399
percorso di, definizione; 399
svantaggio della, in una lista lineare; 434

ridondanza
come fattore di affidabilità nei sistemi distribuiti; 561
nei sistemi RAID; 524

riferimento/i
bit di, come architettura adatta per la sostituzione delle pagine per approssimazione a LRU; 350
contatore dei, come meccanismo di cancellazione del file; 404
nel linguaggio Java 2, aspetti di protezione della memoria; 674
successione dei, utilizzo nella valutazione dell'algoritmo di sostituzione delle pagine; 343

rilascio indietro (*free-behind*)
algoritmo di sostituzione delle pagine; 453

rilevamento
dei guasti, nei sistemi distribuiti; 583
delle intrusioni; 696
delle situazioni di stallo; 266, 269, 632

rilevamento di anomalie; 697

rilocazione
come criterio di valutazione degli algoritmi di gestione della memoria; 320
registro di; 282, 290
rilocazione dinamica tramite un registro di, (figura); 283

rimovibile
mezzi
affidabilità; 544
come tipo di memoria terziaria; 536
costi; 544
prestazioni; 542
problematiche di assegnazione dei nomi ai file; 540

ripristino
basato sulla registrazione delle modifiche; 233
file system; 456
blocchi difettosi; 519

come vantaggio di un servizio con informazioni di stato; 605
da situazioni di stallo; 270
dopo un calo di tensione, nel Windows 2000; 778, 796
dopo un guasto, nei sistemi distribuiti; 584
requisiti della memoria stabile; 535

ripristino di un precedente stato sicuro, annullamento
di transazione, definizione; 232
in cascata, per evitare, nel controllo della concorrenza nei sistemi distribuiti; 630
nella strategia di eliminazione delle situazioni di stallo con la prelazione sulle risorse; 271
non necessario, come problema del metodo per il rilevamento delle situazioni di stallo centralizzato; 634

risoluzione
dei conflitti, nel LINUX; 730
dei nomi
meccanismo di, reti comunicazione; 572
nelle reti TCP/IP; 807
problematiche di progettazione di reti di comunicazione; 572

risorsa/e
vedi CPU; memoria; tempo
acquisizione e rilascio di, cause di situazioni di stallo; 211
assegnazione delle, 227
come servizio di un sistema operativo; 64
condivisione delle; 135, 560
prelazione sulle; 271, 631
prenotazione delle, definizione; 175
processo; 109, 250
utilizzo; 5, 257

RMI (*remote method invocation*); 128

robustezza
come problema dei sistemi distribuiti; 583

roll out, roll in; 287

ROM (*read-only memory*)
utilizzo nel programma d'avviamento iniziale; 518

rotazione
del disco, in relazione alla sua struttura fisica; 510

RPC (*remote procedure call*); 125
accesso alla memoria secondaria connessa alla rete tramite; 533
come meccanismo del servizio remoto; 599
nel protocollo NFS, utilizzo di; 461, 462
nel sistema operativo Mach; 118
nel Windows 2000; 804

RR (*round-robin*)
algoritmo di scheduling; 167

RSA, algoritmo; 706

RSX, sistema operativo; 835

S

SAN (*storage area network*)
come variante della FC; 532
definizione; 18
memoria secondaria connessa alla rete; 533

scalabilità
come attributo dell'AFS; 607
come motivazione per la scomposizione in batterie di macchine, nell'AFS; 608
come problema di progettazione dei sistemi distribuiti; 585
interdipendenza con la tolleranza ai guasti; 586

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

scambio di messaggi
come interfaccia di dispositivi di rete; 487
modello, chiamate del sistema per; 73
nei sistemi distribuiti, 561, 565
per la gestione di I/O; 494
sistemi; 114
come realizzazione di IPC; 115
tipi di; 115
tramite LPC nel Windows 2000; 786
utilizzo nel meccanismo di STREAMS; 499

scappatoia, chiamata del sistema; 485

schede di controllo
nei primi sistemi; 822

scheduler
vedi anche scheduling
a breve termine
definizione; 157
lungo termine
definizione; 106
a medio termine
definizione; 107
diagramma di accodamento con, (figura); 107

scheduling; 101, 735
a breve termine; 157
a code multiple; 169, 171
a lungo termine, definizione; 106
algoritmi; 160
C-LOOK; 515
C-SCAN; 514
FCFS, CPU; 160
FCFS, dischi; 511
LOOK; 515
RR; 167
SCAN; 513
scelta di un algoritmo di; 515
SJF; 162
SSTF; 511
valutazione degli; 176
code di; 104
con diritto di prelazione; 157
definizione; 158
criteri di; 159
dei processi; 103
diagramma di accodamento; 105
modelli di; 182
nel LINUX; 738
dei processi locali, definizione; 183
del disco; 510
della CPU; 157
(capitolo); 155
concetti fondamentali; 155
definizione; 106
informazioni sullo scheduling;103
di I/O; 489
lavori d'elaborazione; 8, 106
nel Windows 2000; 774
non seriale, di transazioni atomiche; 236
per priorità; 165
per sistemi con più unità d'elaborazione; 173
per sistemi d'elaborazione in tempo reale; 174
senza diritto di prelazione, definizione; 158

scheduling per scansione (algoritmo SCAN)
algoritmo di scheduling del disco; 513

scheletro
definizione; 128

schema attesa-morte
prevenzione delle situazioni di stallo, nei sistemi distribuiti; 631

schema conservativo a ordinamento di marche temporali; 630

schema di ridondanza P+Q; 529

schema ferita-attesa
prevenzione delle situazioni di stallo, nei sistemi distribuiti; 631

