

Sistemi Operativi Teoria

Parte 4 – File System e gestione I/O

Appunti a cura di Liccardo Giuseppe
Università degli Studi di Napoli “Parthenope”



Indice – Parte 4

| | |
|--|-----------|
| CAPITOLO 12. File System | 3 |
| 12.1 Elaborazione dei file: panoramica | 3 |
| 12.1.1 <i>File system e IOCS</i> | 3 |
| 12.1.2 <i>Elaborazione dei file in un programma</i> | 4 |
| 12.2 File e operazioni sui file | 5 |
| 12.3 Fondamenti dell'organizzazione dei file e metodi di accesso | 6 |
| 12.3.1 <i>Organizzazione sequenziale dei file</i> | 6 |
| 12.3.2 <i>Organizzazione diretta dei file</i> | 6 |
| 12.3.3 <i>Organizzazione indicizzata dei file</i> | 6 |
| 12.3.4 <i>Metodi di accesso</i> | 7 |
| 12.4 Directory | 7 |
| 12.4.1 <i>Struttura a due livelli</i> | 8 |
| 12.4.2 <i>Struttura ad albero</i> | 9 |
| 12.4.3 <i>Struttura a grafi</i> | 9 |
| 12.4.4 <i>Operazioni sulle directory</i> | 10 |
| 12.4.5 <i>Organizzazione delle directory</i> | 10 |
| 12.5 Montaggio dei file system | 10 |
| 12.6 Protezione dei file | 10 |
| 12.7 Allocazione dello spazio su disco | 11 |
| 12.7.1 <i>Allocazione contigua</i> | 11 |
| 12.7.2 <i>Allocazione non contigua</i> | 11 |
| 12.7.3 <i>Allocazione concatenata</i> | 11 |
| 12.7.4 <i>Allocazione indicizzata</i> | 13 |
| 12.8 Affidabilità del file system | 15 |
| 12.9 Journaling file system – JFS | 15 |
| 12.10 File system virtuale – FSV | 16 |
| 12.11 File system di Unix | 16 |
| 12.11.1 <i>I-node, descrittore di file e struttura file</i> | 16 |
| 12.11.2 <i>Allocazione dello spazio su disco</i> | 17 |
| 12.11.3 <i>Superblocco</i> | 17 |
| CAPITOLO 13. Implementazione delle operazioni su file | 18 |
| 13.1 Livelli dell'input-output control system..... | 18 |
| 13.2 Panoramica dell'organizzazione dell'I/O | 18 |
| 13.3 Dispositivi di I/O..... | 19 |
| 13.3.1 <i>Controller di dispositivo</i> | 20 |
| 13.3.2 <i>I/O mediante DMA</i> | 20 |
| 13.3.3 <i>I/O mappato in memoria</i> | 20 |
| 13.4 Dischi magnetici..... | 21 |
| 13.4.1 <i>Parametri di prestazioni del disco</i> | 21 |

| | |
|---|-----------|
| 13.5 RAID | 22 |
| 13.5.1 RAID di livello 0..... | 24 |
| 13.5.2 RAID di livello 1..... | 24 |
| 13.5.3 RAID di livello 2..... | 25 |
| 13.5.4 RAID di livello 3..... | 25 |
| 13.5.5 RAID di livello 4..... | 26 |
| 13.5.6 RAID di livello 5..... | 27 |
| 13.5.7 RAID di livello 6..... | 27 |
| 13.5.8 RAID ibridi: RAID 0 + 1 e RAID 1 + 0..... | 28 |
| 13.6 Principi del software di I/O | 28 |
| 13.6.1 Scopi del software di I/O | 29 |
| 13.6.2 I/O programmato..... | 29 |
| 13.6.3 I/O guidato dalle interruzioni..... | 30 |
| 13.6.4 I/O mediante DMA | 30 |
| 13.6.5 DMA breakpoint e interrupt..... | 30 |
| 13.7 Driver di dispositivo | 31 |
| 13.8 Scheduling del disco | 31 |
| 13.9 Buffering e blocking dei record | 32 |
| 13.10 Cache del disco e dei file | 33 |
| CAPITOLO 14. Sicurezza e protezione | 34 |
| 14.1 Sicurezza e protezione: introduzione | 34 |
| 14.1.1 Obiettivi di sicurezza e protezione..... | 35 |
| 14.1.2 Minacce alla sicurezza e alla protezione | 35 |
| 14.2 Attacchi alla sicurezza..... | 36 |
| 14.2.1 Trojan, virus e worm..... | 36 |
| 14.3 Cifratura..... | 37 |
| 14.3.1 Attacchi ai sistemi di crittografia | 39 |
| 14.3.2 Tecniche di cifratura..... | 39 |
| 14.3.3 DES – Data Encryption Standard | 40 |
| 14.3.4 AES – Advanced Encryption Standard | 41 |
| 14.3.4 RSA – Rivest-Shamir-Adleman..... | 42 |
| 14.3.5 Algoritmo dello zaino | 43 |
| 14.4 Strutture di protezione | 43 |
| 14.4.1 Granularità della protezione | 43 |
| 14.4.2 Matrice di controllo degli accessi - ACM | 44 |
| 14.4.3 Liste di controllo degli accessi (ACL) | 44 |
| 14.4.4 Capability list (C-list)..... | 44 |
| 14.4.5 Dominio di protezione | 45 |
| 14.5 Unix..... | 45 |

CAPITOLO 12. File System

Tutte le applicazioni hanno bisogno di memorizzare e rintracciare informazioni e dati. Durante l'esecuzione, un processo può memorizzare nella RAM solo una parte di queste informazioni, la restante parte è memorizzata su dischi e/o supporti esterni. Quella parte del SO che si occupa complessivamente dei file è chiamata **file system**.

Un amministratore di sistema si aspetta che un file system assicuri un uso efficiente dei dispositivi di I/O e contribuisca alle alte prestazioni del sistema. Gli utenti, invece, richiedono efficienza nella creazione e nella manipolazione dei file e nella condivisione degli stessi con altri utenti del sistema. Inoltre richiedono che il file system implementi caratteristiche di protezione, sicurezza ed affidabilità in modo che i propri file non siano soggetti ad accessi illegali da parte di altri utenti o siano danneggiati da malfunzionamenti del sistema.

Pertanto, gli obiettivi principali di un file system sono:

- accesso conveniente e veloce ai file
- memorizzazione affidabile dei file
- condivisione dei file con altri utenti

Per raggiungere questi obiettivi in maniera efficace, il file system è strutturato in due strati:

- i moduli del file system che si occupano della condivisione, della protezione e dell'affidabilità dei file
- l'IOCS che si occupa dell'implementazione delle operazioni sui file

Questo capitolo discute i file e i file system dal punto di vista del programmatore.

Describe le fondamentali *organizzazioni dei file*, la *struttura delle directory*, le *operazioni su file e directory* e la *condivisione dei file*. Vengono anche affrontate le problematiche che compromettono l'affidabilità di un file system, e descritte le fault tolerance e il ripristino mediante backup.

12.1 Elaborazione dei file: panoramica

12.1.1 File system e IOCS

I moduli di un file system vedono un file come un insieme di dati:

1. di cui è proprietario un utente
2. che può essere *condiviso* da un insieme di utenti autorizzati
3. che deve essere *memorizzato in modo affidabile* per un periodo esteso di tempo. Inoltre, un file system permette agli utenti di dare un nome ai file senza che questo vada in conflitto con i nomi dei file di altri utenti

L'IOCS vede un file come contenitore di dati:

1. cui è necessario accedere velocemente
2. che sono memorizzati su un dispositivo di I/O che deve essere utilizzato in maniera efficiente

File system

- Strutture delle directory per il raggruppamento conveniente dei file
- Protezione dei file contro gli accessi illegittimi
- Semantica condivisione dei file per gli accessi concorrenti a un file
- Memorizzazione affidabile dei file

Sistema di controllo input-output (IOCS)

- Funzionamento efficiente dei dispositivi di I/O
- Accesso efficiente ai dati in un file

La tabella elenca gli strumenti forniti dal file system e dall'IOCS.

