SISTEMI OPERATIVI – MODULO I

Definizione -capitolo 1-

Un sistema operativo è il software che permette alle applicazioni di interagire con l'hardware. Il software che ne contiene i componenti principali è chiamato kernel.
I sistemi operativi sono principalmente gestori di risorse : gestiscono le risorse hardware come processori, memorie, dispositivi di I/O e di comunicazione, ma anche applicazioni ed altri meccanismi di astrazione creati dal software che, a differenza dell'hardware, non sono oggetti fisicamente esistenti.

Un po' di storia dei sistemi operativi

→ Anni '40
- I primi calcolatori elettronici digitali non erano provvisti di un sistema operativo.
- I programmatori erano spesso costretti a caricare i loro programmi in linguaggio macchina, un bit alla volta per mezzo di gruppi di interruttori meccanici.
- In seguito si utilizzarono e schede perforate per caricare i vari programmi, scritti comunque in linguaggio macchina.
- L'introduzione del linguaggio macchina (che usava abbreviazioni del di parole inglese per rappresentare parole di base del computer) velocizzò ulteriormente il processo.

→ Anni '50
- I laboratori di ricerca della General Motors realizzarono il primo SO per il computer IBM 701.
    - I sistemi dell'epoca eseguivano generalmente solo un job alla volta, utilizzando tecniche per lo alternare l'esecuzione di più lavori allo scopo di massimizzare lo sfruttamento del sistema. Un job costituiva l'insieme delle istruzioni di programma corrispondenti ad un particolare compito computazionale (task). I job restavano spesso in esecuzione per minuti, ore o giorni, senza interazione con l'utente.
    - Questi primi computer erano chiamati single-stream batch-processing system (sistemi di elaborazione a lotti a singolo flusso) perché i programmi e i dati sui quali il sistema doveva lavorare erano sottomessi a gruppi (o lotti) e memorizzati in sequenza su nastri o dischi.
    - Un apposito job stream processor (elaboratore del flusso di job) leggeva il cosiddetto job control language (linguaggio di controllo del job), che serviva a definire ogni job. Quando il job corrente terminava, veniva letto il job control language relativo al successivo job e venivano svolte opportune operazioni di pulizia per facilitarne l'esecuzione.
- Il programmatore era comunque spesso obbligato a controllare direttamente le risorse di sistema, questo risultava un lavoro lento, difficoltoso e noioso, che richiedeva la presenza in memoria di un intero programma per poterlo eseguire. Questo vincolava i programmatori a dover creare programmi piccoli e con limitate funzionalità.

→ Anni '60
- I SO di questi anni erano ancora di tipo batch ma riuscivano a utilizzare le risorse in modo più efficiente portando in esecuzione più job alla volta.
- Tali sistemi includevano molte periferiche come lettori e perforatori di schede, stampanti , nastri e dischi magnetici.

- I SO di questi anni hanno migliorato lo sfruttamento delle risorse permettendo ad un job di utilizzare il processore mentre gli altri sfruttavano le periferiche.
- Infatti mandare in esecuzione un insieme di job con caratteristiche diverse (processor-bound job che usavano principalmente il processore e I/O-bound job che usavano principalmente le periferiche) sembrava essere il modo migliore per ottimizzare l'utilizzo delle risorse.
- Alla luce di queste riflessioni, i progettisti crearono i cosiddetti sistemi a multiprogrammazione (o sistemi multiprogrammati) che gestivano diversi job nello stesso tempo. In un ambiente multiprogrammato il SO scambia rapidamente il processore da un job all'altro permettendogli così' di avanzare tenendo comunque attive le periferiche. Il grado o il livello di multiprogrammazione definisce quanti job erano gestibili concorrentemente.
- L'obbiettivo principale di un sistema multiprogrammato è la condivisione delle risorse. Quando le risorse sono condivise da un insieme di processi e ogni processo mantiene il controllo esclusivo delle risorse che gli sono allocate, un processo potrebbe rimanere in attesa di una risorsa che non diventerà mai disponibile. Se questo accade, il processo in questione non sarà in grado di svolgere il suo compito e probabilmente l'utente sarà costretto a mandarlo in esecuzione di nuovo, perdendo il lavoro svolto fino a quel momento.
- Gli utenti non dovevano più essere necessariamente presenti, i job venivano forniti su schede perforate o nastri magnetici ad un operatore, che si occupava poi di mandarli in esecuzione appena possibile. Spesso un job rimaneva in coda anche ore o giorni. Un minimo errore nel programma bloccava l'esecuzione del job costringendo l'utente a correggere l'errore e riconsegnare il tutto all'operatore rimettendosi in attesa per una nuova esecuzione.
- Sistemi operativi sempre più avanzati furono sviluppati per servire concorrentemente gli utenti interattivi. L'interattività consisteva nel fatto che gli utenti potevano interagire con i loro job durante l'esecuzione (creazione terminali "stupidi" che servivano da interfaccia e non avevano alcuna potenza computazionale).
- Dato che l'utente era fisicamente presente davanti alla macchina era necessario che il sistema rispondesse abbastanza velocemente alle sue richieste per non limitarne la produttività. I sistemi timesharing furono sviluppati per servire contemporaneamente più utenti in modo interattivo. (Questo tipo di sistemi era multimodale cioè sosteneva sia elaborazioni di tipo batch sia elaborazioni in tempo reale, i sistemi real-time dovevano essere in grado di fornire risultati in intervalli di tempo limitati).
- Il tempo di turnaround, cioè il tempo trascorso tra l'invio di un job ed il suo completamento, fu ridotto a minuti e perfino a secondi. Il programmatore non era più obbligato ad attendere ore o giorni ma poteva ora inserire un programma, compilarlo e ricevere una lista degli errori di sintassi per poi correggerli, questo ciclo continuava fino ad avere un programma privo di errori. A quel punto lo si poteva eseguire, correggere in caso di presenza di errori "run-time" (debug) e completarlo con un risparmio di tempo notevole.

→ Anni '70
- Sistemi multiprogrammati multimodali che supportavano le elaborazioni di tipo batch, in timesharing ed in tempo reale.
- I sistemi sperimentali in timesharing degli anni '60 in questo decennio erano ormai divenuti solidi prodotti commerciali.
- I problemi di sicurezza aumentarono di pari passo alla crescita del volume di dati trasmessi attraverso linee di comunicazione vulnerabili. Crebbe così l'importanza della crittografia al fine di rendere meno vulnerabile la sicurezza dei dati.

→ Anni '80
- Decennio dei personal computer e delle workstation (il Personal Computer IBM e l'Apple Macintosh permisero a singoli utenti e piccole industrie di possedere un proprio computer).

- La pratica del personal computer si dimostrò abbastanza semplice da acquisire, anche grazie alla Graphical User Interface (GUI) che ricorse a simboli come finestre, icone e menù per interagire più comodamente con i programmi.
- L'elaborazione distribuita, cioè l'utilizzo di più computer indipendenti per svolgere un compito in comune, si diffuse basandosi sul modello client/server. I client sono computer utenti che richiedono vari servizi mentre i server sono computer che svolgono i servizi richiesti. Spesso i server sono dedicati ad un particolare compito, come il rendering grafico, la gestione di basi di dati o di siti web.

→ Storia di Internet e del World Wide Web

- Verso la fine degli anni '60 l'ARPA (Advanced Research Projects Agency, del dipartimento della difesa americano) mise a punto un progetto per collegare in rete i sistemi di elaborazione di una dozzina di università e centri di ricerca da essi finanziati. Poco dopo realizzò velocemente ciò che venne chiamata ARPAnet, una sorta di "nonna" di quello che oggi è Internet; il beneficio principale fu quello di permettere una forma di comunicazione semplice e rapida attraverso quella che oggi è chiamata posta elettronica (e-mail).
  ARPAnet fu progettata per funzionare senza un controllo centralizzato. Questo significava che se anche una porzione della rete avesse avuto dei problemi, la rimanente porzione sarebbe stata in grado di trasportare dati tra mittente e destinatario per mezzo dei cammini alternativi.
  I protocolli, ossia l'insieme di regole, per comunicare per mezzo di ARPAnet divennero conosciuti come Transimission Control Protocol/Internet Protocol (TCP/IP). Questi furono utilizzati per gestire la comunicazione tra applicazioni, i protocolli assicuravano l'arrivo dei messaggi intatti alla giusta destinazione.
- Il World Wide Web (WWW) permise agli utenti di localizzare e visualizzare documenti multimediali riguardanti ogni possibile argomento. Nel 1989 Tim Berners-Lee del CERN (European Center for Nuclear Research) cominciò a sviluppare una tecnologia per condividere informazioni attraverso documenti testuali con collegamenti ipertestuali (hyperlink). Per mettere in pratica questa tecnologia, Berners-Lee inventò l'HyperText Markup Language (HTML) e l'HyperText Transfer Protocol (HTTP) formando di fatto la spina dorsale di questo nuovo sistema informativo ipertestuale ce lui chiamò World Wide Web.

→ Anni '90

- Alla fine del decennio precedente un tipico personal computer era in grado di eseguire diverse centinaia di milioni di istruzioni al secondo (MIPS, Millions of Instructions Per Seconds) e memorizzare più di un gigabyte di informazioni su un hard disk.
- Il crollo del costo della tecnologia incrementò il numero di computer presenti nelle case, utilizzati sia per lavoro sia per intrattenimento.
- Negli anni '90, la creazione del World Wide Web accrebbe la diffusione dell'elaborazione distribuita. In origine i sistemi operativi svolgevano un' isolata gestione delle risorse locali all'interno di un singolo computer. Con la creazione del WWW e l'avvento di connessioni a Internet sempre più veloci, l'elaborazione distribuita è diventata una realtà comune tra personal computer.
- A seguito della crescita del numero delle richieste di connessione a internet, il supporto al networking da parte dei sistemi operativi è diventato uno standard. Tuttavia la crescita della connettività ha determinato come conseguenza l'aumento delle minacce alla sicurezza.
- La Microsoft Corporation è diventata dominante negli anni '90, nel 1990 rilasciò la versione 3.0 di Windows mentre le versioni 95 e 98 dominarono il mercato domestico degli ultimi anni '90

- Tecnologia ad oggetti: nei sistemi operativi orientati agli oggetti (Object-Oriented Operating Systems, OOSS) gli oggetti rappresentano componenti del sistema operativo e risorse di sistema. I concetti della programmazione orientata agli oggetti, come l'ereditarietà e le interfacce, sono stati sfruttati per creare sistemi operativi modulari più facili da estendere rispetto ai sistemi operativi progettati con le tecniche precedenti e la cui manutenzione fosse più semplice. La modularità facilita il supporto di nuove e diverse architetture da parte del sistema operativo.
- Il movimento Open-Source: un altro sviluppo nella comunità del computing fu quello del movimento open-source. La maggior parte del software è creato scrivendo codice sorgente in uno dei linguaggi di programmazione ad alto livello. Tuttavia gran parte del software commerciale è venduto solo come codice oggetto ossia il codice sorgente già compilato che i computer possono comprendere direttamente. Il codice sorgente non viene incluso per permettere ai produttori di nascondere informazioni riservate e tecniche di programmazione. Nonostante ciò durante gli anni novanta il software gratuito ed open-source ha cominciato a diventare molto comune (es: Linux ed il Web Server Apache). Stallman fondò la Free Software Foundation e creò il progetto GNU (nome ricorsivo che sta per Gnu is not Unix), Stallman era convinto che permettere agli utenti di modificare e di distribuire liberamente il software avrebbe portato a software migliore, guidato dalle necessità degli utenti e non dalle aspirazioni di profitto personali o aziendali. Quando Linus Torvald creò la versione originale del sistema operativo Linux, utilizzò la maggior parte dei tool messi a disposizione gratuitamente dalla GNU sotto la direzione della General Public License (GPL). La GPL specifica che ognuno è libero di modificare e ridistribuire il software disponibile sotto tale licenza, a patto che le modifiche effettuate siano chiaramente indicate e che ogni derivato dal software di partenza sia distribuito a sua volta sotto il controllo della GPL. Sebbene la maggior parte del software controllato da GPL sia disponibile gratuitamente, la GPL richiede solo che il suo software sia libero, nel senso che gli utenti possano modificarlo e ridistribuirlo. Quindi i venditori sono autorizzati a richiedere un pagamento per rilasciare software GPL e relativo codice sorgente, ma non possono impedire agli utenti finali di modificare e ridistribuire tale software. Verso la fine degli anni '90 fu creata la Open Source Initiative (OSI) per proteggere il software open source e per promuovere i benefici della programmazione open source.
- I sistemi operativi divennero negli anni '90 sempre più user-friendly, erano ormai in grado di memorizzare i "profili" degli utenti in modo di utilizzarli tanto per l'autenticazione quanto per la personalizzazione dell'interfaccia.

→ Il 2000 e oltre

- Ai giorni nostri il middleware, un software che collega due applicazioni separate (spesso attraverso una rete) ha assunto un'importanza vitale in quanto molto comune nelle applicazioni web. Il middleware si comporta come un corriere che trasporta messaggi tra il Web server ed il database, semplificando la comunicazione tra architetture diverse.
- I Web service sono in grado di gestire un insieme di servizi standard tale da permettere a due computer qualsiasi di comunicare e scambiarsi dati via Internet. Un Web service comunica attraverso una rete fornendo un insieme di operazioni ben preciso che altre applicazioni possono richiedere (i dati sono trasferiti utilizzando protocolli standard come l'HTTP.
- I linguaggi di programmazione sequenziali che permettono di specificare una sola operazione alla volta fanno ora da complemento ai linguaggi di programmazione concorrente, come Java, che permette la specifica di operazioni parallele; in Java le unità di computazione parallela sono specificate attraverso i thread.
- Un crescente numero di sistemi adotta il cosiddetto parallelismo spinto (massive parallelism), si tratta di sistemi con un gran numero di processori in grado di svolgere in parallelo numerose elaborazioni indipendenti.

- I sistemi operativi open-source, come Linux, diventeranno più diffusi e utilizzeranno lo standard API (Application Programming Interface), per esempio POSIX (Portable Operating System Interface), per migliorare la compatibilità con altri sistemi operativi basati su UNIX.

→ Piattaforme

- Il sistema operativo fornisce una serie di API utilizzabili dai programmatori per interagire con l'hardware e svolgere altre operazioni. Le API mettono a disposizione le cosiddette chiamate di sistema (system call) per mezzo delle quali un programma utente richiede al sistema operativo un certo lavoro.
- Lo spazio utente, contiene componenti software che non sono parte del sistema operativo e che non hanno accesso diretto alle risorse fisiche del sistema. Il *kernel* (nucleo) contiene componenti software che sono parte del sistema operativo ed hanno pieno accesso alle risorse del sistema.
- Se un'applicazione tenta di utilizzare in modo illecito le risorse di sistema o se un'applicazione tenta di utilizzare risorse che non le sono state concesse, il sistema operativo deve intervenire per evitare che l'applicazione in questione possa danneggiare il sistema o interferire con altre applicazioni.
- Quando una piattaforma, cioè la combinazione di hardware, sistema operativo ed ambiente in cui vengono sviluppate le applicazioni, è ben assestata, diventa molto difficile chiedere agli utenti ed agli sviluppatori di convertirsi completamente ad un nuovo ambiente di sviluppo fornito da un sistema operativo sostanzialmente diverso.

→ Ambienti applicativi

- I computer general-purpose (di utilizzo generale) sono dotati di un vasto insieme di risorse , come una quantità dimensionabile dall'utente, processori ad alta velocità, dischi di grande dimensioni e diversi dispositivi periferici, computer del genere sono utilizzati solitamente come personal computer o come workstation.
- I sistemi embedded (dedicati) impongono invece ai loro progettisti una sfida diversa. Si tratta di presentare un piccolo insieme di risorse specializzate che offrono funzionalità a a dispositivi come telefoni cellulari e PDA. Negli ambienti embedded la gestione efficiente delle risorse è la chiave per costruire un sistema operativo di successo.
- I sistemi real-time richiedono che i compiti siano svolti entro un preciso periodo di tempo, spesso anche molto breve. I sistemi operativi real-time devono consentire ai processi di rispondere immediatamente agli eventi critici. I sistemi real-time di tipo soft assicurano che i processi real-rime siano eseguiti con un'alta priorità, ma non garantiscono il rispetto di eventuali vincoli temporali. I sistemi real-time di tipo hard, invece, garantiscono il rispetto di tali vincoli.
- Nei sistemi mission-critical se uno dei processi non rispetta i vincoli temporali, si verifica un vero e proprio fallimento del sistema.
- I sistemi business-critical, come i Web server e i database, devono essere affidabili nel raggiungimento dei loro obbiettivi. Nell'e-business si tratta di garantire velocità di risposta agli utenti che acquistano via Internet; in grandi aziende si tratta di offrire agli impiegati un sistema informativo efficiente e di assicurare che le informazioni importanti siano protette da imprevisti quali la mancata alimentazione o il guasto dei dischi. A differenza dei sistemi mission-critical, per questi, in generale, non è da considerare un fallimento il fatto che a volte il sistema possa non raggiungere i suoi obbiettivi.
- Alcuni sistemi operativi devono gestire dispositivi hardware non sempre presenti fisicamente nella macchina. Una macchina virtuale (VM, Virtual Machine) è un'astrazione software di un computer che spesso esegue un'applicazione utente appoggiandosi al sistema operativo nativo della macchina. Le macchine virtuali si interfacciano con l'hardware del sistema attraverso il sistema operativo; altri programmi utenti possono interagire con le VM.

Le macchine virtuali favoriscono la portabilità cioè la capacità del software di essere su più piattaforme. La Java Virtual Machine (JVM) è una delle macchine virtuali più usate e diffuse. È la base della piattaforma Java e permette alle applicazioni Java di essere eseguite su una qualsiasi JVM della versione corretta, indipendentemente dalla piattaforma su cui è installata. Le macchine virtuali tendono ad essere meno efficienti delle macchine reali perché accedono all'hardware indirettamente o simulano hardware che non è realmente connesso alla macchina. L'accesso indiretto o simulato all'hardware incrementa il numero di istruzioni software richieste per eseguire ogni azione hardware.

Componenti dei sistemi operativi

I sistemi operativi si sono evoluti per rispondere alla crescente richiesta di nuove caratteristiche e maggior efficienza, che esigevano un hardware più potente.
Un utente interagisce con il sistema operativo attraverso una o più applicazioni e spesso attraverso un'applicazione speciale chiamata shell o anche interprete dei comandi. La maggior parte delle attuali shell è implementata come interfaccia basata su testo che permette agli utenti di inoltrare comandi al sistema o come GUI che consentono all'utente di indicare e selezionare (point & click) e trascinare e rilasciare (drag & drop) icone per richiedere vari servizi al SO (es: mandare in esecuzione un'applicazione).
Il software che contiene i componenti principali del SO è chiamato kernel. I tipici componenti di un sistema operativo sono:

- lo scheduler dei processi, che determina quando e per quanto ogni singolo processo sarà in esecuzione su un certo processore.
- Il gestore della memoria, che determina quando e quanta memoria è allocata ai processi e cosa fare quando non c'è più spazio nella memoria principale.
- Il gestore dei dispositivi I/O, che soddisfa le richieste di I/O da e verso i dispositivi hardware.
- Il gestore delle interprocess communication (IPC), che permette ai processi di comunicare tra loro.
- Il gestore del file system, che organizza raccolte di dati su dispositivi di memorizzazione e fornisce un'interfaccia per accedere a tali dati.

Quasi tutti i moderni SO offrono supporto ad un ambiente multiprogrammato in cui più applicazioni sono eseguibili concorrentemente. Una delle principali responsabilità del SO è determinare quale processore debba eseguire un certo processo e per quanto tempo tale processo debba restare in esecuzione.
Un programma può contenere diversi elementi che condividono dati e che sono eseguibili concorrentemente. Tali componenti, che vengono eseguiti indipendentemente ma che svolgono il loro lavoro in uno spazio di memoria comune, sono chiamati thread.

In genere, per utilizzare il processore competono molti processi. Lo scheduler dei processi può basare la sua scelta su diversi criteri come l'importanza di un processo, il tempo di esecuzione stimato o il tempo che questo ha già trascorso in attesa del processore.

Il gestore della memoria alloca memoria al sistema operativo ed ai processi. Per assicurarsi che i processi non possano interferire con il SO o tra di loro, il gestore della memoria impedisce ai processi di accedere a zone di memoria che non sono state loro allocate.

I dispositivi di memorizzazione e le schede di rete sono dispositivi tanto di input quanto di output. Quando un processo desidera utilizzare un dispositivo di I/O, deve effettuare una chiamata di sistema al SO. Tale chiamata viene poi gestita dal device driver, un componente software che interagisce direttamente con l'hardware, per mezzo di istruzioni fortemente dipendenti dal tipo di dispositivo, al fine di permettere il completamento del servizio richiesto.

Molti sistemi di elaborazione possono memorizzare i dati in forma permanente. Dato che la memoria principale è spesso relativamente piccola e volatile, sono utilizzati a tale scopo dispositivi di memoria permanente secondari. Si noti, tuttavia, che le memorie secondarie sono molto più lente rispetto a processore e memoria principale.

Lo scheduler del disco è quel componente del SO responsabile di gestire le richieste di operazioni su disco nel tentativo di massimizzare le prestazioni e minimizzare la quantità di tempo che un processo deve attendere per il loro completamento. I sistemi RAID (Redundant Array of Independent Disks) utilizzano un insieme ridondante di dischi indipendenti per cercare di ridurre i tempi di attesa delle operazioni di I/O da parte dei processi.

I sistemi operativi utilizzano i file system per organizzare ed accedere in modo efficiente a raccolte di dati chiamate file e registrate su dispositivi di memorizzazione.

I processi o i thread cooperano spesso per raggiungere un obbiettivo comune. Molti sistemi operativi forniscono così meccanismi per permettere la comunicazione e la sincronizzazione tra processi (IPC) in modo da semplificare la programmazione concorrente. La comunicazione tra processi permette loro di comunicare attraverso messaggi spediti dai processi stessi, mentre la sincronizzazione tra processi consiste in strutture utilizzabili per condividere opportunamente i dati.

Obbiettivi dei sistemi operativi

- efficienza
- robustezza
- scalabilità
- espandibilità
- portabilità
- sicurezza
- interattività
- usabilità

Un SO efficiente raggiunge un altro throughput e un basso tempo medio di turnaround. Il throughput misura la quantità di lavoro che un processore può completare in un certo intervallo di tempo. Uno dei compiti di un SO è quello di fornire servizi a molte applicazioni, un SO efficiente minimizza il tempo necessario a fornire tali servizi.

Un SO robusto è tollerante ai guasti ed affidabile, il sistema non deve bloccarsi del tutto a causa d errori in applicazioni isolate o nell'hardware e, nel caso dovesse bloccarsi, lo deve fare in modo graduale e controllato nel tentativo di minimizzare la quantità di lavoro perso e di evitare danni all'hardware del sistema. Un tale SO continuerà a fornire servizi alle applicazioni a meno di un guasto all'hardware.

Un SO scalabile è in grado di utilizzare le risorse che vengono aggiunte al sistema, se un sistema non fosse scalabile si potrebbe presto arrivare ad un punto in cui l'aggiunta di ulteriori risorse al sistema non sarebbe pienamente utilizzata. Un SO scalabile può rapidamente variare il suo grado di multiprogrammazione. La scalabilità è una caratteristica particolarmente importante nei sistemi multiprocessore, in quanto al crescere del numero di processori nel sistema dovrebbe idealmente corrispondere la crescita delle capacità di elaborazione in proporzione al numero di processi, mentre in pratica questo non accade.

Un SO espandibile è in grado di adattarsi bene alle nuove tecnologie e presenta la capacità di svolgere compiti che vanno oltre a quelli pensati in fase di progetto del SO stesso.

Un SO portabile è progettato per poter funzionare su un gran numero di configurazioni hardware. La portabilità delle applicazioni è anche importante, in quanto sviluppare applicazioni è costoso; il vantaggio di rendere eseguibile una stessa applicazione su diverse configurazioni hardware permette di ridurre i costi di sviluppo. Il SO è fondamentale per raggiungere questo tipo di portabilità.

Un SO sicuro impedisce agli utenti ed al software di accedere a servizi e risorse senza l'opportuna autorizzazione. Con il termine protezione ci si riferisce ai meccanismi che realizzano la politica di sicurezza del sistema.

Un SO interattivo permette alle applicazioni di rispondere velocemente alle azioni dell'utente o agli eventi.

Un SO usabile ha la potenzialità di essere adatto ad un gran numero di utenti, sistemi di questo tipo sono normalmente provvisti di un interfaccia utente molto facile da usare.

Architetture dei sistemi operativi

I moderni SO tendono ad una certa complessità, in quanto forniscono molti servizi e supportano una gran varietà di risorse hardware e software. Le architetture dei SO aiutano i progettisti a gestire tale complessità fornendo un'organizzazione per i componenti del sistema e specificando i privilegi con cui eseguirli.

- Architettura monolitica: il SO monolitico è l'architettura più vecchia e più comune. Ogni componente fa parte del kernel e può comunicare direttamente con gli altri utilizzando semplici chiamate di procedura, il kernel è solitamente in esecuzione senza restrizioni per quanto riguarda l'accesso al sistema. La comunicazione diretta tra i componenti rende questo tipo di sistemi operativi molto efficiente, però può risultare molto difficile isolare fonti di errore o malfunzionamenti proprio per il fatto che tutti i componenti sono raggruppati nel kernel. Inoltre, dato che l'intero codice viene eseguito con piena possibilità di accesso al sistema, i sistemi con kernel monolitici sono particolarmente esposti a danni derivati da codice accidentalmente o intenzionalmente errato (come quello dei virus).
- Architettura a strati: l'approccio a strati del nel progetto dei sistemi operativi tenta di risolvere queste problematiche raggruppando in strati i componenti che svolgono funzioni simili. Ogni strato comunica esclusivamente con quello immediatamente inferiore o superiore. Gli strati di livello inferiore forniscono servizi a quelli di livello superiore per mezzo di un'interfaccia che nasconde il modo in cui il lavoro viene svolto. I SO a strati sono molto più modulari di quelli monolitici, in quanto ogni strato è modificabile senza apportare necessariamente modifiche agli altri strati. Un sistema modulare è composto da componenti autocontenuti riutilizzabili in tutto il sistema. Ogni componente nasconde il modo in cui svolge il proprio lavoro e presenta un'interfaccia standard utilizzabile dagli altri componenti per richiedere i suoi servizi. La modularità impone che il sistema sia ben strutturato e consistente, spesso semplificando le operazioni di validazione, debug e modifica. Tuttavia una richiesta di un processo utente in un sistema a strati potrebbe doverne attraversare molti prima di poter essere servita. Le prestazioni possono dunque risultare peggiori rispetto a quelle di un kernel monolitico, inoltre dato che i vari strati hanno libero accesso al sistema, i kernel a strati sono anche esposti a danni derivati da codice accidentalmente o intenzionalmente errato.
- Architettura a microkernel: un SO a microkernel fornisce solo un piccolo numero di servizi nel tentativo di mantenere il kernel piccolo e scalabile. Questi servizi riguardano in genere la gestione a basso livello della memoria, la comunicazione tra processi (IPC) e i meccanismi base di sincronizzazione che permettono ai processi di cooperare. Nei progetti a microkernel

la maggior parte dei componenti del SO, come lo schedule, il gestore della rete, il file system ed il gestore dei dispositivi, vengono eseguiti al di fuori dello spazio del kernel con un basso livello di privilegi. Il microkernel offre un altro grado di modularità che lo rende espandibile, portabile e scalabile, inoltre dato che per essere eseguito non ha bisogno di tutti i componenti, anche se se ne bloccasse uno, ciò non causerebbe il blocco dell'intero sistema operativo. Tale modularità si ottiene tuttavia al prezzo di un elevato numero di comunicazioni tra i moduli che finisce per peggiorare le prestazioni del sistema.

- Sistemi operativi di rete e distribuiti: un SO di rete permette ai processi di accedere a risorse che si trovano su altri computer indipendenti collegati in rete. La struttura di molti sistemi operativi di rete e distribuiti si basa spesso sul modello client/server. I computer client richiedono risorse attraverso un appropriato protocollo e i server rispondono fornendo quelle appropriate. In tali reti, i progettisti devono considerare attentamente le problematiche di gestione dei dati e delle comunicazioni tra computer. Alcuni sistemi operativi sono più "di rete" di altri. In un ambiente di rete un processo è eseguibile sul computer in cui è stato creato o su un altro computer della rete. In alcuni SO di rete gli utenti possono specificare esattamente dove i loro processi saranno eseguiti, in altri invece è il SO a prendere queste decisioni. I file system di rete sono componenti molto importanti nei SO di rete, al livello più basso gli utenti acquisiscono le risorse su un'altra macchina collegandosi esplicitamente a quella e recuperando i file; mentre quelli di alto livello, invece, permettono agli utenti di accedere a file remoti come se questi fossero nel sistema locale. Un sistema operativo distribuito è un singolo SO in grado di gestire risorse distribuite su più sistemi di elaborazione. I sistemi distribuiti forniscono l'illusione che diversi computer costituiscano un unico, potente computer, così che un processo possa accedere a tutte le risorse del sistema indipendentemente dalla locazione del processo all'interno della rete di computer del sistema distribuito. I SO distribuiti sono spesso difficili da realizzare e richiedono algoritmi complessi che permettano ai processi di comunicare e condividere dati.

Fondamenti di hardware e software -capitolo 2-

Il computing (elaborazione automatica dei dati) sembra essere una risorsa inesauribile, più cresce il numero di produttori di hardware e quello dei dispositivi, più i sistemi operativi aumentano di complessità. Per facilitare la programmazione di sistema e migliorare l'espandibilità, la maggior parte dei sistemi operativi viene progettata in modo da essere indipendente da una particolare configurazione dell'hardware di sistema. Per questo si utilizzano i device driver (gestori dei dispositivi), i quali permettono al SO di poter utilizzare un nuovo dispositivo semplicemente ricorrendo al driver opportuno.
Molti componenti hardware sono stati progettati per interagire col SO in modo tale da facilitare l'espandibilità del sistema stesso. Ad esempio, i dispositivi plug-and-play, quando vengono connessi al computer, possono segnalare la loro presenza ed identificarsi. Questo permette al SO di selezionare ed utilizzare un device driver corretto con un minimo (alle volte nullo) coinvolgimento dell'utente, semplificando così l'installazione di un nuovo dispositivo.

Componenti hardware che un SO gestisce per soddisfare le esigenze computazionali dell'utente:

- Schede madri: per permettere a dispositivi indipendenti di comunicare tra loro, i computer sono dotati di una o più schede a circuito stampato (Printed Circuit Board, PCB), un PCB è un componente hardware che fornisce connessioni elettriche tra vari dispositivi dislocati sulla scheda. La scheda principale (mainboard) o scheda madre (motherboard), il principale PCB di un sistema, è concepibile come la spina dorsale del computer. La scheda madre fornisce alcuni slot (connettori di forma lineare) in cui vengono inseriti altri componenti, come il processore e la memoria principale/centrale. La scheda madre è uno dei quattro

componenti richiesti per eseguire istruzioni in un computer general-purpose, gli altri tre sono processore, memoria principale e memoria secondaria.

- Le schede madri sono generalmente composte da diversi strati di materiale isolante, derivato dal silicio, estremamente sottili contenenti microscopiche connessioni elettriche chiamate tracce, che servono sia da canali di comunicazione sia da fornitori di connettività alla scheda. Un grande insieme di tracce forma un canale di comunicazione ad alta velocità conosciuto come bus.
- La maggior parte delle schede madri è dotata inoltre di diversi circuiti integrati (chip) che svolgono operazioni di basso livello.
- Le schede madri dispongono in genere di un circuito integrato chiamato Basic Input/Output System (BIOS) che memorizza le istruzioni di base per l'inizializzazione e la gestione dell'hardware. Il BIOS è anche responsabile del trasferimento in memoria principale della porzione iniziale del SO, un'operazione chiamata bootstrapping. Dopo tale operazione, il SO utilizza il BIOS per comunicare con l'hardware di sistema allo scopo di compiere operazioni di I/O di base.
- Le schede madri contengono anche dei circuiti integrati chiamati controller (controllori) che si occupano di gestire il trasferimento dei dati tramite il bus della scheda. Il chipset di una scheda madre è l'insieme di tutti i controller, i processori, i bus e dell'ulteriore hardware integrato sulla scheda, che determina le caratteristiche hardware del sistema.
- Molte schede madri moderne contengono chip che si occupano di elaborazione grafica, networking e gestione del sistema RAID (Redundant Array of Indipendent Disks). Questi on-board device riducono il costo del sistema complessivo e contribuiscono significativamente alla riduzione dei prezzi. Lo svantaggio principale di questi dispositivi è che, essendo integrati nella scheda madre, non possono essere facilmente sostituiti.

- Processori: un processore è un componente hardware che esegue un flusso di istruzioni in linguaggio macchina. I processori dei computer possono essere di diverso tipo: l'unità centrale di elaborazione (Central Processing Unit, CPU same as processore/processore general-purpose), un coprocessore grafico o un processore di segnali digitali (Digital Signal Processor, DSP). Una CPU è un processore che esegue le istruzioni di un programma, essa esegue la maggior parte delle istruzioni ma può incrementare l'efficienza del sistema assegnando compiti computazionali particolarmente intensi e coprocessori progettati specificatamente per gestirli.
  Le istruzioni eseguibili da un processore sono definite dal loro instruction set (insieme di istruzioni). La dimensione di ogni istruzione, detta lunghezza di un'istruzione, può differire tra architetture diverse e nell'ambito della stessa architettura (alcuni processori possono supportare istruzioni con dimensioni diverse). L'architettura del processore determina inoltre la quantità di dati elaborabile contemporaneamente.
  I processori moderni, al fine di incrementare le prestazioni, svolgono in hardware molte operazioni di gestione delle risorse. Tali operazioni includono il supporto per la memoria virtuale e per le interruzioni.
  Alcuni componenti sono comuni a tutti i processori moderni come l'unità di caricamento (fetch) delle istruzioni, il revisore dei salti (branch predictor), l'unità di esecuzione, i registri, le memorie cache ed un'interfaccia con il bus.
  - L'unità di caricamento delle istruzioni copia all'interno delle unità di memoria ad alta velocità chiamate registri delle istruzioni, le istruzioni da eseguire, in modo che siano velocemente eseguibili dal processore. L'unità di decodifica delle istruzioni interpreta l'istruzione e fornisce all'unità di esecuzione le indicazioni necessarie per il suo svolgimento. La parte più significativa dell'unità di esecuzione è l'unità aritmetico-logica (arithmetic and logic unit, ALU) che svolge le operazioni aritmetiche e logiche di base.

- L’interfaccia verso il bus permette al processore di interagire con la memoria e gli altri dispositivi del sistema.
- Le memorie ad alta velocità, memorie cache, servono a contenere copie dei dati presenti nella memoria centrale questo perché i processori lavorano in genere a velocità molto più elevate della memoria principale. Le cache incrementano l’efficienza del processore, permettendo un rapido accesso a dati ed istruzioni. Dato che le cache sono significativamente più costose della memoria centrale tendono ad essere relativamente piccole. Le cache possono essere di livello 1 (L1), le più veloci e costose che si trovano all’interno del processore oppure di livello 2 (L2), più grandi e lente che spesso sono situate sulla scheda madre (nonostante la tendenza sia quella di integrarle nel processore in modo tale da incrementare le prestazioni).
- I registri sono memorie ad altra velocità situate nel processore che contengono dati per utilizzo immediato. I dati vanno memorizzati nei registri prima che un processore possa operare su di loro. Memorizzare le istruzioni in un qualunque tipo di memoria più lenta sarebbe inefficiente, in quanto il processore rimarrebbe inutilizzato durante l’attesa dei dati. Esistono registri dedicati all’esecuzione dei programmi (program execution register) e la metà è di solito può essere riservata alle applicazioni che li usano per accedere velocemente a dati e puntatori durante l’esecuzione. Tali registri sono chiamati general-purpose (registri di uso generale). Gli altri registri (spesso chiamati registri di controllo) memorizzano specifiche informazioni sul sistema, come il contatore di programma (program counter), registro utilizzato dal processore per determinare la successiva istruzione da eseguire.

- Clock: il tempo nei calcolatori è spesso misurato in cicli, chiamati anche tick del clock. Il termine ciclo si riferisce alla completa oscillazione di un segnale elettrico fornito dal generatore di clock. Come un direttore d’orchestra il generatore di clock imposta la cadenza dell’intero sistema, in particolare determina la frequenza seguita dai bus per trasferire i dati, generalmente misurata in cicli al secondo, o hertz (Hz). Il frontside bus (FSB) che connette i processori ai moduli di memoria, lavora generalmente a diverse centinaia di megahertz (Mhz).

- Gerarchia di memoria: la dimensione e la velocità della memoria sono limitate da leggi fisiche ed economiche. Essendoci un limite alla velocità a cui gli elettroni possono viaggiare, più è lunga la distanza tra due terminali, più durerà il trasferimento dei dati. Inoltre, è economicamente proibitivo equipaggiare il processore con grandi quantità di memoria in grado di fornire dati ad una velocità simile alla sua.
  La gerarchia di memoria rappresenta il compromesso costo/prestazioni. La memoria più veloce è in cima alla gerarchia ed è in genere la più piccola, la più lenta ed economica sta in fondo alla gerarchia ed è in genere la più capiente.
  I registri rappresentano la memoria più veloce e costosa del sistema, e lavorano alla stessa frequenza del processore. La velocità delle memorie cache è misurata per mezzo della loro latenza, che rappresenta il tempo necessario per trasferire i dati.
  Nella maggior parte dei processori di oggi le cache L1 (decine di kilobyte di dati) e L2 (da centinaia di kilobyte a diversi megabyte) sono integrate nei processori in modo da poter sfruttare le interconnessioni ad alta velocità presenti al suo interno. Alcuni processori di fascia alta possono contenere anche un terzo livello di cache (cache L3) che è più lenta della cache L2 ma più veloce della memoria centrale.
  Proseguendo nella gerarchia si arriva alla memoria centrale, chiamata anche memoria reale o fisica, essa presenta latenze maggiori in quanto i dati devono passare attraverso il frontside bus, che lavora generalmente ad una frazione della velocità del processore.
  I registri, le cache e la memoria centrale sono generalmente memorie volatili, in quanto i dati memorizzati svaniscono quando non sono alimentati. I dischi rigidi (centinaia di

gigabyte) ed altri dispositivi di memorizzazione come CD, DVD e nastri (dispositivi persistenti, i dati vengono conservati anche quando questi non sono alimentati) sono i dispositivi di memorizzazione di un calcolatore meno costosi, ma i più lenti.
Il SO esegue in genere un altro processo per migliorare il fattore di utilizzo del processore, piuttosto che permettere ad un processore di rimanere inutilizzato nell'attesa di dati dalle memorie secondarie.