SCOPE, sistema operativo; 835

scrittura diretta
criterio di aggiornamento delle cache; 601

scrittura su chiusura
criterio di aggiornamento delle cache; 602

SCSI (*small-computer-systems interface*)
caratteristiche dell'architettura a bus; 532
come dispositivo di controllo; 30
controllore, definizione 473
gestione degli errori; 494

seconda chance
algoritmo di sostituzione delle pagine con; 351

segmentazione; 311
architettura di; 312
(figura); 313
con paginazione; 317
condivisione nella; 314
(figura); 315
definizione; 312
esempio di, (figura); 312
frammentazione nella; 316
metodo di base; 311
nel MULTICS; 831
nel sistema operativo MCP; 835
protezione nella; 314
spazio di memoria, lista delle abilitazioni; 662
su richiesta, realizzazione della memoria virtuale tramite; 327
tabella dei segmenti, definizione; 312

segmento di codice di riferimento
definizione; 125, 128

segnali
definizione; 140
gestione dei; 140
gestione del segnale definita dall'utente, definizione; 141
gestore predefinito del segnale, definizione; 141
per la comunicazione fra processi, nel LINUX; 758

semaforo/i; 207
vedi anche sincronizzazione/sincrono
binario; 211
contatore, definizione; 211
definizione; 207
implementazione nel LINUX di; 758
mutex
come soluzione del problema dei lettori e degli scrittori; 214
come soluzione del problema dei produttori e consumatori con memoria limitata; 212
realizzazione con, (figura); 208
utilizzo nella regione critica; 220
nel sistema operativo THE; 828

semantica
chiamate `read`, strutture di date che utilizzano; 494
del montaggio di un file system; 407
della coerenza
dei file condivisi immutabili; 415
file system Andrew, 415
importanza per la condivisione dei file; 414
negli schemi di replicazione di file; 606
nell'AFS; 610
delle copie
definizione; 491
impiego della memoria di transito per; 491

senza diritto di prelazione
scheduling, definizione; 158
SJF, definizione; 164

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

senza stato
caratteristica dei server NFS; 461
definizione; 461
DFS, problematiche di sicurezza con; 414
server senza informazioni di; 604

sequenziale
accesso
accesso diretto o, come aspetto di un dispositivo; 484
ai file; 391
come metodo di assegnazione concatenata; 440
nel metodo di assegnazione contigua; 436

seriale
porta; 472
controllore di; 473
sequenza d'esecuzione, di transazioni atomiche; 236

serializzabilità
definizione; 235
in transazioni atomiche concorrenti; 236
protocolli basati sulla marcatura temporale per garantire; 239
protocolli d'accesso bloccante per garantire; 238

server, servizi
assenza dell'informazione di stato, caratteristica dei, 461
cluster; 586, 608
comunicazione nei sistemi client-server; 121
confronto tra servizi con e senza informazioni di stato; 604
definizione; 593
dei nomi, nel DNS; 573
meccanismo del servizio remoto, accesso a un file remoto tramite; 599
modello client-server; 411
nel Windows 2000; 805
nell'AFS; 608
remoto, confronto con l'uso delle cache, nei file system distribuiti; 603
sistemi client-server; 15
struttura client-server, nel Windows NT, (figura); 83
utilizzo nei sistemi distribuiti; 559
validità dei dati copiati con metodo iniziato dal; 602

sessione
di file, consistenza della semantica nella; 415
semantica delle, realizzazione tramite AFS; 611
strato di, modello di rete; 580

sessioni di lavoro
remote; 562

settore/i
accantonamento di, come meccanismo di ripristino dei blocchi difettosi; 519
definizione; 30, 38
del disco
corrispondenza sequenziale del vettore di blocchi logici ai; 510
creazione; 517
strategie di strutturazione; 510
traslazione dei, come meccanismo di ripristino dei blocchi difettosi; 520

setuid bit
meccanismo di protezione in UNIX; 653

sezionamento
dei dati, come tecnica di parallelismo nei sistemi RAID; 525
del disco, nel file system del Windows 2000; 797
nell'organizzazione con blocchi di parità intercalati, utilizzo nel RAID di livello 4; 528