Il file system e il sistema IOCS costituiscono una gerarchia in cui ognuno adotta una politica differente e fornisce meccanismi per implementare tali comportamenti.

DATI E METADATI

Un file system consiste di due tipi di dati – i dati contenuti nei file e i dati usati per accedere ai file.

I dati contenuti nei file vengono chiamati ***dati***; i dati usati per accedere ai file sono chiamati ***metadati*** (o *dati di controllo*).

12.1.2 Elaborazione dei file in un programma

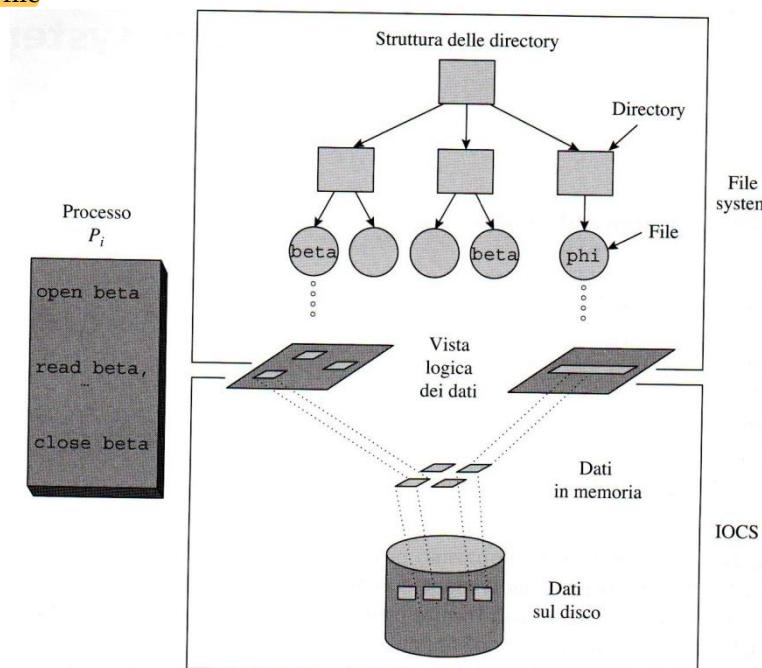
A livello di linguaggio di programmazione, un file è un oggetto che possiede ***attributi*** che descrivono l'organizzazione dei suoi dati e il metodo di accesso agli stessi.

Un programma contiene le *istruzioni per l'elaborazione dei file*, cioè l'istruzione per dichiarare un file, per specificare i suoi attributi, per aprirlo, per eseguire le operazioni di lettura/scrittura e per chiuderlo.

Durante l'esecuzione del programma, l'elaborazione dei file è di fatto implementata dai moduli di libreria del file system e del sistema IOCS.

Col termine ***elaborazione dei file*** si indica la sequenza generale delle operazioni di:

- apertura del file
- lettura dei dati dal file o scrittura dei dati nel file
- chiusura del file



La figura mostra l'organizzazione attraverso la quale un SO implementa le attività di elaborazione dei file dei processi.

Ogni ***directory*** contiene elementi che descrivono alcuni ***file***: nome del proprietario, posizione nel disco, il modo in cui i dati sono organizzati, permessi di accesso per i vari utenti, ecc.

Un file system fornisce diversi ***tipi di file***: un file può essere un file **strutturato**, ovvero può contenere record di dati, o può essere un file **non strutturato, detto anche stream di byte**. Ogni tipo di file fornisce la propria vista astratta dei dati in un file, che chiamiamo ***vista logica dei dati***.

L'IOCS organizza i dati di un file in un dispositivo di I/O relativamente al tipo di file. Questa è denominata ***vista fisica dei dati*** di un file. L'IOCS effettua la mappatura tra la vista logica e la vista fisica dei dati dei file. Inoltre, fornisce un'organizzazione che velocizza l'attività di elaborazione di un file.

12.2 File e operazioni sui file

TIPI DI FILE

Un **file system** contiene e organizza diversi tipi di file, come per esempio i file dati, programmi eseguibili, documenti, fogli di calcoli, file audio, ecc. Ognuno di questi file ha il proprio formato per memorizzare i dati. Fatto sta che questi tipi di file possono essere raggruppati in due classi:

- **file strutturati**
- **file orientati allo stream di byte**

Un **file strutturato** è una collezione di record. Un **record** è una collezione di campi e un **campo** contiene un **singolo elemento dei dati**. Si assume che ogni record in un file contenga un campo chiave, il cui valore è unico nel file, cioè non esistono due record che contengono la stessa chiave.

Un **file orientato allo stream di byte** è un file che non contiene né record né campi, ma viene visto semplicemente come una sequenza di byte dai processi che lo usano.

ATTRIBUTI DI UN FILE

Un attributo di un file è un'importante caratteristica di un file sia per gli utenti che per il file system. Per comodità degli utenti, ogni file ha un nome che si usa come riferimento. Alcuni sistemi, nella composizione dei nomi, distinguono tra maiuscole e minuscole, altri le considerano equivalenti.

Un file ha altri attributi che possono variare secondo il SO, ma che tipicamente comprendono:

- **Nome** – Il **nome simbolico** del file è l'unica informazione in forma umanamente leggibile
- **Identificatore** – Si tratta solitamente di un numero che **identifica il file all'interno del file system**
- **Tipo** – Questa informazione è necessaria ai sistemi che gestiscono tipi di file diversi
- **Dimensione** – Si tratta della dimensione corrente del file ed, eventualmente, di quella massima
- **Posizione sul disco** – Si tratta del "percorso" in cui è memorizzato il file
- **Protezione** – Le **informazioni utili per l'accesso al file in lettura, scrittura ed esecuzione**

Le informazioni sui file sono conservate nella struttura di **directory**, che risiede a sua volta su un disco. Durante l'elaborazione di un file, il file system usa gli attributi di un file per localizzarlo e assicurare che ogni operazione da effettuare su di esso sia consistente con gli attributi.

OPERAZIONI SU FILE

Il SO può offrire chiamate di sistema per creare, scrivere, leggere, spostare, cancellare e troncare un file. La seguente tabella descrive queste operazioni. Ricordiamo che le operazioni come apertura, chiusura, ridenominazione e cancellazione sono eseguite dai moduli del file system, mentre l'accesso (lettura/scrittura) viene implementato dai moduli del sistema IOCS.

| Operazione | Descrizione |
|------------------------------------|--|
| Apertura di un file | Il file system recupera l'elemento della directory corrispondente al file e controlla se l'utente il cui processo sta cercando di aprire il file ha i privilegi di accesso necessari per il file. Successivamente, esegue alcune azioni di gestione per avviare l'elaborazione del file. |
| Leggere o scrivere un record | Il file system considera l'organizzazione del file (Paragrafo 13.3) e implementa le operazioni di lettura/scrittura in maniera appropriata. |
| Chiusura di un file | L'informazione relativa alla dimensione del file nell'elemento della directory relativo al file viene aggiornata. |
| Copia di un file | Viene eseguita una copia del file, viene creato un nuovo elemento della directory per la copia e il suo nome, la sua dimensione, la posizione e le informazioni di protezione vengono memorizzate nella voce corrispondente. |
| Cancellazione del file | L'elemento della directory relativo al file viene cancellato e l'area sul disco occupata viene liberata. |
| Ridenominazione del file | Il nuovo nome viene registrato nell'elemento della directory relativo al file. |
| Specificare i privilegi di accesso | Le informazioni di protezione contenute nell'elemento della directory relativo al file vengono aggiornate. |

12.3 Fondamenti dell'organizzazione dei file e metodi di accesso

Un'organizzazione dei file è una combinazione di due caratteristiche: un modo per organizzare i record in un file e una procedura per accedervi.

Esistono vari modi per accedere ai file salvati sul computer. I due metodi fondamentali di accesso ai record sono l'accesso sequenziale, secondo cui l'accesso ai record avviene nell'ordine in cui si trovano nel file (o nell'ordine inverso) e l'accesso casuale, secondo cui si può accedere ai record in qualsiasi ordine.