- Memoria centrale (principale): la memoria centrale consiste di memoria ad accesso casuale volatile (Random Access Memory, RAM), dove "casaule" sta ad indicare che i processi possono accedere al suo contenuto in qualsiasi ordine. Al contrario, i dati memorizzati su dispositivi di memorizzazione ad accesso sequenziale, come i nastri, devono essere letti sequenzialmente.
  Il tipo più come di RAM è la Ram dinamica (DRAM), che però richiede la presenza di un circuito di refresh che periodicamente, un po' di volte ogni millisecondo, ne rilegga il contenuto rinforzando la memorizzazione dei dati che altrimenti andrebbero persi.
  La RAM statica (SRAM), invece non ha bisogno di alcun refresh per mantenere memorizzati i suoi dati, essa è normalmente utilizzata per realizzare le memorie cache ed è in genere più veloce e costosa della DRAM.
  Un obbiettivo molto importante per i produttori di DRAM è quello di diminuire la differenza di velocità tra il processore e la memoria, i moduli di memoria sono progettati al fine di minimizzare la latenza nell'accesso ai dati del modulo e di massimizzare il numero di dati trasferiti al secondo. Queste tecniche consentono di ridurre la latenza complessiva ed incrementare l'ampiezza di banda (bandwidth), che rappresenta la quantità di dati trasferibile nell'unità di tempo.

- Memoria secondaria: a causa della limitata capacità e della volatilità, la memoria centrale non è adatta a memorizzare dati in grandi quantità o dati che devono permanere anche in assenza di alimentazione. I calcolatori utilizzano quindi i dispositivi di memorizzazione secondaria (secondary storage, memoria secondaria, persistente o ausiliaria) che mantengono i dati anche in assenza di alimentazione. La maggior parte dei computer utilizza i dischi rigidi come memoria secondaria. Nonostante contengano più dati e costino meno, non sono utilizzabili come memoria principale, in quanto presentano tempi di accesso troppo elevati rispetto ad essa.
  Accedere dati memorizzati su disco rigido richiede un opportuno spostamento meccanico della testina di lettura/scrittura, una latenza rotazionale nell'attesa che il dato passi sotto la testina ed il tempo di trasferimento del dato tramite questa. Inoltre, i dati devono essere trasferiti dal disco alla memoria principale prima che questi siano utilizzabili dal processore. Un disco rigido è un esempio di dispositivo a blocchi, in quanto trasmette i dati in quantità costanti di dimensione fissa.
  Per facilitare le operazioni di backup e lo scambio di dati tra computer, alcuni dispositivi di memoria secondaria memorizzano i dati su supporti rimuovibili a bassa capacità (Compact Disc, CD), tuttavia questi presentano in genere latenze maggiori rispetto ad altri dispositivi, tra cui i dischi rigidi.

- Bus: un bus è un insieme di tracce (o altre connessioni elettriche) che trasporta informazioni tra dispositivi hardware. Tali dispositivi inviano segnali elettrici sul bus per comunicare tra loro. La maggior parte dei bus è composta da un bus dati, che trasporta dati, e da un bus indirizzi, che determina il destinatario o il mittente dei dati. Una porta (port) è un bus che connette esattamente due dispositivi. Un bus condiviso da diversi dispositivi per svolgere operazioni di I/O è anche chiamato canale di I/O (I/O channel).
  I canali di I/O ed il processore possono tentare di accedere contemporaneamente alla memoria centrale. Per evitare la collisione dei due segnali sul bus, l'accesso alla memoria è

gestito sulla base di priorità da un dispositivo hardware chiamato controller, che in genere considera i canali di I/O con una priorità maggiore rispetto al processore (meccanismo chiamato cycle stealing, furto di ciclo). I canali di I/O comunque sfruttano solo una piccola percentuale dei cicli di processore totali, cosa che è in genere controbilanciata da un utilizzo avanzato dei dispositivi.
Appena la velocità del FSB aumenta (processore → memoria centrale) si incrementa la quantità di dati trasferiti tra memoria centrale e processore in unità di tempo, con un conseguente miglioramento delle prestazioni del sistema.
La realizzazione chiamata Double Data Rate (DDR) progetta un FSB che è in grado di offrire prestazione come se operasse al doppio della sua velocità iniziale, in quanto riesce a far svolgere due trasferimenti in memoria per ogni ciclo di clock. Un altro tipo di realizzazione è chiamata quad pumping e permette di far avvenire fino a quattro trasferimenti di dati per ciclo, effettivamente quadruplicando l'ampiezza di banda della memoria del sistema.
Il bus PCI (Peripheral Components Interconnect) connette al resto del sistema i dispositivi periferici, come le schede audio o le schede di rete.
La Accelerated Graphics Port (AGP) si utilizza principalmente per le schede grafiche che richiedono in genere decine o centinaia di megabyte di RAM per gestire grafica tridimensionale in tempo reale.

- Accesso diretto alla memoria (DMA): la maggior parte delle operazioni di I/O trasferisce dati tra la memoria centrale ed un dispositivo di I/O. Nei primi calcolatori questo trasferimento avveniva utilizzando l'I/O programmato (programmed I/O, PIO) o a controllo di programma che permetteva di specificare quale byte, o quale parola, si dovesse trasferire e poi attendeva il completamento dell'operazione.
  Questo causava un grosso spreco di cicli di processore, motivo per cui i progettisti realizzarono il cosiddetto I/O guidato dalle interruzioni (interrupt-driven I/O) che permetteva al processore di avviare un'operazione di I/O continuando però nel frattempo ad eseguire istruzioni software. Al termine dell'operazione di I/O, il dispositivo di I/O coinvolto segnalava al processore, per mezzo di un interrupt, il completamento dell'operazione.
  L'accesso diretto alla memoria (Direct Memory Access, DMA) è una tecnica che migliora l'I/O guidato delle interruzioni, permettendo ai dispositivi ed ai controller di trasferire interi blocchi di dati mentre il processore continua ad eseguire istruzioni software. Un canale DMA utilizza un controller di I/O per gestire il trasferimento dei dati tra dispositivo e memoria centrale (o viceversa). Per comunicare al processore l'avvenuto trasferimento tale controller genera un'interruzione al termine dell'operazione.
  La tecnica DMA è compatibile con diverse architetture di bus; nel bus PCI, il DMA viene gestito direttamente utilizzando la tecnica di bus mastering, che consiste nel cedere il controllo del bus a un dispositivo PCI che si occupa di gestire direttamente il trasferimento dati. Questa tecnica è molto più efficiente della precedente ed è stata introdotta nella maggior parte delle moderne architetture di bus.

- Dispositivi periferici: un dispositivo periferico (o semplicemente una periferica), è un qualsiasi dispositivo hardware non strettamente necessario ad un calcolatore per eseguire istruzioni software. Col termine periferica si intende quindi un vasto insieme di dispositivi di I/O (stampanti, scanner e mouse), dispositivi di rete e dispositivi di memorizzazione. I dispositivi periferici interni sono spesso detti "integrati" (schede audio, modem, lettori CD interni). Il disco rigido è forse la periferica più comune. Le tastiere ed i mouse sono un esempio di dispositivi a caratteri, dispositivi che trasferiscono i dati un carattere per volta. Le periferiche sono collegabili al computer tramite porte ed altri bus, le porte seriali trasferiscono i dati un bit alla volta e sono generalmente usate per collegare dispositivi quali

tastiere e mouse; le porte parallele trasferiscono dati diversi un bit per volta e sono di solite utilizzate per collegare stampanti. Lo Universal Serial Bus (USB) è un interfaccia seriale ad alta velocità molto diffusa. La Small Computer System Interface (SCSI) è una nota interfaccia parallela.
Le porte USB permettono di trasferire dati e forniscono anche l'alimentazione a dispositivi come dischi rigidi esterni, fotocamere digitali e stampanti. I dispositivi USB possono essere collegati, riconosciuti e rimossi a computer acceso senza danneggiare l'hardware del sistema (tecnica chiamata hot swapping).
Altre interfacce utilizzare per collegare periferiche al sistema sono la SCSI e al ATA (Advanced Technology Attachment), che realizza l'interfaccia IDE (Integrated Drive Electronics). Queste interfacce permettono il trasferimento dati tra dispositivi come dischi rigidi (o lettori CD/DVD) ed il controller della scheda madre, da dove poi vengono indirizzati sul bus opportuno (interfacce wirless come la Bluetooth per connessioni wirless a breve raggio, sono un esempio di interfacce più recenti).

Supporto hardware per i sistemi operativi

Per realizzare quei meccanismi di protezione che impediscano ai processi di utilizzare istruzioni privilegiate o di accedere ad aree di memoria riservate, la maggior parte dei sistemi operativi si basa sui processori. Se i processori tentano di violare i meccanismi di protezione del sistema, il processore informa il SO, che può quindi intervenire a dovere. Il processore invoca il SO anche quando è necessario rispondere a segnali provenienti da dispositivi hardware.

I calcolatori hanno in genere diverse modalità di esecuzione. Poter cambiare la modalità di esecuzione di una macchina permette di realizzare sistemi più robusti, sicuri e tolleranti ai guasti. Per le applicazioni utente, il sottoinsieme di istruzioni eseguibili in modalità utente (chiamata anche stato utente o stato del problema) non permette, per esempio, l'esecuzione diretta di istruzioni di I/O; un'applicazione utente abilitata a svolgere arbitrariamente operazioni di I/O potrebbe copiare la lista delle password di accesso al sistema, stampare informazioni relative ad un qualsiasi altro utente o addirittura distruggere il SO. Il SO in genere è eseguito in modalità kernel (o in stato supervisore); in tale modalità ha accesso a tutto l'instruction set. In modalità kernel, un processore può eseguire le cosiddette istruzioni privilegiate ed accedere alle risorse. Il binomio "modalità kernel" e "modalità utente" è ancora valido per la maggior parte dei sistemi moderni, tuttavia nei sistemi altamente sicuri è auspicabile avere più di due modalità, così da poter utilizzare meccanismi di protezione a grana più fine. La presenza di stati multipli permette di controllare gli accessi al sistema sulla base del principio del privilegio minimo. Tale principio prevede che ad ogni utente debba essere garantita esclusivamente la minima quantità di privilegi e di accessi necessaria per svolgere i proprio compiti.

Molti processori forniscono meccanismi per la protezione e la gestione della memoria. La protezione della memoria, che serve a impedire ai processi di accedere a locazioni di memoria non a loro assegnate, per esempio la memoria assegnata ad altri utenti o al SO, è realizzata utilizzando alcuni registri del processore modificabili solo tramite istruzioni privilegiate. Il processore controlla il valore di questi registri per assicurarsi che il processo non acceda ad aree di memoria non lecite. Nei sistemi che non usano la memoria virtuale, ai processi viene allocato solo un blocco contiguo di indirizzi di memoria. Il sistema può impedire che tali processi accedano a locazioni di memoria non consentite per mezzo di registri limite (bounds registers), che specificano gli indirizzi di inizio e fine della zona di memoria allocata per un certo processo. La protezione è quindi imposta controllando ogni volta se un dato indirizzo appartenga o meno al blocco di memoria valido per il processo che lo ha generato. Molte operazioni di protezione hardware si svolgono in parallelo all'esecuzione delle istruzioni del programma, in modo tale da limitare le prestazioni.

I sistemi con memoria virtuale permettono ai programmi di referenziare indirizzi che non devono necessariamente corrispondere all'insieme limitato degli indirizzi reali o indirizzi fisici. Durante l'esecuzione dei processi il SO trasforma dinamicamente gli indirizzi virtuali generati dai processi stessi nei corrispondenti indirizzi fisici. I sistemi con memoria virtuale permettono ai processi di referenziare spazi di indirizzamento molto più grandi rispetto allo spazio di indirizzamento disponibile in memoria centrale.

I processori informano il SO del verificarsi di eventi come errori di esecuzione di un programma o dei cambiamenti nello stato di un dispositivo. Un processore fa questo controllando ripetutamente lo stato di ogni dispositivo, tecnica chiamata polling. Questo può però causare un notevole sovraccarico, soprattutto se la maggior parte dei dispositivi non ha cambiato il proprio stato dall'ultimo controllo.
Per questa ragione, quando si verifica un certo evento, la maggior parte dei dispositivi invia al processore un segnale chiamato interruzione (interrupt). Il sistema operativo può rispondere al cambiamento di stato di un dispositivo e notificare la cosa ai processi che sono in attesa per quell'evento. Le eccezioni sono interruzioni generate in risposta a errori, come un guasto nell'hardware, un errore logico o una violazione delle politiche di protezione. Anziché causare un blocco del sistema, per stabilire come reagire un processore chiamerà in causa il SO, il quale potrebbe disporre il blocco del processo responsabile dell'errore oppure il riavvio del sistema.
Se l'intero sistema deve bloccarsi, il SO può fare in modo che avvenga gradatamente, in modo da ridurre al minimo la quantità di lavoro perso. I processi possono anche specificare al SO particolari gestori delle eccezioni, in modo che il SO possa invocare il gestore opportuno al verificarsi di un'eccezione di un particolare tipo.

Un temporizzatore ad intervallo (interval timer) genera periodicamente un'interruzione che porta il processore ad invocare il SO. Spesso i temporizzatori ad intervallo vengono utilizzati per impedire ai processi di monopolizzare il processore.
Un clock orologio (time-of-day clock) permette al computer di tenere traccia del trascorrere del tempo e presenta in genere un'accuratezza di centesimi o millesimi di secondo. Alcuni di questi clock sono alimentati da una batteria, in modo da rimanere aggiornati anche in assenza di alimentazione, questi clock permettono al sistema di gestire il tempo con continuità.

Prima che un SO possa iniziare a gestire risorse, esso deve essere caricato in memoria. Una volta acceso il computer, il BIOS inizializza l'hardware di sistema e successivamente tenta di caricare nella memoria centrale alcune istruzioni prelevandole da una zona particolare della memoria secondaria chiamata settore di boot, effettuando quella che viene chiamata fase di bootstrap o bootstrapping. Il processore è fatto per eseguire tali istruzioni, che si occupano generalmente di caricare in memoria centrale i componenti del SO, di inizializzare i registri del processore e di preparare il sistema alle esecuzioni delle applicazioni utente.
In molti sistemi il BIOS può caricare il SO solo da una locazione predefinita su un limitato numero di dispositivi. Se il settore di boot non viene localizzato su uno dei dispositivi supportati, il sistema non sarà caricato e l'utente non sarà in grado di utilizzare l'hardware del calcolatore. Per permettere un maggior numero di funzionalità in fase di boot, la Intel Corporation ha sviluppato la Extensible Firmware Interface (EFI) in sostituzione del BIOS. La EFI mette a disposizione una shell (interfaccia testuale) con cui l'utente può utilizzare direttamente i dispositivi del calcolatore; dato che contiene anche dei driver, una volta avviato il sistema l'utente può accedere ai dischi rigidi ed alla rete.

La tecnologia plug-and-play permette ai sistemi operativi di configurare ed utilizzare nuovo dispositivi hardware appena installati senza la diretta interazione dell'utente. Un dispositivo hardware plug-and-play:

- permette al SO di identificarlo senza ambiguità.

- Comunica con il SO indicandogli le risorse ed i servizi richiesti per un corretto funzionamento.
- Permette al SO di identificare il corrispondente driver e quindi di utilizzarlo per la configurazione del dispositivo stesso.

Queste caratteristiche permettono all'utente di poter aggiungere hardware ad un sistema e di poterlo utilizzare immediatamente con il supporto adeguato del SO. Il plug-and-play si è evoluto per includere caratteristiche di gestione di potenza in grado di aumentare la durata delle batterie. Lo standard Advanced Configuration and Power Interface (ACPI) definisce un'interfaccia che permette ai SO di configurare i dispositivi e di gestire il loro consumo di potenza.

Caching e buffering

I calcolatori contengono una gerarchia di dispositivi di memorizzazione che lavorano a differenti velocità. Per migliorare le prestazioni, molti calcolatori utilizzano la tecnica di caching, mantenendo copie delle informazioni utilizzate dai processi in dispositivi di memoria veloci. Dato il costo elevato di tali dispositivi, le memorie cache possono contenere solo una piccola porzione delle informazioni presenti nei dispositivi più lenti. Come risultato, i contenuto delle cache (detto linea di cache) deve essere opportunamente gestito, per minimizzare il numero di volte in cui le informazioni richieste non sono presenti in cache; questo evento è chiamato cache miss (fallimento di accesso alla cache). Quando si verifica, il sistema deve recuperare le informazioni richieste dai dispositivi più lenti che le contengono. Un cache hit invece si verifica quando il fato ricercato è presente nella cache; in questo caso il sistema può accedere ai dati ad una velocità relativamente alta.
Per ottenere un incremento di prestazioni tramite caching, il sistema deve fare in modo che un numero significativo di accessi in memoria dia luogo a cache hit. Molte memorie cache sono gestite ricorrendo all'euristica , insieme di regole empiriche e altre approssimazioni che permettono di ottenere buoni risultati con un sovraccarico di esecuzione relativamente basso.
Le cache L1 e L2 del processore memorizzano i dati usati recentemente allo scopo di minimizzare il numero di cicli durante i quali il processore è inutilizzato. Molti SO riservano una porzione della memoria centrale da utilizzare come cache.
Un buffer è un'area di memoria che serve a conservare temporaneamente dei dati durante i trasferimenti tra dispositivi o processi che lavorano a differenti velocità. I buffer migliorano le prestazioni del sistema permettendo al software ed ai dispositivi hardware di scambiarsi dati e comandi in modo asincrono (cioè indipendentemente gli uni dagli altri).
Lo spooling (simultaneous peripheral operations online) è una tecnica in cui un dispositivo intermedio, come un disco, è interposto tra un processo e un dispositivo di I/O molto lento o con buffer limitato. Per esempio, se un processo tenta di stampare un documento ma la stampante è occupata , quel processo, invece di attendere la disponibilità della stampante, scrive i suoi dati su disco, che verranno stampanti non appena la stampante sarà disponibile. Lo spooling permette quindi ai processi di richiedere operazioni da dispositivi periferici anche se in quel momento il dispositivo non è pronto a soddisfare la richiesta.

Concetti base di software

Un calcolatore può comprendere solo il linguaggio macchina, questo in quanto "linguaggio naturale" di particolare un calcolatore è definito dall'architettura hardware del calcolatore stesso (e da essa dipendono strettamente motivo per cui un particolare linguaggio macchina può essere utilizzato solo su un particolare calcolatore). I linguaggi macchina consistono essenzialmente in sequenze di numeri (0, 1) che indicano al calcolatore come svolgere le operazioni più elementari. Più la diffusione dei computer aumentava, più la programmazione in linguaggio macchina si dimostrava lenta e passibile di errori, i programmatori cominciarono quindi ad usare abbreviazioni di parole della lingua inglese per indicare le operazioni di base di un calcolatore. Queste

abbreviazioni formarono le basi del linguaggio assembly. Opportuni programmi di traduzione, chiamati assemblatori (assembler), si occupano di convertire i programmi in linguaggio assembly in programmi in linguaggio macchina.
Il linguaggio assembly risulta più chiaro all'essere umano ma il calcolatore non è in grado di comprenderlo fino a che non viene trasformato in linguaggio macchina da un programma assemblatore.

Il linguaggio assembly richiede parecchie istruzioni anche per svolgere il compito più semplice, per incrementare l'efficienza dei programmatori furono quindi sviluppati i linguaggi ad alto livello (high level languages). Questi permettono di descrivere le operazioni necessarie per lo svolgimento di un determinato compito utilizzando un numero ridotto di istruzioni, ma richiedono l'impiego di un programma di traduzione, chiamato compilatore, per convertire in linguaggio macchina i programmi i programmi scritti con un linguaggio di alto livello. I linguaggi ad alto livello permettono al programmatore di scrivere istruzioni simili all'inglese, contenenti classiche notazioni matematiche.
Mentre i compilatori trasformano il linguaggio ad alto livello in linguaggio macchina, gli interpreti sono programmi che eseguono direttamente il codice sorgente o codice trasformato in un linguaggio a basso livello, che però non è codice macchina. Java, ad esempio, viene prima trasformato da un a specie di compilatore in un formato chiamato bytecode (Java può anche essere trasformato direttamente in linguaggio macchina), che rappresenta una specie di codice macchina per una particolare macchina virtuale (JVM, Java Virtual Machine). Il byte code non è dipendente dalla particolare macchina fisica che lo esegue, cosa che ne aumenta la portabilità; ma a causa del tempo di esecuzione aggiuntivo necessario per la traduzione, un programma interpretato tende ad essere eseguito più lentamente di quanto sia possibile per un programma compilato.

Sebbene siano stati sviluppati centinaia di linguaggi ad alto livello solo un numero relativamente piccolo è stato accettato diffusamente. Ad oggi la tendenza dei linguaggi di programmazione è di essere strutturati o orientati agli oggetti.
Linguaggi di programmazione più diffusi:

- Fortran: sviluppato da IBM a metà degli anni '50 per creare applicazioni scientifiche o ingegneristiche richiedevano elaborazioni matematiche piuttosto complesse.
- COBOL: (Common Business Oriented Language) sviluppato alla fine degli anni '50 da un gruppo composto da produttori di calcolatori, agenzie governative e gruppi industriali. Progettato per applicazioni commerciali che devono manipolare grandi quantità di dati.
- C: sviluppato da Dennis Ritchie nei Bell Laboratories nei primi anni '70, è diventato famoso come il linguaggio usato per realizzare il sistema operativo UNIX.
- C++: estensione del C che permette la programmazione orientata agli oggetti (OOP, Object-Oriented Programming). Gli oggetti sono componenti software riusabili che modellano elementi del mondo reale. I programmi orientati agli oggetti sono spesso più facili da capire, da sottoporre a debug e modificare rispetto a quelli sviluppati con le tecniche precedenti. La maggior parte degli SO di oggi sono scritti in C o C++.
- Java: nel 1993 con l'esplosione della popolarità del WWW, Sun Microsystems intuì l'enorme potenzialità d'utilizzo di Java, il suo nuovo linguaggio orientato agli oggetti per creare applicazioni da scaricare dal Web e da eseguire all'interno dei Web browser. Java fu annunciato nel '95, guadagnando l'attenzione della comunità commerciale in conseguenza del grande interesse per il Web. Ad oggi è diventato un linguaggio molto utilizzato, è impiegato per generare dinamicamente pagine Web, realizzare applicazioni industriali su larga scala, migliorare le funzionalità offerte dai Web server, fornire applicazioni per dispositivi elettronici di consumo e molti altri scopi.
- C#: annunciato nel 2000 dalla Microsoft e progettato appositamente per essere il linguaggio della piattaforma .NET, questo linguaggi deriva dal C, dal C++ e da Java. C# è orientato agli

oggetti ed ha a disposizione una libreria molto ampia di componenti pronti all'uso in modo da permettere ai programmatori di sviluppare velocemente applicazioni.

Negli anni '60 si cominciò a capire che lo sviluppo del software era un'attività più complessa di quanto si potesse immaginare, ci si indirizzò dunque alla programmazione strutturata, un approccio disciplinato allo sviluppo di programmi che fossero chiari, corretti e facili da modificare.

- Pascal: le attività di ricerca portarono allo sviluppo di tale linguaggio, nel 1971 da parte di Nicklaus Wirth; linguaggio pensato per essere utilizzato nell'insegnamento della programmazione strutturata, divenne ben presto il linguaggio preferito per insegnare a programmare. Il linguaggio era però carente di alcune caratteristiche necessarie per lo sviluppo di applicazioni commerciali o industriali.
- Ada: linguaggio di programmazione realizzato con il patrocinio del dipartimento di difesa americano (DoD) durante gli anni '70 e primi anni '80 (il nome deriva dalla figlia del poeta Lord Byron, Ada Lovelace, considerata la prima donna ad aver programmato un calcolatore). Il linguaggio Ada è stato uno dei primi ad essere progettato per supportare la programmazione concorrente.

Una volta compresi i vantaggi della programmazione strutturata, cominciarono ad apparire nuove tecnologie software. La tecnologia ad oggetti è uno schema di impacchettamento per la creazione di unità software significative. Quasi ogni nome (inteso come sostantivo) è rappresentabile come un oggetto. Gli oggetti hanno determinate proprietà (chiamate anche attributi) e svolgono azioni (chiamate anche comportamenti o metodi). Le classi sono i tipi dei relativi oggetti, una classe specifica il formato generale dei suoi oggetti, e le proprietà e le azioni disponibili per un oggetto dipendono dalla classe a cui appartiene.
Grazie ai linguaggi orientati agli oggetti i programmatori possono lavorare in un modo che riflette più naturalmente la percezione che le persone hanno del mondo, il porta ad un notevole guadagno in produttività. La tecnologia ad oggetti permette di poter riutilizzare in molti progetti le classi opportunamente progettate, utilizzare librerie di classi può quindi ridurre enormemente lo sforzo richiesto per realizzare nuovi sistemi. Oltre alla riusabilità del codice un altro grande vantaggio della programmazione ad oggetti è la produzione di software più comprensibile, in quanto meglio organizzato e di più facile manutenzione, in effetti questo tipo di programmazione permette ai programmatori di concentrarsi sull'insieme.

Application Programmiin Interface (API)

Le applicazioni d oggi richiedono l'accesso a molte risorse gestite dal SO, il quale essendo il gestore di tale risorse, in genere non permetterà ai processi di acquisirle senza prima averle richieste esplicitamente. Le API mettono a disposizione un insieme di routine utilizzabili dal programmatore per richiedere servizi al SO, in molti dei SO moderni la comunicazione tra software e SO è svolta esclusivamente tramite API.
I processi eseguono le chiamate di funzione definite dalle API per accedere ai servizi forniti d un sottostrato del sistema. Queste chiamate di funzione possono invocare le chiamate di sistema (system calls) per richiedere servizi al SO. Le chiamare di sistema sono simili alle interruzioni per i dispositivi hardware: quando avviene una chiamata di sistema, il sistema si porta in modalità kernel e il SO si occupa di servire la chiamata di sistema, eseguendola.

Compilazione, collegamento (linking) e caricamento (loading)

Per essere eseguibile, un programma scritto in un linguaggio ad alto livello deve prima essere trasformato in linguaggio macchina, poi collegato con altri programmi in linguaggio macchina dai quali dipende ed infine caricato in memoria.

- Il primo passo nel processo di creazione di programmi eseguibili consiste nel compilare il linguaggio ad alto livello e trasformarlo in linguaggio macchina. Un compilatore accetta come dati di ingresso il codice sorgente, scritto con un linguaggio ad alto livello, e produce come dati di uscita il codice oggetto, contenente le istruzioni in linguaggio macchina da eseguire. Il processo di compilazione è divisibile in diverse fasi, ogni fase modifica il programma in modo che sia interpretabile dalla fase successiva, fino a quando il linguaggio non è tradotto in linguaggio macchina:
    - Per prima cosa il codice sorgente viene passato al lexer (analizzatore lessicale o scanner), che separa i caratteri del programma sorgente in token (= parole chiave, identificatori, operatori e punteggiatura).
    - Il lexer fornisce questo flusso di token al parser (analizzatore sintattico), che raggruppa i token in istruzioni sintatticamente corrette.
    - Il generatore di codice intermedio converte tale struttura sintattica in un flusso di semplici istruzioni che ricordano il linguaggio assembly, anche se queste non specificano i registri utilizzati per le operazioni.
    - L'ottimizzatore cerca di migliorare l'efficienza nell'esecuzione del codice e di ridurre i requisiti di memoria nel programma.
    - Nel'ultimo passaggio, il generatore di codice produce il file oggetto contenente le istruzioni in linguaggio macchina.
- I programmi sono spesso composti da alcuni sottoprogrammi sviluppati indipendentemente, chiamati moduli. Le funzioni per svolgere le tipiche routine di gestione dell'I/O e di generazione di numeri casuali vengono precompilate e raggruppate in moduli libreria. Il collegamento(linking) è il processo di integrazione dei vari moduli referenziati da un programma in una singola unità eseguibile. Quando un programma viene compilato, se questo fa riferimento a funzioni o dati di un altro modulo, il compilatore li trasforma nei cosiddetti "riferimenti esterni". Inoltre se il programma rende le proprie funzioni o i propri dati disponibili per altri programmi, ognuno di questi viene rappresentato con un nome esterno. I moduli oggetto memorizzano tali riferimenti e nomi esterni in una struttura dati chiamata tabella dei simboli. Il modulo integrato prodotto dal linker è chiamato modulo caricabile (load module). I dati in ingresso al linker sono i moduli oggetto, i moduli caricabili, le istruzioni di controllo e la locazione dei file di libreria utilizzati. Il linker ha spesso a che fare con diversi file oggetto che formano un singolo programma. Questi file oggetto, in genere, specificano le locazioni di dati e istruzioni utilizzando indirizzi all'inizio di ogni file, chiamati indirizzi relativi. Il linker deve modificare tali indirizzi in modo che non si riferiscano a dati o istruzioni non validi, una volta che i moduli siano combinati per formare un programma linkato. La rilocazione degli indirizzi assicura che ogni istruzione venga identificata unicamente da un indirizzo all'interno di un file. Una volta modificato l'indirizzo, tutti i riferimenti ad esso vanno aggiornati al nuovo valore. I linker svolgono anche la cosiddetta risoluzione dei simboli, operazione durante la quale i riferimenti esterni all'interno di un modulo sono convertiti nel corrispondente nome esterno in un altro modulo, una volta fatto questo per rispecchiare questa integrazione va modificato l'indirizzo del riferimento esterno. Il linking avviene spesso in due passaggi:
    - Il primo passaggio determina la dimensione di ogni modulo e costruisce la tabella dei simboli; la tabella dei simboli associa ad ogni simbolo un indirizzo, in modo tale che il linker possa localizzare il riferimento.
    - Nel secondo passaggio il linker assegna gli indirizzi alle differenti istruzioni ed ai dati e risolve i riferimenti ai simboli esterni. Il modulo caricabile, poiché può diventare l'input di un successivo passaggio di linking, contiene una tabella dei simboli in cui tutti i simboli sono nomi esterni.
    - Il momento della compilazione dipende dall'ambiente , avviene in fase di compilazione e il programmatore include tutto il codice necessario nel file sorgente, in modo che non

ci siano riferimenti a nomi esterni (caso molto raro in quanto molti programmatori utilizzano le librerie condivise); nel caso contrario avviene dopo la compilazione ma comunque prima del caricamento.
    - Il linking ed il caricamento sono spesso svolti dalla stessa applicazione, detta linking loader. Il linking può anche avvenire a runtime (durante l'esecuzione) con un processo chiamato linking dinamico; in questo caso, i riferimenti a funzioni esterne non sono risolti fino a che il processo non è caricato in memoria o effettua una chiamata a tale funzione. Il linking dinamico permette di risparmiare spazio nei dispositivi di memoria secondaria, in quanto è sufficiente mantenere una sola copia di una libreria condivisa (è memorizzabile separatamente dal restante codice del programma) indipendentemente dal numero di programmi che la utilizzano.
- Una volta che il linker ha creato il modulo caricabile, lo passa ad un programma chiamato loader, il quale è responsabile di posizionare le istruzioni e i dati in una particolare zona di memoria, un'operazione chiamata binding degli indirizzi. Se il modulo caricabile specifica già gli indirizzi fisici in memoria, il loader deve solo posizionare istruzioni e dati negli indirizzi specificati dal programmatore o dal compilatore (supponendo che tali indirizzi siano disponibili), tecnica chiamata caricamento assoluto. Il caricamento rilocabile è utilizzato quando il modulo caricabile contiene indirizzi relativi che devono essere convertiti in reali indirizzi di memoria. Il loader deve quindi richiedere un blocco di memoria in cui posizionare il programma e trasferire gli indirizzi del programma per farli corrispondere alle reali locazioni di memoria. Il caricamento dinamico è una tecnica che permette di caricare moduli di programma al momento dell'utilizzo, in molti sistemi con memoria virtuale, ad ogni processo è assegnato il proprio insieme di indirizzi virtuali che parte da 0, il loader è quindi responsabile del caricamento del programma in una valida regione di memoria.

Firmware e Middleware

Oltre al software e all'hardware molti calcolatori contengono il cosiddetto firmware, che consiste di istruzioni eseguibili conservate su un supporto di memorizzazione non volatile, spesso a sola lettura, collegato ad un dispositivo. Il firmware è programmato con la microprogrammazione, che è uno strato di programmazione al di sotto del linguaggio macchina. Il microcodice (istruzioni del microprogramma) include semplici, ma fondamentali, istruzioni necessarie per realizzare tutte le operazioni in linguaggio macchina. I set di istruzioni implementati in microcodice sono andati via via scomparendo, in quanto l'esecuzione delle istruzioni in microcodice limita la massima velocità raggiungibile dal processore, così le operazioni precedentemente svolte dal microcodice sono ora svolte dall'hardware del processore.

La natura dei sistemi distribuiti richiede il middleware per permettere le interazioni tra diversi processi in esecuzione su uno o più computer collegati in rete. Il middleware permette ad un'applicazione in esecuzione su un calcolatore di comunicare con un'altra applicazione in esecuzione su un calcolatore remoto, consentendo di fatto la comunicazione tra computer in un sistema distribuito. Il middleware permette inoltre alle applicazioni di essere eseguite su piattaforme eterogenee, a patto che ognuna di esse abbia il middleware installato. Il middleware semplifica lo sviluppo delle applicazioni, in quanto gli sviluppatori non devono conoscere i dettagli di come il middleware svolge i suoi compiti e possono concentrarsi sullo sviluppo dei programmi piuttosto che sui protocolli di comunicazione.

Processi -capitolo 3-

Il termine processo è stato usato per la prima volta negli anni '60, tra le definizioni più note ricordiamo: un programma in esecuzione, un'attività asincrona, lo "spirito animato" di una procedura, il luogo deputato al controllo delle procedure. Comunque lo si definisca , esso comporta

l’esistenza di una struttura dati nel SO chiamata descrittore di processo o blocco di controllo del processo, a cui viene assegnato il processore da parte di un’unità di avvio detto dispatcher.
Un processo è un’entità, ogni processo ha il suo spazio di indirizzamento, che di solito consiste in una sezione testo, una sezione dati e uno sezione dello stack (pila). Nella sezione testo è memorizzato il codice eseguito dal processore; nella sezione dati, invece, sono memorizzate le variabili ed i dati allocati dinamicamente che il processo usa durante l’esecuzione. Nella sezione dello stack sono memorizzate le istruzioni e le variabili locali per le chiamate a procedura attive. Il contenuto dello stack aumenta al crescere del numero di chiamate a procedura annidate, e diminuisce mano a mano che le procedure rirtornano.
Un processo è un programma in esecuzione, un programma è un’entità inanimata, solo quando il processore “alita la vita” nel programma, diventa quell’entità che chiamiamo processo.

Stati di un processo: il ciclo di vita

Il SO deve assicurare che ogni processo riceva una sufficiente quantità di tempo di processore. In qualunque sistema possono esserci tanti processi in reale concorrenza quanti sono i processori. Normalmente in un sistema esistono molti più processi che processori, di conseguenza in un dato istante di tempo, possono essere eseguiti solo alcuni processi e non altri.
Durante il suo ciclo di vita un processo si muove attraverso attraverso una serie di stati del processo discreti. Diversi eventi possono causare il cambiamento di stato di un processo.

- Un processo è nello stato running (in esecuzione) se è eseguito da un processore.
- È nello stato ready (pronto o attivo) se può essere eseguito da un processore quando ve ne sia uno disponibile.
- È nello stato blocked (bloccato o in attesa), se è in attesa, prima di poter procedere, di un certo evento (come ad esempio il completamento di un’operazione di I/O)

Se consideriamo un sistema monoprocessore, un solo processo può essere nello stato di running ed molti altri possono trovarsi nello stato di ready o blocked. Il SO mantiene una ready list (lista dei processi ready) e una blocked list (lista dei processi in stato blocked). La ready list è ordinata secondo la priorità dei processi, cosicché il processo che otterrà un processore sarà il primo della lista (ossia il processo con più alta priorità). La blocked list di solito non è ordinata in base alla priorità ma in base all’ordine seguito dagli eventi attesti in quanto i processi in stato blocked non si sbloccano tornando quindi nello stato ready.

Gestione dei processi

Dato che il sistema operativo alterna l’esecuzione dei suoi processi, deve gestirli con attenzione per assicurarsi che non si verifichino errori quando vengono interrotti e poi riavviati. I processi devono poter comunicare con il SO per svolgere semplici operazioni come l’avvio di un nuovo processo o segnalare la fine della sua esecuzione.

- *Stati del processo e transizioni di stato:* quando un utente esegue un programma, alcuni processi vengono creati e messi nella ready list. Un processo si muove verso la testa della lista a mano a mano che altri processi terminano il loro turno di utilizzo del processore. Quando un processo raggiunge la testa della lista e un processore diventa disponibile, gliene viene assegnato uno: avviene cosi una transizione di stato dallo stato ready allo stato running. L’attribuzione di un processore al primo processo della ready list è detto dispatching, è svolta da un’entità del sistema chiamata dispatcher. I processi negli stati running o ready sono detti “svegli” (awake), in quanto si contendono attivamente il tempo del processore. Il SO gestisce le transizioni di stato per servire al meglio i processi nel sistema. Per impedire a un qualunque processo di monopolizzare il sistema, accidentalmente o intenzionalmente, il SO imposta un clock per le interruzioni o temporizzatore ad intervallo per permette ad un processo di essere eseguito per uno specifico intervallo di tempo o

quanto. Se il processo non rilascia volontariamente il processore prima della scadenza del quanto, il clock per le interruzioni genera un'interruzione (interrupt) che permette al SO di prendere il controllo del processore. Il SO può quindi cambiare in ready lo stato del processo precedentemente running, e selezionare il primo processo della ready list, cambiando il suo stato da ready a running. Se un processo running avvia un'operazione di I/O prima che il suo quanto di tempo sia esaurito, deve di conseguenza attendere il completamento di tale operazione prima di poter nuovamente utilizzare un processore; quindi il processo running abbandona volontariamente il processore (in pratica blocca se stesso in attesa del complemento dell'operazione di I/O). I processi nello stato blocked sono detti "addormentati" (asleep), perché non eseguibili anche se un processore diventa disponibile, quando l'operazione di I/O è completata o si verifica l'evento atteso, il SO operativo fa passare il processo in stato blocked allo stato ready.
Si noti che l'unica transizione di stato attivata direttamente dal processo utente è quella che lo porta in stato blocked, le altre tre sono attivare dal SO.
Alcuni dei primi SO funzionanti con processori che non disponevano di clock per le interruzioni, utilizzavano il multitasking cooperativo: ogni processo doveva cioè rilasciare volontariamente il processore sul quale era in esecuzione prima dell'esecuzione di un altro processo. Tuttavia il multitasking cooperativo è raramente utilizzato nei sistemi di oggi perché permette ai processi di monopolizzare, accidentalmente o intenzionalmente,un processore.