sezione critica
vedi anche sincronizzazione
come problema nel processo di sincronizzazione, 197
definizione; 197
nell'approccio alla mutua esclusione centralizzata; 618
sincronizzazione del nucleo nel LINUX; 736
soluzione per due processi; 198
soluzione per più processi; 202

shell
definizione; 63
script, definizione; 389

sicurezza
vedi anche autenticazione; protezione,
nell'AFS; 608
algoritmo di verifica della, per evitare le situazioni di stallo; 263
attacchi basati sull'alterazione della pila; 687
attacchi per rifiuto del servizio; 693
(capitolo); 679
cavalli di Troia; 686
classificazione della, nei sistemi di calcolo; 710
crittografia; 704
definizione; 679
dei tipi, come fondamento del sistema di protezione del linguaggio Java; 674
gestore della sicurezza; 789
in relazione alla protezione; 649
livelli; 680
minacce ai programmi; 686
minacce ai sistemi; 688
nel file system del Windows 2000; 797
nel LINUX; 762
nucleo di protezione e compilatore; 669
panoramica delle problematiche; 679
problematiche di accesso ai dispositivi; 481
rilevamento delle intrusioni; 696-697
sicurezza dei sistemi; 694
sottosistemi di; 793
trabocchetti; 687
virus; 691
worm; 688

sicuro/a
computazione, come protezione contro i virus; 692-693
stato, per evitare le situazioni di stallo; 259

SID (*security ID*)
utilizzo nella gestione della condivisione di file; 410

signaled, stato
degli oggetti dispatcher nel Windows 2000; 230

simmetrico/che
algoritmo di cifratura; 706
batterie di sistemi; 17

sincronizzazione/sincrono
vedi anche concorrente/concorrenza; regioni critiche; sezione critica; stallo dei processi
a basso livello, nel Windows 2000; 777
architetture di; 203
(capitolo) 195
dei processi; 112
del nucleo, nel LINUX; 735
I/O, definizione; 31
in comunicazione tra processi; 117
nel LINUX; 758
messaggi, definizione; 117
monitor; 220
nei sistemi distribuiti, (capitolo); 615

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

nel Solaris 2; 228
nel Windows 2000; 230
o asincrono, come aspetto dei dispositivi; 484
problemi
dei cinque filosofi; 215
dei lettori e degli scrittori; 213
del produttori/consumatori con memoria limitata; 212
problemi tipici di; 212
regioni critiche; 217
scritture, definizione; 453
semafori; 207
sistemi operativi; 228
tramite semafori, nel sistema operativo THE; 828

sistema S/Key
sistema di parole d'ordine monouso; 685

sistema/i
a lotti; 7
a partizione di tempo; 10
batterie di sistemi; 17
classificazione della sicurezza dei sistemi di calcolo; 710
client-server; 15
con più unità d'elaborazione; 12
d'elaborazione in tempo reale; 17
debole, definizione; 19
stretto, definizione; 19
da scrivania; 11
debolmente connessi; 16
di I/O; 471
distribuiti; 15
definizione; 16, 559, 593
problemi di progettazione dei; 585
problemi di robustezza dei; 583
vantaggi dei; 560
mainframe; 7
minacce; 688
multiprogrammati; 9
palmari; 20
paralleli; 12
paritetici; 16
struttura, semplice; 77

sistemi, architetture con più unità d'elaborazione, multielaborazione
algoritmi di scheduling; 173
definizione; 12
gestite dal Solaris 2; 228
gestite dal Windows 2000; 230
multielaborazione asimmetrica; 13, 174
multielaborazione simmetrica; 13, 740
nel sistema operativo Mach, 835
omogenei, algoritmi di scheduling, 173
problematiche di architetture di sincronizzazione; 204

sistemi da scrivania; 11

sistemi debolmente connessi
definizione; 16

sistemi di calcolo interattivi
definizione; 10

sistemi mainframe; 7
effetto sui sistemi distribuiti; 561

sistemi operativi
caratteristiche dei; 3
componenti; 57
di rete; 16, 562
distribuiti; 564
migrazione dei dati nei; 564
migrazione dei processi nei; 565
migrazioni delle computazioni nei; 564
topologie di rete; 566
generazione; 91
gestione, dispositivi di memoria terziaria; 539
migrazione delle funzioni; 21
progettazione; 88-89
realizzazione; 89
scopi; 6
servizi; 63
sicurezza; 680
sincronizzazione nei; 228
storia
dei sistemi, (capitolo); 819
panoramica sullo sviluppo del concetto, (capitolo); 3
struttura; 77
(capitolo); 57
micronucleo; 81
stratificato; 79
strutture di file gestite dal; 390