Chiamiamo "pattern di accesso al record" l'ordine in cui un processo accede ai record in un file. Le azioni di elaborazione dei file saranno eseguite in modo efficiente solo se il pattern di accesso ai record da parte del processo può essere implementato in maniera efficiente nel file system. Per raggiungere questo obiettivo occorre tener conto delle caratteristiche del dispositivo di I/O che si prestano meglio ad un metodo piuttosto che ad un altro.

In definitiva, l'organizzazione dei file viene scelta in base alle caratteristiche del dispositivo di I/O che si usa, in modo tale da fornire un accesso efficiente. Ad esempio, un hard disk può accedere direttamente, mediante l'indirizzo, a qualunque record, mentre un drive a nastro può accedere al record solo in modo sequenziale.

Ora descriveremo tre organizzazioni fondamentali dei file usate dal file system: organizzazione sequenziale dei file, organizzazione diretta dei file, organizzazione indicizzata dei file. Gli accessi ai file sono implementati da un modulo del sistema IOCS chiamato *metodo di accesso*.

12.3.1 Organizzazione sequenziale dei file

Nell'organizzazione sequenziale dei file le informazioni sono memorizzate in ordine crescente o decrescente in base al campo chiave. Di conseguenza l'elaborazione di queste informazioni supporta solo due operazioni:

- legge l'informazione (record) successiva (o precedente)
- salta l'informazione (record) precedente (o successiva)

Un file ad accesso sequenziale viene utilizzato nelle applicazioni se i suoi dati possono essere pre-ordinati convenientemente in ordine crescente o decrescente.

12.3.2 Organizzazione diretta dei file

L'organizzazione diretta dei file fornisce efficienza e convenienza perché permette di accedere alle informazioni (record) in ordine casuale. Di conseguenza l'elaborazione di queste informazioni permette di leggere o scrivere byte senza alcun ordine particolare. Il metodo di accesso diretto si fonda su un modello di file che si rifà al disco: i dischi permettono, infatti, l'accesso diretto ad ogni blocco di file.

In questa organizzazione occorre generare l'indirizzo del record usato dalla periferica. Se il file è memorizzato su un hard disk, la trasformazione genera un indirizzo (*num_traccia, num_record*) in modo tale che le testine dell'hard disk vengono posizionate sulla traccia *num_traccia* prima che venga eseguita l'operazione di lettura o scrittura sul record *num_record*.

Abbiamo detto che questo tipo di organizzazione fornisce efficienza e convenienza, ma presenta anche due svantaggi rispetto all'organizzazione sequenziale:

- il calcolo dell'indirizzo del record consuma tempo di CPU
- una parte della memoria viene sprecata in quanto vengono memorizzati meno dati sulla traccia esterna del disco rispetto a quelli realmente memorizzabili

12.3.3 Organizzazione indicizzata dei file

Nell'organizzazione indicizzata dei file un indice aiuta a determinare la posizione di un record a partire dal valore della sua chiave.

Nell'**organizzazione indicizzata pura**, esiste un indice per ogni record. Un indice di un file è costituito dalla coppia (*valore della chiave, indirizzo del disco*). Per accedere a un record con chiave k , viene trovato l'elemento indice contenente k tramite la ricerca per indice, e si utilizza l'indirizzo del disco annotato nell'elemento trovato per accedere al record.

L'**organizzazione sequenziale indicizzata** è un'organizzazione ibrida che combina gli elementi delle organizzazioni dei file indicizzata e sequenziale: esiste un indice per ogni sezione del disco. Per accedere ad un record, si cerca un indice che punta ad una sezione del disco che può contenere il record. Poi si effettua la ricerca sequenziale dei record in questa sezione del disco per trovare il record desiderato. La ricerca ha buon esito se il record è presente nel file; altrimenti avrà esito negativo. Questa organizzazione richiede un indice molto più piccolo rispetto all'indicizzazione pura poiché l'indice contiene elementi solo per alcuni valori della chiave.

Per un file di grandi dimensioni l'indice può contenere un gran numero di elementi, per cui il tempo richiesto per la ricerca mediante l'indice può risultare molto lungo. Si può utilizzare una "gerarchia di indici": un indice di livello più alto può essere utilizzato per ridurre il tempo di ricerca; un elemento nell'indice di livello più alto punta a una sezione dell'indice.

12.3.4 Metodi di accesso

Un **metodo di accesso** è un modulo del sistema IOCS che implementa gli accessi a una classe di file che utilizza una specifica organizzazione dei file. Il tipo di accesso da utilizzare per accedere ai record in un file dipende dall'organizzazione del file.

E' possibile utilizzare alcune tecniche di programmazione dell'I/O per rendere più efficiente l'accesso al file, due di queste sono il *buffering* e il *blocking* dei record.

BUFFERING DEI RECORD

Il metodo di accesso legge i record di un file di input prima che siano effettivamente richiesti da un processo e li memorizza temporaneamente in aree di memoria chiamate *buffer* finché non vengono richiesti dal processo. Lo scopo del buffering è quello di ridurre o eliminare l'attesa per il completamento di un'operazione di I/O.

BLOCKING DEI RECORD

Il metodo di accesso legge o scrive sempre grandi blocchi di dati, che contengono diversi record di file, da o verso un dispositivo di I/O. Questa caratteristica riduce il numero totale di operazioni di I/O richieste per elaborare un file, migliorando di conseguenza l'efficienza nell'elaborazione di un file da parte di un processo. Il blocking, inoltre, migliora l'utilizzo di un dispositivo di I/O e il throughput di un dispositivo.

12.4 Directory

Un elemento molto importante dei file system sono le **directory**. Una directory contiene le informazioni relative a un gruppo di file. Ogni elemento in una directory contiene gli attributi di un file, come il nome, il tipo, la dimensione, i permessi, ecc.

| Nome del file | Tipo e dimensione | Informazioni sulla posizione | Informazioni sulla protezione | Open count | Lock | Flag | Misc info |
|---------------|-------------------|------------------------------|-------------------------------|------------|------|------|-----------|
| | | | | | | | |

| Campo | Descrizione |
|-------------------------------|---|
| Nome del file | Nome del file. Se questo campo ha dimensione fissa, i nomi lunghi oltre una certa lunghezza saranno troncati. |
| Tipo e dimensione | Il tipo e la dimensione del file. In molti file system, il tipo di file è implicito nella sua estensione; per esempio, un file con estensione .c è un file che contiene un programma C e un file con estensione .obj è un file oggetto, che spesso è un file strutturato. |
| Informazioni sulla posizione | Informazioni sulla posizione del file sul disco. Queste informazioni sono tipicamente sotto forma di una tabella o di una lista concatenata contenente gli indirizzi dei blocchi sul disco allocati al file. |
| Informazioni sulla protezione | Le informazioni relative agli utenti cui è concesso l'accesso a questo file e in che modo. |
| Open count | Numero di processi che attualmente accedono al file. |
| Lock | Indica se un processo sta accedendo al file in maniera esclusiva. |
| Flag | Informazioni sulla natura del file, quali se il file è una directory, un link o un file system montato. |
| Misc info | Informazioni varie come l'id del proprietario, la data e l'ora della creazione, l'ultimo accesso e l'ultima modifica. |

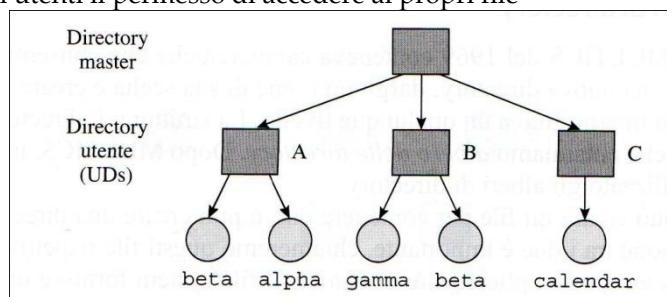
La figura mostra i campi di un tipico elemento di una directory.

I campi *Open count* e *Lock* sono usati quando diversi processi aprono un file in maniera concorrente. Il campo *Lock* viene usato quando un processo richiede l'accesso esclusivo al file. Il campo *Flag* è usato per differenziare i diversi tipi di file di una directory (D=directory, L=link, M=file system). Il campo *Misc_Info* contiene informazioni aggiuntive come il proprietario, la data di creazione e ultima modifica.