- *Blocchi di controllo del processo o descrittori di processo:* quando crea un processo il SO svolge normalmente diverse operazioni. Prima di tutto deve essere in grado di verificarlo, e a tal fine al processo creato viene assegnato un numero, detto identificatore di processo (Process Identification Number, PID). In seguito, il SO crea un blocco di controllo di processo (Process Control Block, PCB), chiamato anche descrittore di processo, che contiene le informazioni necessarie al SO per gestire il processo. Il PCB include generalmente le seguenti informazioni:
    - PID
    - Stato del processo (running, ready, blocked)
    - Contatore di programma (program counter) che indica qual'è l'istruzione successiva da eseguire
    - Priorità di scheduling
    - Diritti (un insieme di dati che indica a quali risorse un processo può accedere)
    - Puntatore al processo padre (che ha creato il processo considerato)
    - Puntatore ai processi figli (eventuali processi creati dal processo)
    - Puntatori per localizzare in memoria i dati e le istruzioni del processo
    - Puntatori per localizzare le risorse

  Il PCB memorizza anche il contenuto dei registri, chiamato contesto di esecuzione del processore sul quale il processo era in esecuzione l'ultima volta, prima che una transizione cambiasse il suo stato running. Il contesto di esecuzione di un processo è dipendente dall'architettura ma di solito include il contenuto dei registri di uso generale (general-purpose registers), che contengono dati ai quali il processore può accedere direttamente, oltre a registri di gestione del processo, come i registri che memorizzano i puntatori allo spazio di indirizzamento del processo. Questo permette al SO di ripristinare il contesto di un processo quando questo torna allo stato running.
  Quando un processore cambia il SO stato deve aggiornare le informazioni contenute nel PCB di quel processo, solitamente il SO memorizza un puntatore per ogni PCB di ciascun processo in una tabella dei processi, generale o divisa per utente in modo da poter accedere velocemente ai PCB. Quando un processo termina, volontariamente o per intervento del SO,

viene rimosso dalla tabella dei processi e vengono liberate la memoria e le altre risorse ad esso associate, rendendole disponibili per altri processi.

- *Operazioni sui processi:* I SO devono essere in grado di svolgere alcune operazioni sui processi tra cui:
  - creare un processo
  - terminare o distruggere un processo
  - sospendere un processo
  - ripristinare o recuperare un processo
  - cambiare la priorità di un processo
  - bloccare un processo
  - risvegliare un processo
  - avviare in esecuzione un processo
  - permettere ai processi di comunicare tra loro (interprocess communication)

  Un processo può generare un nuovo processo. In tal caso il processo creante è chiamato processo padre, mentre il processo creato è chiamato processo figlio. Ogni processo figlio è creato da un solo padre e tale operazione induce una struttura gerarchica dei processi in cui ogni figlio ha un solo padre ma ogni padre può avere molti figli.
  Nei SO UNIX, come Linux, molti processi sono generati a partire da un processo padre comune chiamato init, creato quando viene caricato il kernel. Tra questi processi troviamo kswapd (gestione di memoria), xfs (file system), khubd (dispositivi) e login (che si occupa di autenticare gli utenti che accedono al SO).
  Una volta che il processo login ha autenticato l'utente, genera un processo shell (ad esempio una bash → bourne-again shell) che permette all'utente di interagire con il SO.
  Distruggere un processo significa cancellarlo dal sistema. La memoria e le altre risorse utilizzate sono restituite al sistema e il processo è cancellato da ogni lista o tabella di sistema, il suo PCB rimosso e lo spazio di memoria che occupava il suo PCB messo a disposizione di altri processi. La distruzione di un processo è più complicato quando questo ha creato dei processi figli, in alcuni sistemi ogni processo figlio è automaticamente distrutto quando viene distrutto il processo padre, in altri la distruzione del processo padre non ha effetti sui processi figli, i quali continuano indipendentemente la loro strada.
  Cambiare la priorità di un processo normalmente implica modificare il valore del campo priorità nel PCB. In funzione di come il SO implementa lo scheduling dei processi, può essere necessario memorizzare un puntatore al PCB in una diversa coda di priorità.

- *Sospensione e ripristino:* un processo sospeso è indefinitamente rimosso dalla contesa per il tempo di processore, ma non distrutto. Nella maggior parte degli attuali elaboratori l'esecuzione è troppo veloce per permettere un intervento manuale. Ma un amministratore o un utente scettico sui risultati parziali di un processo, possono comunque decidere di sospenderlo, invece di terminarlo (abort), fino a quando non si siano accertati del suo corretto funzionamento. Questo meccanismo è utile per intercettare minacce alla sicurezza, come l'esecuzione di codice che attacca le protezioni, o per scopi di analisi e rimozione degli errori (debug). Aggiungiamo quindi due nuove stati suspended-ready e suspended-blocked, una sospensione può essere attivata dal processo stesso che verrà sospeso o da un altro processo. In un sistema multiprocessore un processo running può essere sospeso da altri processi running e un processo running può anche sospendere un processo ready blocked. È chiaro che un processo sospende se stesso solo quando è nello stato running, in questa situazione il processo effettua la transizione di stato da running a suspended-ready.
  Un processo nello stato suspended-ready può essere riportato nello stato ready, o ripristinato da un altro processo, che causerà la transizione del primo processo da suspended-ready a ready.
  Un processo blocked effettuerà la transizione dallo stato blocked a suspended blocked

quando verrà sospeso da un altro processo, e anch'esso può essere risvegliato da un altro processo e può effettuare la transizione da suspended-blocked a blocked.
La sospensione viene effettuata immediatamente, poiché è in genere un'attività ad alta priorità. Al termine effettivo dell'operazione di I/O o dell'evento, il processo suspended-blocked effettua la transizione dello stato da suspended-blocked a suspended-ready.

- *Cambio di contesto:* il SO esegue un cambio di contesto (context switching) per arrestare l'esecuzione di un processo running ed iniziare l'esecuzione di un processo ready. Per eseguire un cambio di contesto, il kernel deve prima memorizzare nel relativo PCB il contesto di esecuzione del processo running, poi caricare dal PCB corrispondente il contesto di esecuzione precedente relativo al processo ready da eseguire.
  I cambi di contesto devono necessariamente essere trasparenti ai processi, i processi cioè ignorano di essere stati rimossi dall'esecuzione. Durante un cambio di contesto un processore non può eseguire operazioni "utili"; effettua cioè quelle essenziali per il SO, ma non esegue istruzioni per conto di un dato processo. Il cambio di contesto è un puro costo addizione e avviene così frequentemente che i SO devono minimizzare il tempo impiegato per questa operazione. Il SO accede spesso ai PCB, di conseguenza, molti processori contengono un registro hardware che fa riferimento al PCB del processo attualmente in esecuzione, in modo da facilitare il cambio di contesto. Quando il SO dà avvio a questa operazione, il processore memorizza nel relativo PCB il contesto di esecuzione del processo corrente, evitando così la sovrascrittura da parte del SO (o di altri processi) dei valori contenuti nei registri durante l'esecuzione del processo precedente.

Interruzioni

Le interruzioni permettono al software di rispondere ai segnali proveniente dall'hardware. Il SO può specificare un insieme di istruzioni, denominato gestore delle interruzioni (interrupt handler), da eseguire in risposta ad ogni tipo di interruzione, consentendo al SO di prendere il controllo del processore per la gestione delle risorse di sistema.
Un processore può generare un'interruzione come risultato dell'esecuzione di istruzioni di un processo; in questo caso è spesso denominata trap (cattura) ed è sincrona con le operazioni del processo. Le interruzioni sincrone si verificano, per esempio, quando un processo tenta di eseguire un'azione illegale, come dividere per zero o accedere ad una zona protetta di memoria.
Le interruzioni possono anche essere causate da eventi non relativi all'istruzione corrente di un processo, nel qual caso sono dette "asincrone" rispetto all'esecuzione del processo. I dispositivi hardware utilizzano le interruzioni asincrone per comunicare al processore un cambiamento di stato.
Le interruzioni forniscono un mezzo a basso costo per richiamare l'attenzione di un processore.
Un'alternativa alle interruzioni è quella di richiedere ripetutamente il proprio stato ad ogni dispositivo, questo approccio chiamato polling (interrogazione circolare), incrementa il costo della soluzione a mano a mano che aumenta la complessità del sistema. Le interruzioni eliminano la necessità da parte del processore di interrogare ripetutamente i dispositivi.
I sistemi orientati alle interruzioni possono diventare sovraccarichi: se le interruzioni giungono troppo velocemente, il sistema può non essere in grado di servirle tutte.
Nei sistemi collegati in rete, l'interfaccia di rete contiene una piccola memoria in cui viene memorizzato ciascun pacchetto di dati ricevuto dagli altri calcolatori. Ogni volta che un'interfaccia di rete riceve un pacchetto genera un'interruzione per informare il processore che ci sono dei dati da elaborare. Se il processore non può elaborare i dati provenienti dall'interfaccia di rete prima del riempimento della relativa memoria, i pacchetti potrebbero andare perduti. I sistemi normalmente realizzano code per tener traccia delle interruzioni da elaborare quando il processore diventa disponibile. Queste code, naturalmente, consumano memoria, che è limitata in termini di dimensione. In condizioni di carico consistente, il sistema potrebbe non essere in grado accodare tutte le richieste di interruzione in arrivo, finendo col perderne qualcuna.

Consideriamo come gli elaboratori gestiscono di solito le interruzioni hardware. Si noti comunque che esistono anche altri schemi di interruzione.

- La linea di interruzione, una connessione elettrica tra la scheda madre ed un processore, diventa attiva. Dispositivi quali timer, periferiche e controllori, inviano segnali che attivano la linea di interruzione per informare il processore di un evento. La maggior parte dei processori contiene un controllore delle interruzioni che le ordina in base alla loro priorità allo scopo di servire prima quelle più importanti. Le altre vengono accodate fino a che non siano state servite tutte le interruzioni a priorità più elevata.
- Il processore, dopo che la linea di interruzione è diventata attiva, completa l'esecuzione dell'istruzione corrente, dopodiché arresta l'esecuzione del processo corrente. Per far ciò, il processore deve memorizzare informazioni sufficienti affinché il processo venga risvegliato correttamente e con le giuste informazioni di registro.
- Il processore passa successivamente il controllo all'opportuno gestore dell'interruzione. Ad ogni tipo di interruzione è assegnato un valore univoco che il processore usa come indice all'interno del vettore delle interruzioni, un vettore di puntatori ai gestori delle interruzioni che risiede in una zona di memoria inaccessibile ai processi, allo scopo di prevenirne un'erronea modifica.
- Il gestore delle interruzioni esegue le azioni appropriate basandosi sul tipo di interruzione.
- Dopo la conclusione dell'esecuzione del gestore delle interruzioni, viene ripristinato lo stato del processo interrotto, o di qualche altro processo se il kernel intraprende un cambio di contesto.
- Il processo interrotto, o l'eventuale sostituto, torna in esecuzione. È compito del SO determinare quale processo, tra quello interrotto e quelli in stato ready, vada eseguito.

L'insieme delle interruzioni supportate dal calcolatore dipende dall'architettura del sistema. La specifica IA-32 distingue tra due tipi di segnali che il processore può ricevere: interruzioni ed eccezioni. Le interruzion (interrupts) comunicano al processore che è accaduto un certo evento o che è cambiato lo stato di un dispositivo interno. L'architettura IA-32 prevede anche delle "interruzioni generate dal software", utilizzabili dai processi per effettuare chiamate di sistema. Le eccezioni indicano che si è verificato un errore inerente all'hardware o come risultato di un'istruzione software, oppure vengono utilizzate anche per bloccare temporaneamente un processo quando raggiunge un punto di arresto nel codice.
I dispositivi che generano interruzioni, di solito sotto forma di segnali di I/O e interruzioni del temporizzatore, sono esterni al processore. Queste interruzioni sono asincrone rispetto al processo in esecuzione, perché avvengono indipendentemente dall'istruzione eseguita dal processore. Le interruzioni generate dal software, quali per esempio, le chiamate di sistema, sono sincrone rispetto al processo in esecuzione, perché generate in risposta a un'istruzione.

Tipi di interruzione	**Descrizione**
I/O	Sono avviate da hardware di input/output; notificano al processore il cambiamento di stato di un canale o di un dispositivo. Le interruzioni di I/O sono generate, per esempio, quando è completata un'operazione di I/O.
Temporizzatore	Un sistema può contenere dispositivi che generano periodicamente interruzioni. Queste interruzioni sono utilizzabili per operazioni quali conteggio del tempo trascorso o analisi delle prestazioni. I temporizzatori permettono al SO di determinare anche se è scaduto il quanto di tempo di un processo.
Interruzioni Interprocessore	Permettono a un processore di spedire un messaggio ad un altro processore in un sistema multiprocessore.

La specifica IA-32 classifica le eccezioni come fault, trap o abort. I fault e i trap sono eccezioni a cui un gestore di eccezioni può rispondere per permettere ai processi di continuare la propria esecuzione. Un fault (difetto) indica un errore correggibile da un gestore di eccezioni.

Classe di eccezione	**Descrizione**
Fault	Sono causate da un'ampia gamma di problemi che possono verificarsi durante l'esecuzione delle istruzioni in linguaggio macchina che compongono il programma. Tra i più comuni, la divisione per zero, i dati su cui si sta operando in formato errato, i tentativi di eseguire un codice operativo non valido, di accedere ad una zona di memoria oltre i limiti della memoria reale, di eseguire un'istruzione privilegiata da parte di un processo utente e di accedere ad una risorsa protetta.
Trap	Sono generate da eccezioni queli overflow, per esempio quando il valore memorizzato in registro supera la capacità dello stesso, e quando si incontra un punto di arresto nel codice. Dopo aver eseguito il gestore dei trap, il processore riprende l'esecuzione del processo da dall'istruzione successiva a quella che aveva causato l'eccezione.
Abort	Avvengono quando il processore rileva un errore irrimediabile. Per esempio, quando una procedura di gestione delle eccezioni causa a sua volta un'eccezione, il processore potrebbe non essere in grado di gestire sequenzialmente entrambi gli errori. Si tratta di un'eccezione con doppio fallimento (double-fault exception), che causa la fine del processo che l'ha generata. Nel caso di un abort, il processore non può memorizzare in maniera attendibile il contesto di esecuzione del processo. Normalmente, come risultato, il SO termina prematuramente il processo che ha causato la terminazione.

La maggior parte delle architetture e dei SO assegnano una priorità alle interruzioni, perché alcune richiedono azioni più immediate di altre. Le priorità delle interruzioni sono realizzabili simultaneamente sia nell'hardware che nel software: un processore potrebbe bloccare o accodare le interruzioni con una priorità più bassa rispetto a quella che sta correntemente gestendo. Il kernel può essere così sovraccarico di interruzioni da non riuscire a rispondere. Una rapida risposta alle interruzioni ed una veloce restituzione del controllo ai processi interrotti è essenziale per massimizzare l'utilizzo delle risorse e raggiungere un alto grado di interattività. La maggior parte dei processori, perciò, permette al kernel di disabilitare o mascherare un determinato di interruzione. Il processore può a questo punto ignorare le interruzioni di quel tipo o memorizzarle in una cosa di interruzioni pendenti che verranno servite nel momento in cui quel tipo di interruzione verrà di nuovo riabilitato.

Comunicazione tra processi

In ambienti multiprogrammati e di rete è comune che i processi comunichino fra loro (IPC, InterProcess Comunication). La comunicazione tra processi è anche essenziale per quei processi che devono coordinare attività per raggiungere uno scopo comune.

I segnali sono interruzioni software che notificano il verificarsi di un evento ad un processo, i segnali non permettono ai processi di specificare dati da scambiare con altri processi. Un segnale di sistema dipende dal SO e dalle interruzioni generate dal software supportate da un particolare processore. Quando si verifica un segnale il SO determina quale processo debba riceverlo e come dovrebbe rispondere al segnale. I processi possono catturare, ignorare o mascherare un segnale. Un processo cattura un segnale specificando una procedura richiamata dal SO quando viene spedito il

segnale. Un processo può anche ignorare il segnale: in questo caso il processo si basa sulla gestione standard del segnale del SO, che generalmente corrisponde ad un abort, ovvero alla fine immediata del processo. Un'altra comune modalità di gestione standard è la copia della memoria (memory dump), che è simile alla terminazione. Una copia della memoria causa la fine di un processo ma, prima che ciò avvenga, il processo genera un file (core file) che contiene il contesto di esecuzione del processo e i dati nel suo spazio di indirizzi, informazioni utili per il debug. Altre possibili azioni standard sono quelle di ignorare semplicemente il segnale, di sospendere e, successivamente, risvegliare il processo.
Un processo può anche bloccare un segnale mascherandolo, il SO non invia segnali di quel tipo fino a quando il processo non rimuove la maschera del segnale. I processi in genere bloccano un segnale quando ne stanno gestendo un altro dello stesso tipo. Analogamente alle istruzioni mascherate anche i segnali mascherati possono andare persi a seconda dell'applicazione del SO.

Nella comunicazione tra processi basata sui messaggi, i messaggi possono essere trasmessi seguendo un'unica direzione alla volta, che significa che per un dato messaggio un processo è il mittente e l'altro il destinatario, oppure in senso bidirezionale, che implica che ogni processo agisca sia come mittente che come destinatario.
Normalmente le chiamate send e receive sono implementate come chiamate di sistema accessibili da molti linguaggi di programmazione. Una chiamata send bloccante deve attendere che il destinatario riceva il messaggio del mittente e che ne confermi il ricevimento (questa notifica è detta messaggio di conferma o acknowledgment). Una chiamata send non bloccante permette al mittente di continuare la sua esecuzione anche se il destinatario non ha ancora ricevuto (e confermato) il messaggio.
Tale situazione richiede un meccanismo di memorizzazione temporanea dei messaggi (buffering) per mantenere il messaggio fino al suo ricevimento da parte del destinatario. La send bloccante è un esempio di comunicazione sincrona, mentre una send non bloccante è un esempio di comunicazione asincrona. La chiamata send può indicare esplicitamente il destinatario oppure può ometterlo, indicando che il messaggio è inviato contemporaneamente a tutti i processi (broadcast), o a un "gruppo di lavoro" con il quale il mittente comunica generalmente. La comunicazione asincrona con le send non bloccanti aumenta le prestazioni riducendo il tempo di attesa del processo.
Se non è stato inviato alcun messaggio, una chiamata receive bloccante forza il destinatario a porsi in attesa; una chiamata receive non bloccante permette al destinatario di continuare la sua esecuzione fino al prossimo tentativo di ricezione. Una chiamata receive può specificare il ricevimento di un messaggio da un mittente in particolare o da qualunque mittente, ovvero qualsiasi membro del gruppo dei mittenti.
Un'implementazione diffusa dello scambio di messaggi è la pipe, regione di memoria protetta dal SO che opera come buffer (area tampone), consentendo lo scambio di dati a due o più processi. Il SO sincronizza l'accesso al buffer: dopo che un processo scrittore ha completato la scrittura del buffer (possibilmente riempiendolo), il SO interrompe l'esecuzione dello scrittore e consente ad un processo lettore di leggere i dati del buffer. Quando un processo legge dei dati, questi vengono rimossi dalla pipe. Quando un processo lettore completa la lettura dei dati dal buffer (possibilmente svuotandolo), il SO interrompe l'esecuzione del lettore e permette al processo di scrivere i dati nel buffer.
Nei sistemi distribuiti la trasmissione può causare errori o anche perdere dati. Per questo motivo spesso il mittente e il destinatario collaborano usando un protocollo con conferma per verificare la correttezza di ciascuna trasmissione. Il mittente può utilizzare un meccanismo di timeout (tempo limite) nell'attesa del messaggio di conferma del destinatario; a tempo scaduto, il mittente può ritrasmettere il messaggio se non è giunto il messaggio di conferma. Il sistema dello scambio di messaggi con proprietà di ritrasmissione identifica ogni nuovo messaggio per mezzo di una sequenza numerica. Il destinatario può esaminare questi numeri per assicurarsi di aver ricevuto tutti i messaggi e per riordinare i messaggi che non sono in ordine. Se il messaggio di conferma viene perso e il mittente decide di ritrasmettere, ne invia uno identico al precedente assegnando al

messaggio lo stesso numero sequenziale. Il destinatario, identificando i diversi messaggi che portano lo stesso numero sequenziale, ne conserva uno solo, ignorando le repliche.
Una complicazione dei sistemi distribuiti relativa allo scambio di messaggi è l'attribuzione ai processi di nomi senza ambiguità. La creazione e distruzione dei processi può essere coordinata attraverso un meccanismo centralizzato di nomina; ciò però comporta una appesantimento (overhead) considerevole nella trasmissione quando una macchina richiede il permesso di usare un nuovo nome. Un approccio alternativo consiste nell'assicurare che ogni computer fornisca nomi univoci ai propri processi; in questo modo i processi sono associabili ad un identificatore che combina il nome del computer con il nome dei processi, ma anche questo meccanismo richiederebbe un controllo centralizzato che appesantirebbe il sistema distribuito.
In pratica i sistemi distribuiti si scambiano messaggi usando i numeri di port sulle quali i processi rimangono in ascolto, evitando il problema dei nomi. La comunicazione basata sui messaggi nei sistemi distribuzione presenta seri problemi di sicurezza, tra cui quello di autenticazione.
I processi possono comunicare anche attraverso memoria condivisa, socket, chiamate di procedura remota e semafori e monitor (le ultime due per sincronizzare attività).

I processi UNIX

Durante l'esecuzione, ogni processo deve avere in memoria il codice, i dati e lo stack. In un sistema reale di memoria i processi dovrebbero individuare queste informazioni riferendosi ad un intervallo di indirizzi fisici. L'intervallo di indirizzi di memoria principale validi per ogni processo, è determinato dalla dimensione della memoria principale e dalla memoria utilizzata dagli altri processi. Poiché UNIX implementa la memoria virtuale, tutti i processi UNIX sono dotati di un insieme di indirizzi di memoria, detto spazio di indirizzamento virtuale, in cui il processo può memorizzare informazioni. Lo spazio di indirizzamento virtuale contiene una sezione testo, una sezione dati e uno sezione dello stack.
Il kernel mantiene un PCB del processo in una regione protetta di memoria inaccessibile per i processi utente. Nei sistemi UNIX un PCB memorizza informazioni tra cui il contenuto dei registri de processore, l'identificativo del processo (PID), il contatore di programma e lo stack delle chiamate di sistema. I PCB di tutti i processi sono elencati nella tabella dei processi, che permette al SO l'accesso alle informazioni (per esempio, la priorità) relative ad ogni processo.
I processi UNIX interagiscono con il sistema operativo attraverso le chiamate di sistema, un processo può effettuare la creazione (spawn) di un processo figlio attraverso la chiamata di sitema fork, che crea una copia del processo padre. Il processo figlio riceve un copia dei dati del processo padre e dei segmenti dello stack, insieme a tutte le altre risorse. Il segmento testo, che contiene le istruzioni in sola lettura del padre, è condiviso con il figlio, immediatamente dopo la fork i processi padre e figlio dispongono dei medesimi dati e istruzioni. Questo significa che i due processi devono eseguire esattamente le stesse azioni, a me che il padre o il figlio determinino ciascuno la propria identità. La chiamata di sistema fork restituisce valori diversi: il processo padre riceve il PID del figlio e il processo figlio riceve un valore zero. Questa convenzione permette al processo figlio di riconoscere che è stato appena creato. I programmatori di applicazioni possono usare questa convenzione per specificare nuove istruzioni da far eseguire al processo figlio.
Un processo può richiamare exec per caricare da un file un nuovo programma; spesso exec viene eseguita da un processo figlio immediatamente dopo l'operazione di creazione. Quando un padre crea un processo figlio, il padre può invocare la chiamata di sistema wait, che lo blocca fino a quando il processo figlio specificato non termina. Dopo che un kernel ha completato il suo lavoro, invoca la chiamata di sistema exit, che comunica al kernel la fine del processo; il kernel risponde liberando tutta la memoria del processo e tutte le altre risorse. Quando termina un processo padre, i suoi processi figli vengono di solito riposizionati nella gerarchia dei processi, diventando figli del processo init. Se il processo padre termina a causa di un segnale di kill da parte di un altro processo, tale segnale viene inviato anche ai figli.

La priorità dei processi UNIX è rappresentata da un numero che va da -20 a 19 (estremi inclusi) che il sistema usa per determinare quale sia il successivo processo da eseguire. Un valore numericamente basso di priorità indica un'elevata priorità in fase di scheduling. I processi che appartengono al SO, detti processi kernel, assumono spesso valori negativi di priorità e generalmente hanno una priorità di scheduling maggiore rispetto ai processi utente. I processi si SO che eseguono periodicamente operazioni di manutenzione, detti "demoni", vengono generalmente eseguiti con la priorità più bassa possibile.

Chiamate di sistema	**Descrizione**
fork	Crea un processo figlio e assegna a quel processo una copia delle risorse del padre.
exec	Carica da un file le istruzioni e i dati del processo nello spazio di indirizzamento.
wait	Blocca il processo che l'ha invocata fino a quando il processo figlio non abbia terminato la sua esecuzione.
signal	Consente ad un processo di specificare un gestore dei segnali per un particolare tipo di segnale.
exit	Termina il processo che l'ha invocata.
nice	Modifica la priorità di scheduling del processo.

Thread -capitolo 4-

Sebbene i primi SO permettessero ai calcolatori di eseguire concorrentemente programmi diversi, i linguaggi di programmazione allora disponibili non consentivano ancora di specificare attività concorrenti. Questi linguaggi disponevano generalmente solo di un semplice insieme di strutture di controllo che permetteva ai programmatori di specificare un flusso di controllo singolo.
Il programmatore può specificare che un'applicazione contiene più "flussi di controllo" o thread, ognuno dei quali individua una porzione del programma eseguibile in contemporanea con altri thread, questa tecnologia è chiamata multithreading.
Il supporto del SO è fondamentale per i linguaggi che forniscono la semantica per il multithreading, scrivere programmi multithread, cioè con la tecnica del multithreading, può essere impegnativo.

Un thread chiamato a volte anche processo leggero (lightweight process, LWP), condivide molte delle caratteristiche di un processo. I thread sono avviati all'esecuzione su un processore e ogni thread può eseguire un insieme di istruzioni indipendente da altri processi e thread. Tuttavia i thread non sono pensati per esistere da soli perché solitamente appartengono a processi tradizionali, chiamati a volte anche processi pesanti (heavyweight process, HWP). Al fine di migliorare l'efficienza nello svolgimento dei loro compiti, i thread di un processo condividono molte risorse con il processo stesso, in particolare lo spazio di indirizzamento e i file aperti. Il termine "thread" si riferisce ad un singolo flusso di istruzioni o flusso di controllo; i thread di un processo sono eseguibili concorrentemente per contribuire così al raggiungimento di un obbiettivo comune. Su un sistema multi-processore più thread potranno così essere eseguiti parallelamente. I thread possiedono quindi un sottoinsieme delle risorse di un processo, risorse come i registri del processore, lo stack e altri dati specifici del thread (Thread-Specific Data, TSD), per esempio le maschere dei segnali (ossia dati che descrivono quali segnali non debba riceve un certo thread), sono locali per ogni thread, mentre lo spazio di indirizzamento appartiene al processo che contiene i thread, ed è globale per tutti loro. I thread sono gestibili dal SO o dall'applicazione che li crea in base alla realizzazione dei thread adottata da una particolare piattaforma.
Sebbene molti S supportino i thread, le realizzazioni variano notevolmente.

Perché usare i thread

Alcuni fattori motivanti per il multithreading sono:

- *Progettazione software:* molte applicazioni moderne contengono segmenti di codice eseguibili indipendentemente dal resto dell'applicazione. Separare i segmenti di codice indipendenti in singoli thread migliora le prestazioni e semplifica l'attività di programmazione inerente a quelle attività intrinsecamente parallele.
- *Prestazioni:* in un'applicazione multithread, i thread possono condividere uno o più processori, così da poter svolgere più attività in parallelo, tale tipo di esecuzione riduce significativamente il tempo necessario per eseguire un'applicazione multithread.
- *Cooperazione:* Per comunicare e sincronizzare le attività, molte applicazioni si basano su componenti indipendenti. Il livello delle prestazioni raggiungibili utilizzando processi leggeri è spesso molto superiore, perché i thread di un processo possono comunicare utilizzando lo spazio di indirizzamento condiviso.

Un Web server è un tipo di applicazione in cui i thread possono notevolmente migliorare le loro prestazioni e interattività. Per ogni richiesta ricevuta,viene creato un nuovo thread che si occupa di interpretarla, recuperare la pagina web cercata e trasmetterla al client (in genere un web browser). Una volta creato il nuovo thread, il suo thread padre può continuare ad ascoltare eventuali nuove richieste. Poiché molti Web server sono sistemi multiprocessore, thread differenti possono ricevere e soddisfare concorrentemente diverse richieste, migliorando sia il throughput si il tempo di risposta. Tuttavia, il sovraccarico dovuto alla continua creazione e distruzione dei thread è significativo, per cui la maggior parte dei Web server moderni mantiene attivo un insieme (pool) di thread i cui elementi vengono assegnati al servizio delle richieste man mano che arrivano. Tali thread non vengono però distrutti dopo aver servito una richiesta ma, anzi, sono riassegnati al pool per essere riutilizzati per nuove richieste.

Stati dei thread: ciclo di vita

Quando i processi contengono diversi flussi di controllo, è possibile concepire ogni thread come in continua transizione da una serie di stati dei thread discreti all'altra.
(Il seguente insieme di stati si basa largamente sulla realizzazione dei thread in Java).
Un nuovo thread inizia il suo ciclo di vita nello stato born, rimane in tale stato fino a che il programma non avvia il thread, il che lo fa passare allo stato ready (detto anche stato runnable). In alcuni SO, un thread si trova già nello stato ready al momento della creazione, eliminando così lo stato born. Il thread in stato ready con la priorità più alta va nello stato running appena un processore diventa disponibile e il thread lo ottiene.
Un thread in uno stato running entra in stato dead quando completa il suo compito o comunque termina. Alcune librerie permettono a un thread di porre termine ad un altro thread, il che forza quest'ultimo a entrare in stato dead. Quando un thread entra in stato dead le sue risorse vengono rilasciate e viene eliminato dal sistema.
Un thread entra nello stato blocked quando deve attendere il completamento di un'operazione di I/O, un thread nello stato blocked non è assegnabile ad un processore fino a ce non sia stata completata la sua richiesta di I/O. A quel punto, il thread ritorna nello stato ready, e riprende l'esecuzione nel momento in cui si libera un processore.
Quando un thread è in attesa di un evento, entra in stato waiting quindi ritorna nello stato ready solo quando un altro thread effettua un'operazione di notify (notifica), chiamata anche operazione di wake. Quando un thread in stato waiting riceve un evento di notify, il thread passa dallo stato waiting allo stato ready.
Un thread in stato running può passare allo stato sleeping per un certo periodo di tempo, chiamato intervallo di sleep. Un thread in stato sleeping ritorna allo stato ready quando è trascorso il suo

intervallo di sleep, un thread in stato sleeping non può usare un processore, anche se ce n'è uno disponibile. I thread entrano in stato sleeping quando al momento non hanno lavoro da svolgere.

Operazioni sui thread

I thread e i processi hanno molte operazioni in comune, ad esempio:

- creazione
- terminazione
- sospensione
- recupero
- blocco
- risveglio

La creazione dei thread è per molti aspetti simile a quella dei processi. Quando un processo crea un thread, la libreria dei thread inizializza una struttura dati specifica che memorizza informazioni tra cui, il contenuto dei registri, il contatore di programma e l'identificativo del thread. A differenza dei processi, un thread non richiede al SO di inizializzare le risorse condivise con il processo padre e con gli altri thread del processo. La condivisione delle risorse richiede un numero di istruzioni minore rispetto a quello necessario per la loro inizializzazione, così la creazione di un thread è più veloce della creazione di un processo. Inoltre anche la terminazione di un thread è spesso più veloce di quella di un processo. Il ridotto carico di sistema indotto dalla creazione e terminazione di thread spinge gli sviluppatori software a realizzare, quando possibile, attività parallele per mezzo di thread multipli invece che con processi multipli.
Alcune operazioni sui thread, però, non corrispondono precisamente alle operazioni sui processi:

- *cancellazione:* un thread o un processo possono causare la terminazione prematura di un thread cancellandolo. A differenza dei processi, la cancellazione dei thread non ne garantisce la terminazione. Questo accade perché un thread può mascherare o disabilitare dei segnali; se un thread maschera il segnale di cancellazione, non lo riceverà fino a che non lo avrà riabilitato. Tuttavia, un thread non può mascherare un segnale di abort.
- *join:* in alcune realizzazioni thread, per esempio in Windows XP, l'inizializzazione di un processo dà luogo a un thread primario. Il thread primario si comporta come qualsiasi altro thread eccetto che, quando termina, termina anche il processo corrispondente. Per impedire che un processo termini prima che tutti i suoi thread abbiano completato la loro esecuzione, il thread primario generalmente "dorme" fino a che ogni thread creato non abbia ultimato l'esecuzione. In tal caso, si dice che il thread primario ha effettuato un join con i thread da esso creati. Quando un thread fa un join con un altro thread, il primo non va mai in esecuzione fino a che non termina il secondo.

Modelli di threading

- *Thread a livello utente:* questi thread a livello utente svolgono le loro operazioni in spazio utente, nel senso che i thread sono creati da librerie che non possono eseguire istruzioni privilegiate o accedere direttamente a primitive del kernel. I thread a livello utente sono trasparenti al SO, che tratta un processo multithread come un unico contesto di esecuzione. Di conseguenza il SO avvia in esecuzione il processo multithread considerandolo come una singola entità, invece che avviare in esecuzione i singoli thread.
  Tale implementazione dei thread è anche chiamata associazione dei thread molti-a-uno, in quanto il SO associa tutti i thread di un processo multithread a un singolo contesto di esecuzione. Quando un processo utilizza thread a livello utente, alcune librerie a livello utente si occupano di effettuare lo scheduling e l'avvio in esecuzione di ogni singolo thread, in quanto il SO è ignaro del fatto che il processo contenga più thread. Il processo multithread continua la sua esecuzione fino alla scadenza del suo quanto di tempo o fino a che non è interrotto dal kernel. I thread a livello utente non richiedono il supporto dei thread

da parte del SO, quindi sono più portabili in quanto non sfruttano alcuna API di sistema operativo. Un altro vantaggio è che, essendo una libreria, e non il SO a controllare lo scheduling dei thread, le applicazioni possono tarare l'algoritmo di scheduling dei thread della libreria in base alle esigenze delle specifiche applicazioni. Inoltre, un thread a livello utente non richiama il kernel per decisioni relative allo scheduling o a questioni di sincronizzazione. I processi multithread a livello utente, che svolgono un gran numero di operazioni legate a i thread (come lo scheduling o la sincronizzazione), possono quindi trarre vantaggio dal basso sovraccarico rispetto a thread che su basano sul kernel per svolgere tali operazioni. Le prestazioni dei thread a livello utente variano a seconda del sistema e del comportamento dei processi. Molti dei problemi dei thread a livello utente dipendono dal fato che il kernel percepisce un processo multithread come un singolo flusso di controllo. Un thread a livello utente, per esempio, non è adattabile a sistemi multiprocessore in quanto il kernel non può avviare simultaneamente in esecuzione un processo multithread su più processori, i thread a livello utente possono quindi ottenere prestazioni non ottime in sistemi multiprocessore.
Quando un thread a livello utente si blocca, il kernel manda in esecuzione un altro processo, che potrebbe contenere thread di priorità più bassa rispetto ai thread in stato ready contenuti nel processo bloccato. I thread a livello utente non supportano quindi una politica di scheduling basata su priorità visibili a livello dell'intero sistema, il che potrebbe essere particolarmente dannoso per i processi multithread con vincoli di tempo reale. Per risolvere questo problema alcune librerie trasformano le chiamate di sistema bloccanti in chiamate non bloccanti.

- *Thread a livello kernel:* mappando ogni thread con il proprio contesto di esecuzione i thread a livello kernel tentano di eliminare le limitazione dei thread a livello utente. Le implementazioni dei thread a livello kernel vengono chiamate associazione dei thread uno-a-uno, queste richiedono al SO di fornire ad ogni thread utente un thread a livello kernel che il SO sia in grado di avviare in esecuzione. Ogni thread a livello kernel si occupa di memorizzare i dati specifici per ogni thread nel sistema, per esempio il contenuto dei registri e l'identificatore. Quando un processo richiede a un thread a livello kernel tramite un'opportuna chiamata di sistema, il SO crea un thread a livello kernel che esegua le istruzioni del thread utente. Le associazioni uno-a-uno presentano diversi vantaggi, il kernel può inviare contemporaneamente in esecuzione i thread di un processo su diversi processori, il che migliora le prestazioni delle associazioni progettate per un'esecuzione concorrente. Il kernel può inoltre gestire individualmente ogni thread, nel senso che il SO può inviare in esecuzione i thread in stato ready di un processo, anche se uno più thread dello stesso processo sono in stato blocked. Le applicazioni che svolgono operazioni di I/O bloccante sono quindi in grado di eseguire altri thread mentre attendono il completamento di tali operazioni. Questo incrementa l'interattività, migliora le prestazioni, purché l'applicazione si possa avvantaggiare dell'esecuzione concorrente. I thread a livello kernel permettono al dispatcher del SO la gestione individuale di ogni thread utente. Se il SO implementa un algoritmo di scheduling basato sulle priorità, un processo che usa i thread a livello kernel può, per mezzo di esse, scegliere l'opportuno livello di servizio che ogni thread deve ricevere dal SO. I thread a livello kernel, però, non rappresentano sempre la soluzione migliore per le applicazioni multithread. Le implementazioni dei thread a livello kernel tendono ad essere meno efficienti di quelle a livello utente, perché le operazioni di scheduling e sincronizzazione invocano il kernel, il che sovraccarica il sistema. Un altro svantaggio è che i thread a livello kernel tendono a consumare più risorse di quelli a livello utente. In conclusione, i thread a livello kernel richiedono al SO la gestione di tutti i thread presenti nel sistema.