sistemi operativi, esempi, vedi
Atlas, sistema operativo
BeOS, sistema operativo
BSD UNIX, sistema operativo
CP/67, sistema operativo
CP/M, sistema operativo
CTSS, sistema operativo
LINUX, sistema operativo
Mach, sistema operativo
Macintosh, sistema operativo
MCP, sistema operativo
MS-DOS, sistema operativo
MULTICS, sistema operativo
NeXT, sistema operativo
OS/2, sistema operativo
OS/360, sistema operativo
OS/MFT, sistema operativo
OS/MVT, sistema operativo
RC 4000, sistema operativo
SCOPE, sistema operativo
Solaris 2, sistema operativo
Tenex, sistema operativo
THE, sistema operativo
TOPS-20, sistema operativo
Tru64 UNIX, sistema operativo
TSO, sistema operativo
TSS/360, sistema operativo
UNIX, sistema operativo
VM, sistema operativo
VMS V, sistema operativo
Windows 2000, sistema operativo
Windows NT, sistema operativo
XDS-940, sistema operativo

sistemi palmari; 20

sistemi paralleli
definizione; 12

sistemi paritetici; 16

sistemi server di calcolo; 16

SJF (*shortest-job-first*)
algoritmo di scheduling; 162
con prelazione, definizione; 164
scheduling del disco, SSTF paragonato a; 512
senza prelazione, definizione; 164

Slackware
pacchetto di distribuzione del LINUX; 723

SLS (*Softlanding Linux System*)
pacchetto di distribuzione del LINUX; 723

SMB (*server message-block*), protocollo
vedi anche CIFS (Common Internet File nel Windows 2000); 802

SMP (*symmetric multiprocessing*)
definizione; 13
nel LINUX; 740

sniffing
vulnerabilità delle parole d'ordine; 682

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

socket; 121
definizione; 121
interfaccia di rete, come meccanismo di accesso ai dispositivi di rete; 487
nel Windows 2000; 803
realizzazione tramite STREAMS; 501

Solaris 2, sistema operativo
associazione alla memoria nel; 338
come gestore di thread al livello del nucleo; 136
elaborazioni sia a partizione del tempo sia in tempo reale nel; 374
gestione dell'area di avvicendamento nel; 523
realizzazione della memoria virtuale; 364
scheduling dei processi nel; 183
sincronizzazione nel; 228
thread del; 145
UI-threads del, come thread al livello d'utente; 136

sostituzione (delle pagine); 339
algoritmi
difficoltà di scelta di; 451
lettura anticipata; 453
rilascio indietro; 453
algoritmi alternativi di assegnazione dei blocchi di memoria; 354
algoritmi con memorizzazione transitoria delle pagine; 353
algoritmo di, con bit supplementari di riferimento; 350
algoritmo FIFO di; 344
algoritmo LRU di; 347, 350
basata su conteggio, algoritmo di; 353
con seconda chance, algoritmo di; 351
con seconda chance migliorato, algoritmo di; 351
necessità di (figura); 341
ottimale, algoritmo di; 346
schema di base; 340

sottosistemi
d'ambiente, Windows 2000; 790
mutuamente sospetti, soluzione in Hydra ai problemi dei; 666

sovrapposizione di sezioni; 284
definizione; 284
per un assemblatore a due passi, (figura); 286

spaccone, algoritmo dello; 639

spazio
come problematica di assegnazione contigua; 436
degli indirizzi fisici, definizione; 282
degli indirizzi logici, definizione; 282
determinazione dell'assegnazione, dimensione come problematica di assegnazione contigua; 437
di indirizzi, logici e fisici; 282
gestione dello spazio libero, per lo spazio dei dischi; 446
identificatori dello spazio d'indirizzi (ASID), definizione; 300
nel disco; 435, 448
requisiti di
come svantaggio dell'assegnazione concatenata; 440
come svantaggio dell'assegnazione indicizzata; 441
schema spazio-temporale; 616, 617
spazio degli indirizzi sparso, utilizzo nelle tabelle delle pagine a gruppi; 308

spazio degli indirizzi
logici e fisici; 282
utilizzo nella tabella delle pagine a gruppi; 307
virtuali, definizione; 326

spazio degli indirizzi sparso
utilizzo nelle tabelle delle pagine a gruppi; 308

spazio libero (*free-space*)
elenco dei blocchi liberi, definizione; 446
gestione, per lo spazio dei dischi; 446

spinlock
definizione; 208
nel LINUX; 740
utilizzo nel Solaris 2; 229
utilizzo nel Windows 2000; 230

spoofing
come problematica dell'identificazione certa dei clienti; 411
vulnerabilità delle barriere di sicurezza; 696