12.4.1 Struttura a due livelli

Una directory contiene i file appartenenti a diversi utenti, quindi deve garantire due importanti prerogative:

- libertà nella scelta del nome – possibilità per gli utenti di dare gli stessi nomi ai propri file
- condivisione dei file – possibilità per un utente di accedere ai file creati da altri utenti e possibilità di concedere ad altri utenti il permesso di accedere ai propri file



La figura mostra una semplice struttura di directory, chiamata **struttura di directory a due livelli**, dove con i rettangoli sono indicate le directory, mentre con i cerchi sono indicati i file.

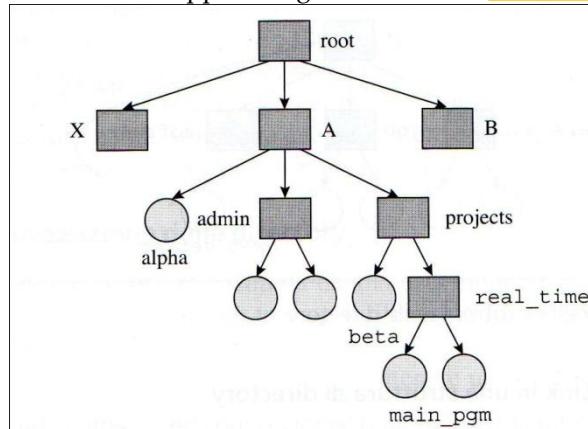
Questa struttura presenta due tipi di directory:

- la *directory master* contiene informazioni relative alle directory utente di tutti gli utenti del sistema; ogni elemento di un directory master è una coppia che consiste di un ID utente e di un puntatore a una directory utente
- una *directory utente* (UD) contiene elementi che descrivono i file appartenenti a un utente

Questa soluzione richiede che i nomi siano unici solo all'interno dell'area del singolo utente. Infatti l'uso di UD separate permette la libertà nell'assegnazione di un nome ad un file, infatti la figura mostra che vi sono più file con lo stesso nome (beta). Una tale organizzazione garantisce l'accesso al file corretto anche se nel sistema esistono molti file con lo stesso nome. Tuttavia l'utilizzo delle UD ha uno svantaggio: impedisce agli utenti di condividere i loro file con altri utenti. Per raggiungere tale obiettivo sono necessarie istruzioni speciali che permettono a degli utenti di accedere ai file di altri utenti; per fare ciò occorre verificare i permessi dei file (che si trovano nel campo *Prot_Info*).

12.4.2 Struttura ad albero

Un approccio più potente e flessibile è l'approccio gerarchico o *ad albero delle directory*.



In questa struttura, il file system fornisce all'utente una directory chiamata *root* che contiene la *directory home* per ciascun utente, ovvero una directory che, solitamente, ha lo stesso nome del nome utente. Un utente può creare file oppure può creare directory all'interno della sua *home*, con la possibilità di organizzazione le proprie informazioni in una struttura basata su diversi livelli di *sottodirectory*.

Ad ogni istante, si dice che un utente si trova in una specifica directory, chiamata *directory corrente*. Quando l'utente vuole aprire un file, il file viene cercato in questa directory. Quando l'utente effettua il login, il SO permette all'utente di operare solo nella sua directory *home*.

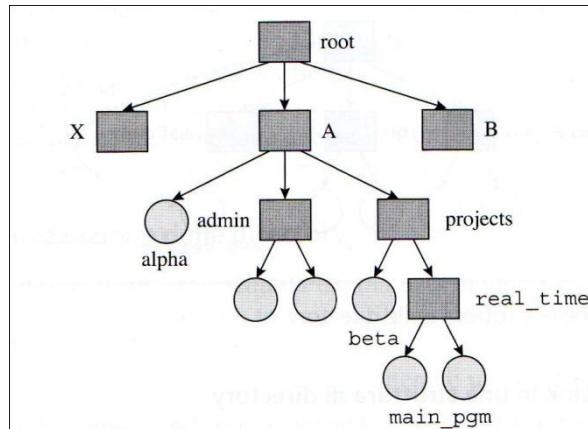
Il nome dato a uno specifico file può non essere unico nel file system, per cui un utente o un processo utilizza un *path* per identificarlo in maniera univoca. Il path è il percorso di tutte le sottodirectory in cui si trova il file. Nel path ogni riferimento è un directory tranne l'ultima che è, appunto, il file stesso.

I path per localizzare un file a partire dalla directory corrente sono detti *path relativi*, che sono spesso corti e convenienti da usare; tuttavia, possono essere fonte di confusione poiché un file può avere diversi path relativi quando vi si accede a partire da diverse directory correnti.

Il *path assoluto* di un file parte dalla directory *root* dell'albero delle directory del file system. I file con lo stesso nome creati in directory differenti differiscono nei rispettivi path assoluti.

12.4.3 Struttura a grafi

In un albero di directory, ogni file eccetto la directory *root* ha esattamente una directory genitore. Una struttura del genere separa totalmente i file di utenti differenti; in pratica non ammette la condivisione di file e directory tra più utenti.



Questo problema può essere risolto organizzando le directory in una *struttura a grafo aciclico*. Un grafo aciclico permette di avere sottodirectory e file condivisi. Il fatto che il file sia condiviso non significa che ci siano due copie del file.

In questa struttura, un file può avere molte directory genitore, per cui un file condiviso può essere puntato dalle directory di tutti gli utenti che hanno accesso ad esso. Le strutture a grafo aciclico possono essere implementate in vari modi.

LINK

Un metodo molto diffuso prevede la creazione di un nuovo elemento di directory, chiamato *link* (o *collegamento*). Un *link* è un puntatore ad un altro file o directory, in pratica è una connessione diretta tra due file esistenti.

12.4.4 Operazioni sulle directory

La ricerca di informazioni o file è l'operazione più frequente sulle directory. Altre operazioni sulle directory sono operazioni come la creazione o la cancellazione di file, l'aggiornamento degli elementi relativi ai file quando questo viene chiuso da un programma, l'elencazione del contenuto di una directory o la cancellazione della stessa.

L'operazione di cancellazione risulta complessa quando la struttura è un grafo poiché un file può avere molti genitori. Un file è cancellato quando ha un solo genitore; altrimenti viene semplicemente reso inaccessibile dalla directory che si vuole cancellare. Per semplificare l'operazione di cancellazione, il file system associa ad ogni file un contatore di link: il contatore è impostato a 1 quando viene creato il file, viene incrementato di 1 quando un link punta al file, viene decrementato di 1 quando viene eseguita una cancellazione. Il file viene cancellato definitivamente quando il contatore dei link diventa 0.

12.4.5 Organizzazione delle directory

Una directory dovrebbe essere costituita da una lista lineare in modo da poter effettuare ricerche lineari per trovare il file richiesto. Tuttavia, questa organizzazione è inefficiente se la directory contiene un elevato numero di elementi.

Per avere una maggiore efficienza, cioè per diminuire notevolmente il tempo di ricerca nelle directory, sono usate organizzazioni che fanno uso di una tabella hash o di un albero B+.

12.5 Montaggio dei file system

In un sistema operativo possono convivere più file system. Ogni file system è creato su un disco logico, ovvero su una partizione del disco. I file contenuti in un file system possono essere utilizzati solo quando il file system è *montato*.

L'operazione di *montaggio* (*mount*) "connette" il file system alla struttura delle directory del sistema. Questa operazione è utile quando ci sono più file system nel sistema e dura finché il file system non viene smontato o finché il sistema non viene riavviato.

L'operazione di *smontaggio* (*unmount*) "disconnette" un file system.

12.6 Protezione dei file

Il SO deve anche implementare la protezione dei file, in particolare deve dare la possibilità ad un utente di voler condividere i suoi file solo con una parte degli altri utenti del sistema. Questa esigenza è detta *condivisione controllata* e viene implementata in questo modo: il proprietario del file specifica quali utenti possono accedere al file e in che modo. Queste informazioni sono contenute nel campo *Protection_Info* dell'elemento della directory relativo al file.