- *Combinazione di thread a livello utente e di thread a livello kernel:* si è cercati di ridurre le distanze tra le associazioni di thread molti-a-uno e uno-a-uno creando un'implementazione ibrida dei thread. La combinazione delle implementazioni dei thread a livello utente e a livello kernel prende il nome di associazione molti-a-molti, infatti questa implementazione associa diversi thread a livello utente ad un insieme di thread a livello kernel. Il numero di thread a livello utente e il numero di thread a livello kernel non deve necessariamente essere uguale. L'associazione molti-a-molti riduce questo sovraccarico implementando il thread pooling, tecnica che permette a un'applicazione di specificare il necessario numero di thread kernel. Il thread pooling può ridurre notevolmente il numero, e quindi il costo, delle operazioni di creazione e terminazione dei thread. Il thread kernel è quindi riutilizzabile per un nuovo thread utente creato successivamente. Ciò migliora i tempi di risposta in sistemi come i Web server, perché le richieste sono assegnabili a thread già esistenti nel pool. Questi thread kernel permanenti sono chiamati worker thread, perché svolgono in genere- diverse funzioni, a seconda del thread utente che viene a loro assegnato.
  Un'attivazione dello scheduler è un thread kernel in grado di avvisare di certi eventi una libreria di threading a livello utente, per esempio del blocco di un thread o della disponibilità di un processore. Questo tipo di thread kernel è chiamato "attivazione dello scheduler", perché la libreria di threading a livello utente può svolgere operazioni di scheduling quando "attivata" da una notifica di evento, a volte chiamata upcall.
  Al momento della creazione di un processo multithread, il SO genera un'attivazione dello scheduler che esegue il codice di inizializzazione della libreria di thread a livello utente del processo che, a sua volta, crea i thread e, se necessario, richiede processori addizionali per i suoi thread. Il SO genera un'attivazione dello scheduler per ogni processore allocato ad un processo, permettendo alla libreria a livello utente di assegnare diversi thread per l'esecuzione simultanea su più processori.
  Quando si blocca un thread utente, il SO salva lo stato del thread nella sua attivazione dello scheduler e ne crea una nuova per notificare alla libreria a livello utente il blocco di uno dei suoi thread. La libreria di threading a livello utente può quindi salvare lo stato del thread in stato blocked dalla sua attivazione dello scheduler e assegnare ad essa un thread diverso. Questo meccanismo evita il blocco di un processo multithread quando uno solo dei suoi thread si blocca. Il limite principale del modello di threading molti-a-molti è che complica il progetto del SO, oltre a mancare di tecniche standard per l'implementazione.

Thread di Linux

Nel sistema operativo Linux il supporto ai thread fu introdotto a livello utente nella versione 1.0.9 a livello kernel nella versione 1.3.56. Molti sottosistemi kernel di Linux non fanno distinzione fra thread e processi. Infatti, Linux alloca a questi lo stesso tipo di descrittore di processo, chiamato per entrambi task. Per la creazione di task "figli" Linux utilizza la chiamata di sistema fork, basata sul sistema UNIX. Linux risponde a tale chiamata di sistema creando un nuovo task che contiene una copia di tutte le risorse del task "padre", per esempio lo spazio di indirizzamento, il contenuto dei registri e lo stack.
Per permettere il threading, Linux fornisce una versione modificata di fork, clone crea una copia del task chiamante, che è anche il padre del task creato. A differenza di fork, clone accetta in ingresso dei parametri che permettono di specificare quali risorse vanno condivise tra padre e figlio. A partire dalla versione 2.6, il kernel di Linux adotta un modello di associazione dei thread di tipo uno-a-uno che supporta un numero arbitrario di thread nel sistema. Tutti i task sono gestiti dallo stesso scheduler, il che implica che processi e thread con la stessa priorità ricevano lo stesso livello di servizio. Lo scheduler è stato progettato per adeguarsi nel modo migliore alla crescita del numero di processi e di thread. La combinazione del modello uno-a-uno e un algoritmo di scheduling efficiente forniscono a Linux un'implementazione dei thread altamente scalabile.

Ogni task nella tabella dei processi memorizza informazioni relative al suo stato del momento, per esempio running, stopped e dead. Un task in stato running viene inviato in esecuzione sul processore, entra in stato sleeping quando si blocca o comunque non è eseguibile dal processore, entra in stato stopped quando riceve un segnale di sospensione (suspend). Lo stato zombie indica che un task è stato fatto terminare, ma non è ancora stato rimosso dal sistema. Un task nello stato dead è rimovibile dal sistema.

Thread di Windows XP

I componenti di un processo Windows XP sono codice, contesto di esecuzione (spazio di indirizzamento virtuale e vari attributi tra cui quelli relativi alla sicurezza), risorse e uno o più thread associati al processo. I thread rappresentano la reale unità di esecuzione, eseguono parte del codice del processo nel contesto del processo, utilizzandone le risorse. Oltre al contesto del suo processo, un thread contiene anche il proprio contesto di esecuzione che comprende il stack, i registri del processore e vari attributi, per esempio la priorità di scheduling.
Quando il sistema inizializza un processo, il processo crea un thread primario, che si comporta come un normale thread, eccetto per il fatto che, se il thread primario termina, ha fine anche il processo, a meno che tale thread non richieda esplicitamente al processo di non terminare. Un thread può creare altri thread appartenenti al suo stesso processo, tutti i thread appartenenti allo stesso processo ne condividono lo spazio di indirizzamento virtuale. I thread possono memorizzare i proprio dati privati in un'area chiamata Thread Local Storage (TLS).

I thread possono creare i cosiddetti fiber, somiglianti ai thread, a parte il fatto che l'ordine di esecuzione di un fiber è sotto il controllo del thread che lo ha creato. I fiber semplificano agli sviluppatori il porting delle applicazioni verso sistemi diversi che fanno uso dei thread a livello utente. Un fiber viene eseguito nel contesto del thread che lo ha creato. Il fiber deve quindi avere accesso alle informazioni di stato, il thread memorizza queste informazioni per ognuno dei sui fiber. Il thread stesso, essendo un'unità di esecuzione, deve convertirsi in fiber per tenere separate le sue informazioni di stato da quelle degli altri fiber in esecuzione nel proprio contesto. Infatti, le API di Windows forzano un thread a convertirsi in fiber prima di creare o mandare in esecuzione altri fiber. Il contesto del thread rimane, e tutti i fiber associati al thread vengono eseguiti in quel contesto. Una volta che un fiber ottiene il processore, viene eseguito fino a che non è interrotto il thread nel contesto corrispondente. Come i thread, anche i fiber posseggono la loro area di memoria privata, la Fiber Local Storage (FLS), che funzione per i fiber esattamente come la TLS funzione per i thread. Da notare che un fiber può anche accedere alla TLS del proprio thread, se un fiber termina allora termina anche il suo thread.

Windows XP mette a disposizione di ogni processo un pool di thread, che consiste in un certo numero di worker thread; si tratta di thread a livello kernel che eseguono le funzioni specificate da thread a livello utente. Dato che le funzioni specificate dai thread a livello utente superano a volte il numero di worker thread, Windows XP, mantiene in una cosa le richieste di esecuzione di funzioni. Il pool di thread è composto da worker thread che "dormono" fino a che non viene messa in cosa una richiesta di esecuzione di una funzione. Il thread che accoda la richiesta deve specificare la funzione da eseguire e deve fornire le informazioni sul contesto. Il pool di thread viene creato la prima volta che un thread richiede al pool l'esecuzione di una funzione. L'utilizzo di pool di thread può semplificare un'applicazione e renderla più efficiente, in quanto lo sviluppatore non deve preoccuparsi della creazione e distruzione di così tanti thread. Tuttavia, la tecnica del pool di thread trasferisce una parte del controllo dal programmatore al sistema, e questo può causare alcune inefficienze.

Gli stati in cui si può trovare un thread in Windows XP sono otto:

- al momento della creazione il thread si trova nello stato initialized

- conclusa l’inizializzazione il thread entra nello stato ready (in attesa di utilizzare un processore)
- un thread selezionato dal dispatcher come il successivo da eseguire entra nello stato standby in cui rimane in attesa del processore
- una volta che il thread ha ottenuto il processore, entra nello stato running, un thread esce da tale stato se termina la sua esecuzione, se il suo quanto di tempo scade, se è oggetto di preemption, se è sospeso o se è in attesa su qualche oggetto
- quando un thread completa le sue istruzioni, entra nello stato terminated, il sistema non elimina necessariamente subito tale thread, il che può ridurre il sovraccarico di creazione nel caso in il processo lanci di nuovo il thread in esecuzione. Il sistema elimina un thread dopo che sono state eliminate le risorse ad esso associate
- se su un thread in stato running viene fatta preemption oppure se esaurisce il suo quanto di tempo ritorna allo stato ready
- un thread in stato running entra in stato waiting quando è in attesa di un evento, oppure un thread potrebbe venir sospeso da un altro thread (con sufficienti diritti di accesso) o dal sistema, e forzato nello stato waiting fino al suo recupero
- una volta completata l’attesa, ritorna nello stato ready o entra nello stato transition, il sistema posiziona un thread nello stato transition se i dati del thread non sono al momento disponibili altrimenti il thread è pronto per l’esecuzione, il thread rientra nello stato ready nel momento in cui il sistema riporta in memoria centrale le pagine contenenti lo stack a livello kernel del sistema
- il sistema posiziona un thread nello stato unknown quando lo stato di un thread non è definibile (in genere a causa di qualche errore)

Scheduling del processore

Quando si ha la possibilità di scegliere il processo per decidere quale debba essere eseguito in un certo istante, si deve adottare una strategia, definita come politica di scheduling o modalità di scheduling. Una politica di scheduling deve tentare di raggiungere determinati obbiettivi, quali la massimizzazione del numero di un processi completati per unità di tempo (throughput) e quella di utilizzo del processore, la minimizzazione del tempo d’attesa prima dell’esecuzione per ogni processo (latenza) e la prevenzione di posticipazioni indefinite, in modo da garantire che ogni processo termini regolarmente.

Consideriamo tre livelli di scheduling:

- *Scheduling ad alto livello:* chiamato anche job scheduling o scheduling a lungo termine, determina quali lavori (job) competono attivamente tra loro per le risorse del sistema. Questo livello è anche chiamato scheduling di ammissione, perché determina quali lavori ottengono l’ammissione al sistema. Una volta ammessi, i lavori hanno inizio e diventano processi o gruppi di processi. La politica di scheduling ad alto livello definisce il grado di multiprogrammazione, ossia il numero totale di processi presenti in un sistema in un certo istante. L’ingresso di troppi processi può saturare le risorse di un sistema, portando al degrado delle prestazioni. In questo caso la politica di scheduling di alto livello può decidere di proibire temporaneamente l’ingresso a nuovi lavori, finché alcuni degli altri non siano finiti.
- *Scheduling di livello intermedio:* dopo che la politica di ammissione ha accettato un lavoro contenente uno o più processi lo scheduling di livello intermedio determina quali siano autorizzati a competere per i processori. Esso sospende e ripristina temporaneamente i processi per raggiungere un livello operativo costante e per realizzare certi obbiettivi a livello di sistema. Lo scheduling intermedio opera da cuscinetto tra l’ammissione di lavori al sistema e l’assegnamento dei processori ai processi che costituiscono questi lavori.

- *Scheduling di basso livello:* determina quali processi attivi il sistema assegnerà a un processore appena diventato libero. Le politiche di scheduling a basso livello assegnano in genere una priorità a ogni processo, che ne rispecchia il rilievo: più un processo è importante, più è probabile che la politica di scheduling lo selezioni per un'esecuzione successiva. Lo scheduler di basso livello, chiamato anche dispatcher, assegna (dispatches) un processore al processo selezionato: effettua quindi il dispatching. Il dispatcher opera molte volte al secondo e deve quindi sempre risiedere in memoria.

Scheduling preemptive e non-preemptive

Le tecniche di scheduling possono essere preemptive o non-preemptive. Una politica è non-preemptive (senza diritto di prelazione) se, una volta assegnato un processore o un processo, il sistema non può rimuovere quel processore dal processo. Una politica di scheduling è preemptive (con diritto di prelazione) se il sistema può muovere il processore dal processo che è in esecuzione. Sotto il controllo di una politica non-preemptive, ogni processore, una volta ricevuto il processore, prosegue l'esecuzione fino al complemento, o finché non lo rilascia volontariamente. Adottando una tecnica di scheduling preemptive, il processore può eseguire una porzione di codice di un processo e poi effettuare un cambio di contesto.
Lo scheduling preemptive è utile in sistemi in cui ci sono processi ad alta priorità che richiedono rapide risposte. Per rendere efficace la preemption, il sistema deve mantenere molti processi in memoria principale, in modo da averne uno disponibile appena si libera un processore.
Nei sistemi non-preemptive, i processi brevi possono subire elevati ritardi di servizio mentre quelli lenti giungono a termine: il vantaggio è che l'intervallo di tempo tra l'inizio e la fine (tempo di turnaround) è più predicibile, perché nuovi processi ad altra priorità non possono scalzare quelli già presenti. I problemi principali di questa politica derivano dal fatto che programmi errati potrebbero non rilasciare mai il controllo del sistema e che l'esecuzione di processi poco importanti potrebbe ritardare quelli più critici.
Il clock delle interruzioni permette di garantire agli utenti interattivi un tempo di risposta accettabile, prevenendo stalli dovuti a cicli infiniti e permettendo ai processi di rispondere a eventi dipendenti dal tempo.

Priorità

Per determinare come ordinare e assegnare i processi, gli scheduler usano spesso la priorità, le priorità possono essere assegnate staticamente o cambiare dinamicamente, queste quantificano l'importanza relativa dei processi.
Le priorità statiche rimangono fisse, quindi i meccanismi basati su tale tipo di priorità sono relativamente facili da implementare e aggiungono un carico addizionale abbastanza basso, questi meccanismi, tuttavia, non reagiscono ai cambiamenti di ambienti e nemmeno a quelli che possono incrementare il throughput o diminuire la latenza.
I meccanismi a priorità dinamica sono invece reattivi ai cambiamenti, il problema è che schemi a priorità dinamica sono più complessi da realizzare e hanno un carico aggiuntivo maggiore rispetto a quelli a priorità fissa. La speranza è che il sovraccarico sia giustificato dall'aumento di prestazioni del sistema.
Nei sistemi multiutente, un SO deve fornire un servizio accettabile ad un elevato numero di utenti, ma deve anche offrire il supporto a situazioni in cui a un particolare utente occorra un trattamento preferenziale.

Obbiettivi di scheduling

- Massimizzare il throughput, una politica di scheduling deve servire il maggior numero di processi per unità di tempo.

- Massimizzare il numero di processi interattivi che ricevono tempi di risposta accettabili.
- Massimizzare l'uso delle risorse, il meccanismo di scheduling deve tenere occupate le risorse del sistema.
- Evitare posticipazioni indefinite, un processo non deve sperimentare un tempo d'attesa illimitato prima o mentre è servito.
- Garantire la priorità, se il sistema assegna una priorità ai processi, la politica di scheduling deve favorire i processi a priorità maggiore.
- Minimizzare il sovraccarico, in quanto il carico aggiuntivo porta spesso ad uno spreco di risorse, se opportunamente investite queste risorse potrebbero migliorare le prestazioni.
- Garantire la predicibilità, minimizzando la varianza statistica dei tempi di risposta dei processi, un sistema può garantire che essi ricevano un livello di servizio predicibile.

Lo scheduler può prevenire le posticipazioni indefinite per mezzo dell'invecchiamento, ossia dell'incremento graduale della priorità di un processo durante l'attesa. Alla fine la priorità diverrà talmente alta da portare lo scheduler a selezionarlo per l'esecuzione.
Lo scheduler può aumentare il throughput favorendo quei processi le cui richieste possano velocemente essere soddisfatte, o il cui completamento permette l'esecuzione di altri processi, un esempio è favorire i processi che mantengono risorse chiave.
Tecnica dell'inversione di priorità: le priorità relative di due processi vengono invertite per permettere a quelle a quello a priorità maggiore di ottenere le risorse necessarie alla sua esecuzione; analogamente, lo scheduler può scegliere di favorire un processo che richiede risorse poco sfruttate, perché il sistema sarà maggiormente in grado di soddisfare la richiesta in un breve periodo di tempo.
Il modo migliore per minimizzare i tempi di risposta è la disponibilità di risorse sufficienti al momento opportuno, il prezzo di questa strategia è però che lo sfruttamento complessivo delle risorse sarà basso.
Nonostante le differenze di obbiettivi tra vari sistemi, molte modalità di scheduling presentano proprietà simili:

- Equità, una politica di scheduling è equa se tutti i processi sono trattati allo stesso modo e nessun processo viene posticipato in modo indefinito a causa di problemi dell'algoritmo.
- Predicibilità: in condizioni simili di carico di sistema, un certo processo dovrebbe essere sempre eseguito più o meno nello stesso tempo.
- Scalabilità: in gravose condizioni di carico le prestazioni di un sistema devono degradare gradualmente, non crollare immediatamente.

Criteri di Scheduling

Per realizzare gli obbiettivi di uno scheduling di sistema, lo scheduler deve considerare il comportamento dei processi. Un processo processor-bound tende a usare tutto il tempo di processore che gli viene allocato. Un processo I/O-bound usa il processore per poco tempo prima di rilasciarlo dopo aver generato una richiesta di I/O. Questo tipo di processi impiega molto del proprio tempo nell'utilizzo del processore, ed è frequentemente in attesa che un risorsa esterna serva la propria richiesta utilizzando il processore in modo irrisorio.
Un criterio di scheduling deve inoltre considerare se un processo sia interattivo o batch, un processo batch fornisce carico al sistema senza interagire con l'utente. Un processo interattivo interroga frequentemente l'utente per ottenere dei dati. Il sistema deve fornire ai processi interattivi tempi di risposta accettabili, mentre un processo batch può sopportare ritardi ragionevoli, analogamente una politica di scheduling deve tenere conto dell'urgenza di un processo.
Gli scheduler possono basare e loro decisioni tenendo in considerazione quanto frequentemente un processo ad alta priorità ne interrompa altri a priorità inferiore. I processi frequentemente interrotti sono meno favoriti, perché il breve periodo d'esecuzione prima della successiva interruzione non giustifica il carico dovuto al cambio di contesto necessario per rimetterlo in esecuzione.

Anche se alcuni progettisti sostengono che un processo che abbia ricevuto poco tempo d'esecuzione debba essere favorito mentre altri sostengono che debba essere favorito un processo in esecuzione da molto tempo perché potrebbe essere prossimo alla fine.

Algoritmi di scheduling

Questi algoritmi decidono quando e quanto a lungo vada eseguito ogni processo, determinano quando e per quanto sia possibile interromperlo, oltre alle priorità, al tempo d'esecuzione, al tempo che lo separa dal completamento, all'equità ed altre caratteristiche del processo.

- ✗ *Scheduling First-In-First-Out (FIFO)*
  L'algoritmo di scheduling più semplice è quello FIFO, detto anche Firs-Come-First-Served. I processi vengono assegnati secondo il loro tempo d'arrivo nella coda dei processi pronti. L'algoritmo FIFO è non-preemptive e, una volta che il processo ha ottenuto il processore esegue fino al suo completamento. Il criterio FIFO è equo, perché ordina i processi secondo il loro tempo di arrivo, così da fornire lo stesso trattamento a tutti i processi. Tale criterio è comunque un po' scorretto perché i processi lunghi fanno attendere quelli brevi, e i processi di minor rilievo fanno attendere quelli più importanti. Risulta inoltre poco utile per quanto riguarda i processi interattivi perché non può garantire tempi brevi di risposta.

- ✗ *Scheduling Round-Robin (RR)*
  I processi sono attivati in modalità FIFO ma viene loro assegnato un tempo di processore chiamato quanto o time slice. Se un processo non si completa prima che il suo quanto sia esaurito, il sistema lo interrompe lasciando il processore al processo successivo in attesa. Il sistema inserisce poi il processo interrotto in fondo alla coda dei processi pronti". Questo tipo di scheduling è efficace negli ambienti interattivi in cui il sistema deve garantire un tempo di risposta ragionevole. Il sistema può minimizzare il carico dovuto alla preemption attraverso un efficiente cambio di contesto e mantenendo i processi in attesa in memoria prinicpale.

- ✗ *Scheduling selfish Round-Robin (RR)*
  Usa l'invecchiamento delle priorità dei processi per farle crescere gradualmente nel tempo. In questo schema ogni processo entrato nel sistema resta inizialmente in una coda d'attesa dove invecchia fino a che la sua priorità non raggiunge lo stesso livello di quelli nella coda dei processi attivi. A questo punto viene inserito in questa coda e selezionato sulla base di una politica RR che considera anche gli altri processi. Lo scheduler comprende solo quelli nella cosa dei processi attivi, quindi i più vecchi sono favoriti rispetto a quelli appena entrati nel sistema.
  La priorità di un processo cresce ad una velocità 'a' mentre staziona nella cosa d'attesa e ad una velocità 'b', con b<=a, nella coda degli attivi. Quando b<a, i processi nella coda d'attesa invecchiamo ad una velocità più elevata rispetto a quelli nella cosa attiva, così possono entrare nella coda dei processi attivi e contendersi il processore. La modifica dei parametri a e b influisce su come l'età di un processo influenzi la latenza ed il throughput medi.

- ✗ *Scheduling Shortest-Process-First (SPF)*
  È una politica di scheduling non-preemptive in cui lo scheduler seleziona il processo in attesa con il più breve tempo d'esecuzione stimato. L'algoritmo SPF riduce il tempo medio di attesa, il tempo d'attesa ha tutta via una notevole varianza, è quindi meno predicibile che in FIFO, specialmente per processi grandi. In sostanza, SPF favorisce i processi brevi a spese di quelli più lunghi, i processi interattivi, in modo particolare, tendono a essere più corti di quelli processor-bound; questa politica sembra quindi fornire un buon tempo di

risposta. Il problema è che non è preemptive dunque i processi interattivi in arrivo non otterranno un servizio immediato.
SPF seleziona i processi da servire in modo tale da garantire che il suo successivo si completi e lasci il sistema non appena possibile. Tale procedura tende a ridurre il numero di processi in attesa, oltre che quello dei processi in attesa dietro a quelli grandi.
Un problema chiave di SPF è la richiesta di una stima precisa de tempo di esecuzione di un processo: tale informazione normalmente non è disponibile. Quindi SPF deve fare affidamento sulla stima dei tempi di esecuzione fornita dall'utente o dal sistema.
SPF deriva da una politica denominata Short Job First (SJF), che funziona in modo soddisfacente nello scheduling delle attività produttive nelle industri, ma che risula chiaramente inappropriata per lo scheduling low-level nei SO. SPF è non-preemptive quindi non adatto agli ambienti in cui debba essere garantito un tempo di risposta ragionevole.

- ✗ *Scheduling Highest-Response-Ratio-Next (HRRN)*
  Politica che corregge alcuni dei problemi di SPF, in particolare l'eccessiva tendenza a sfavorire i processi più lunghi e l'esagerata preferenza e l'esagerata preferenza accordata a quelli più brevi. HRRN è una politica di scheduling non-preemptive in cui la priorità di ogni processo è una funzione non solo del suo tempo di servizio, ma anche del tempo speso in attesa di quel servizio. Una volta che il processo ottiene il processore, esegue fino al completamento.
  HRRN calcola le priorità dinamiche con la seguente formula:
  priorità = (tempo atteso + tempo d'esecuzione)/tempo d'esecuzione
  Essendo il denominatore il tempo d'esecuzione, i processi più brevi sono privilegiati. Tuttavia, poiché il tempo d'attesa appare come numeratore, i processi lunghi in attesa riceveranno anch'essi un trattamento di favore. Questa tecnica è simile a quella dell'invecchiamento ed evita allo scheduler di posporre un processo indefinitamente.

- ✗ *Scheduling Shortest-Remaining-Time (SRT)*
  Questa politica è la controparte preemptive di SPF che tenta di far crescere i throughput servendo i piccoli processi in arrivo. SRT è efficace per sistemi con sequenze di lavori in ingresso, ma non lo è più nella maggior parte dei SO odierni. In SRT, lo scheduler seleziona il processo con il minor tempo d'esecuzione stimato fino al completamento. Un nuovo processo in arrivo con un tempo d'esecuzione stimato più breve può interrompere quelli in corso con un tempo maggiore.
  Questa politica richiede però una stima efficace del comportamento futuro dei processi: l'algoritmo deve mantenere informazioni circa il tempo di servizio attualmente sfruttato dal processo ed effettuare preemption occasionali. I nuovi processi in arrivo con tempo d'esecuzione più brevi vengono eseguiti quasi immediatamente. I processi più lunghi hanno tuttavia un tempo d'attesa con media e varianza maggiore che in SPF. Questi fattori portano un carico aggiuntivo superiore rispetto a SPF.
  La politica SRT offre teoricamente un tempo d'attesa minimo, ma in certe situazioni, a causa del carico aggiuntivo della preemption, SPF può dare prestazioni migliori.
  Bisognerebbe inoltre garantire che un processo in esecuzione non sia interrompibile quando il suo rimanente tempo d'esecuzione raggiunge una certa soglia minima, lo stesso problema ha luogo nel momento in cui arriva un processo con tempo d'esecuzione leggermente minore rispetto a quello che è in esecuzione sul processore in quel momento (la politica impone di interromperlo in favore di quelli in arrivo ma potrebbe non rivelarsi la scelta migliore); infatti se il carico aggiuntivo dovuto all'interruzione fosse per esempio più grande della differenza tra i tempi dei due processi, la preemption porterebbe a scarse prestazioni.

- ✗ *Code multilivello a retroazione:*
  - Un nuovo processo entra nella rete di accomodamento in fondo alla coda a livello più alto.
  - Il processo avanza in modalità FIFO fino ad ottenere il processore.
  - Se completa la sua esecuzione, se rilascia il processore in attesa di I/O o in attesa di qualche altro evento, esce dalla rete di accomodamento.
  - Se il quanto di un processo si esaurisce prima che questo rilasci volontariamente il processore, il sistema mette il processo alla fine della coda ad un livello più basso.
  - Finché il processo usa appieno il quanto fornitogli ad ogni livello, continua a muoversi verso code a più basso livello. Di solito c'è una coda di livello minimo in cui i processi circolano in modalità RR fino al loro completamento.
  - Il processo che successivamente otterrà il processore è quello all'inizio della lista non vuota di livello più alto nella coda multilivello a retroazione.
  - Un processo può essere interrotto da un altro in arrivo in una coda più alta.
  - In questo sistema, se una coda più alta contiene sempre almeno un processo, un processo in una coda più bassa può subire posticipazione indefinita. Per questo, solitamente, lo scheduler accresce il quanto di un processo man mano che avanza verso code a basso livello. Così, più un processo rimane nel sistema di code, maggiore sarà il suo quanto.
  - Questa tecnica favorisce i processi I/O-bound e quelli che necessitano solo di piccoli intervalli del tempo di processore, poiché entrano nel sistema ad un alto livello di priorità, ottenendo rapidamente il processore.
  - (Caso di un processo processor-bound) Il processo entra nel sistema con un'elevata priorità e viene messo nella coda più ad alto livello. Il processo ottiene il processore velocemente, usa completamente il suo quanto, il quanto si esaurisce e lo scheduler sposta il processo nella coda inferiore successiva. I processi interattivi continueranno a ricevere ragionevoli tempi di risposta e alla fine i processi processor-bound otterranno il processore con un quanto di tempo maggiore di quello della coda più alta e lo usano completamente. I processi processor-bound raggiungono la coda di più basso livello dove continuano ad essere eseguiti in modalità RR con gli altri processi fino al completamento.
  - Questa politica è dunque ideale per suddividere i processi in categorie secondo il loro utilizzo del processore.
  - Lo scheduler usa l'approssimazione secondo cui il comportamento tenuto più recentemente sia un buon indicatore di quello futuro. Ma se lo scheduler rimette sempre un processo già eseguito nella coda più bassa precedentemente occupata, non potrà adeguarsi ai cambiamenti di comportamento del processo. Lo scheduler può risolvere questo problema registrando anche la quantità di quanto non sfruttato nell'ultima esecuzione. Se il processo consuma completamento il suo quanto, verrà messo in una coda inferiore viceversa potrà essere messo in una coda superiore. Se un processo dà inizio ad una trasformazione da processor-bound a I/O-bound, potrà subire dei ritardi durate il cambio di comportamento del sistema ma lo scheduling sarà comunque in grado di reagire a tale cambiamento. Lo scheduler può anche far invecchiare il processo, promuovendolo in una coda superiore dopo un certo tempo d'attesa.
  - Questa politica è un buon esempio di meccanismo adattivo per cui la risposta cambia al variare del comportamento del sistema controllato. I meccanismi adattivi richiedono, solitamente, un maggior carico ma la risultante sensibilità ai cambiamenti rende il sistema più reattivo e ne giustifica l'uso.

- ✗ *Scheduling Fair Share (FSS):*
  Questa politica permette ad un sistema di garantire l'equità su gruppi di processi, restringendo ogni gruppo ad un certo sottoinsieme di risorse, FSS è stato sviluppato

specificamente per “attribuire uno specifico grado di risorse ad un gruppo correlato di utenti”. (funzionamento in un sistema UNIX) Il sistema suddivide le risorse tra vari gruppi fair share, e distribuisce le risorse non utilizzare da un gruppo a un altro, proporzionalmente alle loro relative necessità. I comandi UNIX stabiliscono i gruppi fair share e vi associano specifici utenti. Si ipotizzi che UNIX usi uno scheduler dei processi RR a priorità. Ogni processo ha un priorità e lo scheduler associa processi di una certa priorità con un coda per quel valore. Lo scheduler seleziona i processi pronti in cima alla coda a più alta priorità, i processi a pari priorità sono allocati RR. Un processo che richiede un ulteriore servizio dopo aver subito un’interruzione riceve priorità più bassa. Le priorità del kernel sono alte e relative a processi che vengono eseguiti nel kernel, mentre le priorità utente sono più basse. Eventi del disco ricevono una priorità maggiore di quella del terminale. Lo scheduler assegna la priorità dell’utente secondo il rapporto tra il tempo di utilizzo recente del processore e il tempo complessivo finora utilizzato; più basso è il tempo complessivo più alta sarà la priorità. I gruppi fair share sono ordinati per priorità rispetto alla vicinanza ai loto obbiettivi d’uso delle risorse, i gruppi lontani dagli obbiettivi riceveranno alta priorità, quelli vicini una priorità più bassa.

Scheduling a deadline

Nello scheduling a deadline alcuni processi sono selezionati in base al loto tempo di completamento predeterminato, detto anche deadline. Questi processi possono avere un altro valore se terminati in tempo o essere inutili negli altri casi.
Lo scheduling a deadline è piuttosto complicato. Innanzitutto, per garantire che i processi terminino in tempo, l’utente deve fornire le richieste di risorse in modo preciso ed in anticipo, queste informazioni sono raramente disponibili. Secondariamente, il sistema deve eseguire questi processi senza compromettere il servizio agli altri utenti. Inoltre il sistema deve pianificare attentamente le proprie risorse in previsione di ogni deadline, cosa difficile, perché possono arrivare nuovi processi con richieste non predicibili. Infine, se molti processi con deadline fossero attivi contemporaneamente, lo scheduling potrebbe diventare ancora più complesso.
L’intensa gestione di risorse richiesta dallo scheduling a deadline può generare un sostanziale carico aggiuntivo, il consumo netto di risorse può essere alto, peggiorando il servizio degli altri processi.
Lo scheduling a deadline è importante soprattutto per i processi con vincoli di tempo reale. Dertouzos dimostrò che tutti i processi possono rispettare la loro deadline indipendentemente dall’ordine in cui sono eseguiti, e che a scelta migliore sia quella di selezionare per primi i processi con la deadline più imminente. Tuttavia, quando il sistema diventa sovraccarico, deve allocare un tempo significativo allo scheduler per determinare l’ordine appropriato in cui eseguire i processi, in modo che possano rispettare la propria deadline.

Scheduling real-time

Lo scheduling real-time soddisfa le necessità di quei processi che devono produrre un risultato corretto entro un certo istante, ossia che hanno dei vincoli temporali (timing constraint). Un processo con vincoli di tempo reale può dividere le sue istruzioni in compiti separati, ognuno dei quali deve essere completato entro una scadenza precisa.
Gli scheduler real-time devono garantire il rispetto dei vincoli, le politiche di scheduling real-time sono classificate a seconda della precisione con rispettano questi vincoli, ossia la deadline di una processo.

- Lo scheduling soft real-time garantisce che i processi real-time siano allocati prima degli altri processi del sistema, ma non che qualcuno di loro riuscirà a rispettare i propri vincoli. Questi sistemi di scheduling possono trarre vantaggio da frequenti interruzioni di clock per impedire al sistema di rimanere bloccato nell’esecuzione di un processo mentre gli altri non

rispettano i propri vincoli. Tuttavia, se le interruzioni sono troppo frequenti, il sistema può incorrere in un elevato carico aggiuntivo, portando a scarse prestazioni o deadline mancate.

- Lo scheduling hard real-time garantisce sempre il rispetto dei vincoli temporali di un processo, ogni compito di un processo con forti vincoli di tempo reale deve completarsi prima della scadenza del proprio tempo limite, il fallimento di tale obbiettivo può portare a conseguenze devastanti. I sistemi con forti vincoli di tempo reale possono contenere processi periodici che eseguono calcoli ad intervalli regolari, e processi asincroni che vengono eseguiti a seguito di eventi.

Gli algoritmi di scheduling real-time statico non modificano nel tempo la priorità dei processi, poiché le priorità sono calcolate solo una volta, questi algoritmi risultano essere più semplici e incorrono in un minore carico aggiuntivo questa politica è però limitata dall'impossibilità di adeguarsi ai cambiamenti di comportamento dei processi e dalla dipendenza della funzionalità delle risorse per garantire il rispetto dei vincoli temporali. I sistemi con forti vincoli di tempo reale prediligono gli scheduling statici, perché richiedono un minore carico aggiuntivo e perché risulta facile dimostrare il rispetto dei vincoli temporali di ogni processo.

- L'algoritmo di scheduling *Rate Monotonic (RM)* è un algoritmo round-robin preemptive basato su priorità, che incrementa linearmente, ovvero in modo monotono, la priorità di un processo con la frequenza o tasso con cui deve essere eseguito. Questo algoritmo di scheduling statico favorisce i processi periodici eseguiti frequentemente. L'algoritmo di scheduling RM con deadline è utilizzabile quando un processo periodico specifica un deadline che non corrisponde al suo periodo.

Gli algoritmi di scheduling real-time dinamici selezionano i processi adattando la loro priorità durante l'esecuzione, cosa che può comportare un significativo carico aggiuntivo. Alcuni algoritmi tentano di minimizzare tale carico assegnando delle priorità statiche ad alcuni processi e dinamiche ad altre.

- *Earliest-Deadline-Firs (EDF)* è un algoritmo di scheduling preemptive che assegna i processi con la scadenza più prossima. Se un processo in arrivo ha una deadline più stringente di quella del processo in esecuzione, il sistema interrompe quest'ultimo a favore di quello in arrivo. L'obbiettivo è massimizzare il throughput soddisfacendo le deadline del maggior numero di processi per unità di tempo, e minimizzare il tempo d'attesa medio, che impedisce ai processi brevi di mancare le proprie scadenze mentre quelli lunghi sono in esecuzione.
- L'algoritmo *minimum-laxity-first* è simile a quello di EDF, ma basa la priorità sulla laxity del processo. La laxity è una misura dell'importanza di un processo basata sul tempo che precede la scadenza della sua deadline e sul tempo d'esecuzione per completare tutti i suoi compiti, che possono essere periodici. La laxity può essere calcolata usando la formula:

  $$L = D – (T + C)$$

  dove L è la laxity, D la deadline del processo, T è l'istante corrente e C il tempo di esecuzione rimanente del processo.
  Le priorità in questo tipo di algoritmo sono più accurate di quelle di EDF, perché sono determinate considerando il tempo necessario al processo per completarsi. Tuttavia, queste informazioni sono spesso non disponibili.

Gestione e organizzazione della memoria -capitolo 9-

Le strategie di gestione della memoria determinano il comportamento di una particolare organizzazione con diversi carichi di lavoro, di solito la gestione della memoria viene eseguita sia dal software sia da componenti hardware dedicati.
Il gestore della memoria è un componente del SO legato allo schema di organizzazione e alla strategia di gestione della memoria. Il gestore della memoria decide come allocare a un processo

uno spazio libero di memoria e come rispondere ai suoi cambiamenti di richiesta di memoria. Inoltre, se disponibili, interagisce con i componenti hardware dedicati alla gestione della memoria con lo scopo di migliorare le prestazioni.

Gerarchie di memoria

I programmi e i dai di cui il sistema non ha una necessità immediata possono essere mantenuti in un dispositivo di memorizzazione secondario e, quando richiesti, trasferiti alla memoria principale.
La gerarchia di memoria è composta da livelli, caratterizzati dalla velocità e dal costo della memoria che risiede in ciascuno di essi. Il sistema sposta i programmi e i dati avanti e indietro tra i vari livelli. Questa spola tra livelli consuma risorse di sistema, come il tempo di processore, dedicabili ad un uso più produttivo. Per incrementare l'efficienza, i sistemi attuali includono unità hardware dette controllori della memoria (memory controller) che eseguono le operazioni di trasferimento di memoria senza introdurre virtualmente un sovraccarico computazionale. Di conseguenza, i sistemi che sfruttano la gerarchia di memoria offrono il vantaggio di un costo minore e di una capacità maggiore.
La memoria cache è un livello addizionale molto più veloce rispetto alla memoria principale e si trova generalmente su ogni processore dei sistemi di oggi. Un processore può leggere programmi e dati direttamente dalla sua cache. Le memorie cache sono estremamente costose se confrontate con quella principale e quindi se ne usano di relativamente piccole.
Le memorie cache introducono un livello in più nel trasferimento dati nel sistema. Prima di essere eseguiti, i programmi in memoria principale sono trasferiti nella cache, essendo l'esecuzione dei programmi della cache molto più veloce rispetto alla memoria principale. Per il fenomeno della località temporale, molti processi che accedono a dati ed istruzioni uno per volta, accedono agli stessi dati ed istruzioni anche in futuro; quindi anche una cache relativamente piccola può incrementare significativamente le prestazioni, motivo per cui alcuni sistemi arrivano ad utilizzare diversi livelli di cache.