SRI-NIC
sito di Internet di registrazione dei nomi; 574

SSTF (*shortest-seek-time-first*)
algoritmo di scheduling del disco; 511

stallo dei processi
vedi anche processo/i; sincronizzazione (capitolo); 249
caratterizzazione delle; 251
condizioni necessarie per; 251
definizione; 210, 249
e attesa; 210
gestione, nei sistemi distribuiti; 630
metodi per la gestione delle; 255
nel controllo della concorrenza in ambienti distribuiti; 626
per evitare lo; 259
definizione; 255
nell'algoritmo per la mutua esclusione con metodo totalmente distribuito; 619
uso dei monitor; 223
prevenire; 255, 256, 630
rilevamento; 266, 268, 632
ripristino da; 270

statico/a
associazione tra un processo e un dominio; 652
collegamento, definizione; 283
nella matrice d'accesso; 657

stato/i
dei componenti di I/O, strutture di dati del nucleo per; 494
del processo; 100
come componente del blocco di controllo di un processo; 101
definizione; 100
diagramma di, (figura); 101
dell'I/O, come componente del blocco di controllo di un processo; 103
informazioni di, necessarie al recupero dai malfunzionamenti; 414
informazioni di, nei programmi di sistema; 74
non sicuro; 260
registro `status`; 475
busy bit del, utilizzato dal controllore di I/O; 475
servizio con informazioni di; 604
sicuro; 259

strato, stratificazione
di applicazione, modello di rete; 580
di collegamento dei dati, modello di rete; 579
di presentazione, modello di rete; 580
di rete, modello di rete; 580
di sessione, modello di rete; 580
di trasporto; modello di rete; 580
file system
(figura); 427
interfaccia basata sulle chiamate del sistema come primo strato; 433
realizzazione; 432
realizzazione di un tipo di, come strato più basso; 434
virtuale come secondo strato; 433

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

fisico, modello di rete; 578
nel sistema operativo dell'RC 4000; 829
nella struttura dell'interfaccia di I/O per le applicazioni; 483
protocollo di comunicazione, modello di rete; 579
struttura del sistema operativo; 79

streams
definizione; 499
di caratteri, o a blocchi, come aspetto dei driver di dispositivi; 483

STREAMS; 499
come interfaccia di dispositivi di rete; 487

struttura
dei sistemi di calcolo, (capitolo); 27
impatto su numero minimo di blocchi di memoria
assegnazione; 355
traduzione degli indirizzi nell'Intel 80386, (figura); 319

struttura/e
dei dischi; 509
dei domìni di protezione; 651
dei file; 389
interna; 391
dei programmi, effetto sulla prestazione di memoria virtuale; 370
dei sistemi di calcolo, (capitolo); 27
dei sistemi distribuiti, (capitolo); 559
dei sistemi operativi
(capitolo); 57
micronucleo; 81
semplice; 77
stratificata; 79
del file system; 425
gestione della; 426
delle reti; 50
nel LINUX; 759
di directory; 394
di I/O; 30, 34
memoria; 35
memoria secondaria e terziaria, (capitolo); 509
per la memorizzazione terziaria; 536

strutturazione dei parametri (*marshalling*)
definizione; 125

Sun Microsystems
Enterprise 6000, velocità di trasferimento dei dispositivi di, (tabella); 491
NFS, *vedi* NFS (network file system)

superblocco
come blocco di controllo delle partizioni nell'UFS; 428

SuSE
versone commerciale di LINUX; 723

sysgen
di sistemi operativi; 91

T

T1; 570

T3; 570

tabella delle pagine
a due livelli
algoritmo; 304
(figura); 304
traduzione degli indirizzi per un'architettura a 32 bit, (figura); 305
a gruppi, definizione; 307
ad associazione diretta, definizione; 305
bit di validità in; 302, 303

definizione; 294
di tipo hash; 307, 307
invertita; 308, 309, 369
meccanismo d'ausilio alla paginazione su richiesta; 332
PTLR, definizione; 302
registro di base della (PTBR), definizione; 299
struttura della; 303

tabella delle pagine ad associazione diretta
definizione; 305

tabella delle pagine invertita; 308, 309, 369

tabella di simboli
directory considerata come; 395

tabelle hash; 307, 307
vantaggi e svantaggi; 435

task control block
vedi PCB

tasso di successi (*hit ratio*)
definizione, 300
proporzionale alla portata del TLB; 368

tastiera
vedi anche I/O
come dispositivo a flusso di caratteri, meccanismo di chiamata del sistema per; 486

TCB (*trusted computer base*)
come criterio di sicurezza; 710

TCP (*transmission control protocol*)
accesso alla memoria secondaria connessa alla rete tramite; 533
pila di protocolli di rete; 580
socket, nel Java; 122

TCP/IP, protocollo
nel Windows 2000; 802
risoluzione dei nomi gestita dal; 587
risoluzione dei nomi nelle reti; 807
strati del, (figura); 582

TDI (*transport driver interface*)
nel Windows 2000; 802

TEB (*thread environment block*)
definizione; 148

tecniche biometriche
come meccanismo di sicurezza; 685

Telnet
demone, problematiche di prestazione; 503
paragoni con il meccanismo FTP; 563
utilizzo nelle sessioni di lavoro; 562

TEMPEST, sistema di certificazione; 711

tempo
come attributo dei file; 384
d'accesso diretto, definizione; 38
d'attesa; 159
d'attesa nei sistemi distribuiti; 583, 642
di completamento, definizione; 159
di ricerca, definizione; 38, 510
di risposta; 160
di servizio dell'eccezione di pagina; 335
effettivo d'accesso alla memoria, definizione; 301
partizione del
CP67/CMS; 833
CTSS; 830
MULTICS; 831
sistemi; 10
strategie di scheduling in; 107
TSS/360; 833
quanto di, definizione; 167