L'informazione relativa alla protezione è solitamente memorizzata nella forma di una *access control list* (ACL). Ogni elemento della ACL è una coppia (*nome utente, lista dei privilegi di accesso*). Quindi prima di eseguire un'operazione viene anche controllata l'informazione relativa alla protezione, in particolare viene controllato se l'utente può accedere a quel file, e in tal caso, con quale accesso.

La dimensione della ACL di un file dipende dal numero di utenti e dal numero di privilegi di accesso definiti nel sistema. Per ridurre le dimensioni, è possibile specificare una ACL per ogni classe di utenti piuttosto che per ogni utente. In questo modo una ACL ha solo tante coppie quante sono le classi di utenti. Per esempio, Unix specifica i privilegi di accesso per tre classi di utenti: il proprietario del file, gli utenti nello stesso gruppo del proprietario, tutti gli altri utenti del sistema.

In molti file system i privilegi sono di tre tipi: lettura, scrittura ed esecuzione.

12.7 Allocazione dello spazio su disco

Come menzionato nei paragrafi precedenti, un disco può contenere molti file system, ognuno nella sua partizione del disco. Il file system ha informazioni sulla partizione in cui è presente un file, ma il sistema IOCS no; dunque l'allocazione dello spazio sul disco è eseguita dal file system.

12.7.1 Allocazione contigua

I primi file system usavano il modello di *allocazione contigua* della memoria, cioè allocavano una singola area di memoria a ogni file al momento della creazione.

In questa allocazione ogni file occupa un insieme di blocchi contigui sul disco, quindi questa allocazione risulta essere particolarmente semplice poiché basta conoscere il blocco iniziale e la lunghezza del file.

Questa allocazione porta però alla *frammentazione esterna*, cioè si hanno aree di memoria troppo piccole per poter essere riutilizzate; ma portava anche alla *frammentazione interna* poiché il file system era progettato per allocare spazio extra sul disco per consentire ai file di crescere.

12.7.2 Allocazione non contigua

I moderni file system adottano l'*allocazione non contigua* della memoria per l'allocazione dello spazio sul disco. In questo approccio, una parte di spazio sul disco è allocata *su richiesta*, ovvero, quando viene creato un file oppure quando la sua dimensione aumenta a seguito di un'operazione di aggiornamento.

Il file system deve risolvere tre problemi per implementare questo approccio:

- *gestione dello spazio libero sul disco*: deve tenere traccia dello spazio libero sul disco e allocarlo quando un file richiede un nuovo blocco del disco
- *evitare movimenti eccessivi della testina del disco*: garantire che un file non sia "sparpagliato" in diverse parti del disco, poiché ciò causerebbe un movimento eccessivo delle testine del disco durante l'elaborazione del file
- *accesso ai dati del file*: mantenere le informazioni sui file per trovare i blocchi del disco che lo contengono

L'allocazione non contigua può essere concatenata o indicizzata.

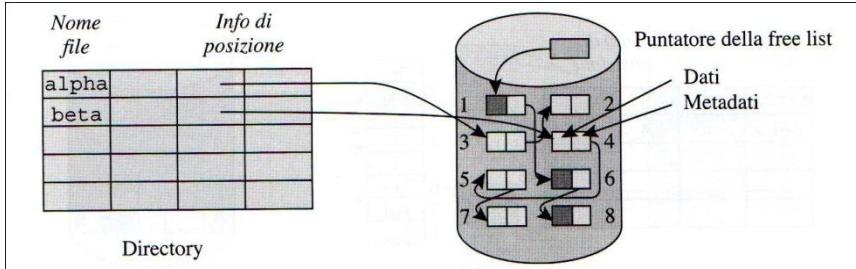
12.7.3 Allocazione concatenata

L'allocazione concatenata risolve il problema della frammentazione esterna e quello della dichiarazione delle dimensioni del file, entrambi presenti nell'allocazione contigua della memoria. Tuttavia, in mancanza di una FAT, l'assegnazione concatenata non è in grado di sostenere un efficiente accesso diretto, poiché i puntatori ai blocchi sono sparsi, con i blocchi stessi, per tutto il disco e si devono recuperare in ordine.

In questa allocazione, ogni file è rappresentato da una lista concatenata di blocchi del disco, che possono essere sparpagliati ovunque sul disco.

Ogni blocco del disco ha due campi al suo interno:

- **dati**, che contiene, appunto, i dati
- **metadati**, che è un campo di tipo link e punta al prossimo blocco



La figura mostra l'allocazione concatenata.

Il campo **Info di posizione** dell'elemento della directory punta al primo blocco sul disco del file. Agli altri blocchi si accede seguendo i puntatori dei vari blocchi. L'ultimo blocco del disco contiene un'informazione **null** nel campo metadati.

Lo spazio libero sul disco è rappresentato da una **free list** in cui ogni blocco libero contiene un puntatore al successivo. Quando viene richiesto un blocco, viene estratto un blocco dalla free list per poi essere aggiunto alla lista dei blocchi del file. Per cancellare un file, la lista di blocchi del file viene semplicemente aggiunta alla free list.

Ad esempio, nella figura:

- il file *alpha* è costituito da due blocchi, il numero 3 e il numero 2;
- il file *beta* è costituito da tre blocchi, il numero 4, seguito dal 5, seguito dal 7
- la *free list* è di tre blocchi, l'1, il 6 e l'8.

VANTAGGI:

Il vantaggio principale di questa allocazione è che è sufficiente memorizzare in ogni elemento della directory solamente l'**indirizzo** su disco **del primo blocco**, mentre la parte rimanente si può trovare a partire da esso. Questo porta anche ad una **lettura sequenziale semplice** da effettuare.

SVANTAGGI:

Tuttavia ci sono anche degli svantaggi.

L'**accesso diretto** è estremamente **lento**, perché per arrivare al blocco *n* occorre necessariamente aver letto gli *n-1* blocchi che lo precedono.

Un ulteriore svantaggio riguarda lo **spazio richiesto per i puntatori**. La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi continui in gruppi (*cluster*) e nell'assegnare i gruppi di blocchi anziché i singoli blocchi.

Un altro problema riguarda l'**affidabilità**. Se si danneggiasse il campo **metadati** di un blocco, cioè se si danneggiasse il puntatore, i **dati** successivi al blocco danneggiato potrebbero essere **persi**. Ci sono delle soluzioni a questo problema, come l'uso di liste doppiamente concatenate, che però sono onerose da implementare.

FILE ALLOCATION TABLE (FAT)

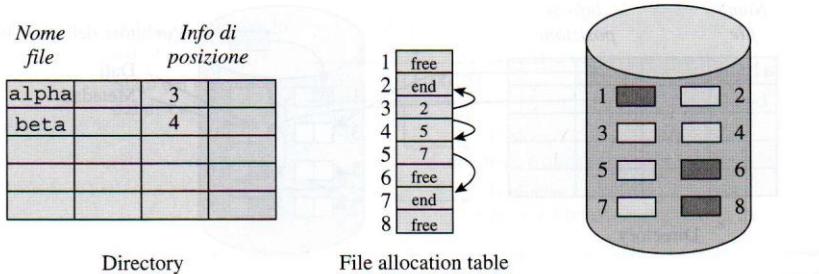
Una variante importante del metodo di assegnazione concatenata consiste nell'uso della **tabella di assegnazione dei file**, una tabella di questo tipo, tenuta in memoria, si chiama **FAT** (File Allocation Table).

Questo metodo di allocazione è usato nei sistemi operativi MS-DOS e OS/2.

Per contenere tale tabella si riserva una sezione del disco all'inizio di ciascuna partizione; la FAT ha un elemento per ogni blocco del disco. Per un blocco allocato a un file, il corrispondente elemento della FAT contiene l'indirizzo del blocco successivo. In questo modo il blocco e il suo elemento nella FAT insieme

formano una coppia che contiene la stessa informazione contenuta nel blocco nel classico schema dell'allocazione concatenata.

L'elemento di una directory relativo a un file contiene l'indirizzo del primo blocco sul disco. L'elemento della FAT corrispondente a questo blocco contiene l'indirizzo del secondo blocco e così via. L'elemento della FAT corrispondente all'ultimo blocco contiene un valore speciale di fine file.



La figura illustra la FAT per il disco della figura precedente.