Strategie di gestione della memoria

Si suddividono in:
- strategie di fetch (prelievo)
- strategie di posizionamento
- strategie di sostituzione

Le strategie di fetch determinano quando spostare la sezione successiva di un programma o di dati dai dispositivi secondari alla memoria principale. Si dividono in due tipi: strategie di fetch a richiesta e strategie di fetch a previsione. In quelle a richiesta il sistema posiziona nella memoria principale la sezione successiva di programma o dati quando il programma in esecuzione ne fa richiesta, mentre quelle a previsione tentano di caricare una sezione di programma o dati nella memoria prima che questa sia richiesta.
Le strategie di posizionamento determinano dove il sistema dovrebbe memorizzare i programmi o i dati nella memoria principale.
Quando la memoria è piena e non può ospitare un nuovo programma, il sistema deve rimuovere qualcosa (o tutto), dei dati o dei programmi, che risiedono al momento in memoria. La strategia di sostituzione del sistema determina quale sezione di programmi o dati rimuovere.

Allocazione contigua e non contigua della memoria

Per eseguire un programma nei primi sistemi di elaborazione, l'operatore o il SO dovevano individuare una zona di memoria principale contigua sufficiente a contenere l'intero programma

(metodo noto come allocazione contigua della memoria). Se il programma era più grande della memoria disponibile, il sistema non poteva eseguirlo.
Nell'allocazione non contigua della memoria un programma è diviso in blocchi o segmenti che il sistema può posizionare in spazi prefissati di memoria principale non adiacenti. Questo permette l'uso di buchi (spazi inutilizzati) in memoria troppo piccoli per ospitare un intero programma; sebbene il SO debba così sostenere un sovraccarico maggiore, è giustificato dall'aumento del livello di multiprogrammazione, cioè il numero di processi ospitabili dalla memoria principale.

Allocazione contigua della memoria per sistemi monoutente

- *Overlay:* l'allocazione contigua della memoria può limitare le dimensioni di un programma eseguibili da un sistema, un modo per superare questi limiti è creare delle sezioni di codice dette overlay (sovrapposizioni), che consentano al sistema l'esecuzione di programmi più grandi della memoria principale. Il programmatore divide i programmi in sezioni logiche, quando il programma non ha bisogno di memoria per una sezione, il sistema la utilizza per caricare un'altra sezione che è stata richiesta. Gli overlay permettono ai programmatori di ampliare la memoria principale ma la loro gestione richiede comunque molta attenzione ed una lunga pianificazione. I sistemi a memoria virtuale hanno evitato al programmatore la necessità di controllare gli overlay, così come l'IOCS (Input/Output Control System) lo ha sollevato dalla manipolazione delle operazioni ripetitive di I/O di basso livello.

- *La protezione in un sistema monoutente:* un processo può interferire con la memoria del SO, intenzionalmente o inavvertitamente, sostituendo tutto o in parte il suo contenuto con altri dati, rovinando così il SO. La protezione in un sistema monoutente con allocazione contigua della memoria è realizzabile con un solo registro di boundary, situato all'interno del processo, modificabile solo da istruzioni privilegiate. Il registro di boundary contiene l'indirizzo di memoria dell'inizio del programma utente. Ogni volta che un processo richiede un indirizzo di memoria, il sistema determina se l'indirizzo è maggiore o uguale a quello contenuto nel registro di boundary. Verificata la condizione, il sistema esegue la richiesta di accesso alla memoria; in caso contrario, il programma cerca di accedere al SO. Il sistema intercetta questa richiesta ed elimina (kill) il messaggio con un messaggio d'errore appropriato. Il controllo degli indirizzi di boundary viene eseguito da componenti hardware che operano molto velocemente per evitare rallentamenti nell'esecuzione delle istruzioni. Ovviamente un processo per ottenere determinati servizi deve poter accedere al SO, motivo per cui esistono le chiamate a sistema, anche dette chiamate supervisore; il sistema individua la chiamata e passa dalla modalità di esecuzione utente alla modalità kernel o modalità executive. In modalità kernel, il processore esegue le istruzioni del SO e accede ai dati. Una volta esaudita la richiesta, il sistema ritorna alla modalità utente passando il controllo al processo. Il registro di boundary rappresenta un meccanismo di protezione semplice.

- *Elaborazione batch a singolo flusso:* i lavori (job) richiedevano generalmente un considerevole tempo di inizializzazione (setup time) ed al termine del lavoro era necessario un altrettanto considerevole tempo di terminazione (teardown time). Durante questi due periodi il sistema di elaborazione rimaneva inattivo. I progettisti capirono che se avessero potuto automatizzare alcuni aspetti della transizione di un processo all'altro (job-to-job transition), lo spreco di tempo poteva essere ridotto di molto. Queste considerazioni portarono allo sviluppo di sistemi a elaborazione batch (a lotti). Nell'elaborazione batch a singolo flusso, i lavori vengono raggruppati in lotti e caricati consecutivamente su nastri magnetici o dischi. Un esecutore legge le istruzioni del linguaggio di controllo dei job che definisce ciascun lavoro, facilita l'inizializzazione del successivo, interpreta le direttive dell'operatore di sistema ed esegue molte funzioni precedentemente svolte manualmente dall'operatore. Al termine del lavoro, si leggono le istruzioni del linguaggio di controllo per

quello successivo.
Il sistema di elaborazione batch ha notevolmente migliorato l'utilizzazione delle risorse ed ha aiutato a dimostrare il vero valore si un SO e di una gestione rigorosa delle risorse.

Multiprogrammazione con partizioni fisse

Nella memoria principale dell'elaboratore devono risiedere contemporaneamente più processi per trarre il massimo vantaggio dalla multiprogrammazione. Quando un processo richiede operazioni di I/O, può passare a un altro processo e continuare ad eseguire calcoli senza provocare il ritardo dovuto al caricamento dei programmi dai dispositivi secondari di memorizzazione. Una volta che questo nuovo processo cede il processore, un altro processo è pronto ad usarlo. I primi sistemi multiprogrammati usavano la multiprogrammazione con partizioni fisse; secondo questo schema, il sistema divide la memoria principale in un certo numero di partizioni di dimensione fissa. Ogni partizione contiene un solo processo e il sistema passa da un processo ad un altro rapidamente creando l'illusione della simultaneità, chiaramente la multiprogrammazione richiede in genere molta più memoria rispetto ad un sistema monoutente.

Nei primi sistemi multiprogrammati il programmatore elaborava il codice usando un assemblatore o un compilatore assoluto. Ciò rendeva il sistema di gestione della memoria relativamente semplice da realizzare, ma implicava che un processo avesse una precisa posizione in memoria, determinata prima che fosse lanciato, e che potesse essere eseguito solo in quella specifica partizione. Questa restrizione comportava uno spreco di memoria; infatti, se un processo era già pronto per essere eseguito e la partizione del programma occupata, il processo era costretto all'attesa anche se un'altra partizione fosse stata disponibile.

Per superare il problema dello spreco di memoria, gli sviluppatori crearono strumenti di sviluppo, compilatori con rilocazione, assemblatori, collegatori (linker) e loader (caricatori). Questi strumenti producono un programma rilocabile, eseguibile in una qualsiasi partizione disponibile che sia sufficientemente grande per contenere il programma. Nonostante la relativa eliminazione dello spreco di memoria dovuto agli indirizzi di memoria assoluti, le traduzioni con rilocazione ed i loader sono più complessi rispetto a quelli con indirizzamento assoluto.

In un sistema con multiprogrammazione, il sistema deve proteggere il SO da tutti i processi utente ed ogni processo da tutti gli altri. La protezione viene spesso realizzata attraverso l'uso di più registri di boundary. Il sistema può delimitare ogni partizione con registri di boundary, uno basso ed uno alto detti anche registro base e registro limite. Come nei sistemi monoutente, anche quelli con multiprogrammazione forniscono chiamate di sistema che permettono ai programmi utente di accedere ai servizi del SO.

Un problema rilevante in tutte le organizzazioni di memoria è la frammentazione, fenomeno che non permette al sistema di usare certe aree disponibili di memoria principale. I sistemi multiprogrammati con partizioni fisse risentono della frammentazione interna, che si manifesta quando la dimensione della memoria del processo e dei dati è più piccola della partizione in cui si esegue il processo.

Multiprogrammazione con partizioni variabili

Una partizione può essere troppo piccola per ospitare un processo o così grande da provocare una perdita considerevole di risorse a causa della frammentazione interna. Un semplice miglioramento fu quello di permettere ai processi di occupare solo lo spazio di cui necessitano, fino a coprire tutta la memoria principale disponibile. Questo schema è detto multiprogrammazione con partizioni variabili.

Rimaniamo nell'ambito di schemi di allocazione contigua in cui un processo deve occupare le locazioni di memoria adiacenti. La multiprogrammazione con partizioni variabili non risente della frammentazione interna, poiché le partizioni dei processi sono esattamente della dimensione del processo; in questo schema lo spreco non diventa evidente fino a quando i processi terminano e lasciano dei buchi in memoria principale.
Il sistema può continuare a posizionare nuovi processi in questi buchi però, poiché i processi vengono creati e si esauriscono, i buchi diventano più piccoli, fino a raggiungere una dimensione troppo ridotta per ospitare un nuovo processo. Questo fenomeno è detto frammentazione esterna, in cui la somma dei buchi è sufficiente per ospitare un altro processo, ma la dimensione di ogni buco è troppo piccola per ospitare qualsiasi processo disponibile.
Al termine di un processo di un sistema multiprogrammato con partizioni variabili, il sistema può determinare se l'area di memoria che si è resa di nuovo disponibile sia adiacente ad altre aree libere. Il sistema registra in una lista della memoria libera, sia l'esistenza di un nuovo buco sia l'ingrandimento di quello già esistente, in conseguenza dell'unione di due buchi adiacenti, questo tipo di unione è detta coalescing.
Nonostante il SO si occupi di unire queste zone inutilizzate di memoria, quelle separate, distribuite lungo tutta la memoria principale, ne costituiscono ancora una quantità significativa, che potrebbe essere sufficiente a soddisfare persino le richieste di memoria di un processo.
Un'altra tecnica per ridurre la frammentazione esterna è detta compattazione della memoria, che effettua la rilocazione di tutte le aree occupate di memoria lungo l'intera memoria principale, lasciando un unico grande buco libero al posto dei tanti piccoli buchi, tipici dei sistemi con partizioni variabili. Ora tutta la memoria libera disponibile è contigua, e di conseguenza un processo può essere eseguito se le sue richieste di memoria sono soddisfatte dal buco ottenuto dalla compattazione; la compattazione della memoria viene anche chiamata garbage collection (raccolta dei rifiuti).
La compattazione non è esente da svantaggi, il sovraccarico che introduce consuma le risorse di sistema destinabili ad un uso più produttivo; durante la compattazione il sistema cessa infatti tutte le altre elaborazioni. Questo implica un tempo di risposta imprevedibile per gli utenti interattivi e può avere effetti devastanti nei sistemi real-time.

In un sistema con partizioni variabili il sistema deve spesso scegliere quale buco di memoria assegnare ad un processo in ingresso. La strategia di posizionamento in memoria del sistema determina in quale posizione della memoria principale situare i programmi ed i dati.

- *Strategia first-fit:* il sistema posiziona un processo nuovo in memoria principale nel primo buco disponibile e grande a sufficienza per contenerlo. Questa strategia è intuitiva e permette al sistema di prendere velocemente decisioni di posizionamento.

- *Strategia best-fit:* il sistema posiziona un processo nuovo nel buco in memoria principale ce lo contiene lasciando inutilizzata la minima quantità di spazio possibile. Questa è in genere considerata la strategia più intuitiva, ma richiede un sovraccarico dovuto alla ricerca del buco più adatto, lasciando inutilizzati molti buchi.

- *Strategia worst-fit:* inizialmente può sembrare una scelta strana, ma ad un'attenta analisi questa strategia è intuitivamente efficace. Questa strategia suggerisce di posizionare un processo in memoria principale in un buco in cui il riempimento sia il peggiore, per esempio un buco molto grande. La sua efficacia è semplice: una volta copiato il programma nel buco molto grande, rimane abbastanza spazio da contenere un altro programma. I lati negativi sono il sovraccarico per individuare i buchi gradi e il fatto che tende comunque a lasciare molti buchi piccoli utilizzati.

Una variante della strategia first-fit, detta strategia next-fit, inizia le ricerche di un buco disponibile dal punto in cui la precedente ricerca era terminata.

Multiprogrammazione con swapping di memoria

In tutti gli schemi con multiprogrammazione finora trattati il sistema mantiene un processo in memoria principale fino al suo completamento. Un'alternativa a questo schema è lo swapping, che permette ad un processo di non rimanere necessariamente nella memoria principale durante la sua esecuzione.
In alcuni sistemi con swapping, un solo processo occupa la memoria principale in un dato istante. Il processo procede fino a quando deve terminare quindi rilascia al processo successivo sia la memoria sia il processore. Il sistema dedica quindi l'intera memoria ad un processo per un breve periodo. Quando il processo rilascia le risorse, il sistema cambia (swap) il vecchio processo con il successivo. Per effettuare lo swapping, il sistema mantiene nel dispositivo secondario di memorizzazione il contenuto della memoria del processo, oltre al suo PCB. Quando il sistema richiama il processo che aveva subito lo swap, vengono caricati dal dispositivo secondario il contenuto della memoria del processo e gli altri valori; normalmente un processo subisce lo swap diverse volte prima di terminare.

Organizzazione della memoria virtuale -capitolo 10-

L'idea alla base della memoria virtuale (1960, Università di Manchester) consiste nel creare l'illusione che ci sia più memoria. I sistemi con memoria virtuale forniscono infatti ai processi l'illusione di disporre di molta memoria installata nel calcolatore.
Esistono, nei sistemi con memoria virtuale, due tipi di indirizzi: quelli utilizzati dai processi (indirizzi virtuali) e quelli disponibili nella memoria principale (indirizzi fisici).
Ogni volta che un processo accede ad un indirizzo virtuale, il sistema deve trasformarlo in un indirizzo reale. Se quest'operazione venisse eseguita da un processore generico vi sarebbe un crollo delle prestazioni infatti i sistemi con MV dispongono di un componente hardware dedicato detto unità di gestione della memoria (MMU) che trasforma velocemente gli indirizzi virtuali in indirizzi reali.
Uno spazio di indirizzamento virtuale, V, è l'intervallo di indirizzi virtuali utilizzabili da un processo. L'intervallo di indirizzi reali in un particolare elaboratore è detto spazio di indirizzamento reale, R. Solitamente lo spazio di indirizzamento virtuale è molto maggiore di quello reale, se ciò è permesso risulta necessari mantenere i programmi ed i dati in una grande memoria ausiliaria.
In genere un sistema raggiunge questo obbiettivo utilizzando uno schema di memorizzazione a due livelli:

- un livello è composto dalla memoria principale e dalle cache, in cui devono risiedere le istruzioni ed i dati per essere disponibili al processore.
- l'altro livello è il dispositivo di memorizzazione secondario che consiste in un dispositivo ad alta velocità, normalmente un disco, capace di contenere i programmi ed i dati che non possono risiedere nella limitata memoria principale. Quando il sistema è pronto ad eseguire un processo, carica il codice ed i dati del processo del dispositivo secondario nella memoria principale.

La traduzione dinamica degli indirizzi (Dynamic Address Translation, DAT) è un meccanismo che converte durante l'esecuzione gli indirizzi virtuali in indirizzi fisici. Gli indirizzi contigui in uno spazio di indirizzamento virtuale non devono essere contigui anche nella memoria fisica: in questo caso si ha contiguità artificiale. La traduzione dinamica degli indirizzi e la contiguità artificiale permettono al programmatore di non preoccuparsi del posizionamento in memoria.

Mapping dei blocchi

Il meccanismo di traduzione dinamica degli indirizzi deve mantenere una mappa di traduzione degli indirizzi che indichi quali regioni dello spazio di indirizzamento virtuale, V, sono correntemente in memoria principale ed il loro posizionamento. La quantità di informazioni circa la mappatura deve occupare solo una piccola frazione della memoria principale, per non rubare quella necessaria sia al SO sia ai processi utente.
La soluzione più comune è quella di raggruppare le informazioni in blocchi, il sistema tiene traccia di dove viene posizionato ogni blocco di memoria virtuale nella memoria principale. Maggiore è la dimensione media dei blocchi, minori sono le informazioni di mappatura. Blocchi più grandi comportano comunque una frammentazione interna maggiore e prolungano i tempi di trasferimento tra il dispositivo secondario e la memoria principale.
Quando i blocchi hanno una dimensione fissa sono chiamati pagine e l'organizzazione della memoria virtuale è detta paginazione. Quando i blocchi hanno dimensioni differenti sono chiamati segmenti e l'organizzazione della memoria virtuale è detta segmentazione.
In un sistema con memoria virtuale e mapping dei blocchi, il sistema rappresenta gli indirizzi come coppie ordinate. Il processo, per riferirsi ad un particolare elemento dello spazio di indirizzamento virtuale di un processo, specifica il blocco in cui risiede l'elemento e lo spiazzamento (displacement) dell'elemento dall'inizio del blocco.
Un indirizzo virtuale v è composto da una coppia ordinata (b,d) in cui b è il numero del blocco in cui risiede l'elemento richiesto e d è il suo spiazzamento dall'inizio del blocco.
La traduzione da un indirizzo viruale v=(b,d) ad un indirizzo di memoria reale, r, avviene nel seguente modo: il sistema mantiene in memoria una tabella della mappa dei blocchi per ogni processo. Questa tabella contiene una riga per ogni blocco del processo e le righe sono ordinate sequenzialmente. Durante il cambio di contesto il sistema determina l'indirizzo reale,a , che corrisponde all'indirizzo in memoria principale della tabella della mappa dei blocchi relativa al nuovo processo. Il sistema carica questo indirizzo in un registro dedicato ad alta velocità chiamato registro origine della tabella dei blocchi. Il sistema aggiunge il numero di blocco b all'indirizzo base a e ottiene la riga della tabella che contiene l'indirizzo reale del blocco b.
Si noti che per semplificare, si assume che la dimensione di ogni riga della tabella sia fissa e che ogni indirizzo in memoria contenga una riga di quella dimensione. Questa riga contiene l'indirizzo b', che rappresenta l'inizio del blocco b in memoria principale. Il sistema aggiunge lo spiazzamento d all'indirizzo iniziale b' per ottenere l'indirizzo reale r. Quindi il sistema calcola l'indirizzo reale r dall'indirizzo virtuale v=(b,d) attraverso la formula r = b'+d, dove b' è memorizzato nella tabella della mappa dei blocchi all'indirizzo reale a+b .
È importante notare che il mapping dei blocchi viene eseguito dinamicamente durante l'esecuzione di un processo da un veloce dispositivo hardware dedicato.
Il meccanismo di traduzione degli indirizzi dei blocchi deve accedere alla tabella della mappa dei blocchi molto più velocemente di qualsiasi altra informazione contenuta in memoria. In genere, il sistema posiziona le righe della tabella in una cache ad alta velocità che riduce notevolmente il tempo necessario al loro reperimento.

Paginazione

In un sistema paginato un indirizzo virtuale è una coppia ordinata (p, d), dove p è il numero della pagina nella memoria virtuale in cui risiede l'indirizzo e d è lo spiazzamento all'interno della pagina p. Un processo è eseguibile se la pagina che lo contiene risiede nella memoria principale. Quando un sistema trasferisce una pagina dal dispositivo secondario alla memoria principale, posiziona la pagina in un blocco di memoria principale, detto page frame (cornice di pagina), della stessa dimensione della pagina entrante.
Un processo in esecuzione fa riferimento ad un indirizzo di memoria virtuale =(p,d). il meccanismo di mappatura delle pagine cerca la pagina p nella tabella di mappatura delle pagine del processo

(detta tabella delle pagine) e determina ce la pagina p è nel page frame p'. Si noti che p' è il numero di page frame e non un indirizzo fisico.
Un vantaggio dei sistemi con memoria virtuale paginata è che non tutte le pagine di un processo devono risiedere nella memoria principale nello stesso istante, ma solo quelle che sta utilizzando il processo. Il lato positivo è che molti processi possono risiedere simultaneamente nella memoria principale; quello negativo è rappresentato dalla crescente complessità del meccanismo di traduzione degli indirizzi. Poiché la memoria principale normalmente non contiene contemporaneamente tutte le pagine di un processo, la tabella delle pagine deve indicare se una pagina risiede in memoria principale o meno. Se la pagina è in memoria principale, la tabella delle pagine indica il numero del page frame in cui risiede, altrimenti restituisce la posizione nel dispositivo secondario a cui fare riferimento per accedere alla pagina richiesta. Quando un processo richiede una pagina che non è nella memoria principale, il processore genera un errore di pagina (page fault) che invoca il sistema operativo per caricare la pagina mancante dal dispositivo secondario verso la memoria reale.

Traduzione degli indirizzi di una pagina con mappatura diretta (direct mapping)

Un processo richiede un indirizzo virtuale v = (p,d). Durante il cambio di contesto, il SO carica l'indirizzo di memoria principale della tabella delle pagine del processo nel registro origine della tabella delle pagine. Per determinare l'indirizzo reale che corrisponde all'indirizzo virtuale, il sistema deve aggiungere all'indirizzo base della tabella delle pagine b il numero della pagina richiesta p (cioè l'indice della tabella delle pagine). Questo risultato, b+p, è l'indirizzo in memoria principale della PTE (Page Table Entry) per la pagina pagina p. Si ipotizza che la dimensione di ogni riga della tabella delle pagine sia fissa e che ogni indirizzo in memoria possa memorizzare una riga di quella dimensione.
Il sistema concatena p' con lo spiazzamento d per formare l'indirizzo reale r. La mappatura diretta è molto simile all'accesso dei dati in un array: il sistema può localizzare direttamente qualsiasi riga della tabella con un solo accesso ad essa. Il sistema mantiene l'indirizzo virtuale e l'indirizzo base della tabella delle pagine in registri ad alta velocità del processore, in modo da consentire l'esecuzione rapida di operazioni che utilizzano questi valori. Tuttavia, un sistema mantiene di solito la tabella delle pagine, che può anche essere abbastanza grande, nella memoria principale. Un riferimento alla tabella delle pagine richiede un ciclo completo di lettura della memoria principale. Poiché il tempo di accesso alla memoria principale rappresenta la pare maggiore del ciclo di esecuzione di un'istruzione e poiché si richiede un accesso addizionale alla memoria per il mapping della pagina, l'uso della paginazione direct mapping può comportare il dimezzamento della velocità del sistema di elaborazione. Per ottenere una traduzione più veloce, si potrebbe realizzare completamente la tabella di paginazione in una memoria cache ad alta velocità ma dato che il costo della memoria ad alta velocità è elevato e la dimensione dello spazio di indirizzamento virtuale potenzialmente notevole, non è possibile mantenere l'intera tabella delle pagine nella memoria cache.

Traduzione degli indirizzi di una pagina con mappatura associativa (associative mapping)

Questa tecnica consiste nel posizionare l'intera tabella delle pagine in una memoria associativa indirizzata in base al contenuto (content-address) invece che attraverso un indirizzo. La traduzione dinamica degli indirizzi con una mappatura associativa funziona così: un processo fa riferimento ad un indirizzo virtuale v=(p,d). Tutte le righe della memoria associativa sono analizzate contemporaneamente per cercare la pagina p. La ricerca restituisce p' come page frame corrispondente alla pagina p e la concatenazione di p' con d costituisce l'indirizzo reale r.
Utilizzare una memoria cache per implementare la mappatura diretta pura o una memoria associativa per realizzare la mappatura associativa pura è, nella gran maggioranza dei sistemi,

troppo costoso. Di conseguenza numerosi progettisti hanno raggiunto un compromesso che offre molti dei vantaggi dell'approccio con cache o con memoria associativa ad un costo più modesto.

Traduzione degli indirizzi di una pagina con mappatura diretta/associativa combinata (direct/associative mapping)

Si ricorre ad una memoria associativa, detta Translation Lookaside Buffer (TLB), capace di mantenere solo una piccola percentuale di tutta la tabella delle pagine del processo. A seconda dell'architettura il contenuto della TLB è controllabile attraverso il sistema operativo o via hardware. Le righe delle tabelle delle pagine presenti in questa mappa corrispondono solo alle pagine di più recente utilizzo, basandosi sull'euristica secondo cui una pagina utilizzata recentemente sarà presto di nuovo riutilizzata. Questo è un esempio di località, più precisamente di località temporale. La TLB è parte integrante degli attuali MMU.
La traduzione dinamica degli indirizzi funziona nel seguente modo: un processo richiede un indirizzo virtuale v=(p,d). Il meccanismo di mappatura delle pagine prova prima ad individuare nella TLB la pagina p. Se la TLB la contiene, si ottiene il page frame p' che, concatenato con lo spiazzamento d, forma l'indirizzo reale r. Quando il sistema ottiene dalla TLB il mapping di p, si dice che avviene un TLB hot (successo nell'accesso alla TLB). Poiché la TLB è una memoria associativa ad alta velocità, la traduzione avviene alla massima velocità possibile per il sistema.
Il problema è che a causa dei costi proibitivi la mappa associativa può coprire solo una piccola parte dello spazio di indirizzamento virtuale. La difficoltà sta nello stabilire la dimensione della mappa entro i vincoli economici del progetto, pur garantendo un numero di pagine sufficiente ad assicurare un'alta percentuale di TLB hit. Se la TLB contiene non contiene la riga della pagina p, ovvero avviene un TLB miss (fallimento nell'accesso alla TLB), il sistema localizza la riga della tabella della pagina usando la mappatura diretta accedendo alla memoria principale, azione che causa un incremento del tempo di esecuzione.
Il sistema posiziona quindi la PTE nella TLB così che i successivi riferimenti alla pagina p sano velocemente traducibili. Se la TLB è piena, il sistema rimuove una riga, in genere quella utilizzata meno recentemente, per far spazio alla riga relativa alla pagina corrente. Grazie al fenomeno empirico della località, per ottenere buone prestazioni non deve essere elevato il numero di righe residenti nella TLB.

Tabella delle pagine multilivello

Una limitazione nella traduzione degli indirizzi usando la mappatura diretta è che tutte le PTE di una tabella devono essere mappate e memorizzate in modo contiguo ed in ordine sequenziale per numero di pagina. Inoltre, queste tabelle possono occupare una quantità significativa di memoria.
Le tabelle delle pagine multilivello o gerarchiche permettono al sistema di memorizzare in locazioni non contigue la tabella delle pagine attivamente utilizzate da un processo. Le altre pagine possono essere create la prima volta che vengono caricare dal dispositivo secondario e spostate nel dispositivo secondario quando cessano di essere usate attivamente. Le tabelle delle pagine multilivello sono implementate creando una gerarchia: ogni livello contiene una tabella che memorizza i puntatori alla tabella nel livello sottostante. Il livello più basso è composto da tabelle che contengono la mappatura tra pagine e page frame.
In molti sistemi ogni tabella nella gerarchia ha la dimensione di una pagina, permettendo al SO di trasferire facilmente le tabelle delle pagine tra memoria principale e dispositivo secondario.
Se lo spazio è occupato in modo sparso, cioè se sono utilizzare poche righe, gran parte della tabella delle pagine rimane inutilizzata, situazione che implica un significativo spreco di memoria detto frammentazione della tabella. La frammentazione si riduce utilizzando una tabella delle pagine a due livelli, infatti le tabelle delle pagine multilivello permettono al sistema di ridurre la frammentazione di oltre il 99 percento.

Il sovraccarico introdotto dalle tabelle multilivello corrisponde ad un accesso in memoria in più per il meccanismo di mappatura delle pagine. Inizialmente pu sembrare ce questo ciclo di memoria addizionale peggiori le prestazioni ma grazie alla località dei riferimenti e alla disponibilità di una TLB ad alta velocità, una volta che una pagina virtuale è stata mappata con il suo page frame, i riferimenti successivi non diretti alla pagina non subiscono il sovraccarico nell'accesso alla memoria. Quindi, per ottenere prestazioni elevate, i sistemi che utilizzano una tabella dele pagine multilivello possono contare su di un valore di TLB miss estremamente basso.

Tabelle inverse delle pagine

Si presume che i processi usino solo una regione ristretta e contigua del loro spazio virtuale di indirizzamento, in modo tale che il sistema possa gestire solo le righe relative allo spazio di indirizzamento effettivamente utilizzate. Se usano una porzione significativa dello spazio di indirizzamento a 32 bit, le tabelle multilivello non riducono necessariamente la frammentazione. Una tabella inversa delle pagine risolve questo problema memorizzando esattamente una PTE per ogni page frame del sistema. Di conseguenza, il numero di PTE da memorizzare nella memoria principale è proporzionale alla dimensione della memoria fisica, non alla dimensione dello spazio di indirizzamento virtuale. Le tabelle sono invertite rispetto a quelle tradizionali, poiché le PTE sono indicizzate per numero di page frame invece che per numero di pagina virtuale.
Le tabelle inverse non memorizzano le locazioni del dispositivo secondario delle pagine non residenti in memoria principale, ma queste informazioni devono essere gestite dal SO e non devono necessariamente risiedere nelle tabelle.
Le tabelle inverse usano funzioni hash per mappare gli indirizzi virtuali nelle PTE. Una funzione hash è una funzione matematica che riceve un numero come ingresso e restituisce un numero, compreso in un intervallo finito, detto valore hash. Una tabella hash memorizza ogni elemento nella cella corrispondente al valore hash dell'elemento. Poichè il dominio della funzione hash (lo spazio di indirzzamento virtuale) è generalmente più grande del suo intervallo (il numero di page frame), diversi valori d'ingresso restituiscono la stessa uscita: tale fenomeno è detto collisione. Per impedire che diversi elementi sovrascrivano gli altri quando sono mappati nella stessa cella della tabella hash, le tabelle inverse possono realizzare una variante con il meccanismo del concatenamento (chaining) che risolve il problema delle collisioni. Quando un valore hash mappa un elemento in una locazione che è occupata, al valore hash viene applicata una nuova funzione. Il valore ottenuto è usato come posizione nella tabella dove è stato collocato l'ingresso. Per assicurare che l'elemento sia reperibile quando ricercato, alla riga viene aggiunto un puntatore alla nuova posizione nella cella corrispondente al valore hash originale. Questo procedimento viene ripetuto ogni volta che si incorre in una collisione. Generalmente per concatenare gli elementi le tabelle inverse usano una lista collegata. In un sistema paginato che adotta tabelle inverse, ogni indirizzo virtuale è rappresentato da una coppia ordinata v=(p,d). Per individuare velocemente la riga della tabella hash corrispondente alla pagina virtuale p, il sistema applica la funzione hash su p, che produce il valore q. Se la q-esima cella della tabella inversa contiene p, l'indirizzo virtuale richeisto è nel page frame q, se invece la cella non contiene p, il sistema controlla il valore nella catena dei puntatori di quella cella. Se il puntatore è nullo, la pagine non è in memoria ed il processore genera un page fault; il sistema può quindi recuperare la pagina dal dispositivo secondario. Se invece c'è stata una collisione a questo indice, la riga della tabella memorizza un puntatore alla PTE successiva della catena. Il sistema percorre la catena fino a quando trova la PTE che contiene p, il processore genera un page fault indicando che la pagina non è disponibile e il SO deve recuperarla dal dispositivo secondario.
Nonostante una scelta attenta della funzione hash possa ridurre il numero di collisioni nella tabella hash, ogni puntatore addizionale della catena richiede che il sistema acceda alla memoria principale, con conseguente incremento del tempo necessario a tradurre un indirizzo. Per migliorare le prestazioni , il sistema può tentare di ridurre il numero di collisioni aumentando l'intervallo dei valori hash prodotti dalla funzione. Poiché la dimensione della tabella inversa deve rimanere fissa

per fornire una mappatura diretta verso i page frame, molti sistemi aumentano l'intervallo della funzione hash usando una tabella hash di ancoraggio, cioè una tabella hash che contiene i puntatori alle righe della tabella inversa. La tabella hash di ancoraggio impone un accesso addizionale in memoria per tradurre un indirizzo da virtuale a fisico usando la tabella inversa, incrementando così la frammentazione della tabella. Se la tabella di ancoraggio è sufficientemente grande, tuttavia, il numero di collisioni nella tabella inversa è significativamente riducibile velocizzando, così la traduzione degli indirizzi. La dimensione della tabella di ancoraggio va scelta attentamente per bilanciare le prestazioni della traduzione degli indirizzi con il sovraccarico in memoria.

Condivisione in un sistema paginato

In un sistema multiprogrammato, è frequente che gli utenti eseguano gli stessi programmi contemporaneamente. La soluzione più ovvia è che il sistema condivida le pagine comuni ai singoli processi. Il sistema deve controllare attentamente la condivisione, in modo da impedire che un processo non modifichi i dati a cui sta accedendo un altro processo. In molti dei sistemi odierni che realizzano la condivisione, i programmi sono suddivisi in aree separate destinate ai dati ed alle procedure. Le procedure non modificabili sono dette procedure pure o procedure rientranti. I dati modificabili non sono condivisibili senza un attento controllo di simultaneità, mentre i dati non modificabili sono condivisibili. Le procedure modificabili non possono essere condivise, sebbene si possa immaginare un esempio estremo in cui abbia senso farlo con un sistema di controllo di simultaneità.
Esiste dunque la necessità di identificare se ogni pagina sia o meno condivisibile. Quando tutte le pagine di un processo sono state classificate in questo modo, è realizzabile la condivisione in un sistema con paginazione pura. La condivisione riduce la quantità di memoria principale richiesta da un gruppo di processi per un'esecuzione efficiente e permette ad un sistema di aumentare il suo grado di multiprogrammazione.
Il SO deve assicurare che i due processi non interferiscano quando si modificano le pagine. Per risolvere questo problema molti SO usano una tecnica detta copia-su-scrittura (copy-on-write). Inizialmente il sistema mantiene una copia di ogni pagina condivisa in memoria. Se una pagina tenta di modificare una pagina condivisa, il SO crea una copia di questa pagina, applica la modifica ed assegna la nuova copia allo spazio di indirizzamento virtuale di quella stessa pagina. La copia non modificata della pagina rimane mappata negli spazi di indirizzamento di tutti gli altri processi che la condividano. Questo assicura che, quando un processo modifica una pagina condivisa, nessun altro processo ne sia influenzato.
Molte architetture forniscono per ogni PTE un bit di lettura/scrittura che il processore controlla ogni volta che un processo richiede un indirizzo. Quando il bit è disattivato, la pagina è leggibile, ma non modificabile. Quando il bit è attivo, la pagina è sia leggibile sia modificabile. La tecnica copia-su-scrittura può essere realizzata marcando ogni pagina condivisa in solo lettura. Quando un processo tenta di scrivere in una pagina, il processore che ha generato un page fault invoca il kernel. A questo punto,il kernel è in grado di determinare che il processo sta tentando di scrivere in una pagina condivisa ed esegue la copia-su-scrittura. La copia-su-scrittura velocizza la creazione dei processi e riduce il consumo di memoria per i processi che durante la loro esecuzione non modificano una quantità significativa di dati. Se una porzione significativa dei dati condivisi è modificata durante l'esecuzione del programma, la copia-su-scrittura può ridurre le prestazioni. In questo caso, i processi causeranno un page fault ogni volta che modificheranno una pagina ancora condivisa. I vantaggi della copia-su-scrittura vengono rapidamente annullati dal sovraccarico associato alla condivisione del kernel per ogni eccezione.

Segmentazione

Un'alternativa alla paginazione è la segmentazione fisica della memoria, le istruzioni e i dati di un programma sono divisi in blocchi detti segmenti. Ogni segmento è composto da locazioni contigue, comunque non è necessario che i segmenti abbiano le stesse dimensioni, né che debbano trovarsi in posizioni adiacenti in memoria principale.
Rispetto alla paginazione, la segmentazione ha il vantaggio di essere un concetto logico invece che fisico; i segmenti possono essere grandi o piccoli a seconda della necessità.
In un sistema con memoria virtuale segmentata è possibile mantenere in memoria solo quei segmenti richiesti da un programma per la sua esecuzione in un certo istante; il resto dei segmenti risiede nel dispositivo secondario. Un processo può essere in esecuzione mentre le sue istruzioni ed i dati correnti risiedono in un segmento collocato in memoria principale. Se un processo fa riferimento ad un segmento che non risiede in memoria principale, il sistema a memoria virtuale deve recuperarlo dal dispositivo secondario.
Un nuovo segmento è allocabile in una qualsiasi area disponibile della memoria principale grande a sufficienza per contenerlo. Un indirizzo di memoria virtuale segmentata è una copia ordinata v=(s,d) dove s è il numero del segmento in memoria virtuale in cui l'elemento referenziato risiede, mentre d è lo spiazzamento interno al segmento s dove è allocato l'elemento referenziato.

Traduzione degli indirizzi segmentati con mappatura direttamente

Consideriamo la traduzione degli indirizzi segmentati usando la mappatura diretta e mantenendo la tabella di mappatura completa dei segmenti in una memoria cache ad accesso veloce.
Un processo fa riferimento ad un indirizzo virtuale v=(s,d) per determinare dove risiede il segmento referenziato in memoria principale. Il sistema aggiunge il numero del segmento s all'indirizzo base della tabella di mappatura dei segmenti b (indirizzo base della tabella dei segmenti), memorizzato nel registro origine della tabella dei segmenti. Il valore risultante b+s è la locazione della riga della tabella di mappatura dei segmenti. Per semplificare si ipotizza che la dimensione di ciascuna riga della tabella è fissa e che ogni indirizzo in memoria contiene una riga di quella dimensione.
Se il segmento risiede al momento in memoria principale, la riga contiene l'indirizzo iniziale del segmento s' in memoria principale. Il sistema aggiunge a questo indirizzo lo spiazzamento d per formare l'indirizzo di memoria reale richiesto (r=s'+d).
Tutti i riferimenti ad un segmento sono controllati rispetto alla sua lunghezza l per assicurarsi che cadano all'interno dell'intervallo di indirizzi del segmento. Per determinare se l'operazione che si sta eseguendo sia ammessa, vengono anche controllati i bit di protezione per ogni riferimento.
Durante la traduzione dinamica degli indirizzi, una volta individuata la riga della tabella per il segmento s, viene esaminato il resident bit r per determinare se il segmento è in memoria. Se non lo è, viene generato un missing-segment fault (fallimento di accesso al segmento), e il controllo passa al SO che carica il segmento con indirizzo a sul dispositivo secondario. Una volta caricato, la traduzione prosegue controllando che lo spiazzamento d sia minore o uguale alla lunghezza del segmento l; in caso negativo viene generata una segment-overflow exception (eccezione di superamento dell'area del segmento), passando il controllo al SO che può terminare il processo. In caso positivo, vengono controllati i bit di protezione per assicurarsi che sia ammessa l'operazione in esecuzione. Se tutto è a posto, l'indirizzo base del segmento s' viene sommato allo spiazzamento d per formare l'indirizzo reale r=s'+d corrispondente all'indirizzo virtuale v=(s,d). Se l'operazione non è ammessa, viene generata una segment-protection exception (eccezione di protezione del segmetno) che passa il controllo al SO per la conclusione del processo.