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

tempo medio
di guasto, in sistemi RAID; 524
di perdita di dati, in sistemi RAID; 525
di riparazione, in sistemi RAID; 524

tempo reale debole, sistemi in; 175
abilitazione logica; 667, 670
eccezione, utilizzo nell'architettura delle interruzioni; 479
nei primi sistemi; 819
stratificazione dei, nella struttura dell'interfaccia di I/O per le applicazioni; 483

temporizzatore
caratteristiche e meccanismi di chiamate del sistema per; 487
programmabile, caratteristiche e meccanismi di chiamate del sistema per; 487
uso nella protezione della CPU; 48
variabile, uso nella protezione della CPU; 48

Tenex, sistema operativo; 835

terminato/terminazione
a cascata. definizione; 111
come strategia per il ripristino da situazioni di stallo; 270
di un processo; 110
stato del processo, definizione; 100

`TestAndSet`, istruzione; 204, 205, 206

THE, sistema operativo; 828

thread/threading;
al livello d'utente; 136
al livello del nucleo; 136
architettura del nucleo basata sul, per la gestione delle interruzioni; 480
bersaglio; 139
(capitolo); 133
creazione dei, nel liguaggio Java; 150
dati specifici dei; 142
definizione, 142
definizione; 133
del sistema Solaris 2; 145
e processi, nel LINUX; 734
gruppi di; 142
definizione; 142
idle, definizione; 185
in relazione ai processi; 103
introduzione; 133
memoria locale del, definizione; 148
modelli di programmazione multithread, 136
motivazioni; 133
multithread, caratteristica del Solaris 2; 228
nel linguaggio Java; 149
nel sistema LINUX; 148
nel sistema Windows 2000; 147
nel Windows 2000; 774
Pthreads; 143
questioni di programmazione multithread; 139
relazione tra scheduling dei processi e; 182
utilizzo nei server cluster, nei sistemi distribuiti; 587
utilizzo nella sovrapposizione di I/O; 488
vantaggi; 135

tipo
come attributo dei file; 384
di dati astratti, protezione basata sul linguaggio; 668
di file; 388
comuni (tabella); 388

tipo di dato astratto
come file di; 385
definizione; 650
uso nei meccanismi di protezione; 668

TLB (*translation look-aside buffer*)
architettura di paginazione con, (figura); 301
cancellazione del, definizione; 300
definizione; 300
insuccesso del, definizione; 300
strategie di aumento della portata del; 368

tolleranza ai guasti
definizione; 13
nel file system Windows 2000; 797
problemi di progettazione nei sistemi distribuiti; 585
relazioni con la scalabilità di un sistema; 585

topologie
di reti (figura); 567
di sistemi operativi distribuiti; 566

TOPS-20, sistema operativo; 835
gestione dei tipi di file; 389
metodo di protezione in; 653
utilizzo del HSM nel: 542

tornelli
utilizzo nel Solaris 2; 228

Torvalds, Linus
creatore del sistema operativo LINUX; 719

tracce
rilevamento basato sulle; 697

trace tapes
utilizzo nelle simulazioni; 180

Transarc, DFS della
definizione; 593
storia del; 607

transazione/i
atomiche; 231
concorrenti; 235
coordinatore delle, nei sistemi distribuiti; 622
definizione; 232, 457
distribuzione delle, nei sistemi distribuiti; 625
fallita, definizione; 232
file system basati sull'annotazione delle modifiche e orientati alle; 456
nome della, come campo del giornale delle modifiche, nel sistema di ripristino basato sulla registrazione delle modifiche; 233
terminata con successo, definizione; 232

translation look-aside buffer (TLB)
vedi TLB

transparenza
come problema di progettazione dei sistemi distribuiti; 585
di locazione; 595, 596
file system distribuiti; 594, 595
negli schemi di nominazione dei file system distribuiti; 608

trap
door, violazione della sicurezza; 687
eccezione di pagina mancante, gestione della memoria virtuale; 328
evento segnalato da; 28
utilizzo nel meccanismo delle interruzioni; 479

trasferimento
dati, memorizzazione transitoria come meccanismo per la gestione del; 491
dei dispositivi, comparazione di velocità (tabella); 491
tempo di, in relazione con il tempo d'avvicendamento; 288
temporizzazione, come aspetto dei dispositivi; 483
velocità di, definizione; 38

trasporto
protocolli di; 581
strato di, modello di rete; 580

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

Tripwire
strumento per il rilevamento di anomalie; 700-701

troncamento
di un file; 386

Tru64 UNIX, sistema operativo
gestore dei thread al livello del nucleo; 136
orientamento a micronucleo; 81