Ad esempio:

- il file *alpha* si compone dei blocchi 3 e 2; il campo *Info_di_posizione* contiene 3; l'elemento della FAT relativo al blocco 3 contiene 2 e l'elemento della FAT relativo al blocco 2 indica che il file termina con quel blocco
- il file *beta* si compone dei blocchi 4, 5 e 7; il campo *Info_di_posizione* contiene 4; l'elemento della FAT relativo al blocco 4 contiene 5, e così via

La FAT può anche essere usata per memorizzare l'informazione relativa allo spazio libero. La lista dei blocchi liberi è costruita nello stesso modo in cui viene costruita la lista dei blocchi di un file. In alternativa, per ogni blocco libero, può essere utilizzato un valore speciale (*free*) che indica che quel blocco è libero.

VANTAGGI:

L'uso della **FAT** fornisce **maggiori affidabilità** rispetto alla classica allocazione concatenata poiché il danneggiamento di un blocco contenente i dati del file comporta danni limitati.

SVANTAGGI:

In questa allocazione, il **danneggiamento di un blocco** usato per **memorizzare la FAT** risulta disastroso.

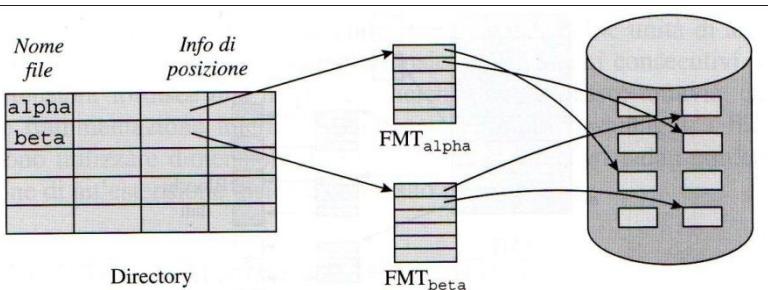
Inoltre, le **prestazioni sono peggiori** poiché è necessario accedere alla **FAT** per ottenere l'indirizzo del blocco successivo.

12.7.4 Allocazione indicizzata

L'allocazione indicizzata risolve il problema dell'accesso diretto, presente nell'allocazione concatenata, raggruppando tutti i puntatori in una sola locazione: il **blocco indice**.

Nell'allocazione indicizzata si mantengono tutti i puntatori ai blocchi di un file in una **tabella indice** chiamata **file map table (FMT)**. In questa tabella sono riportati gli indirizzi dei blocchi del disco allocati a un file.

Ogni file ha il proprio blocco indice (o tabella indice): nella sua forma più semplice, un FMT è un array di indirizzi di blocchi del disco. Ogni blocco ha un solo campo, il campo **dati**. L'i-esimo elemento del blocco indice punta all'i-esimo blocco del file.



Il campo *Info_di_posizione* dell'elemento della directory relativo a un file contiene l'indirizzo della FMT, cioè punta alla FMT. Una volta creato il file, tutti i puntatori del blocco indice sono impostati a *null*. Quando le dimensioni del file crescono, viene localizzato un blocco libero e l'indirizzo di questo blocco viene aggiunto alla FMT del file.

VANTAGGI:

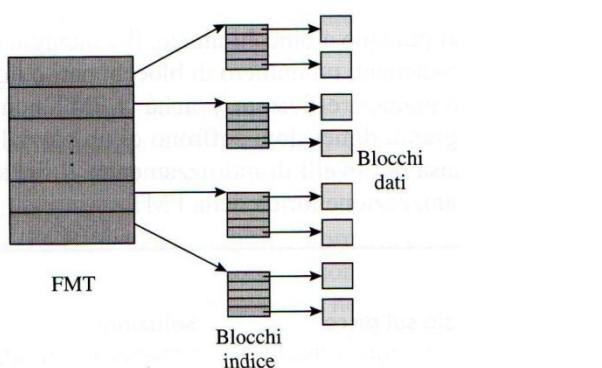
Questa allocazione permette di accedere direttamente a un blocco del file direttamente dalla FMT senza avere frammentazione esterna. Inoltre l'affidabilità è migliore in quanto il danneggiamento di un elemento della FMT non compromette l'intero file, ma solo una parte di esso.

SVANTAGGI:

Il problema principale consiste nella dimensione del blocco indice, cioè della FMT. Se essa è troppo piccola non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni, quindi è necessario disporre di un meccanismo per gestire questa situazione.

INDICE A PIU' LIVELLI

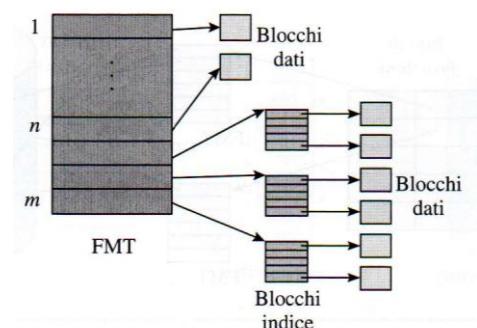
Una prima soluzione consiste nell'utilizzo di un blocco indice a più livelli.



In questa organizzazione, ogni elemento della FMT contiene l'indirizzo di un *blocco indice*. Un blocco indice non contiene dati; contiene elementi che contengono gli indirizzi dei blocchi dati. Per accedere ai blocchi dati, prima accediamo a un elemento della FMT e otteniamo l'indirizzo di un blocco indice. Successivamente, accediamo a un elemento del blocco indice per ottenere l'indirizzo di un blocco dati.

ORGANIZZAZIONE IBRIDA

Alcuni file system implementano un'organizzazione ibrida della FMT che include alcune delle caratteristiche dell'allocazione classica e dell'allocazione indicizzata multilivello.



I primi elementi nella FMT, ad esempio n elementi, puntano a blocchi dati come nell'allocazione indicizzata. Gli altri elementi puntano a blocchi indice.

Il vantaggio di questa organizzazione è che piccoli file continuano a essere accessibili in maniera efficiente, poiché la FMT non utilizza i blocchi indice. I file di medie o grandi dimensioni soffrono di un parziale degrado delle prestazioni di accesso a causa dei livelli di indirizzamento.

12.8 Affidabilità del file system

L'affidabilità del file system è il grado di funzionamento corretto di un file system, anche quando si verificano malfunzionamenti come la corruzione dei dati nei blocchi del disco e fault di sistema dovuti a interruzioni di corrente.

I due aspetti principali dell'affidabilità del file system sono:

- garantire la correttezza della creazione, cancellazione e aggiornamento dei file
- prevenire la perdita dei dati contenuti nei file

Il primo riguarda la consistenza e la correttezza dei metadati, ovvero i dati di controllo del file system; mentre il secondo riguarda la consistenza e la correttezza dei dati memorizzati nei file.

Quando si parla di affidabilità bisogna tenere conto del *fault* (o guasto) e del *failure* (o insuccesso).

Un *fault* è un difetto in qualche parte del sistema. Un *failure* è un comportamento erroneo, o che differisce dal comportamento atteso. L'occorrenza di un fault causa un failure.

12.9 Journaling file system – JFS

Un file system durante l'esecuzione mantiene in memoria una parte dei file dati e dei metadati, in particolare mantiene in memoria i *file control block*, le *file map table* e le *free list*.

Quando l'esecuzione di un file system viene terminata dall'utente del sistema, il file system copia tutti i dati e i metadati dalla memoria RAM sul disco, in modo che la copia sul disco sia completa e consistente.

Tuttavia, quando si verifica una mancanza di corrente elettrica o quando il sistema viene spento all'improvviso, il file system non ha l'opportunità di copiare i file dati e i metadati sul disco. Questo spegnimento è detto spegnimento *sporco* e causa una perdita dei dati dei file e dei metadati contenuti in memoria.

Tradizionalmente, i file system utilizzavano delle *tecniche di ripristino* per proteggersi contro la perdita di dati e metadati poiché erano molto semplici da implementare. Per questo motivo, si creavano backup periodici e i file erano ripristinati a partire dalle copie di backup nel momento in cui si verificavano dei malfunzionamenti. La creazione delle copie di backup comportava un piccolo overhead durante il normale funzionamento del sistema. Tuttavia, quando si verifica un malfunzionamento, l'overhead per la correzione delle inconsistenze era elevato ed, inoltre, il sistema non era disponibile durante il ripristino.