Condivisione nei sistemi segmentati

La condivisione dei segmenti può introdurre un sovraccarico minore rispetto alla condivisione in un sistema con paginazione pura e mappatura diretta.
Due processi condividono un segmento quando le righe delle loro tabelle dei segmenti puntano allo stesso segmento in memoria principale. Nonostante la condivisione offra ovvi benefici, comporta anche certi rischi. Per esempio, un processo può eseguire intenzionalmente o meno delle operazioni che influiscono negativamente sugli altri processi che condividono il segmento. Un sistema che fornisce la condivisione deve anche fornire meccanismi di protezione appropriati, per assicurare che solo gli utenti autorizzati possano accedere o modificare un segmento.

Protezione e controllo degli accessi in un sistema segmentato

La segmentazione favorisce la modularità dei programmi e consente un miglior uso della memoria attraverso l'allocazione non contigua e la condivisione. Con questi vantaggi aumenta però la complessità, diventa più difficile limitare l'intervallo degli accessi di un dato programma. Uno schema per realizzare la protezione della memoria nei sistemi segmentati fa uso delle chiavi di protezione della memoria (memory protection keys). A ogni processi viene associato un valore, detto chiave di protezione. Il SO controlla rigidamente queste chiavi, manipolabili solo da istruzioni privilegiate. Il SO utilizza le chiavi di protezione nel modo seguente.
Durante un cambio di contesto, il SO carica la chiave di protezione del processo all'interno di un registro del processore. Quando il processo fa riferimento ad un segmento particolare, il processore controlla la chiave di protezione del blocco che contiene l'elemento referenziato. Se la chiave di protezione del processo e del blocco richiesto coincidono, il processo può accedere al segmento. Se il processo tenta di accedere ad un blocco con una chiave di protezione differente, il sistema hardware impedisce l'accesso in memoria e passa il controllo al kernel, attraverso l'eccezione di protezione del segmento.
Il SO può eseguire un controllo di protezione ulteriore specificando in che modo un segmento sia accessibile e da quali processi. Questo controllo è compiuto assegnando ad ogni processo alcuni diritti d'accesso ai segmenti. Se un processo ha accesso in lettura a un segmento, può leggere dati a qualsiasi indirizzo in quel segmento. Se ha accesso in scrittura, il processo può modificare qualsiasi contenuto del segmento e può aggiungere ulteriori informazioni. Un processo con accesso in esecuzione su un segmento può passare il controllo del programma alle istruzioni contenute nel segmento per essere eseguite da un processore. L'accesso in esecuzione è normalmente vietato ad un segmento dati. Un processo con accesso in aggiunta ad un segmento può scrivere informazioni addizionali alla fine del segmento, ma non modificare le informazioni esistenti.
Sia consentendo sia vietando ognuno dei quattro tipi di accesso, è possibile creare 16 diverse modalità di controllo degli accessi, alcune delle quali sono interessanti, altre invece non hanno senso.

- Modalità 0: non è permesso nessun tipo di accesso al segmento, modalità utile per la sicurezza.
- Modalità 1: un processo ha solo l'accesso in esecuzione, modalità utile quando un processo può eseguire le istruzioni del segmento, ma non può copiarle o modificarle.
- Modalità 2 e 3: queste modalità non sono utili, non ha senso dare il diritto di modificare un segmento se non si ha il diritto di leggerlo.
- Modalità 4: garantisce al processo l'accesso in sola lettura al segmento, modalità utile per l'accesso ai dati non modificabili.
- Modalità 5: utile per il codice rientrante, garantisce al processo l'accesso in lettura/esecuzione al segmento; un processo può farsi la propria copia del segmento che in seguito potrà modificare.

- Modalità 6: permette l’accesso in lettura/scrittura al segmento, è utile per i segmenti che contengono dati che possono essere letti o scritti dai processi, ma che devono essere protetti da esecuzioni accidentali, poiché il segmento non contiene istruzioni.
- Modalità 7: consente un accesso senza restrizioni al segmento, modalità utile per permettere al processo un accesso completo al proprio segmento e per dare lo stato di massima fiducia nell’accesso agli altri segmenti del processo.

Sistemi con segmentazione/paginazione

A partire dai sistemi costruiti a metà degli anni ‘60 molti sistemi sono stati progettati combinando la paginazione e la segmentazione. In un sistema con paginazione e segmentazione, i segmenti occupano solitamente più pagine. Le pagine di un segmento non sono necessariamente tutte in memoria principale e le pagine virtuali, che sono contigue nella memoria virtuale, non necessitano di esserlo anche nella memoria principale.
Con questo schema un indirizzo virtuale è realizzato come una tripla ordinata v=(s,p,d) dove s è il numero del segmento, p il numero della pagina all’interno del segmento e d è lo spiazzamento all’interno della pagina dove è posizionato l’elemento richiesto.

Traduzione dinamica degli indirizzi in un sistema con segmentazione/paginazione

Un processo richiede un indirizzo virtuale v=(s,p,d). La pagina richiesta più recentemente ha la riga della tabella nella memoria associativa, la TLB. Il meccanismo di traduzione esegue una ricerca associativa per localizzare (s,p). Se la TLB contiene (s,p), la ricerca restituisce p’, il page frame in cui risiede la pagina p. Questo valore è concatenato con lo spiazzamento d per formare l’indirizzo di memoria reale r.
Se la TLB non contiene la riga per (s,p), il processore deve eseguire una mappatura completamente diretta. L’indirizzo base della tabella dei segmenti b viene sommato al numero del segmento s per formare l’indirizzo b+s. Questo indirizzo corrisponde alla locazione fisica in memoria della riga della tabella dei segmenti. Questa riga indica l’indirizzo base s’ della tabella delle pagine in memoria principale per il segmento s. Il processore aggiunge il numero di pagina p all’indirizzo base s’ per formare l’indirizzo della riga della tabella delle pagine relativa alla pagina p del segmento s. Questa riga indica che p’ è il numero di page frame corrispondente alla pagina virtuale p. Il numero di page frame p’ viene concatenato con lo spiazzamento d per formare l’indirizzo reale r. La traduzione viene caricata nella TLB. La traduzione può richiedere passi extra o fallire. La ricerca nella tabella dei segmenti può indicare che il segmento s non è in memoria principale, generando un missing-segment fault (fallimento di accesso al segmento) che il SO risolve localizzando il segmento sul dispositivo secondario, creando una tabella delle pagine per il segmento e caricando la pagina opportuna in memoria principale. Se il segmento è in memoria principale, la tabella delle pagine, può indicare che la pagina richiesta non è in memoria, avviando un page fault. Il controllo passa quindi al SO che localizza la pagina sul dispositivo secondario e la carica in memoria principale. È possibile che un processo faccia riferimento ad un indirizzo virtuale che superi il limite del segmento, generando quindi un’eccezione di segment-overflow (superamento dell’area del segmento). Se i bit di protezione del segmento indicano che l’operazione richiesta non è permessa, viene generata un’eccezione di protezione del segmento.
Il SO deve gestire tutte queste possibilità.
Il vantaggio di mantenere tutte le tabelle in memoria principale è che la traduzione procede più velocemente. Tuttavia, maggiore è il numero delle tabelle che il sistema deve mantenere in memoria principale, minore è il numero di processi che può supportare, con conseguente diminuzione della produttività.

Condivisione e protezione nei sistemi con segmentazione/paginazione

I sistemi con segmentazione/paginazione della memoria virtuale traggono vantaggio dalla semplicità architetturale della paginazione e dalla capacità di controllo degli accessi della segmentazione. In questi sistemi i benefici della condivisione dei segmenti diventano importanti. Due processi condividono memoria quando ciascuno di loro ha una riga della tabella dei segmenti che punta alla stessa tabella delle pagine.
La condivisione, sia nei sistemi paginati sia in quelli con segmentazione o con segmentazione/paginazione, richiede una gestione attenta da parte del SO

Gestione della memoria virtuale -capitolo 11-

Località

È il concetto che sta alla base di molte strategie di gestione, essa si manifesta sia nel tempo, sia nello spazio. La località temporale è la località nel tempo mentre la località spaziale implica che elementi vicini tendano ad essere simili.
È una proprietà empirica (osservata) piuttosto che teorica, non dà nessuna garanzia, ma un'ottima approssimazione. Nella paginazione si osserva che i processi tendono a favorire un certo sottoinsieme delle loro pagine e che queste pagine tendono a essere vicine fra loro nello spazio di indirizzamento virtuale.

Paginazione a richiesta

La politica di fetch più semplice realizzata nei sistemi con memoria virtuale è la paginazione a richiesta. Quando un processo inizia la sua esecuzione, il sistema carica una pagina in memoria solo quando il processo fa esplicitamente riferimento a quella pagina.
La paginazione a richiesta garantisce che il sistema porti in memoria principale solo le pagine necessarie al processo, permettendo potenzialmente ad un numero maggiore di processi di occupare la memoria principale, infatti lo spazio non è sprecato da pagine senza riferimenti.
La paginazione a richiesta non è esente da problemi: un processo deve accumulare le pagine una alla volta, mentre fa riferimento ad una nuova pagina, deve aspettare che il sistema trasferisca le pagine in memoria principale. Se il processo ha già molte pagine in memoria, il costo dovuto al tempo d'attesa può essere particolarmente alto, poiché una grande porzione di memoria principale è occupata da un processo che non può procedere nella sua esecuzione. Questo fattore spesso influenza il valore del prodotto spazio-tempo del processo. Il prodotto spazio-tempo indica non solo quanto tempo il processo passi in attesa, ma anche quanta memoria non sia utilizzabile durante l'attesa.

Paginazione a previsione

Un modo per ridurre questi tempi di attesa è evitare i ritardi della paginazione a richiesta. Nella paginazione a previsione (detta anche prefetching e prepaginazione), il SO prova a prevedere quali pagine serviranno ad un processo e le carica anticipatamente quando è disponibile lo spazio in memoria. I criteri fondamentali per determinare il successo della prepaginazione sono:

- allocazione della prepaginazione, la quantità di memoria allocata lla prepaginazione
- numero di pagine caricate anticipatamente ogni volta
- politica, ossia l'euristica che determina quali pagine vengono caricate in anticipo, per esempio quelle previste dalla località spaziale o dalla località temporale.

Le strategie a previsione sono spesso combinate con quelle a richiesta, il sistema carica con anticipo alcune pagine del processo quando viene riferita una pagina non presente. Questa strategia sfrutta la località spaziale.

Quando un processo genera un page fault, il sistema carica nello spazio di indirizzamento virtuale la pagina richiesta e alcune pagine vicine che sono contigue. Il tempo richiesto per caricare più pagine dal dispositivo secondario può essere significativamente maggiore rispetto a quello per una sola pagina.

Sostituzione delle pagine

Nei sistemi con memoria virtuale paginata può capitare che tutti i page frame siano già occupati e che un processo faccia riferimento ad una pagina residente. In questo caso, il sistema deve non solo caricare una nuova pagina dal dispositivo secondario, ma prima decidere quale pagina in memoria spostare, cioè rimuovere o sovrascrivere, per far spazio alla pagina entrante.
Ricordiamo che un page fault avviene quando un processo fa riferimento ad una pagina non residente. In questo caso, il sistema di gestione della memoria deve localizzare la pagina referenziata nel dispositivo secondario, caricarla nella memoria principale e aggiornare opportunamente la riga della tabella delle pagine.
Se la pagina scelta per la sostituzione non è stata modificata dall'ultimo caricamento dal disco, la nuova pagina può semplicemente sovrascriverla. Se la pagina è stata modificata, deve prima essere scritta sul dispositivo secondario per conservare il suo contenuto. Il bit di modifica o dirty bit nella riga della tabella delle pagine è impostato a 1 se la pagina è stata modificata, altrimenti a 0.
La scrittura (flush) di una pagina modificata sul disco, che richiede un'operazione di I/O sul disco, se viene eseguita in fase di sostituzione aumenta il tempo d'attesa di una pagina. Alcuni SO effettuano periodicamente il flush delle pagine dirty su disco per aumentare la probabilità di poter eseguire una sostituzione senza dover scrivere prima le pagine modificate sul disco.
Poiché questo fluah periodico è asincrono rispetto all'esecuzione dei processi, il sistema introduce un piccolo sovraccarico.
Quando si valuta una strategia di sostituzione, spesso la si mette a confronto con la cosiddetta strategia di sostituzione ottima delle pagine (Optimal Page Replacement, OPT), secondo cui, per ottenere ottime prestazioni, bisogna sostituire quelle pagine che non saranno più richieste in futuro. Si aumentano così le prestazioni minimizzando il numero di page fault. Questa strategia si dimostra essere ottimale, ma non realizzabile, poiché non si può prevedere con precisione il comportamento dei processi. La strategia viene utilizzata invece come riferimento per confrontare strategie realizzabili.

Strategie di sostituzione delle pagine

Sostituzione casuale delle pagine

La sostituzione casuale delle pagine (Random Page Replacement, RAND) è una strategia con un sovraccarico piccolo e una realizzazione semplice. In questa strategia ogni pagina in memoria ha la stessa probabilità di essere selezionata per la sostituzione. Un problema della sostituzione RAND è che può scegliere accidentalmente come pagina da sostituire la pagina che prossimamente verrà referenziata che è, ovviamente, la peggior pagina da rimuovere. Un vantaggio di questo tipo di sostituzione è che le decisioni sono veloci ed eque.

Sostituzione delle pagine First-In-First-Out (FIFO)

Nella strategia di sostituzione FIFO si sostituiscono le pagine in memoria da più tempo. Nella sostituzione FIFO il sistema tiene traccia dell'ordine in cui le pagine entrano in memoria. Quando una pagina deve essere sostituita, la strategia sceglie quella che è in memoria da più tempo. Sfortunatamente questa strategia può sostituire le pagine pesantemente utilizzate.

Nella sostituzione FIFO, certi modelli di riferimento delle pagine causano un aumento del numero di page fault quando aumenta il numero di page frame allocati al processo. Questo fenomeno è detto anomalia FIFO o anomalia di Belady.

Sostituzione Least-Recently-Used (LRU)

La strategia di sostituzione della pagina LRU si fonda su questa euristica: il comportamento del passato recente di un processo è un buon indicatore del suo comportamento futuro (località temporale). Quando il sistema deve sostituire una pagina, la strategia LRU sostituisce quella pagina che è da più tempo in memoria senza essere referenziata. Nonostante le prestazioni della sostituzione LRU siano migliori di quella FIFO, i vantaggi devono fare i conti con il sovraccarico del sistema. La strategia LRU è realizzabile con una struttura a lista che contiene un elemento per ogni page frame occupato.
Ogni volta che un page frame viene referenziato, il sistema posiziona l'elemento della pagina in testa alla lista, indicando che quella è la pagina referenziata più recentemente. Gli elementi più vecchi si spostano verso la coda della lista. Quando una pagina deve essere sostituita per far spazio ad una pagina entrante, il sistema sostituisce l'elemento in coda alla lista. Il sistema libera il page frame corrispondente, eventualmente scrivendo la pagina modificata sul dispositivo secondario, posiziona la pagina entrante nel page frame e muove l'elemento di quella pagina in testa alla lista, poiché la pagina è ora quella referenziata più recentemente.
Questo schema realizza fedelmente la strategia LRU, comporta però un sovraccarico notevole, poiché il sistema deve aggiornare la lista ogni volta che una pagina viene referenziata.

Sostituzione Least-Frequently-Used (LFU)

La strategia di sostituzione LFU si basa sull'intensità di utilizzo di ciascuna pagina. Il sistema sostituisce le pagine usate meno frequentemente. L'idea di base è la seguente: seuna pagina non è intensivamente richiamata, non lo sarà neanche in futuro. La sostituzione LFU è realizzabile usando un contatore che si aggiorna ogni volta che la pagina corrispondente viene referenziata, ma con un notevole sovraccarico.
Il problema è che una pagina appena portata in memoria risulterà usata meno frequentemente di conseguenza verrebbe rimossa da questo tipo di strategia, quando in realtà la probabilità di usarla immediatamente è alta.

Sostituzione Not-Used-Recently (NUR)

La strategia di sostituzione della pagina NUR è un diffuso schema di approssimazione della strategia LRU. La sostituzione NUR si basa sull'idea che una pagina non usata recentemente non sarà probabilmente richiesta nemmeno nel prossimo futuro. La strategia NUR è realizzata usando due bit hardware nella riga della tabella delle pagine:

- bit di referenza (referenced bit): posto a 0 se la pagina non è stata referenziata e posto a 1 altrimenti
- bit di modifica (modified bit): posto a 0 se la pagina non è stata modificata e posto a 1 altrimenti

Il bit di referenza viene talvolta chiamato bit di accesso (accessed bit), il sistema imposta a 0 i bit di referenza. Quando un processo fa riferimento a una pagina particolare, il sistema imposta a 1 il bit di referenza di quella pagina. I bit di modifica di tutte le pagine sono anch'essi posti a 0.
Quando una pagina viene modificata, il sistema pone a 1 il bit di modifica della pagina.
Quando il sistema deve sostituire una pagina, la strategia NUR tenta prima di trovare una pagina che non sia stata referenziata, se non la trova il sistema deve sostituirne una referenziata. In questo caso, la strategia NUR controlla i bit di modifica per determinare se la pagina è stata modificata. Se la pagina non è stata modificata, il sistema la sceglie per la sostituzione, altrimenti viene scelta una

pagina modificata. Teniamo a mente che la sostituzione di una pagina modificata comporta un ritardo notevole per le operazioni di I/O addizionali poiché, per preservarne i contenuti, la pagina modificata viene scritta sul dispositivo secondario.
La memoria principale di un sistema multiutente verrà referenziata attivamente e molte o tutte le pagine avranno il bit di referenza posto a 1. In questo caso, la strategia NUR non riesce ad identificare la pagina migliore da sostituire. Per evitare il problema, una tecnica ampiamente utilizzata è quella di impostare periodicamente a 0 tutti i bit di referenza. Questa tecnica rende però vulnerabili alla sostituzione pagine attive, anche solo per un breve momento dopo l'azzeramento dei bit di referenza, dopodiché le pagine attive avranno quasi subito i bit di referenza posti a 1.
Nello schema NUR, le pagine sono classificabili in quattro gruppi. Le pagine del gruppo con il numero più basso sono le prime ad essere sostituite, mentre quelle nel gruppo col numero più alto sono le ultime. La selezione delle pagine all'interno di un gruppo avviene casualmente. Si noti che il gruppo due sembra descrivere una situazione non realistica, ovvero una pagina è stata modificata ma non referenziata. Questa situazione capita perché l'inizializzazione periodica riguarda i bit di referenza, non quelli di modifica.

Varianti della strategia FIFO: seconda chance e sostituzione a orologio

Il lato debole della strategia FIFO è che può scegliere di sostituire una pagina usata pesantemente che risiede in memoria da molto tempo. Questa eventualità è evitabile adottando la strategia FIFO con un bit di referenza per ciascuna pagina e sostituendo solo quelle pagine con il bit di referenza pari a zero.
La variante data dalla strategia di sostituzione con seconda chance esamina il bit di referenza della pagina più vecchia; se il bit è nullo, la strategia seleziona immediatamente quella pagina per la sostituzione. Se il bit di referenza è attivo, l'algoritmo azzera il bit e sposta la pagina alla fine della coda FIFO. Questa pagina viene considerata come se fosse un nuovo arrivo. Mentre il tempo avanza, la pagina si sposta gradualmente in testa alla coda. Quando la raggiunge, verrà scelta per la sostituzione solo se il bit di referenza sarà ancora nullo.
Le pagine attive verranno selezionate per ritornare in coda alla lista, poiché il loro bit di referenza sarà attivo e quindi rimarranno in memoria principale. Una pagina modificata deve essere scritta sul dispositivo secondario prima che il sistema la possa sostituire; così quando il suo bit di referenza è nullo, la pagine rimane "temporaneamente insostituibile" fino a quando il sistema non completa il trasferimento. Se un processo fa riferimento a questa pagina prima che il flush sia terminato, l'evento viene catturato, risparmiando una costosa operazione di caricamento della pagina dal dispositivo secondario.

La strategia di sostituzione ad orologio, che produce essenzialmente lo stesso risultato dell'algoritmo della seconda chance, sistema le pagine in una lista circolare invece che in una lista lineare. Ogni volta che capita un page fault, un puntatore si muove introno alla lista come la lancetta di un orologio. Quando il bit di referenza di una pagina viene annullato, il puntatore viene spostato verso l'elemento successivo della lista, simulando il movimento in fondo alla coda FIFO.
L'algoritmo dell'orologio posiziona i nuovi arrivi nella prima pagina che incontra con il bit di referenza nullo.

Sostituzione far delle pagine

La strategia di sostituzione far (legata cioè alla lontananza delle pagine) usa dei grafi basati su questi modelli prevedibili per prendere le decisioni. Questa strategia si è dimostrata funzionare ad un livello quasi ottimo, ma la sua complessità di realizzazione implica un significativo sovraccarico durante l'esecuzione.

La strategia far crea un grafo degli accessi che caratterizza il modello dei riferimenti di un processo. Ogni vertice del grafo rappresenta una pagina del processo. Un arco dal vertice v al vertice w significa che il processo può riferire la pagina w dopo che ha fatto riferimento alla pagina v.
Il grafo, che può diventare piuttosto complesso, descrive come un processo riferisce le pagine durante la sua esecuzione. Il grafo degli accessi può essere creato analizzando un programma compilato per determinare quali pagine siano accessibili da ciascuna istruzione in ogni pagina, il che può richiedere un notevole tempo di esecuzione. Si noti che molti studi sulla strategia far ipotizzano che il grafo venga costruito prima dell'esecuzione del processo, mentre la costruzione del grafo durante l'esecuzione è in fase di indagine.

Far marca inizialmente tutti i vertici nel grafo degli accessi come non referenziati; quando il processo accede ad una pagina, l'algoritmo marca come referenziato il vertice che corrisponde a quella pagina. Quando l'algoritmo deve selezionare una pagina per la sostituzione, viene scelta la pagina non referenziata che è la più lontana da qualsiasi pagina referenziata nel grafo degli accessi. L'idea alla base è che la pagina non referenziata più lontana da quelle referenziate sarà probabilmente riferita in un futuro più lontano. Se il grafo non contiene un vertice non referenziato, la fase corrente è completa e la strategia marca tutti i vertici come non referenziati iniziando una nuova fase. A questo punto, l'algoritmo sostituisce la pagina più lontana da quella riferita più recentemente.
La teoria dei grafi fornisce gli algoritmi per la costruzione e la ricerca nel grafo nella strategia far; tuttavia questa strategia non è implementata nei sistemi reali, a causa della sua complessità e del sovraccarico introdotto nel tempo di esecuzione.

Modello del working set

La teoria del working set (insieme di lavoro) sul comportamento dei programmi si basa sul determinare quale sia il sottoinsieme di pagine favorito e su come mantenerlo in memoria principale per ottenere le prestazioni migliori.
Il principio della località e il comportamento dei processi si basano sulla teoria di Denning del working set. Questa teoria afferma che per eseguire un programma efficientemente, il sistema deve mantenere in memoria il sottoinsieme di pagine favorite, cioè il working set; altrimenti, il sistema sarà impegnato in un'attività eccessiva di paginazione causando una bassa utilizzazione del processore, fenomeno detto thrashing, poiché il processo richiede ripetutamente il caricamento delle stesse pagine dal dispositivo secondario. Un metodo per evitare il thrashing consiste nell'assegnare a ciascun processo un numero sufficiente di page frame per contenere metà del proprio spazio di indirizzamento virtuale.
Una politica di gestione della memoria con working set cerca di mantenere in memoria solo quelle pagine che compongono il working set corrente del processo. La decisione di aggiungere un nuovo processo all'insieme dei processi attivi, cioè di aumentare il livello di programmazione, si basa sulla possibilità che il sistema disponga di memoria principale a sufficiente ad ospitare il working set del nuovo processo.
Il working set W(t,w) del processo è definito come l'insieme di pagine riferite dal processo nell'intervallo di tempo di proceso che va da (t-w) a t.
Il "vero" working set di un processo è semplicemente l'insieme di pagine che deve risiedere in memoria per un'esecuzione efficiente del processo. Durante l'esecuzione del processo il working set cambia; talvolta, le pagine vengono aggiunte o rimosse, e avvengono cambiamenti notevoli quando il processo entra in una nuova fase, cioè l'esecuzione richiede working set differenti. Quindi, qualsiasi ipotesi circa la dimensione ed il contenuto del working set iniziale non è necessariamente applicabile ai working set successivi accumulati dal processo.

Sostituzione Page-Fault-Frequency (PFF)

Una misura della bona qualità dell'esecuzione di un processo in un sistema paginato è il suo tasso di page fault. Un processo che genera costantemente page fault ha un utilizzo del processo molto basso, poiché i page frame allocati non contengono tutto il suo working set. Un processo che non solleva mai page fault, tuttavia, significa che dispone di troppi page frame e quindi limita il grado di multiprogrammazione del sistema. Idealmente, un processo dovrebbe operare in un qualche punto compreso fra questi due estremi. L'algoritmo di frequenza dei page fault o PFF adatta l'insieme delle pagine residenti cioè quelle pagine che sono al momento in memoria, basandosi sulla frequenza con cui il processo solleva dei page fault. Alternativamente, PFF può adattare l'insieme delle pagine residenti basandosi sul tempo che intercorre tra due page fault, detto tempo di interfault.
La sostituzione PFF comporta un sovraccarico inferiore rispetto alla sostituzione basata sul working set. Con la strategia PFF, quando un processo fa una richiesta che comporta un page fault, la strategia calcola il tempo trascorso dall'ultimo page fault. Se questo intervallo di tempo è più grande di un certo valore limite, il sistema rilascia tutte le pagine non referenziate nell'intervallo. Se l'intervallo è inferiore ad un altro valore limite, la pagina entra a far parte dell'insieme delle pagine residenti.
Se un processo sta passando ad un working set più grande, solleverà frequentemente dei page fault e PFF allocherà un maggior numero di page frame. Il fattore chiave dell'efficienza di questa sostituzione è assegnare valori appropriati al limite inferiore e superiore dell'intervallo di tempo.

Rilascio delle pagine

Nei sistemi con working set, un processo indica quali sono le pagine che vuole usare referenziandole esplicitamente. Le pagine non richieste più dal processo dovrebbero essere rimosse dal suo working set. Nei sistemi esistenti, le pagine necessarie spesso rimangono in memoria fino a quando la strategia di gestione individua che il processo non ne ha più bisogno. Una strategia alternativa prevede che il processo, per liberare un page frame di cui non ha più bisogno, una strategia alternativa prevede che il processo, per liberare un page frame di cui non hai più bisogno, effettui un rilascio volontario della pagina. Questo eliminerebbe il ritardo dovuto al tempo necessario ad accorgersi dei cambiamenti nel working set.
Il rilascio volontario delle pagine può velocizzare l'esecuzione dei programmi per l'intero sistema.

Dimensione delle pagine

Molti dei risultati citati in letteratura, sia teorici sia empirici, puntano sulla necessità di pagine piccole. Poiché sia la memoria sia la dimensione dei programmi sono rapidamente aumentate, vengono privilegiate le pagine grandi.

- Una pagina grande aumenta l'intervallo di memoria che viene indirizzato con una riga della TLB. Questo accresce la probabilità di TLB hit, migliorando le prestazioni della traduzione dinamica degli indirizzi.
- Una pagina grande può ridurre il numero delle costose operazioni di I/O che trasferiscono informazioni tra memoria principale e dispositivo secondario. Un sistema che trasferisce queste informazioni usando pagine piccole può richiedere diverse operazione di I/O separate, che incrementerebbero il prodotto spazio-tempo del processo.
- I processi tendono ad esibire la località dei riferimenti rispetto a piccole porzioni del loro spazio di indirizzamento; in questo modo le pagine più piccole aiuterebbero un processo a stabilire un working set più piccolo e snello, lasciando disponibile agli altri processi una maggiore quantità di memoria.
- Le pagine piccole portano a gestire un gran numero di pagine e page frame a cui corrisponde una più grande tabella delle pagine. Le pagine grandi riducono la frammentazione delle

tabelle diminuendo il numero delle righe della tabella, al prezzo di un aumento della frammentazione interna.
- Poiché le procedure ed i dati riempiono raramente un numero intero di pagine, in un sistea con segmentazione/paginazione combinate si può osservare la frammentazione interna.

Comportamento dei programmi con la paginazione

Il numero di fault incontrato da un processo dipende dalla dimensione e dal comportamento dei processi del sistema. Se hanno working set piccoli, il numero di page fault in esecuzione tende ad aumentare con l'aumento della dimensione delle pagine. All'aumentare della dimensione delle pagine, aumentano anche le procedure ed i dati portati in memoria che non saranno referenziati. Inoltre, poiché le pagine continuano ad aumentare, il sistema introduce una maggiore frammentazione interna.
Quindi, sarà referenziata una piccola percentuale di procedure e dati contenuti nella limitata memoria principale. Un processo con un working set grande richiede un elevato numero di piccoli page frame, che possono comportare un grande numero di page fault ad ogni esecuzione del processo. Se il working set contiene pagine contigue nello spazio di indirizzamento virtuale, il numero dei page fault tende a diminuire con l'aumentare della dimensione delle pagine.

Strategie di sostituzione globali o località

Quando si realizza un sistema con memoria virtuale, il progettista del SO deve decidere se la strategia di sostituzione debba essere applicata a tutti i processi visti come un'unica entità, adottando una strategia globale, oppure se considerare ogni processo individualmente, adottando una strategia locale. Le strategie globali di sostituzione tendono ad ignorare le caratteristiche del comportamento di un singolo processo; le strategie locali di sostituzione consentono al sistema di adattare l'allocazione della memoria in relazione alla necessità del processo di migliorare le prestazioni.
La strategia di sostituzione globale della pagine usata meno recentemente o gLRU (global LRU) sostituisce la pagina usata meno recentemente nell'intero sistema. Quando seleziona la pagina da sostituire, questa semplice strategia non analizza il comportamento del singolo processo o la sua importanza.

Sostituzione delle pagine in Linux

Le pagine sono classificate in pagine attive e inattive. Per essere considerata attiva, una pagina deve essere stata referenziata recentemente. Un obiettivo della gestione della memoria è mantenere il working set al momento in uso all'interno delle pagine attive.
Linux usa una variante dell'algoritmo dell'orologio a imitazione della strategia LRU; il gestore della memoria usa due liste collegate: la lista attiva contiene le pagine attive, la lista inattiva quelle inattive. Le liste sono organizzate in modo tale che le pagine usate più recentemente siano vicine alla testa della lista attiva, mentre quelle usate meno recentemente siano prossime alla coda della lista inattiva.
Quando una pagina è portata in memoria la prima volta, viene posizionata nella lista inattiva ed è marcata come se fosse stata referenziata attivando il suo bit di referenza. Il gestore della memoria determina periodicamente se la pagina è stata referenziata successivamente, oppure durante il page fault. Se la pagina è attiva o inattiva ed il suo bit di referenza è zero, i bit viene attivato.
Analogamente all'algoritmo dell'orologio, questa tecnica assicura che le pagine referenziate recentemente non siano selezionate per la sostituzione.
Se la pagina è inattiva e viene referenziata una seconda volta, il gestore della memoria sposta la pagina in testa alla lista attiva azzerando il bit di referenza. Questo permette al kernel di determinare se una pagina è stata referenziata recentemente più di una volta. In questo caso, la pagina viene

spostata nella lista attiva così da non essere selezionata per la sostituzione. Per assicurare che la lista attiva contenga solo le pagine pesantemente referenziate, il gestore della memoria sposta periodicamente tutte le pagine della lista attiva non referenziate in testa alla lista inattiva.
Questo algoritmo è ripetuto fino a quando un numero specificato di pagine non è stato spostato dalla coda della lista attiva all'inizio della lista inattiva. Una pagina della lista inattiva rimarrà in memoria fino a quando non viene selezionata per la sostituzione. In ogni caso, finché una pagina è nella lista attiva, non è comunque selezionabile per la sostituzione.

Ottimizzazione delle prestazioni dei dischi -capitolo 12-

Per accedere ai dati, un dispositivo che trasporta corrente, detto testina di lettura e scrittura, sorvola il mezzo quando si muove. La testina legge i dati misurando come il mezzo magnetizzato modifichi la corrente generata, la scrittura dei dati avviene usando una corrente in grado di modificare lo stato di magnetizzazione del mezzo. La testina deve rimanere estremamente vicina alla superficie del mezzo, senza però toccarlo.

- 1951, introduzione di una memoria su nastro magnetico, sia riscrivibile sia persistente.
- 1957, IBM introdusse il primo lettore di dischi rigidi commerciale, il RAMAC (Random Access Method of Accounting and Control). I lettori di dischi rigidi sono ad accesso casuale (detto anche accesso diretto), poiché non sono limitati all'accesso sequenziale dei dati.

Col passare degli anni i dischi rigidi hanno migliorato capacità e prestazioni, mentre i costi si sono ridotti. A causa dei vincoli meccanici, la velocità dei dischi rigidi è aumentata più lentamente rispetto alle loro capacità.

Caratteristiche dei dischi con testina mobile

I dischi con testina mobile hanno una velocità d'accesso variabile, dipendente dalla posizione relativa della testina di lettura e scrittura dei dati richiesti. I dati sono memorizzati su una serie di dischi magnetici, detti piatti, connessi ad un rotore che ruota ad alta velocità. I dati su ciascuna delle superficie del disco sono accessibili attraverso la testina di lettura e scrittura, vicinissima alla superficie. Una testina può accedere ai dati immediatamente sottostanti o sovrastanti; quindi, prima di poter accedere ai dati, la porzione di superficie del disco i cui dati devono essere letti (o scritti) deve ruotare fino a quando non si trova al di sotto (o al di sopra) della testina di lettura e scrittura. Il tempo necessario per far ruotare ii dati dalla posizione attuale all'inizio della testina è detto tempo di latenza rotazionale.
Durante la rotazione dei piatti, ogni testina sorvola una traccia circolare di dati sulla superficie del disco. Ciascuna testina è posizionata alla fine del braccio del disco, collegato ad un attuatore (detto anche boom o braccio mobile di assemblaggio). Il braccio del disco si sposta parallelamente alla superficie del disco; quando muove la testina in una nuova posizione diventa accessibile a un nuovo insieme verticale di tracce circolari, dette cilindri. Il processo di spostamento del braccio su un nuovo cilindro è detto operazione di seek (ricerca). Per localizzare piccole unità di dati, i dischi suddividono le tracce in diversi settori.
Per accedere ad un particolare settore di dati il braccio deve spostarsi sul cilindro appropriato, eseguendo cioè un'operazione di seek; il tempo impiegato dalla testina per spostarsi dall'attuale cilindro a quello contenente i dati è detto tempo di seek. Quindi, la porzione di disco su cui i dati sono memorizzati deve ruotare fino a quando non si trovi immediatamente sopra (o sotto) la testina. I dati, di dimensione arbitraria, per essere letti devono quindi ruotare rispetto alla testina; operazione chiamata tempo di trasmissione. Poiché ciascuna di queste operazioni richiede movimenti meccanici, il tempo totale d'accesso al disco aumenta ad una frazione di secondo, tempo in cui il processore è in grado di eseguire decine o anche centinaia di milioni di istruzioni.

Neccesità della pianificazione o scheduling del disco

Molti processi possono generare richieste di lettura e scrittura di dati sul disco contemporaneamente. Poiché questi processi effettuano le richieste più velocemente rispetto alla possibilità del disco di servirle, le richieste vengono accumulate in code o liste d'attesa. I primi sistemi di elaborazione utilizzavano il principio First-Come-First-Served, il quale risulta essere un metodo equo per gestire il servizio ma quando quando il tasso di richiesta, cioè il carico, diventa pesante, i tempi di attesa si allungano . FCFS esibisce sequenze di seek casuali, in cui le richieste successive possono causare seek lunghi dal cilindro più interno verso quello più esterno. Questo processo detto scheduling del disco o pianificazione del disco, può migliorare significativamente le prestazioni (throughput).
Per determinare il modo più efficiente di soddisfarle, lo scheduling del disco richiede un esame attento delle richieste pendenti. Lo scheduler del disco esamina la relazione tra le posizioni delle richieste in attesa e riordina la coda in modo tale che l'evasione delle richieste comporti il minimo movimento meccanico.
Poiché FCFS non riordina le richieste, è considerato da molti l'algoritmo di scheduling del disco più semplice. I due tipi di scheduling più comuni sono l'ottimizzazione del tempi di seek e l'ottimizzazione rotazionale. Il tempo di seek tende a superare il tempo di latenza, molti algoritmi di scheduling si concentrano sulla minimizzazione del tempo di seek di un insieme di richieste. Poiché la differenza tra la latenza rotazionale ed il tempo di seek va diminuendo, la minimizzazione della latenza rotazionale di un insieme di richieste può anch'essa contribuire a migliorare le prestazioni del sistema, soprattutto se il carico è alto.

Strategie di scheduling del disco

Gran parte delle strategie sono valutate con i criteri seguenti:

- Throughput: numero di richieste servite nell'unità di tempo.
- Tempo medio di risposta: tempo medio di attesa per il servizio di una richiesta.
- Varianza del tempo di risposta: misura della prevedibilità della risposta. Ogni richiesta dovrebbe essere evasa entro un periodo di tempo accettabile, cioè la strategia dovrebbe evitare posticipazioni indefinite.

Una politica di scheduling dovrebbe chiaramente tentare di massimizzare le prestazioni e minimizzare il tempo medio di risposta.
La varianza misura come sono servite le singole richieste relativamente alle prestazioni medie del sistema. Più piccola è la varianza, più alta è la probabilità che la gran parte delle richieste d'accesso al disco siano esaudite in un tempo di attesa simile. La varianza è quindi concepibile come la misura dell'equità e della prevedibilità. Per evitare tempi di servizio imprevedibili, è quindi opportuna una politica di scheduling che minimizzi la varianza, o almeno che rimanga al di sotto di un livello ragionevole.