TSO, sistema operativo
nell'OS/360; 833

TSS/360, sistema operativo; 833

U

UDP (*user datagram protocol*)
accesso alla memoria secondaria connessa alla rete tramite; 533
interdipendenza con l'IP; 580
socket, nel linguaggio Java; 122
utilizzo nella migrazione delle computazioni, nei sistemi operativi distribuiti; 564

UFD (*user file directory*)
vedi anche file system
come componente della directory a due livelli; 397

UFS (*UNIX File System*)
come file system basato su dischi; 427

UI-threads
come thread al livello d'utente; 136

UID (*user ID*)
utilizzo nella gestione di condivisione di file; 410

UltraSparc, architettura
strategia di gestione del TLB; 369

UMA (*uniform memory access*); 173

UNC (*uniform naming convention*)
nel Windows 2000; 803

unico punto d'accesso sicuro
come meccanismo d'autenticazione; 413

Unicode, codice dei caratteri
nel Windows 2000; 772

Unifix
versone commerciale di LINUX; 723

unità d'elaborazione del terminale (*front-end processors*)
per incrementare le prestazioni; 503

UNIX, sistema operativo
4.2 BSD, utilizzo nel sistema operativo Mach; 834
albero di processi (figura); 109
bit del, forniti con liste d'accesso dall'AFS; 609
confronto con LINUX; 719
domìni di protezione; 653
gestione dell'area d'avvicendamento nel; 522
gestione della struttura interna dei file; 391
gestore di strutture di file; 390
identificatore del processo del, definizione; 110
inode dello (figura); 444
interfacce degli interpreti dei comandi; 337
NFS, *vedi* NFS
protezione delle directory nel; 420

realizzazione dell'AFS nel; 611
semantica della consistenza nel; 415
strategie di protezione; 663
STREAMS nel; 499
struttura; 77
supporto del Windows 2000, nel sottosistema d'ambiente POSIX; 792
terminazione di un processo nel; 112
utilizzo di un codice per la gestione dei tipi di file; 389
versione 4.3 BSD, gestione dell'area d'avvicendamento nel; 522

utente/i
autenticazione; 681
come dominio di protezione; 652
gestione del segnale definita da, definizione; 141
identificazione
come attributo dei file; 384
controllo degli accessi dipendente dalla; 417
in relazione ai domìni di protezione; 653
interfaccia di, difficoltà nell'elenco di controllo degli accessi; 418
mobilità dell'
come problema di progettazione dei sistemi distribuiti; 585
facilitazione nell'NFS; 460
modo, in duplice modo di funzionamento; 44
multipli, problemi relativi alla condivisione dei file tra; 409
programmi, nel LINUX, caricamento ed esecuzione; 746
punto di vista; 4
thread al livello d'utente; 136
definizione; 136
thread del livello d'utente, in relazione allo scheduling dei processi; 183

utilità di sistema
come componenti del sistema LINUX; 725

utilizzo
come obiettivo nei primi sistemi; 821
della CPU, come criterio di scheduling; 159
di più unità d'elaborazione, come vantaggio della programmazione multithread; 135

UUCP (*UNIX news network*)
come protocollo delle WAN; 571

V

V, operazione di semaforo; 207

validazione
dei dati nella cache, nei file system distribuiti; 602
del file system; 407
delle cache, nell'AFS; 610

validità, bit di
definizione; 302
in una tabella delle pagine (figura); 303

valutazione
vedi anche criteri di valutazione
degli algoritmi di scheduling; 176
analisi delle reti di code; 179
metodi analitici della; 177
degli algoritmi di sostituzione delle pagine; 343
delle strutture di nominazione nei file system distribuiti; 598

VAX, architettura
tabella di paginazione a due livelli gestita dalla; 305
/VMS, algoritmi con memorizzazione transitoria delle pagine utilizzati nel; 353

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.

VAXcluster, sistema
tolleranza ai guasti nel; 585

Venus, sistema operativo
correlazioni con il sistema operativo THE; 829

Venus (sottosistema dell'AFS)
gestione delle cache; 612
interazione con il Vice; 610
traduzione dei nomi di percorso; 611

vettore
delle interruzioni, definizione; 30, 478
di bit, gestione dell'elenco dei blocchi liberi con; 446

vettore/i
block buffer cache, nel LINUX; 755
buffer cache unificata; 450
limitato; 113, 195
overflow, come minaccia ai programmi di sicurezza; 687
problema del produttore/consumatore con memoria illimitata; 113
problema del produttore/consumatore con memoria limitata; 212
translation look-aside buffer (TLB); 300
utilizzo nel sottosistema per l'I/O; 490

VFS (*virtual file system*)
come strato della realizzazione del file system; 433
integrazione dell'NFS nel sistema operativo tramite; 462
nel LINUX; 749
schema di un file system, (figura); 433