Un file system moderno utilizza tecniche di *fault tolerance* in modo da poter riprendere l'esecuzione velocemente dopo uno spegnimento improvviso. Un *journaling file system* implementa la *fault tolerance* mantenendo un *journal*, un diario giornaliero, dove vengono salvate le azioni che il file system si accinge ad eseguire prima di eseguirle effettivamente. Quando l'esecuzione del file system viene ripristinata dopo uno spegnimento improvviso, il file system consulta il *journal* per identificare le azioni non ancora eseguite e le esegue, garantendo in questo modo la correttezza dei dati dei file e dei metadati.

L'uso di tecniche di *fault tolerance* genera un elevato overhead. Per questo motivo il JFS offre diverse modalità *journaling*. Un amministratore di sistema può scegliere una modalità *journaling* da adattare al tipo di affidabilità necessaria nell'ambiente di elaborazione.

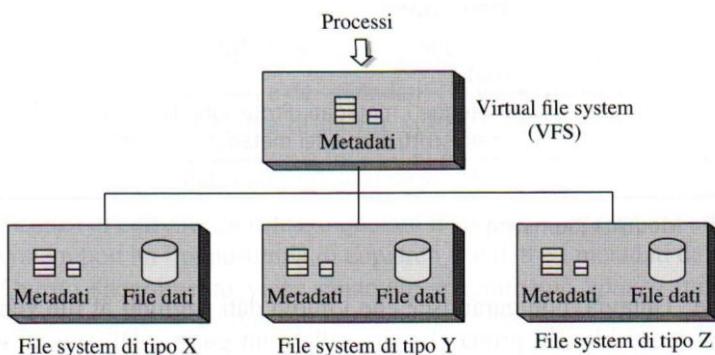
| Modalità | Descrizione |
|--------------|--|
| Write behind | Protegge solo i metadati. Non fornisce alcuna protezione per i dati. |
| Ordered data | Protegge i metadati. Protezione limitata per i file dati – vengono scritti prima dei metadati a essi relativi. |
| Full data | Protegge sia i dati che i metadati. |

12.10 File system virtuale – FSV

Gli utenti richiedono requisiti differenti a un file system, come convenienza, alta affidabilità, risposte veloci e accesso ai file su altri sistemi. Un singolo file system non può fornire tutte queste caratteristiche, per cui un sistema operativo fornisce un **file system virtuale** (VFS) che facilita l'esecuzione simultanea di diversi file system. In questo modo ogni utente può usare il file system che preferisce.

In pratica un **file system virtuale** è un livello software che consente a diversi file system di essere in funzione su un computer simultaneamente, in modo che un utente possa scegliere il file system che è più adatto per le sue applicazioni.

Un processo invoca il livello VFS utilizzando comandi generali per l'accesso ai file e il livello VFS redireziona il comando al file system appropriato.



Ciò è implementato da un layer VFS situato tra un processo e un file system. Il layer VFS ha due interfacce: un'interfaccia col file system e un'interfaccia con i processi. Ogni file system conforme alle specifiche dell'interfaccia del file system VFS può essere installato per funzionare con il VFS. Questa caratteristica rende facile aggiungere un nuovo file system.

L'interfaccia del VFS con il processo fornisce le funzionalità per eseguire le generiche operazioni sui file (come open, close, read, write) e le operazioni mount e umount sul file system.

L'interfaccia con il file system serve per determinare a quale file system appartiene il processo, invocando le operazioni open, close, read e write dello specifico file system.

Il FSV risulta molto utile con i dispositivi rimovibili come le penne USB o i CD/DVD in quanto consente all'utente di montare il file system presente in questi dispositivi nella sua directory corrente e accedere ai file senza preoccuparsi del fatto che i dati sono memorizzati in un formato differente.

12.11 File system di Unix

12.11.1 I-node, descrittore di file e struttura file

INODE

Le informazioni che costituiscono l'elemento della directory relativo a un file (nome, tipo, dimensione, locazione, protezione, open count, lock, flag, Misc Info), in Unix, sono divise tra l'elemento della directory e l'**i-node** del file. L'elemento della directory contiene solo il nome e il numero di inode; la maggior parte delle informazioni di un file è contenuta nel suo inode. Quindi, un file in Unix è rappresentato da un **inode** (nodo indice).

In Unix, l'amministratore di sistema può specificare una quota disco per ogni utente. Questo impedisce a un utente di occupare uno spazio elevato su disco rigido oppure tutto lo spazio disponibile.

La struttura dati **inode** viene mantenuta sul disco. Essa contiene le seguenti informazioni:

- Tipo di file (directory, link o file speciale)
- Numero di link al file

- Dimensione del file
- ID del dispositivo su cui è memorizzato il file
- Numero seriale dell'inode
- ID utente e gruppo del proprietario
- Permessi di accesso
- Informazione sull'allocazione

La divisione dell'elemento della directory tra l'elemento della directory e l'inode facilita la cancellazione dei link. Un file può essere cancellato quando il suo numero di link va a zero.

ORGANIZZAZIONE IN MEMORIA

In Unix, in memoria vengono mantenute gli inode, i **descrittori di file** e le **strutture file**.

Una struttura file contiene due campi, la posizione corrente in un file aperto e un puntatore all'inode del file.

In questo modo un inode e una struttura file insieme contengono tutte le informazioni necessarie per accedere al file.

I descrittori di file sono memorizzati in una tabella per ogni processo.

12.11.2 Allocazione dello spazio su disco

Unix utilizza l'allocazione indicizzata dello spazio su disco, con dimensione del blocco di 4KB.

Ogni file ha una FAT memorizzata nel proprio inode. Infatti gli inode contengono, oltre agli attributi elencati in precedenza, altri 15 elementi, di cui 12 sono indirizzi diretti e 3 sono indirizzi indiretti. Questi ultimi vengono allocati su richiesta, cioè quando servono effettivamente.

- i primi 12 puntano direttamente ai blocchi dati del file
- gli altri 3 puntano a blocchi indice, in particolare
 - o il 13° punta a un blocco indiretto di primo livello, cioè un blocco che contiene puntatori a blocchi dati
 - o il 14° punta a un blocco indiretto di secondo livello
 - o il 15° punta a un blocco indiretto di terzo livello (non viene quasi mai usato)

In questo modo la dimensione totale del file può arrivare a un massimo di 2^{42} byte. Ma, visto che gli indirizzi sono a 32 bit, la dimensione di un file è limitata a $2^{32}-1$ byte.

Il numero di indirizzi contenuti in ogni blocco indiretto dipende dalla dimensione dei singoli blocchi e dalla dimensione degli indirizzi. In Unix, la dimensione comune dei blocchi è 4KB, mentre gli indirizzi sono di 32 bit, cioè 4B. Riassumendo:

Dimensione blocco = 8KB = 8192 byte

(dimensione blocco = 4kb = 4096 byte)

Dimensione indirizzo = 32 bit = 4 byte

(dimensione indirizzo = 32 bit = 4 byte)

Ogni blocco indiretto contiene $8192/4= 2048$ indirizzi

($4096/4=1024$ indirizzi)

12.11.3 Superblocco

La radice di un file system è chiamata **superblock**.

Una parte del disco, solitamente quella iniziale, viene utilizzata per il codice di avvio del SO.

Dopo lo spazio che viene lasciato per questo codice, si colloca il **superblocco**. Esso si può considerare come una tabella riassuntiva delle caratteristiche e dello stato del file system.

Le informazioni che contiene consentono di sapere:

- dimensione dei blocchi
- posizione degli i-node
- numero di i-node
- blocchi liberi
- blocchi allocati

CAPITOLO 13. Implementazione delle operazioni su file

Una delle funzioni principali di un SO è il controllo dei dispositivi di I/O: esso deve inviare i comandi ai dispositivi, catturare le interruzioni e trattare le condizioni di errori. Deve fornire un'interfaccia tra dispositivi fisici ed il resto del sistema che sia semplice e facile da usare; possibilmente, l'interfaccia dovrà essere la stessa per tutti i dispositivi (*indipendenza dal dispositivo*).