Scheduling FCFS

FCFS usa una coda FIFO cosicché le richieste siano servite nell'ordine di arrivo. Questa tecnica di scheduling è equa nel senso che la posizione di una richiesta in coda non è influenzata da una nuova richiesta. Ciò assicura che nessuna richiesta sia posticipabile indefinitamente, ma significa anche che FCFS può eseguire delle operazioni di seek lunghe per servire la richiesta successiva, anche se un'altra richiesta in coda è più vicina e potrebbe essere soddisfatta più velocemente. Nonostante questa tecnica introduca un sovraccarico basso durante la sua esecuzione, ottiene un throughput altrettanto ridotto dovuto alla lunghezza delle operazioni di seek.
FCFS introduce un modello di seek casuale pichè ignora la relazione tra le posizioni delle richieste pendenti. Ciò è accettabile quando il carico sul disco è leggero. Tuttavia, crescendo il carico, FCFS tende a saturare, cioè a sovraccaricare il dispositivo ed il tempo di risposta diventa alto.

Scheduling SSTF

L'algoritmo SSTF (Shortest-Seek-Time-First) serve la richiesta più vicina al cilindro corrispondente all'attuale posizione della testina, sfruttando il tempo di seek più breve, anche se non serve la prima richiesta della coda. SSTF non assicura equità e può causare posticipazioni indefinite, poiché la sua sequenza di seek tende ad essere altamente localizzata, inducendo un tempo di risposta inaccettabile per le richieste delle tracce più interne o più esterne.
Riducendo il tempo medio di seek, SSTF ottiene un throughput maggiore rispetto a FCFS ed un tempo medio di risposta che tende ad essere inferiore per carichi moderati. Uno svantaggio significativo è l'elevata varianza del tempo di risposta, a causa della discriminazione delle tracce interne ed esterne. Se le nuove richieste tendono a concentrarsi presso i cilindri centrali, si può verificare starvation, ovvero la mancata evasione delle richieste lontane dalla testina. SSTF è accettabile per i sistemi batch ma non è adatta ai sistemi interattivi, in cui bisogna assicurare un tempo di risposta rapido e prevedibile.

Scheduling SCAN

SCAN sceglie la richiesta che necessita della più breve distanza di seek in una data direzione preferenziale. SCAN non cambia a direzione preferenziale fino a quando non raggiunge il cilindro più esterno o quello più interno. In questi termini, viene detto anche algoritmo dell'ascensore, poiché come un ascensore continua a servire le richieste in una direzione prima di invertire il senso. Si comporta analogamente a SSTF in termini di prestazioni elevate e di tempo di risposta ma offre una varianza inferiore rispetto ad esso.
Poiché le richieste che arrivano possono essere servite prima di quelle in attesa, sia SSTF sia SCAN sono soggette a posticipazione indefinita.

Scheduling C-SCAN

Nello SCAN circolare il braccio si sposta dal cilindro esterno a quello interno, servendo le richieste in base al seek più breve. Quando il braccio ha completato il suo percorso verso il cilindro interno, salta a quello esterno senza servire alcuna richiesta, ricominciando il suo percorso verso il cilindro interno ed evadendo le richieste d'accesso al disco. C-SCAN mantiene un throughput elevato, mentre limita ulteriormente la varianza del tempo di risposta, evitando discriminazioni tra i cilindri mediani e quelli periferici. Come in SCAN, anche in C-SCAN le richieste che continuano ad arrivare allo stesso cilindro sono posticipabili indefinitamente, sebbene sia meno probabile rispetto a SCAN o SSTF.

Scheduling FSCAN e SCAN a n-passi

Le varianti FSCAN e SCAN a n-passi eliminano la possibilità di richieste di posticipazione indefinita. FSCAN usa la strategia SCAN per soddisfare solo quelle richieste in attesa all'inizio di un attraversamento del braccio del disco (la F sta per freezing, cioè congelamento della coda delle richieste in un certo istante). Le richieste che arrivano durante un attraversamento vengono raggruppate ed ordinate per essere servite in modo ottimale durante l'attraversamento di ritorno.
Lo SCAN a n-passi serve le prime n richieste nella cosa per mezzo della strategia SCAN. Quando l'attraversamento è completo, vengono soddisfatte le n richieste successive. Le richieste in arrivo sono posizionate alla fine della coda delle richieste. Lo SCAN a n-passi è calibrabile variando il valore di n. Quando n=1, lo SCAN a n-passi degenera in FCFS; con n che tende all'infinito degenera in SCAN.
FSCAN e SCAN a n-passi offrono buone prestazioni, un throughput elevato e un tempo di risposta medio basso. Poiché prevengono la posticipazione indefinita, la loro caratteristica distintiva è una varianza del tempo di risposta più bassa rispetto a SSTF e SCAN.

Scheduling LOOK e C-LOOK

La variante LOOK di SCAN guarda avanti (look ahead) fino al termine dell'attuale attraversamento per determinare la richiesta successiva da servire; se non ci sono più richieste nella direzione attuale, LOOK cambia la direzione preferenziale ed inizia l'attraversamento successivo. In questo senso, è appropriato definire questa strategia algoritmo dell'ascensore, poiché un ascensore continua in una direzione fino a quando raggiunge l'ultima richiesta, per poi cambiare direzione. Questa strategia elimina i seek superflui effettuati dalle altre varianti della strategia SCAN.
La variante alla strategia LOOK, detta LOOK circolare usa la stessa tecnica di C-SCAN per la riduzione della discriminazione, rispetto alle richieste posizionate agli estremi della superficie del disco. Quando non ci sono più richieste nell'attuale attraversamento verso l'interno, la testina di lettura e scrittura si sposta verso le richieste vicine al cilindro più esterno, senza servirne alcuna durante il tragitto ed iniziando così un nuovo attraversamento.
La politica C-LOOK è caratterizzata da una varianza del tempo di risposta più bassa rispetto a LOOK ed un throughput piuttosto alto, nonostante sia generalmente inferiore a quello di LOOK.

Ottimizzazione rotazionale

I processi che accedono ai dati sequenzialmente tendono ad accedere a intere tracce di dati e quindi non si avvantaggiano granché dell'ottimizzazione rotazionale. Tuttavia, quando ci sono numerose richieste di piccole porzioni di dati, distribuite casualmente sulla superficie del disco, l'ottimizzazione rotazionale può migliorare le prestazioni significativamente.

Scheduling SLTF

Una volta che il braccio del disco raggiunge un certo cilindro, su diverse tracce di quel cilindro possono trovarsi molte richieste pendenti. La strategia SLTF (Shortest-Latency-Time-First) di scelta delle richieste con il minor tempo di latenza, esamina tutte queste richieste e serve per prima quella con il ritardo di rotazione minore. Questa strategia è quella teoricamente ottimale ed è relativamente semplice da realizzare. L'ottimizzazione rotazionale è conosciuta anche come accodamento dei settori: le richieste sono accodate a seconda della posizione del settore sul disco ed i settori più vicini sono serviti per primi.

Scheduling SPTF e SATF

La strategia SPTF (Shortes-Positioning-Time-First) sceglie le richieste con il minor tempo di posizionamento, cioè la somma del tempo di seek e del tempo di latenza rotazionale. Questa strategia ottiene un throughput elevato, un basso tempo medio di risposta e può anche posticipare indefinitamente le richieste ai cilindri più interni o esterni.
Una variante di SPTF è SATF (Shortest-Access-Time-First) di scelta delle richieste con il minor tempo d'accesso, cioè la somma del tempo di posizionamento più il tempo di trasmissione. SATF ottiene un throughput migliore di SPTF ma può accadere che le richieste di maggiori dimensioni siano rinviate indefinitamente da una serie di piccole richieste, e che le richieste ai cilindri interni o esterni siano rinviate indefinitamente dalle richieste ai cilindri mediani. Sia SPTF che SATF possono migliorare le prestazioni, realizzando il meccanismo di visione anticipata.
SPTF e SATF richiedono la conoscenza delle caratteristiche del disco, oltre al tempo di latenza, il tempo di seek traccia per traccia e le relative posizioni dei settori. Sfortunatamente, molti dei moderni dischi rigidi mostrano una geometria fuorviante.
In alcuni sistemi, quando vengono individuati i settori danneggiati, i settori alternativi sono assegnati al loro posto.

Queste funzionalità che migliorano l'integrità dei dati, come quelle dei settori alternativi, tendono a neutralizzare l'effetto del miglioramento delle prestazioni usando le strategie di scheduling, poiché il disco fornisce al SO informazioni incomplete o fuorvianti sulla posizione dei dati.
Una semplice strategia di miglioramento delle prestazioni è la riduzione della latenza rotazionale attraverso l'aumento della velocità di rotazione dei dischi. I progettisti hanno tuttavia incontrato problemi significativi nell'aumento dei giri al minuto (RPM) dei dischi rigidi. La rotazione rapida dei dischi consuma una grande quantità di energia, irradia più calore, produce più rumore e richiede meccanismi elettromeccanici più costosi per controllarla.

Considerazioni di sistema

Quando i dispositivi di memorizzazione secondaria si dimostrano essere il collo di bottiglia (bottleneck) del sistema, alcuni progettisti raccomandano l'aggiunta di dischi. Questo non sempre risolve il problema, poiché il collo di bottiglia potrebbe essere causato da carichi di grandi richieste su un numero relativamente piccolo di dischi. Quando si verifica questa situazione, lo scheduling del disco è utilizzabile come mezzo per migliorare le prestazioni ed eliminare il collo di bottiglia.

Lo scheduling del disco può non essere utile in un sistema di elaborazione batch con un basso grado di multiprogrammazione. Lo scheduling diventa più efficace quando la casualità e la multiprogrammazione aumentano, situazioni che comportano un incremento del carico del sistema e sequenze di richieste d'accesso al disco imprevedibili.
Analogamente, i sistemi di elaborazione al volo delle transizioni (Online Transaction Processing, OLTP), come i database e i server Web, ricevono in genere richieste di posizioni casuali contenenti piccole quntità di dati, come file HTML e tuple di databse. Alcune ricerche hanno dimostrato come gli algoritmi di scheduling del disco, quali C-LOOK e SATF (con visione anticipata), possano migliorare le prestazioni di questo tipo di ambiente.

Le distribuzioni non uniformi delle richieste sono diffuse in certe situazioni e le loro conseguenze vanno analizzate. La maggior parte dei riferimenti al disco sono sullo stesso cilindro del riferimento immediatamente precedente. Una causa comune dell'alta localizzazione della distribuzione non uniforme delle richieste è il grande uso di file sequenziali sui dischi dedicati. Quando un SO alloca lo spazio per i blocchi adiacenti di un file utente sequenziale, solitamente posiziona i blocchi adiacenti sulla stessa traccia. Quando una traccia è piena, i blocchi aggiuntivi sono posizionati sulle tracce adiacenti dello stesso cilindro; quando il cilindro è pieno, i blocchi aggiuntivi sono posizionati sui cilindri adiacenti. Le richieste dei blocchi adiacenti in un file sequenziale spesso non causano nessuna operazione di seek; possono però comportare operazioni di seek brevi, poiché si muovono verso i cilindri immediatamente adiacenti. Ovviamente, una politica di scheduling come quella di FCFS potrebbe essere adeguata per questa situazione; infatti, il sovraccarico introdotto dalle strategie di scheduling più complesse potrebbe peggiorare le prestazioni.

Le tecniche più sofisticate di organizzazione dei file possono causare una proliferazione di richieste con lunghi tempi di seek. In questi casi, il recupero dei dati può richiedere un riferimento all'indice master (principale), un riferimento all'indice dei cilindri e alla locazione del blocco corrente, comportando un notevole ritardo dovuto ai numeri tempi di seek. Poiché solitamente l'indice master e l'indice dei cilindri sono memorizzati sul disco, ma separatamente dall'area principale dei dati, queste operazioni di seek possono essere costose. Tali tecniche di organizzazione dei file sono vantaggiose per gli sviluppatori delle applicazioni, ma possono peggiorare le prestazioni.

Cache e buffer del disco

Molti sistemi mantengono una memoria buffer/cache del disco, ossia una regione della memoria principale che il SO riserva ai dati inerenti al disco. In un certo contesto, la memoria riservata agisce come cache, consentendo ai processi l'accesso veloce ai dati che dovrebbero essere altrimenti recuperati dal disco. La memoria riservata può anche agire come una memoria buffer o tampone, permettendo al SO il differimento della scrittura dei dati modificati, fino a quando il carico del disco non sia basso, oppure fino a quando la testina del disco si trovi in una posizione favorevole, migliorando le prestazioni di I/O. Poiché la dimensione della cache del disco deve essere limitata per lasciare memoria sufficiente ai processi attivi, i progettisti devono realizzare alcune strategie di sostituzione. Il problema della sostituzione nella cache è simile a quello della sostituzione delle pagine. I progettisti scelgono spesso una strategia che sostituisca dalla memoria buffer/cache l'elemento usato meno recentemente.
La memoria buffer/cache del disco è mantenuta in una memoria volatile e, se il sistema perde energia mentre i dati modificati sono nella buffer/cache, queste modifiche vanno perse. A salvaguardia dei dati, il contenuto della memoria buffer/cache subisce periodicamente un flush verso il disco rigido, riducendo la probabilità di perdere i dati nel caso in cui il sistema incontri qualche problema.
Un sistema che utilizza una cache di tipo write-back (scrivi alla fine) non scrive immediatamente i dati modificati sul disco, mentre la cache viene scrita periodicamente sul disco, permettendo al SO di accorpare più operazioni di I/O servite con una sola richiesta, migliorano le prestazioni. Un sistema che utilizza una cache di tipo write-through (scrivi subito) scrive i dati sia sulla memoria buffer/cache sia sul disco ogni volta che i dati sono modificati. Questa tecnica evita che il sistema possa accorpare le richieste, ma riduce la possibilità di avere dati incongruenti in caso di problemi al sistema.
Molti dei moderni dischi rigidi incorporano una memoria buffer/cache indipendente ad alta velocità, chiamata spesso memoria cache on-board. Se i dati richiesti sono memorizzati nella cache on-board, il disco può spedire i dati ad una velocità prossima a quella della memoria principale. Quando i dati richiesti non sono presenti nella memoria buffer/cache in memoria principale, le memorie on-board migliorano le prestazioni delle operazioni di I/O, servendo le richieste senza compiere le lente operazioni meccaniche tipiche dei dischi.

Altre tecniche di miglioramento delle prestazioni dei dischi

Poiché i file e i record vengono aggiunti e rimossi, i dati dei dischi tendono a disperdersi in tutto il disco, cioè a frammentarsi. Anche i file sequenziali, da cui ci si aspetterebbe una latenza bassa, si frammentano abbastanza, con un incremento del tempo d'accesso. Molti SO forniscono programmi di deframmentazione o riorganizzazione del disco, utilizzabili periodicamente per riorganizzare i file. Questo permette ai record consecutivi di file sequenziali di occupare posizioni contigue del disco.
Inoltre i SO possono posizionare vicino allo spazio libero i file che saranno più probabilmente modificati, per ridurre la frammentazione futura, poiché se il file cresce, i nuovi dati possono essere collocati nello spazio libero adiacente invece che in altre parti del disco. Alcuni SO permettono agli utenti la partizione del disco in aree separate: i file sono quindi limitati in queste partizioni per ridurre la frammentazione. Tuttavia, le partizioni possono portare ad uno spreco di memoria simile a quello dovuto alla frammentazione interna nei sistemi con memoria virtuale paginata.
Alcuni sistemi, per ridurre la quantità di spazio richiesta dalle informazioni sul disco, si avvalgono della tecnica della compressione dei dati. La compressione dei dati riduce la dimensione di un record sostituendo la sequenza normale di bit con una più corta; i dati compressi usano così uno spazio minore senza perdere alcuna informazione, con una conseguente riduzione del numero di seek, del tempo di latenza e del tempo di trasmissione. Tuttavia, questa tecnica può richiedere un

tempo di elaborazione notevole per comprimere i dati e successivamente decomprimerli al fine di renderli disponibili alle applicazioni.
I sistemi che necessitano di un veloce accesso a certe informazioni si avvantaggiano della collocazione di copie multiple di quei dati in posizioni differenti del disco, con un conseguente notevole risparmio del tempo di seek e del tempo di rotazione; le copie ridondanti possono però consumare una parte significativa del disco. Questa tecnica è più utile per i dati in sola lettura o per i dati che cambiano raramente. Le modifiche frequenti riducono le prestazioni di questo schema, poiché ciascuna delle copie va aggiornata regolarmente; un altro lato negativo è che il blocco del sistema possa lasciare le copie multiple in uno stato incongruente.
Inoltre l'accorpamento dei record (blocking) può dar luogo ad un significativo miglioramento delle prestazioni. Quando i record contigui sono letti o scritti come un blocco unico, è richiesta una sola operazione di seek; se questi record fossero lettio scritti individualmente, sarebbe necessaria un'operazione di seek per ogni record.
In molti ambienti ci sono spesso brevi periodi in cui non si verificano richieste d'accesso al disco pendenti ed il braccio del disco è inattivo, in attesa di richieste successive. Se il braccio si trova in un lato del disco, è probabile che la richiesta successiva necessiterà di una lunga operazione di seek. Al contrario, se il braccio si trova al centro del disco o sull'hot spot (punto caldo) dell'attività del disco, il tempo medio di attesa sarà minore. Lo spostamento del braccio del disco in una posizione che minimizza il tempo di seek successivo è noto come anticipazione del braccio del disco.
L'anticipazione del braccio del disco può essere utile negli ambienti in cui la sequenza delle richieste del disco esibisce la località. Per ottenere risultati migliori, il braccio dovrebbe muoversi verso l'hot spot invece che verso il centro del disco.

Array ridondanti di dischi indipendenti (RAID)

RAID rappresenta un insieme di tecniche che usano più dischi, detti array o vettori di dischi, organizzati per fornire migliori prestazioni e/o affidabilità.
Nel documento originale, Patterson e gli altri autori proposero cinque diverse organizzazioni o livelli degli array di dischi. Ogni livello RAID è caratterizzato dal data striping (suddivisione dei dati in strisce) e dalla ridondanza. Il data striping suddivide la memoria secondaria in blocchi di dimensioni fisse dette strisce (strip). Generalmente le strisce contigue di un file sono posizionate su dischi separati, affinché le richieste dei dati possano essere soddisfatte usando più dischi alla volta, migliorando così il tempo d'accesso. Una banda (stripe) consiste in un insieme di strisce poste nella stessa posiziona su ciascun disco dell'array.
Lo striping distribuisce i dati su tutti i dischi, consentendo prestazioni migliori rispetto a quele di un sistema con un unico disco, poiché i dati sono prelevabili dai dischi contemporaneamente.
Se le strisce sono piccole, dette quindi strisce a grana fine, il sistema tende a spalmare il file su diversi dischi. Poiché i dischi recuperano le porzioni dei dati richiesti contemporaneamente, le strisce a grana fine sono in grado di ridurre il tempo d'accesso di ogni richiesta e aumentare la velocità di trasferimento. Mentre i dischi evadono una richiesta, non sono utilizzabili per servire le altre, contenute nella coda delle richieste.
Le strisce grandi, dette strisce a grana grossa, permettono di contenere alcuni file in un'unica striscia. In questo caso, alcune richieste possono essere servite solo da una parte dell'array dei dischi, così è probabile che più richieste possano essere servite contemporaneamente.
Con l'aumento del numero di dischi dell'array, la probabilità di guasto ad un disco cresce. Conseguentemente, molti sistemi RAID memorizzano informazioni che permettono al sistema di ristabilirsi dagli errori, tecnica chiamata ridondanza. I sistemi RAID usano la ridondanza per fornire la tolleranza ai guasti, cioè resistere ai guasti evitando la perdita di dati.
Un modo semplice per assicurare la ridondanza è il disk mirroring (replicazione speculare), tecnica di posizionamento di ogni singolo elemento dei dati su due dischi. Uno svantaggio del mirroring è che è utilizzabile solo la metà della capacità di memorizzazione dell'array di dischi per i singoli elementi dei dati.

Il sistema deve suddividere i file in strisce, ricostruire i file dalle strisce, determinare la posizione delle strisce nell'array e realizzare lo schema di ridondanza.
Molti sistemi RAID contengono quindi dispositivi hardware dedicati, detti controllori RAID, che eseguono queste operazioni velocemente. I controllori RAID semplificano la realizzazione dei sistemi RAID, consentendo al SO di passare semplicemente le operazioni di lettura e scrittura al controllore RAID, che agisce sulle strisce e mantiene le informazioni ridondanti necessarie; i controllori RAID, però, possono aumentare significativamente il costo del sistema.

Livello 0 (striping)

Il livello RAID 0 usa un array di dischi con strisce senza ridondanza. Esso non è uno dei cinque livelli originali e non è considerato un "vero" RAID, poiché non fornisce la tolleranza ai guasti. Se uno dei dischi si guasta, tutti i dati nell'array che dipendono da quel disco vanno persi. A seconda della dimensione delle strisce, tutti i dati memorizzati nell'array diventano inutilizzabili con la perdita di un solo disco. Un sistema RAID 0 con n dischi esegue le letture e le scritture ad una velocità fino a n volte maggiore di un singolo disco. I sistemi RAID 0 sono indicati per quei sistemi dove le prestazioni elevate ed il basso costo sono considerati aspetti più importanti dell'affidabilità.

Livello 1 (mirroring)

Per fornire la ridondanza, in modo che ogni disco dell'array sia duplicato, il livello RAID 1 effettua il mirroring dei dischi, detto anche replicazione speculare o shadowing. Le strisce non sono utilizzate nel livello 1, riducendo così sia la complessità hardware sia le prestazioni del sistema.
Per assicurare la coerenza, i dati modificati devono essere scritti sulla copia di dischi replicati; quindi, le richieste di scrittura multiple destinate alla stessa coppia di dischi devono essere servite una alla volta.
Benché il livello RAID 1 fornisca il grado di tolleranza ai guasti più alto rispetto agli livelli, per memorizzare i singoli dati è utilizzabile solo metà della capacità totale dell'array di dischi; quindi, il costo per unità di memorizzazione in un array RAID 1 è il doppio di quello in array RAID 0. poiché ciascun blocco di dati è memorizzato in una coppia di dischi, il sistema può sostenere guasti di disco multipli senza perdita di dati.
Il recupero e la ricostruzione dei dati da un disco guasto, detta rigenerazione dei dati, comporta la copia dei dati ridondanti dal disco replicato. Tuttavia, se entrambi i dischi di una coppia si guastano, i loro dati sono irrecuperabili.
Alcuni sistemi RAID contengono dei dischi di scorta, anche detti dischi di ricambio a caldo (hot spare disk) o di ripristino al volo (online spare), in grado di sostituire i dischi guasti.
Alcuni sistemi dispongono di dischi sostituibili a caldo che possono essere cambiati mentre il sistema è attivo e consentono la rigenerazione dei dati quando il sistema sta funzionando.
Vantaggi e svantaggi:

- sovraccarico di memoria elevato
- velocità media di trasferimento in lettura superiore: due diverse richieste di lettura memorizzate nella stessa coppia di dischi possono essere servite contemporaneamente
- velocità media di trasferimento inferiore: le richieste di scrittura su una coppia di dischi devono essere eseguite una alla volta. Tuttavia, le richieste di scrittura a diverse coppie di dischi possono essere eseguite contemporaneamente
- tolleranza ai guasti elevata
- costo elevato: il sovraccarico aumenta il costo per unità di memorizzazione.

Il livello RAID 1 è il migliore per ambienti in cui l'affidabilità abbia una priorità superiore al costo o alle prestazioni,

Livello 2 (parità di Hamming ECC a livello di bit)

Gli array di questo livello hanno strisce a livello di bit, quindi ogni striscia memorizza un bit. Il livello 2 non è replicato, riducendo così il sovraccarico di memoria introdotto dal livello 1. Nel caso di guasto la sostituzione è analoga a quella di una porzione di messaggio che viene compromessa durante il trasferimento, come durante una trasmissione di rete.
Questo livello prende in prestito una tecnica che è utilizzata solitamente nei moduli di memoria e che si chiama codice a correzione d'errore di Hamming (Hamming ECCs); tali codici usano i bit di parità per il controllo degli errori dei dati trasmessi dai dischi e, quando possibile, li correggono.
I bit di parità si calcolano come segue. Quando il sistema riceve la richiesta di scrivere i dati (una serie di bit) in una banda, determina la parità della somma dei bit, cioè se la somma è pari o dispari. Generalmente i sistemi RAID posizionano i bit di parità in un disco separato, cosicché le letture e le scritture di ciascuna banda siano eseguibili su tutti i dischi contemporaneamente. Quando si sta per accedere ad una banda, il sistema legge la banda ed il suo bit di parità, calcola la parità della banda e la confronta con il valore letto dal disco di parità.
Un limite di questa particolare forma di parità è che non può rilevare un numero pari di errore e non permette al sistema di determinare quali siano, se ce ne sono, i bit con errori. I codici a correzione di Hamming usano un approccio molto sofisticato che permette al sistema di individuare fino a due errori, di correggere fino ad un errore e di determinare la posizione dell'errore nella banda.
I codici di Hamming vengono calcolati e poi scritti sui dischi, questo processo riduce le prestazioni, poiché il sistema deve accedere all'array due volte per ogni scrittura, ed è detto ciclo di lettura-modifica-scrittura. Sebbene i dischi ECC, in confronto a quelli replicati, riducano il sovraccarico associato alla ridondanza, possono diminuire le prestazioni, poiché richieste multiple non possono venire evase e contemporaneamente. Ogni richiesta di lettura necessita l'accesso a tutti i dischi dell'array, il calcolo dei codici di correzione ed il confronto con quelli recuperati dai dischi di parità; alcuni sistemi riducono questo collo di bottiglia, calcolando la parità solo per le richieste di scrittura. Tuttavia, per mantenere la parità nell'esecuzione di una scrittura, il sistema deve accedere a tutti i dischi; quindi le richieste di scrittura devono essere servite una alla volta. Un modo per aumentare il numero massimo di letture e scritture concorrenti per mezzo di un RAID di livello 2 consiste nella suddivisone dei dischi del sistema in diversi piccoli array RAID di livello 2. Sfortunatamente, questa tecnica incrementa il sovraccarico di memorizzazione, poiché i gruppi più piccoli richiedono un rapporto maggiore tra dischi di parità e dischi di dati.

Livello 3 (parità XOR ECC a livello di bit)

Il livello RAID 3 organizza i dati a livello di bit o di byte. Invece dei codici di Hamming ECC per la generazione della parità, RAID 3 usa i codici di correzione degli errori XOR (esclusive-or) detti XOR ECC. [a xor b = 0 quando a e b sono entrambi o zero o uno, 1 altrimenti].
Il livello RAID 3 sfrutta questa condizione per eseguire operazioni XOR annidate su ciascun byte, così da generare il proprio XOR ECC.
Per contenere la parità XOR ECC usa un solo disco, indipendentemente dalla dimensione dell'array. XOR ECC non permette di individuare quale bit contiene i dati sbagliati. Questo è accettabile, poiché molti errori di parità nei sistemi RAID sono causati da guasti all'intero disco, facili da individuare. Se è il disco della parità a subire un guasto, il recupero comporta il calcolo della parità dai dischi dei dati.
A causa delle strisce a grana fine, molte letture richiedono l'accesso all'intero array; inoltre, a causa della generazione della parità una sola parità può essere eseguita alla volta. Con questo livello si ottengono velocità di trasferimento elevate nella lettura e scrittura di file grandi ma, in generale, si può evadere solo una richiesta per volta. Il vantaggio principale di questo livello di RAID è che è semplice da realizzare, offre un'affidabilità simile al RAID 2 e comporta un sovraccarico di memoria notevolmente inferiore.

Livello 4 (parità XOR ECC a livello di blocco)

I sistemi RAID di livello 4 sono organizzati con blocchi di dimensione fissa, generalmente molto più grandi di un byte, e usano XOR ECC per generare la parità, che richiede un unico disco di parità. Poiché RAID 4 permette strisce a grana grossa, è possibile che i dati richiesti siano memorizzati in una piccola frazione dei dischi dell'array. Quindi se la parità non è determinata per ciascuna lettura, il sistema può potenzialmente servire richieste di lettura contemporaneamente. Poiché i codici ECC sono usati principalmente per la rigenerazione dei dati, invece che per il controllo e la correzione degli errori, molti sistemi eliminano il calcolo della parità nell'esecuzione delle letture, così che si possano servire letture multiple contemporaneamente.
Quando si serve una richiesta di scrittura, tuttavia, il sistema deve aggiornare la parità per assicurare che, in caso di guasto ad un disco, non vada perso alcun dato. Con le strisce a grana grossa, le richieste di scrittura modificano raramente i dati su tutti i dischi dell'array; l'accesso a ciascun disco dell'array per il calcolo della parità può portare ad un sovraccarico notevole.
Le scritture non richiedono l'accesso all'intero array. Tuttavia, poiché le richieste di scrittura necessitano la modifica del disco di parità, devono essere eseguite una alla volta, creando così un collo di bottiglia. Il livello 4 è usato raramente.

Livello 5 (parità XOR ECC distribuita a livello di blocco)

Gli array RAID di livello 5 sono organizzati a livello di blocco e usano la parità XOR ECC, ma i blocchi di parità sono distribuiti sui dischi dell'array. Poiché i blocchi di parità sono distribuiti su più dischi, si può accedere a diverse strisce di parità contemporaneamente, rimuovendo così il collo di bottiglia nella scrittura di molte richieste.
Sebbene il livello RAID 5 migliori le prestazioni in scrittura distribuendo la parità, questi array devono ancora eseguire un ciclo di lettura-modifica-scrittura per ciascuna richiesta di scrittura, richiedendo almeno quattro operazioni di I/O per servire ciascuna richiesta; se un sistema scrive frequentemente piccole quantità di dati, il numero di operazioni di I/O può causare un notevole degrado delle prestazioni.
Sono stati sviluppati diversi metodi per risolvere questo inconveniente. L'utilizzo di una cache per le parità e i dati usati più recentemente può ridurre il numero di operazioni di I/O del ciclo di lettura-modifica-scrittura. La registrazione della parità (parity logging) può migliorare le prestazione del livello RAID 5 salvando in memoria la differenza tra la vecchia parità e quella nuova, chiamata immagine di aggiornamento (update image).
Poiché una singola immagine di aggiornamento può contenere le parità relative a diverse richieste di scrittura, dopo l'esecuzione di diverse scritture il sistema, può ridurre il sovraccarico di I/O eseguendo un unico aggiornamento al blocco di parità nell'array.

> AFRAID (A Frequently RAID), invece di eseguire il ciclo di lettura-modifica-scrittura ad ogni scrittura, la generazione della parità viene ritardata a quando il carico del sistema è basso. Ciò può migliorare notevolmente le prestazioni in ambienti caratterizzati da blocchi intermittenti di richieste che generano piccole scritture.

Sebbene il livello RAID 5 aumenti le prestazioni rispetto ai livelli 2 e 4, è più complesso da realizzare, oltre che più costoso; inoltre, poiché la parità è distribuita nell'array, la rigenerazione dei dati è molto più complicata rispetto agli altri livelli RAID. Nonostante le sue limitazioni il livello 5 è largamente utilizzato, grazie al buon equilibrio tra prestazioni, costi e affidabilità. Gli array di livello 5 sono considerati array general-purpose e si riscontrano spesso nei sistemi aziendali.

Altri livelli RAID

- Livello RAID 6: estende il RAID 5 distribuendo due blocchi di parità per banda per ottenre un'affidabilità più elevata rispetto ai guasti al disco.
- Livello RAID 0+1: un insieme di dischi a livello 0 le cui immagini sono replicate in un secondo insieme di dischi (livello 1).
- Livello RAID 10: un insieme di dischi replicati (livello 1) suddivisi in strisce in un altro insieme di dischi, che richiede un minimo di quattro dischi.

File system e database -capitolo 13-

Le informazioni sono memorizzate nei calcolatori secondo una gerarchia dei dati. Il livello inferiore di questa gerarchia è composto dai bit; per rappresentare tutti gli elementi d'informazione di interesse per un sistema di elaborazione, i bit vengono raggruppati in sequenze di bit (bit pattern). Per una stringa di n bit, vi sono 2 alla n possibili sequenze di bit.
Il livello successivo nella gerarchia dei dati è rappresentato da sequenze di bit di dimensione fissa, come byte, caratteri e parole. Una parola (word) è il numero di bit che un processore può elaborare alla volta. Quindi, una parola è 4 byte su un processore a 32 bit e 8 byte su un processore a 64 bit.
I caratteri associano i byte o i gruppi di byte a simboli, quali lettere, numeri, caratteri d'interpunizione e caratteri di ritorno a capo. Molti sistemi usano i caratteri ad 8 byte e quindi possono avere 256 (2 alla 8) possibili caratteri nel loro insieme di caratteri. I tre insieme di caratteri più in uso oggi sono ASCII, EBCDIC, Unicode.
ASCII memorizza caratteri come byte da 8 bit e può quindi disporre di 256 possibili caratteri nel suo insieme. Data la sua piccola dimensione, ASCII non supporta insiemi di caratteri internazionali. Unicode è uno standard riconosciuto a livello internazionale di uso comune nelle applicazioni multilingua e in Internet. Il suo obbiettivo è di usare un numero unico per la rappresentazione di tutti i caratteri delle lingue del mondo.
Un campo è un gruppo di caratteri, come un nome, un indirizzo o un numero di telefono. Un record è un gruppo di campi. Un file è un gruppo di record correlati. Il livello più alto della gerarchia è rappresentato dai file system o dai database. I file system sono una raccolta di file, mentre i database, trattati più avanti, sono raccolte di dati.
Il termine volume indica un'unità di memorizzazione di dati che può contenere più file. Un volume fisico è limitato ad un singolo dispositivo di memorizzazione; un volume logico, come quelli utilizzabili nelle macchine virtuali, può risiedere su molti dispositivi.

File

Un file è una raccolta di dati associata ad un nome, manipolabile come un'unica entità da operazioni quali:

- apertura, prepara il file ad essere referenziato
- chiusura, evita ulteriori referenze ad un file fino alla riapertura
- creazione, crea un nuovo file
- distruzione, rimuove un file
- copia, copia il contenuto di un file in un altro
- rinomina, cambia il nome del file
- elenco, stampa o mostra il contenuto di un file

I singoli elementi di dato interni al file sono manipolabili per mezzo di operazioni quali:

- lettura, copia dei dati da un file alla memoria del processo
- scrittura, copia dei dati dalla memoria del processo al file
- modifica, modifica di dati esistenti in un file
- inserimento, aggiunta di nuovi dati ad un file

- cancellazione, rimozione di dati da un file

I file possono essere caratterizzati da attributi quali:

- dimensione, quantità di dati memorizzati nel file
- posizione, posizione del file in un dispositivo di memorizzazione o nell'organizzazione logica dei file del sistema
- accessibilità, restrizioni poste all'accesso dei dati del file
- tipo, modalità d'uso dei dati del file
- volatilità, frequenza con cui sono effettuate le cancellazioni e le aggiunte ad un file
- attività, percentuale dei record di un file a cui si ha avuto accesso durante un dato periodo di tempo

I file possono consistere in uno o più record. Un record fisico o blocco fisico, è un'unità d'informazione letta o scritta in un dispositivo di memorizzazione. Un record logico o blocco logico è una raccolta di dati gestita dal software come un'unica entità. Quando ciascun record fisico contiene esattamente un record logico, si dice che il file consiste in record non bloccati; quando ciascun record fisico può contenere diversi record logici, si dice che il file consiste in record bloccati. In un file con record di dimensione fissa, tutti i record sono della stessa lunghezza e la dimensione dei blocchi è un multiplo intero della dimensione dei record. In un file con record di dimensione variabile, i record possono variare fino a raggiungere la dimensione del blocco.

File system

Un file system organizza il file e gestisce gli accessi ai dati. I file system sono responsabili delle fasi seguenti:

- Gestione dei file, fornisce ai file meccanismi di memorizzazione, accesso, condivisione e sicurezza
- gestione della memoria ausiliaria, alloca spazio per i file sui dispositivi di memorizzazione secondaria o terziaria
- meccanismi d'integrità dei file, assicurano che le informazioni memorizzate in un file non siano modificate
- metodi d'accesso, regolano l'accesso ai dati memorizzati

Il file system interagisce principalmente con la gestione dello spazio di memorizzazione secondaria, in particolare dei dischi.
I file system permettono all'utente creazione, modifica e cancellazione di file; inoltre sono in grado di strutturare i file per ogni applicazione nel modo più appropriato e di avviare il trasferimento dei dati tra file.
Il meccanismo di condivisione dei file dovrebbe fornire diversi tipi di controllo d'accesso come accesso in lettura, accesso in scrittura, accesso in esecuzione o loro diverse combinazioni. Il file system dovrebbero essere indipendenti dal dispositivo, gli utenti dovrebbero cioè dovrebbero poter far riferimento ai loro con nomi simbolici, invece dei nomi fisici dei dispositivi.
I nomi simbolici permettono al file system di dare agli utenti una vista logica dei loro dati, assegnando nomi significativi ai file e alle operazioni sui file. Una vista fisica è legata alla struttura del file sul dispositivo e alle sue operazioni specifiche, al fine di manipolare i dati. Gli utenti non dovrebbero preoccuparsi dei particolari dispositivi sui quali sono memorizzati i dati, della loro forma su questi dispositivi o del significato fisico del trasferimento dei dati attraverso di essi.
Per prevenire perdite accidentali o distruzioni intenzionali di informazioni, il file system dovrebbe fornire anche la funzionalità di backup, che facilita la creazione di copie ridondanti di dati e la funzionalità di recupero, che permette agli utenti di ripristinare qualsiasi dato perso o danneggiato.
Il file system potrebbe fornire anche le funzionalità di cifratura (encryption) e decifratura (decryption). Queste funzionalità rendono le informazioni disponibili solo ai destinatari autorizzati, cioè a chi è in possesso di chiavi di cifratura.

Directory

Per organizzare velocemente i file, i file system usano le directory (cartelle), file che contengono nomi e locazioni di altri file presenti nel sistema; diversamente dagli altri file, le directory non possono contenere dati utente.

La più semplice organizzazione per un file system è la struttura delle directory a livello singolo o piatto. In questa realizzazione il file system memorizza tutti i file usando una sola directory. In un file system a singolo livello, nessun file può avere lo stesso nome (implementati raramente).

In un file system strutturato gerarchicamente, una radice (root) indica l'inizio della directory radice (root directory) sul dispositivo. Le directory sono file che possono puntare ad altre directory e file. La directory radice punta alle diverse directory utente. Una directory utente contiene una riga per ogni file utente; ciascuna riga punta alla posizione sul dispositivo fisico corrispondere al file.
I nome dei file devono essere unici solo all'interno della stessa directory utente. In un file system strutturato gerarchicamente ogni directory può contenere diverse sottodirectory, ma non più di una directory padre. Il nome di un file è solitamente costruito come il pathname, che parte dalla directory radice e arriva fino al file.
I file system gerarchici sono realizzati in molti file system general-purpose, ma il nome della directory radice ed il tipo di delimitatore possono variare tra file system.