Vice (sottosistema dell'AFS); 608
identificatori di file, nell'AFS; 609
interazione con il Venus; 610
interfaccia, come meccanismo di sicurezza nell'AFS; 608

virtuale/i
disco, come strumento per migliorare le prestazioni di un disco basato sulla memoria; 453
file system; 432
come strato della realizzazione del; 433
integrazione dell'NFS nel sistema operativo tramite; 462
nel LINUX; 749
schema di un, (figura); 433
indirizzi; 282, 326
instradamento; 575
macchina; 82
Java come esempio di; 87
realizzazione; 84
vantaggi; 85
memoria
attività di paginazione degenere; 358
(capitolo); 325
definizione; 10, 325, 326
del nucleo, nel LINUX; 745
di rete; 599
durata di uno spazio d'indirizzi, nel LINUX; 744
(figura); 327
fork, come alternativa all'uso della `fork` con copiatura su scrittura; 337
introduzione; 325
LINUX, gestione della; 742
nel sistema operativo XDS-940; 828
nel Windows 2000; 781, 813
nell'MVS; 833
realizzazione nel sistema operativo Solaris 2; 365
realizzazione nel sistema operativoWindows NT; 364
regioni, nel LINUX; 743
utilizzo del DVMA per DMA; 481
utilizzo dell'area d'avvicendamento nella; 521
reti private, uso di IPSec; 709

Virtue, protocollo
nell'AFS; 608

virus
come minaccia ai sistemi; 691

vittima
selezione di, nella prelazione su risorse; 271

VM, sistema operativo
concetto di macchina virtuale; 83
partizione del tempo nel; 833

VMS, sistema operativo; 835
creazione di processo nel; 110

volume/i
base di dati locazione-volume, nell'AFS; 609
come componente dello spazio dei nomi condivisi dell'AFS; 609
del disco, *vedi* partizioni
gestione dei, nel file system del Windows 2000; 797
tabella dei contenuti del, *vedi* directory

W

WAN (rete geografica); 533, 569
definizione; 15
struttura; 51

Web clipping
definizione; 20

Win16
come sottosistema d'ambiente del Windows 2000; 792

Win32
come sottosistema d'ambiente del Windows 2000; 792

Win32 API
come interfaccia per il programmatore di Windows 2000; 808

Windows 2000, sistema operativo
APCs nel; 141
(capitolo); 769
chiamata di procedura locale; 785
come esempio di scambio di messaggi; 120
componenti del sistema; 772
esecutivo; 778
file system; 793
compressione dei dati; 800
gestione dei volumi; 797
punti d'interpretazione; 801
ripristino nel; 796
sicurezza; 797
tolleranza ai guasti; 797
gestione della memoria virtuale; 781
gestore degli oggetti; 778
gestore dei processi; 785
gestore dei thread al livello del nucleo; 136
gestore del sistema Plug-and-Play; 789
gestore della sicurezza; 789
gestore dell'I/O; 786
nucleo; 773
principi di progettazione; 770
risoluzione dei nomi nelle reti TCP/IP nel; 807
scheduling; 774
scheduling dei processi nei; 185
segnali di eccezione; 775
segnali di interruzione; 775
servizi di rete; 801
domini; 806
elaborazione distribuita; 803
protocolli; 802
reindirizzamento; 805
server; 805

sincronizzazione a basso livello nel; 777
sincronizzazione nel; 230
sottosistemi d'ambiente; 790
di sicurezza; 793
MS-DOS; 791
OS/2; 793
POSIX; 792
Win16; 792
Win32; 792
Thread nel; 147, 774
Win32 API; 808
accesso agli oggetti del nucleo; 808
gestione dei processi; 809
gestione della memoria; 813
IPC; 812

Windows NT, sistema operativo
creazione del processo nel; 111
garanzie di sicurezza; 711
gestore dei thread al livello del nucleo; 136
metodo stratificato; 79-80
orientamento a micronucleo; 81-82
realizzazione della memoria virtuale nel; 364

Windows, sistema operativo
CIFS, *vedi* CIFS

WINS (*Windows Internet Name Service*)
risoluzione dei nomi nelle reti TCP/IP nel; 807

Winsock
nel Windows 2000; 804

WorkGroup Solutions
versone commerciale del LINUX; 723

WORM (*write-once read-many*), **dischi**
come memoria terziaria; 537

worm
come minaccia ai sistemi; 688

worst-fit, algoritmo di assegnazione della memoria; 292

`write/` **scrittura**
di file; 385
lettura o, come aspetto dei dispositivi; 484
sincrone, definizione; 453
su nastri; 540

WWW (*World Wide Web*)
as remote file sharing mechanism meccanismo di condivisione di file remoti; 410
come ambiente d'elaborazione distribuito; 566
utilizzato dal DNS; 412

X

XDR (*external data representation*), **protocollo**
definizione; 126
nell'NFS, costruite su; 460

XDS-940, sistema operativo; 828

Y

yellow pages
vedi NIS (network information service)

http://www.adultpdf.com
Created by TIFF To PDF trial version, to remove this mark, please register this software.