Molti file system supportano la nozione di directory di lavoro per semplificare la navigazione per mezzo dei pathname. La diretory di lavoro nei file system UNIX e Windows permette all'utente di specificare un pathname che non inizi dalla directory radice ma utilizzando un percorso relativo. Quando un file system incontra un pathname relativo, costruisce il percorso assoluto, cioè il percorso che inizia dalla radice, concatenando la directory di lavoro con il percorso relativo; quindi, il file system attraversa la struttura delle directory per individuare il file richiesto.
Generalmente il file system mantiene un riferimento alla directory padre della directory di lavoro, cioè la directory al livello superiore della gerarchia.

Un link (collegamento) è un elemento di una directory che punta ad un file di dati o a una directory residente in genere in un'altra directory. Un soft link (collegamento debole) detto anche link simbolico nei sistemi UNIX, scorciatoia nei sistemi Windows e alias nei sistemi MacOS, è un elemento di una directory che contiene il pathname di un altro file. Il file system individua la destinazione del soft link attraversando la struttura delle directory usando il pathname specificato. Un hard link (collegamento forte) è un elemento di una directory che specifica la posizione del file, solitamente il numero del blocco, sul dispositivo fisico. Il file system individua il file puntato dal link accedendo direttamente al blocco fisico che referenzia.

La riorganizzazione del disco e la deframmentazione sono utilizzabili per migliorare le prestazioni del disco; durante queste operazioni, le posizioni fisiche di un file possono cambiare; si richiede quindi che il file system aggiorni le posizioni dei file presenti nella directory. Poiché un hard link specifica la posizione fisica di un file, in caso di spostamento di quel file, il link referenzierà dati non validi; per risolvere questo problema, un file system può memorizzare in un file un puntatore ai suoi hard link: quando cambia la posizione fisica del file, il file system può usare questi puntatori per individuare gli hard link che richiedono un aggiornamento.
Poiché i soft link memorizzano la posizione logica del file nel file system, che non cambia durante la deframmentazione o la riorganizzazione del disco, non richiedono aggiornamenti al momento dello spostamento dei dati del file. Tuttavia, se un utente sposta un file in una directory differente o rinomina il file, qualsiasi soft link a questo file non sarà più valido.
Quando un utente distrugge un link a un file, il file system deve determinare se distruggere anche il file corrispondente. A questo scopo, i file system in genere mantengono un conteggio del numero di

hard link al file; quando raggiunge zero, il file system non contiene riferimenti al file e quindi può tranquillamente rimuoverlo. Poiché i soft link non puntano alle posizioni fisiche dei file, non sono considerati nella determinazione della rimozione di un file.

Metadati

La maggior parte dei file system, per assicurare che i blocchi in uso non vengano sovrascritti da dati di un nuovo file, memorizza la posizione di blocchi liberi di un dispositivo e l'ora corrispondente all'ultima modifica del file. Queste informazioni, dette metadati, proteggono l'integrità del file system e non sono direttamente modificabili dagli utenti.
Prima che un file system possa accedere ai dati, il dispositivo di memorizzazione viene in genere formattato. Formattare è un'operazione dipendente dal sistema, ma generalmente comporta il controllo del dispositivo e la creazione della directory radice del file system. Molti file system creano inoltre un superblocco per memorizzare le informazioni che proteggono l'integrità del file system. Un superblocco potrebbe contentere:

- un identificativo del file system che ne identifica univocamente il tipo
- il numero di blocchi nel file system
- la posizione dei blocchi liberi del dispositivo
- la posizione della directory radice
- la data e l'ora in cui il file system ha subito l'ultima modifica
- le informazioni che indicano se il file system richiede di essere controllato

Se il superblocco è danneggiato o distrutto, il file system potrebbe non essere in grado di accedere ai dati. Piccoli errori nel superblocco, come la posizione dei blocchi liberi, possono causare la sovrascrittura di file esistenti. Per ridurre il rischio di perdita di dati, molti file system distribuiscono sul dispositivo di memoria copie ridondanti del superblocco. Il file system quindi può usare le copie ridondanti del superblocco per determinare se il superblocco principale sia danneggiato e, in caso positivo, per sostituirlo.

Il sistema mantiene in memoria principale una tabella che tiene traccia dei file aperti. In molti sistemi l'operazione di apertura dei file restituisce un descrittore del file, un numero intero non negativo che è un indice della tabella dei file aperti. Da questo punto in poi, l'accesso al file avviene per mezzo del descrittore del file.
La tabella dei file aperti contiene spesso il blocco di controllo del file (file control block), che specifica le informazioni necessarie al sistema per gestire il file, dette attributi del file. Questi attributi dipendono fortemente dalla struttura del sistema.

Montaggio

L'operazione di montaggio (mounting) combina più file system in uno spazio di nomi (namespace), un insieme di file identificabili da un unico file system. Lo spazio di nomi unificato permette agli utenti l'accesso ai dati da diversi dispositivi, come se tutti i file fossero posizionati interamente al file system nativo. Il comando di montaggio assegna una directory del file system nativo, detta punto di montaggio, alla radice del file system montato.
Quando un file system è montato, il contenuto della directory che è punto di montaggio è irraggiungibile agli utenti fino a quano il file system non viene smontato. I file system gestiscono le directory montate con la tabella dei montaggi. Questa tabella contiene le informazioni sui pathname dei punti di montaggio e i dispositivi che memorizzano ogni file system montato.
Il comando di smontaggio permette all'utente di scollegare il file system montato. Questo comando aggiorna la tabella dei montaggi e permette agli utenti l'accesso ai file nascosti dal file system montato.

Un hard link specifica un blocco associato al dispositivo che memorizza il collegamento. In generale, gli utenti non possono creare hard link tra due file system, poiché spesso associati a dispositivi di memorizzazione differenti.

Organizzazione dei file

L'organizzazione dei file fa riferimento al modo in cui i record di un file sono sistemati sul dispositivo secondario. Gli schemi di organizzazione implementati sono i seguenti:

- Sequenziale, i record sono posizionati fisicamente in ordine sequenziale. Il record successivo è quello che fisicamente segue il record precedente. Questa organizzazione è naturale per i file memorizzati sui nastri magnetici, un mezzo intrinsecamente sequenziale. I file dei dischi che possono essere anch'essi organizzati sequenzialmente, ma per diversi motivi esaminati più avanti, i record di un file sequenziale non lo sono necessariamente.
- Diretta, i record sono referenziati direttamente, ovvero casualmente attraverso l'indirizzo fisico su un dispositivo di memoria ad accesso diretto (DASD). Le applicazioni degli utenti posizionano i record sul DASD nell'ordine più appropriato.
- Sequenziale indicizzata, i record sul disco sono organizzati in sequenze logiche, secondo una chiave contenuta in ogni record. Il sistema mantiene un indice che contiene l'indirizzo fisico di alcun record principali. I record sequenziali indicizzati sono accessibili sequenzialmente nell'ordine delle chiavi, o direttamente, attraverso una ricerca nell'indice creato dal sistema.
- Partizionata, questo è essenzialmente un file di sottofile sequenziali. Ogni sottofile sequenziale è detto membro. L'indirizzo iniziale di ogni membro è memorizzato nella directory del file; i file partizionati sono usati per memorizzare librerie di programmi o macro librerie.

Allocazione dei file

Poichè i file vengono allocati e liberati, lo spazio in memoria secondaria tende sempre più a frammentarsi, con i file sparpagliati in blocchi dispersi sul disco; questo aspetto può causare problemi di prestazioni. Il sistema può eseguire la deframmentazione, ma l'esecuzione può comportare tempi di risposta lunghi mentre il file system è in uso.
La località spaziale indica che una volta che un processo ha referenziato dei dati su una pagina, è molto probabile che ne referenzierà altri sulla stessa pagina; è anche probabile che referenzi i dati delle pagine contigue a quella nello spazio di indirizzamento virtuale. Meglio quindi memorizzare le pagine non residenti e contigue nello spazio d'indirizzamento virtuale come pagine fisicamente contigue in memoria secondaria, specialmente se diverse pagine sono memorizzate in un blocco fisico.
Poiché col tempo i file spesso crescono o si riducono, e poiché gli utenti raramente conoscono in anticipo quanto sarà grande il loro file, i sistemi di allocazione contigua sono generalmente sostituiti da sistemi di allocazione non contigua più dinamici. Per sfruttare la località, questi sistemi tentano di allocare parti di file in modo contiguo, ma consentono ai file di cambiare la dimensione con un sovraccarico minimo.

Allocazione contigua dei file

I file system che utilizzano l'allocazione contigua posizionano i file in indirizzi contigui sul dispositivo di memorizzazione secondaria. L'utente specifica in anticipo la quantità di spazio necessario a memorizzare il file; se la quantità di spazio contiguo desiderata non è disponibile, il file non può essere creato.

Un vantaggio è che i record logici successivi sono in genere fisicamente adiacenti agli altri; questo accelera l'accesso. Individuare i file di dati è semplice, poiché le directory necessitano di memorizzare solo l'indirizzo di inizio del file e la sua lunghezza.
Uno svantaggio è che mostrano lo stesso tipo di frammentazione esterna tipico dell'allocazione della memoria nei sistemi multiprogrammati a partizioni variabili. Inoltre, l'allocazione contigua può causare basse prestazioni qualora i file crescano o si riducano nel tempo. Se un file cresce oltre la dimensione originale specificata e non ci sono blocchi liberi adiacenti, il file deve essere trasferito in una nuova area di dimensioni adeguate, introducendo così operazioni di I/O aggiuntive. Per fornire un'espansione anticipata, gli utenti possono sovrastimare le loro necessità di memoria, con conseguente allocazione inefficiente.

Allocazione dei file con liste collegate non contigue

Molti file system realizzati su dispositivi riscrivibili usano l'allocazione non contigua, che può essere impostata realizzando una lista collegata basata sui settori. In questo schema, ogni elemento delle directory punta al primo settore di un file di un dispositivo a testina mobile, come i dischi rigidi. La porzione di dati del settore memorizza il contenuto del file; la porzione del puntatore memorizza un puntatore al settore successivo del file. Poiché i file spesso occupano più settori, la testina di lettura e scrittura deve accedere sequenzialmente a ogni settore del file, fino all'individuazione del record richiesto.
L'allocazione non contigua risolve alcuni problemi inerenti agli schemi di allocazione contigua, ma ha i propri svantaggi; poiché i record del file possono essere dispersi sul disco, l'accesso diretto o sequenziale ai record logici può comportare oltre al primo seek del file operazioni di seek aggiuntive. I puntatori nella struttura della lista riducono, inoltre, in ogni settore la quantità di spazio disponibile per i dati dei file.
L'allocazione a blocchi è lo schema adottato per una gestione più efficiente della memoria secondaria e per la riduzione del sovraccarico degli attraversamenti. In questo schema, invece di allocare i singoli settori, vengono allocati blocchi di settori contigui, spesso detti estensioni. Scegliendo i blocchi liberi più vicini ai blocchi già esistenti, preferibilmente sullo stesso cilindro, il sistema tenta di allocarne di nuovi. Ogni accesso al file richiede la determinazione del blocco appropriato e del settore all'interno del blocco.
Con il concatenamento (chaining) dei blocchi, ogni elemento di una directory punta al primo blocco di un file. I blocchi che includono un file contengono ciascuno due porzioni: un blocco dati ed un puntatore al blocco successivo. La più piccola unità di allocazione è un blocco di dimensioni fisse che generalmente consiste in molti settori. Individuare un particolare record richiede la ricerca di quel blocco nella catena dei blocchi e la ricerca nel blocco del record appropriato.
Se i blocchi sono occupati parzialmente dai file, quelli grandi possono comportare un'elevata frammentazione interna. I blocchi grandi riducono comunque il numero di operazioni di I/O richieste dell'accesso ai dati. I blocchi di piccole dimensioni possono causare la dispersione di file su più blocchi, causando una riduzione delle prestazioni.

Allocazione non contigua tabellare dei file

L'allocazione non contigua tabellare dei file memorizza i puntatori in modo contiguo in una tabella. Gli elementi delle directory indicano il primo blocco del file. Il numero del blocco corrente è usato come indice di una tabella di allocazione dei blocchi per determinare la posizione del blocco successivo.
Poiché i puntatori che individuano i file sono memorizzati in posizioni centrali, la tabella può essere posizionata in memoria, affinché i blocchi che compongono un file siano attraversabili velocemente, migliorando così il tempo d'accesso. Per ridurre il tempo d'accesso, la tabella di allocazione dovrebbe essere memorizzata sul disco in modo contiguo e collocata in una cache in memoria principale.

La dimensione di un elemento della tabella, e quindi la dimensione della tabella stessa, è magiore per un file system contenente un gran numero di blocchi.
Se i dati dei file sono sparsi sul disco, gli elementi del file della tabella dei blocchi saranno distribuiti sull'intera tabella; il sistema, quindi, dovrà caricare diversi blocchi della tabella di allocazione, causando un tempo d'accesso elevato e un consumo significativo di memoria, nel caso la tabella di allocazione disponga di un meccanismo di cache.
Un'implementazione di uso comune dell'allocazione non contigua tabellare dei file è il file system FAT di Microsoft. Microsoft ha introdotto la tabella di allocazione dei file (File Allocation Table), questa memorizza le informazioni di ogni blocco, inclusa l'allocazione attuale del blocco ed il numero del blocco successivo del file.

Allocazione indicizzata non contigua dei file

Un'altra diffusa strategia di allocazione usa i blocchi indice per puntare ai dati di un file. Ogni file possiede uno o più blocchi indice; un blocco indice contiene un elenco di puntatori ai blocchi di dati dei file. Gli elementi delle directory puntano ai blocchi indice dei file. Per individuare un record, il file system attraversa la struttura delle directory alla ricerca della posizione del blocco indice del file su disco; quindi, carica il blocco indice in memoria e usa i puntatori per determinare la posizione fisica del particolare blocco. I file grandi consumano spesso molti blocchi in più rispetto al numero di puntatori contenibile in un singolo blocco indice.
Il vantaggio principale di questa tecnica è che la ricerca può limitarsi ai blocchi indice stessi, che si possono tenere vicini tra loro nel dispositivo secondario per memorizzare i seek. Per velocizzare l'attraversamento dei file, i blocchi indice sono memorizzati in una memoria cache. Una volta localizzato il record con i blocchi indice, il blocco dati contenente il record viene portato nella memoria principale. Il concatenamento dei blocchi indice è analogo alla memorizzazione di una tabella di allocazione separata per ogni file, che può essere più efficiente delle tabelle di allocazione a livello di sistema, poiché i riferimenti a ciascun file sono memorizzati in modo contiguo nel suo blocco indice. Generalmente i file system posizionano i blocchi indice vicino ai blocchi di dati che referenziano, in modo che siano accessibili rapidamente dopo il caricamento del loro blocco indice.
Nei SO UNIX i blocchi indici sono detti inode, cioè nodi indice. Un inode di un file memorizza gli attributi di quel file, memorizza l'indirizzo di alcuni blocchi dati ed i puntatori alle continuazioni dei blocchi indice, che sono detti blocchi indiretti. La struttura degli inode supporta fino a tre livelli di blocchi indiretti. Il primo blocco indiretto punta a blocchi di dati (singolo livello di indirezione); il secondo blocco indiretto contiene i puntatori verso altri blocchi indiretti, questi puntano a blocchi di dati (due livelli di indirezione); il terzo blocco indiretto punta ad altri blocchi indiretti che puntano ancora ad altri blocchi indiretti che a loro vola puntano a blocchi di dati (tre livelli di indirezione).
La potenza di questa struttura gerarchica consiste nel porre un limite relativamente basso al numero massimo di puntatori da seguire per individuare i dati di un file: permette l'individuazione di qualsiasi dato seguendo un massimo di quattro puntatori.

Gestione dello spazio libero

I file system mantengono un elenco delle posizioni dei blocchi che sono disponibili alla memorizzazione di nuovi dati, cioè di blocchi liberi. Per tenere traccia dello spazio disponibile, un file system può usare una lista dello spazio libero. La lista dello spazio libero è una lista collegata di blocchi contenenti le posizioni dei blocchi liberi. L'ultimo elemento di un blocco della lista memorizza un puntatore al blocco successivo; l'ultimo elemento dell'ultimo blocco memorizza un puntatore nullo, indicando che non ci sono ulteriori blocchi nella lista. Quando il sistema ha bisogno di allocare un nuovo blocco a un file, trova l'indirizzo di un blocco libero nella lista dello spazio libero, scrive i nuovi dati nel blocco libero e lo rimuove dalla lista dello spazio libero.

Il file system alloca i blocchi all'inizio della lista dello spazio libero e aggiunge alla fine della lista quelli che si sono liberati. I puntatori alla testa ed alla coda della lista sono memorizzabili nel superblocco del file system. Un blocco libero è individuabile seguendo un solo puntatore; questa tecnica richiede, quindi, un piccolo sovraccarico per l'esecuzione delle operazioni di gestione della lista.
Un altro metodo diffuso di gestione dello spazio libero è la bitmap (mappa di bit). Una bitmap contiene un bit per ogni blocco del file system, dove l'i-esimo bit corrisponde all'i-esimo blocco del file system. Un bit della bitmap è posto a 1 se il blocco è in uso, e 0 altrimenti.
La bitmap generalmente occupa diversi blocchi. Uno dei vantaggi principali delle bitmap rispetto alle liste è che il file system può determinare velocemente la disponibilità dei blocchi contigui in una certa posizione del disco.
Uno svantaggio delle bitmap è che per trovare un blocco libero il file system può aver bisogno di cercare nell'intera bitmap, con un sovraccarico di calcolo. Questo sovraccarico in molti casi è banale, grazie al fatto che nei sistemi odierni la velocità dei processori è molto elevata, rispetto alla velocità delle operazioni di I/O.

Controllo d'accesso ai file

Matrice di controllo degli accessibile

Un modo per controllare gli accessi ai file è dato dalla creazione di una matrice di controllo degli accessi bidimensionale che elenchi tutti gli utenti e tutti i file di sistema. L'elemento a_{ij} è 1 se l'utente i può accedere al file j, altrimenti $a_{ij}=0$.
In un sistema con un gran numero di utenti e di file, questa matrice sarebbe abbastanza grande. Inoltre, permettere ad un utente l'accesso ad un file di un altro utente è un'eccezione piuttosto che una regola, quindi la matrice sarebbe estremamente sparsa. Per rendere il concetto di matrice utile, sarebbe necessario usare dei codici per indicare i diversi tipi d'accesso, come: solo lettura, solo scrittura, solo esecuzione, lettura e scrittura, e così via, con un incremento notevole della dimensione della matrice.

Controllo degli accessi con le classi di utente

Una tecnica che richiede uno spazio considerevolmente inferiore a quello usato dalle matrici di controllo, è il controllo degli accessi con diverse classi di utenti. Uno schema di classificazione comune degli accessi ai file è il seguente.

- Proprietario, generalmente l'utente che ha creato il file. Il proprietario ha un accesso non ristretto al file e in genere ne può cambiare i permessi.
- Utente specifico, il proprietario specifica che un altro individuo può usare il file.
- Gruppo (di progetto), i diversi membri del gruppo sono autorizzati all'accesso dei file relativi al progetto.
- Pubblico, molti sistemi permettono a un file di essere classificato come pubblico, e di essere referenziato da qualsiasi utente del sistema. Generalmente permette agli utenti di leggere ed eseguire un file, ma non di scriverlo.

Questi dati di controllo degli accessi possono essere memorizzati come parte del blocco di controllo del file e spesso occupano una quantità di spazio minima.

Tecniche d'accesso ai dati

I metodi di accesso ai dati sono classificabili in: metodi d'accesso a coda e metodi d'accesso di base, i metodi a coda forniscono migliori capacità rispetto ai metodi di base.
I metodi a coda sono usati quando può essere prevista la sequenza in cui sono stati processati i record, come negli accessi sequenziali o sequenziali indicizzati. I metodi a coda eseguono la

memorizzazione a previsione (anticipatory buffering) e lo scheduling delle operazioni di I/O. Questi metodi tentano di rendere disponibile all'elaborazione il record successivo, prima che sia elaborato quello precedente. Più di un record alla vola viene mantenuto in memoria principale, permettendo la sovrapposizione dell'elaborazione e delle operazioni di I/O, migliorando così le prestazioni.
I metodi d'accesso di base sono usati normalmente quando la sequenza in cui vengono elaborati i record non è prevedibile, in particolare con l'accesso diretto. Nei metodi di base, il metodo d'accesso legge e scrive i blocchi fisici; l'accorpamento e l'estrazione dei dati in blocchi, se utili per l'applicazione, vengono eseguiti dall'applicazione utente.
I file mappati in memoria associano i dati di un file a uno spazio d'indirizzamento virtuale invece di usare la cache del file system. Poiché i riferimenti a questi file avvengono in uno spazio d'indirizzamento virtuale, la gestione della memoria virtuale può indirizzarsi a decisioni di sostituzione delle pagine in base alle sequenze di riferimenti di ogni processo.
Quando un processo effettua una richiesta di scrittura, i dati sono in genere memorizzati in memoria principale, per migliorare le prestazioni di I/O, e la pagina corrispondente viene marcata come dirty. Quando viene sostituita la pagina modificata di un file mappato in memoria, viene scritta sul file corrispondente sul dispositivo secondario. Quando il file viene chiuso, il sistema copia tutte le pagine dirty nel dispositivo secondario; per ridurre il rischio di perdita dei dati dovuto a guasti del sistema, le pagine dirty sono copiate periodicamente sul dispositivo.

Protezione dell'integrità dei dati

Backup e recupero

Molti sistemi realizzano tecniche di backup per la memorizzazione di copie ridondanti delle informazioni, oltre a tecniche di recupero per il ripristino dei dati originali dopo un guasto del sistema.
I backup fisici duplicano i dati del dispositivo a livello di bit; in alcuni casi, il sistema copia solo i blocchi allocati. I backup fisici sono di semplice realizzazione, ma non memorizzano nessuna informazione circa la struttura logica del file system. I dati del file system sono memorizzabili in formati differenti, a seconda dell'architettura del sistema; i backup fisici quindi non sono facilmente recuperabili su computer che usano architetture differenti. Inoltre, poiché un backup fisico non legge le strutture logiche del file system, non distingue i file che contiene; i backup fisici, quindi, devono memorizzare e recuperare l'intero file system, assicurando che tutti i dati siano stati duplicati, anche se molti dei dati del file system non sono stati modificati dall'ultimo backup fisico.
Un backup logico memorizza i dati del file system e la sua struttura logica; i backup logici quindi osservano la struttura delle directory per determinare quali file necessitino di backup, e scrivono questi file sul dispositivo di backup, in un formato di archiviazione comune, spesso compresso. Nel caso di cancellazione accidentale, i backup logici permettono anche all'utente il recupero di un singolo file dal backup, generalmente più veloce di quello dell'intero file system. Poiché i backup logici possono leggere solo i dati mostrati dal SO, possono omettere informazioni come i file nascosti ed i metadati, copiati dal backup fisico nella duplicazione di ciascun bit del dispositivo di memorizzazione.
I backup incrementali sono backup logici che memorizzano solo i dati del file system modificati rispetto al backup precedente. Il sistema registra quali file sono stati modificati e li scrive periodicamente in un file di backup. Poiché i backup incrementali richiedono meno tempo e poche risorse rispetto al backup dell'intero file system, sono eseguibili più spesso, riducendo così al minimo il rischio di perdita dei dati causati da calamità.

Integrità dei dati e file system strutturati a log

Se un sistema si guasta durante un'operazione di scrittura, è possibile che i dati del file system siano rimasti in uno stato incoerente. La registrazione o logging delle transizioni riduce il rischio di

perdita dei dati per mezzo delle transizioni atomiche; esse eseguono un gruppo di operazioni nel loro complesso o non ne eseguono nessuna. Se accade un errore che impedisce il completamento della transazione, avviene l'annullamento della transazione (roll back) che riporta il sistema nello stato precedente l'inizio della transazione.
Le transazioni atomiche sono realizzabili registrando il risultato di ogni operazione in un file registro o log, invece di modificare i dati esistenti; una volta completata la transazione, viene confermata registrando un valore speciale nel log. In un determinato momento successivo, il log viene trasferito in memoria permanente. Se il sistema si guasta prima del completamento della transizione, qualsiasi operazione registrata successivamente viene ignorata.
Per determinare lo stato del file system a transazione confermata (committed transaction), un sistema effettua il recupero leggendo il file di log e confrontandolo con i dati nel file system.
Per permettere al sistema l'annullamento di un qualsiasi numero di operazioni, in molti sistemi il file di log non viene cancellato dopo la scrittura delle due operazioni sul disco. Poiché i log possono diventare grandi, la rielaborazione delle transazioni per determinare lo stato del sistema all'ultimo punto di commit può essere molto lunga; per ridurre il tempo speso nella rielaborazione delle transizioni nel log, molti sistemi mantengono dei checkpoint (posto di blocco), che puntano all'ultima transazione trasferita in memoria permanente. Se il sistema si danneggia, è sufficiente esaminare le transazioni successive al checkpoint.
La paginazione shadow (paginazione ombra) realizza le transazioni atomiche scrivendo i dati modificati nei blocchi liberi invece che nei blocchi originali. Una volta confermata la transazione (committed), il file system aggiorna i suoi metadati puntando al nuovo blocco e liberando quello vecchio; questa operazione viene anche detto oscuramento della pagina (pagina shadow), poiché nasconde il blocco vecchio trasformandolo in spazio libero. Se la transazione fallisce il file system ne effettua l'annullamento, liberando così il blocco nuovo come spazio libero.

La registrazione delle transazioni e la paginazione shadow impediscono ai dati del file system l'ingresso in uno stato incoerente, ma non ne garantiscono necessariamente la coerenza.
I file system strutturati a log (LFS), detti anche journaling file system, eseguono le operazioni di file system come transazioni registrate.
In un LFS l'intero disco è a servizio come file di log per la registrazione delle transazioni; i nuovi dati sono scritti sequenzialmente nello spazio libero del file di log.
Poiché le directory modificate ed i metadati sono sempre scritti alla fine del log, per individuare un file particolare un LFS può aver bisogno di leggere l'intero log, causando prestazioni di lettura scadenti. Per limitare questo problema, un LFS memorizza in una cache le posizioni dei metadati del file system e occasionalmente scrive le mappe degli inode o il superblocco, indicanti la posizione degli altri metadati. Questo permette al SO di individuare e porre in cache i metadati dei file subito all'avvio del sistema; successivamente, i dati dei file, determina la loro posizione dalla cache del file system, sono accessibili velocemente. Con l'aumento della dimensione della cache del file system, avviene un miglioramento delle prestazioni a spese della quantità di memoria disponibile ai processi utenti, che viene ridotta.
Per limitare il sovraccarico, alcuni file system memorizzano nel log solo i metadati. In questo caso il file system modifica prima i metadati scrivendo nel log e, quindi, aggiorna gli elementi nel file system. L'operazione effettua il commit solo dopo l'aggiornamento dei metadati sia nel log sia nel file system, assicurando l'integrità del file system con un sovraccarico relativamente basso, ma non l'integrità dei file in caso di guasto al sistema.
Poiché in un LFS i dati sono scritti sequenzialmente, anche l'esecuzione di ogni richiesta segue lo stesso ordine, riducendo così notevolmente i tempi di scrittura. Per contro, le implementazioni delle allocazioni non contigue possono richiedere al file system l'attraversamento della struttura delle directory, che può essere distribuita in tutto il disco, comportando lunghi tempo d'accesso.
Quando il log si riempie, a LFS spetta il compito di determinare il recupero dello spazio libero per i dati in ingresso. Per determinare quali blocchi possono essere liberati, un LFS può ispezionare periodicamente il contenuto del log, in quanto contiene una copia modificata dei loro dati. LFS può

sistemare i nuovi dati in questi blocchi, probabilmente altamente frammentati. Questa sistemazioni può sfortunatamente ridurre le prestazioni di lettura e scrittura a livelli inferiori del file system convenzionali, poiché il disco può aver bisogno di eseguire molti seek per accedere ai dati. Per superare il problema, un LFS può creare zone contigue di spazio libero nel log, copiando i dati in una regione contigua alla fine del log; durante l'esecuzione di queste operazioni di I/O, gli utenti sperimenteranno tempi di risposta elevati.

Controllo degli accessi -capitolo 19, paragrafo 4-

Come gestore delle risorse, un SO deve proteggersi dal loro potenziale uso da parte di utenti malintenzionati. Di conseguenza i moderni SO sono progettati per tutelare i servizi e le informazioni sensibili da utenti o software che abbiano accesso alle risorse dell'elaboratore. I diritti d'accesso proteggono le risorse di sistema ed i servizi degli utenti potenzialmente pericolosi, circoscrivendo o limitando le azioni che possono essere svolte su di loro. Questi diritti sono gestite da liste di controllo degli accessi o liste di capacità.

Diritti d'accesso e domini di protezione

La chiave per ottenere la sicurezza a livello di SO è il controllo degli accessi a dati interni ed alle risorse. I diritti d'accesso stabiliscono come vari soggetti possano accedere a vari oggetti. I soggetti possono includere utenti, processi, programmi o altre entità. Gli oggetti comprendono risorse, hardware o software, e dati; possono inoltre esserci oggetti fisici, che corrispondono a dischi, processori o memorie, oppure oggetti astratti, che corrispondono a strutture dati, processi o servizi. I soggetti possono essere oggetti del sistema: un soggetto potrebbe avere i diritti per accedere ad un altro soggetto. I soggetti sono entità attive, gli oggetti sono passivi. Durante il funzionamento di un sistema, la popolazione di soggetti e oggetti tende a cambiare: il modo in cui un soggetto può accedere ad un oggetto è chiamato privilegio, e include lettura, scrittura e stampa.
Gli oggetti devono essere protetti dai soggetti: se a un processo è consentito l'accesso ad ogni risorsa del sistema, un utente potrebbe, intenzionalmente o inavvertitamente, compromettere la sicurezza o causare la terminazione anomala di altri programmi. Per impedire il verificarsi di tali eventi, ogni soggetto deve ottenere l'autorizzazione all'accesso agli oggetti del sistema.
Un dominio di protezione è un insieme di diritti d'accesso. Ogni diritto d'accesso in un dominio di protezione è rappresentato da una coppia ordinata con un campo dedicato al nome dell'oggetto ed ai corrispondenti privilegi. Un dominio di protezione è unico per un soggetto.
I diritti d'accesso più comuni sono quelli di lettura, scrittura ed esecuzione. Alcuni soggetti possono anche garantire diritti d'accesso ad altri soggetti. Nella maggior parte dei sistemi di elaborazione, l'amministratore detiene tutti i diritti d'accesso ed è responsabile della gestione dei diritti degli altri. I diritti d'accesso sono copiabili, trasferibili o propagabili da un dominio all'altro.
La copia dei diritti d'accesso implica semplicemente la garanzia ad un utente del diritto d'accesso di un altro utente. Quando un diritto d'accesso è trasferito dal soggetto A al soggetto B, il diritto del soggetto A è revocato al completamento del trasferimento. La propagazione del diritto d'accesso è simile alla copia; oltre a condividere il diritto d'accesso originale, entrambi i soggetti possono anche copiare i diritti d'accesso di altri soggetti.
Quando un soggetto non necessita più l'accesso ad un oggetto, i diritti d'accesso sono revocabili.

Modelli e politiche di controllo degli accessi

Il controllo degli accessi è suddivisibile in tre livelli concettuali: modelli, politiche e meccanismi.
Un modello di sicurezza definisce i soggetti, gli oggetti ed i privilegi di un sistema.
Una politica di sicurezza, normalmente specificata dall'utente o dall'amministratore di sistema definisce quali privilegi relativi agli oggetti sono assegnati ai soggetti. Il meccanismo di sicurezza è il metodo con cui il sistema realizza la politica di sicurezza.

In molti sistemi la politica si modifica nel tempo a mano a mano che cambiano l'insieme delle risorse di sistema e degli utenti; il modello di sicurezza ed i meccanismi realizzati dal controllo d'accesso non richiedono modifica; la politica di sicurezza è quindi separata dal meccanismo e dai modelli.
Nel modello di controllo degli accessi basato sui ruoli (Role-Based Access Control, RBAC), agli utenti vengono assegnati ruoli; ad ogni ruolo è attribuito un insieme di privilegi che definisce gli oggetti a cui un utente ricoprente quel ruolo può accedere. Gli utenti possono appartenere a diversi ruoli. La caratteristica interessante di RBAC è che assegna relazioni significative tra soggetti ed oggetti non limitate a classi quali proprietari e gruppi.
Sebbene le politiche di sicurezza varino a seconda delle esigenze degli utenti nel sistema, la maggior parte incorpora il principio del privilegio minore: ad un soggetto sono garantiti i diritti d'accesso solo agli oggetti strettamente necessari per il completamento delle operazioni. Le politiche possono anche realizzare un controllo degli accessi a discrezione o obbligatorio, in base alle necessità di sicurezza dell'ambiente. La maggior parte dei sistemi basati su UNIX adottano il modello di controllo discrezionale degli accessi (Discretionary Access Control, DAC), dove il creatore di un oggetto controlla i permessi per quell'oggetto. Al contrario, le politiche di controllo obbligatorio degli accessi (Mandatory Access Control, MAC) predefiniscono uno schema di permessi centralizzato tramite cui sono controllati tutti i soggetti e gli oggetti.

Meccanismi di controllo degli accessibile

- Matrici di controllo degli accessi: i vari soggetti sono elencati nelle righe; mentre gli oggetti a cui essi richiedono accesso sono elencati nelle colonne. Ogni cella della matrice specifica le azioni che il soggetto, definito dalla riga, può svolgere sull'oggetto, definito dalla colonna. I diritti d'accesso di una matrice di controllo degli accessi sono garantiti sulla base del privilegio minimo: in questo modo se un diritto d'accesso non è esplicitamente descritto nella matrice, l'utente non ha quel diritto per quell'oggetto. Poiché una matrice di controllo degli accessi posiziona tutte le informazioni relative ai permessi in una posizione centrale, essa dovrebbe essere una delle entità più protette del SO. Se la matrice è compromessa, ogni risorsa protetta dai diritti d'accesso da essa definiti è suscettibile di attacco. Sebbene la generazione e l'interpretazione della matrice di controllo degli accessi sia semplice, può assumere dimensioni considerevoli ed essere piuttosto vuota.
- Liste di controllo degli accessi: memorizza gli stessi dati di una matrice di controllo degli accessi, ma mantiene solo le voci che specificano un diritto d'accesso realmente attribuito. La lista di controllo degli accessi di un sistema può basarsi sulle righe di una matrice, ovvero sui soggetti, o sulle colonne, quindi sugli oggetti. Per ogni oggetto gestito dal SO, la lista di controllo degli accessi contiene voci che specificano i privilegi di cui ogni soggetto gode sull'oggetto. Quando un soggetto tenta di accedere ad un oggetto, il sistema cerca nella lista di controllo degli accessi i privilegi di quel soggetto sul determinato oggetto. Uno svantaggio è dato dall'inefficienza con cui il SO determina i privilegi degli utenti per un particolare oggetto. La lista di controllo degli accessi per ogni oggetto contiene una voce per ogni soggetto che ha dei privilegi su quell'oggetto: potenzialmente si tratta di un elenco molto lungo. Ogni volta che si richiede accesso ad un oggetto, il sistema deve cercare in questo elenco per individuare i privilegi riguardanti l'utente che ha inoltrato la richiesta. È difficile determinare quali diritti d'accesso appartengano ad un determinato dominio di protezione tramite le liste di controllo degli accessi: si dovrebbero cercare nella lista di ogni oggetto le voci riguardanti quel particolare soggetto.
- Liste di capacità: una capacità è un riferimento o un token che garantisce i privilegi ad un soggetto che lo possiede. Le capacità solitamente non sono modificate, però possono essere riprodotte. Si ricordi che un dominio di protezione definisce l'insieme di privilegi tra soggetti ed oggetti. In alternativa, è possibile definire il dominio di protezione come l'insieme delle capacità che appartengano ad un soggetto. Una capacità è solitamente

realizzata come un identificatore d’oggetto univoco. Le capacità sono garantite ai soggetti che presentano il token per tutti gli accessi successivi all’oggetto; sono create da routine di SO particolarmente protette. Un soggetto in possesso di una capacità può eseguire determinate operazioni, inclusa la creazione di copie della capacità o il passaggio della stessa come parametro. Una capacità è creata a fronte della creazione di un oggetto; questa capacità originale include tutti i privilegi sul nuovo oggetto. Il soggetto che crea l’oggetto può passare copie di questa capacità anche ad altri soggetti, anche riducendone i privilegi associati. Una capacità può quindi propagarsi all’intero sistema, mantenendo lo stesso insieme di privilegi o riducendolo.
Gli utenti devono impedire la creazione arbitraria di capacità, memorizzando le capacità in segmenti in accessibili ai processi utente. L’identificatore in una capacità è realizzabile come un riferimento all’oggetto desiderato, il che semplifica l’accesso all’indirizzo in cui la risorsa è memorizzata, ma se l’oggetto viene spostati bisogna aggiornare tutti i riferimenti nel sistema (conseguente degrado delle prestazioni). Altrimenti, è possibile realizzare l’identificatore attraverso una sequenza univoca di bit, ad esempio un token; così le capacità non si basano sulla posizione dell’oggetto in memoria. Poiché un token non specifica la posizione dell’oggetto a cui fa riferimento, richiede che l’indirizzo dell’oggetto sia stabilito la prima volta che è usato il token. Un meccanismo di mascheramento (hashing) realizza efficientemente le capacità basate sui token; le memorie cache ad alta velocità spesso riducono il costo ulteriore dovuto a ripetute richieste d’accesso allo stesso oggetto. I sistemi che impiegano le capacità risentono del problema dell’oggetto perduto; se viene distrutta l’ultima capacità di un oggetto, l’oggetto associato non sarà più accessibile. Per prevenire questo problema, molti SO assicurano che il sistema mantenga sempre almeno una capacità per ogni oggetto.
I sistemi generalmente non permettono agli utenti la manipolazione diretta delle capacità; la manipolazione è invece eseguita dal SO a nome degli utenti. Tener traccia delle capacità è un’operazione importante, che diventa complessa in sistemi multiutente contenenti un gran numero di capacità. Molti sistemi impiegano una directory per la gestione delle proprie capacità.