Come visto nel capitolo precedente, l'elaborazione di un file è implementata utilizzando i moduli del file system e il sistema di controllo input/output (IOCS – Input/Output Control System). In particolare i moduli del file system consentono di avere libertà nella scelta dei nomi, la condivisione e la protezione dei file e l'affidabilità; le operazioni sui file vengono implementate dall'IOCS.

In questo capitolo esamineremo le caratteristiche di un dispositivo di I/O. Successivamente discuteremo di come le operazioni di I/O sono eseguite al livello dei dispositivi di I/O, delle funzionalità offerte dal sistema IOCS fisico per semplificare le operazioni di I/O e di come lo *scheduling del disco* consenta di ottenere valori elevati di throughput.

13.1 Livelli dell'input-output control system

Descriviamo brevemente come l'IOCS implementi le operazioni sui file.

Quando un processo effettua una richiesta di lettura o scrittura di dati da un file, il file system passa la richiesta al sistema IOCS. Ricordiamo che questo sistema mantiene alcuni dati in aree di memoria chiamate *buffer* (o *cache del disco*) per velocizzare l'elaborazione dei file.

La prima operazione che effettua il sistema IOCS è quella di controllare se i dati richiesti dal processo sono presenti in memoria; in caso affermativo, il processo può accedere direttamente ai dati; altrimenti, l'IOCS esegue una o più operazioni di I/O per caricare i dati nella cache del disco, e durante queste operazioni il processo deve attendere. Questo rende efficiente l'elaborazione dei file.

Visto che molti processi eseguono operazioni di I/O in modo concorrente, **queste operazioni vengono schedulate da algoritmi di scheduling del disco**. Questo fornisce un elevato throughput dei dispositivi.

L'IOCS è organizzato in due livelli: metodo di accesso e IOCS fisico.

Il metodo di accesso gestisce la lettura e scrittura dei dati per rendere efficiente l'elaborazione di un file.

L'IOCS fisico esegue l'I/O a livello di dispositivo e utilizza le politiche di scheduling per migliorare il throughput dei dispositivi di I/O.

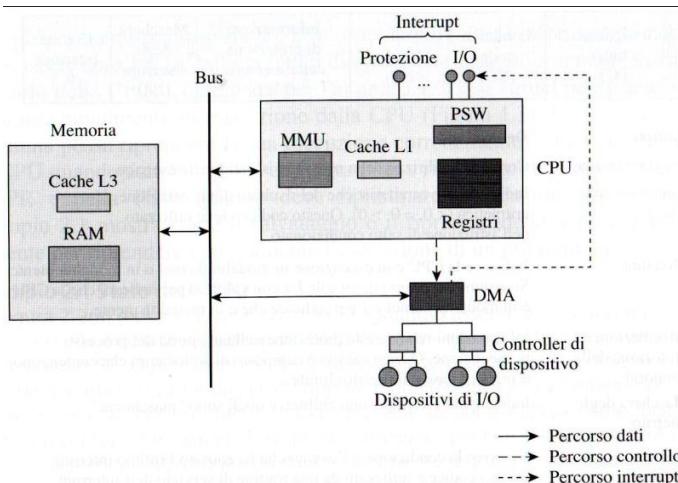
Questa struttura dell'IOCS consente di separare le problematiche relative all'implementazione delle operazioni sui file a livello di processo da quelle a livello di dispositivo.

13.2 Panoramica dell'organizzazione dell'I/O

| Modalità di I/O | Descrizione |
|---|---|
| I/O programmato | Il trasferimento dati tra la periferica di I/O e la memoria avviene attraverso la CPU. La CPU non può eseguire nessun'altra istruzione mentre è in esecuzione un'operazione di I/O. |
| Interrupt di I/O | La CPU è libera di eseguire altre istruzioni dopo aver eseguito l'istruzione di I/O. Un interrupt viene generato quando un byte di dati deve essere trasferito dalla periferica di I/O alla memoria e la CPU esegue la routine di servizio dell'interrupt che gestisce il trasferimento del byte. Questa sequenza di operazioni viene ripetuta finché tutti i byte sono trasferiti. |
| I/O basato sul direct memory access (DMA) | Il trasferimento di dati tra la periferica di I/O e la memoria avviene direttamente sul bus. La CPU non è coinvolta nel trasferimento dei dati. Il controller DMA genera un interrupt quando il trasferimento di tutti i byte è stato completato. |

Nella tabella sono riassunti i tre modi per effettuare le operazioni di I/O:

- modalità programmata
- modalità ad interrupt
- modalità basata su DMA



La figura mostra come i dispositivi di I/O sono connessi ai controller di dispositivo che, a loro volta, sono connessi al controller DMA. Ogni controller di dispositivo ha un ID numerico univoco. Ogni dispositivo ad esso connesso ha un ID numerico univoco.

Un indirizzo di dispositivo è una coppia nella forma (*controller_ID, device_ID*).

Un'operazione di I/O è caratterizzata:

- dal tipo di operazione da eseguire (read, write, ecc.)
- dall'indirizzo del dispositivo di I/O
- dal numero di byte da trasferire
- indirizzi delle aree di memoria e del dispositivo di I/O che sono coinvolti nel trasferimento dei dati

Quando un'operazione di I/O viene eseguita in modalità DMA, la CPU avvia l'operazione di I/O, ma non è coinvolta nel trasferimento dei dati tra il dispositivo e la memoria. Questa operazione di I/O è chiamata **istruzione di I/O**. Essa richiede la partecipazione del controller DMA, del controller del dispositivo e del dispositivo di I/O, ma non richiede la partecipazione della CPU. In questo modo, la CPU è libera di eseguire altre istruzioni mentre l'operazione di I/O è in corso.

13.3 Dispositivi di I/O

I dispositivi di I/O funzionano con diversi supporti di I/O, servono per scopi differenti e organizzano e accedono ai dati in modi differenti, per cui possono essere classificati secondo i seguenti criteri:

- **Scopo**: dispositivi di input, output e memorizzazione
- **Tipo di accesso**: dispositivi sequenziali e ad accesso casuale
- **Modalità di trasferimento dati**: dispositivo a caratteri e a blocchi

TIPO DI ACCESSO

L'informazione scritta (letta) con un comando di I/O si dice che forma un **record**.

Un dispositivo ad accesso sequenziale utilizza il supporto in maniera sequenziale; dunque un'operazione è sempre eseguita su un record contiguo a un record cui si è avuto accesso nell'operazione precedente.

Un dispositivo ad accesso casuale consente di eseguire operazioni di lettura e scrittura su un record posizionato in una qualsiasi parte del supporto di I/O.

Un dispositivo di I/O può essere anche **rimovibile**, basti pensare ai CD/DVD o alle penne USB, mentre altri dispositivi, come gli hard disk interni non sono supporti rimovibili.

MODALITA' DI TRASFERIMENTO DEI DATI

La modalità di trasferimento dei dati di un dispositivo dipende dalla velocità di trasferimento. In base a questa caratteristica, i dispositivi di I/O possono essere suddivisi in **dispositivi a blocchi** e **dispositivi a carattere**.

I **dispositivi a carattere** sono dispositivi di I/O lenti perché trasferiscono un carattere alla volta tra la memoria e il dispositivo. Questo tipo di dispositivi non sono indirizzabili e non possono eseguire nessuna operazione di posizionamento. Alcuni esempi sono la tastiera, il mouse e la stampante.

I **dispositivi a blocchi** sono dispositivi di I/O più rapidi in quanto trasferiscono blocchi di dati di lunghezza fissa, ognuno dei quali ha un proprio indirizzo. Per questo motivo, questi dispositivi sono indirizzabili, ovvero è possibile leggere o scrivere ciascun blocco in maniera indipendente. Un esempio sono i dischi.

13.3.1 Controller di dispositivo

I dispositivi di I/O sono dotati, tipicamente, di una componente meccanica e di una componente elettronica ed è spesso possibile separare le due parti.

La componente elettronica prende il nome di **controllore del dispositivo**.

Ciò che esce realmente dal dispositivo fisico è una sequenza di bit che comincia con un **preamble** iniziale seguito da un **checksum**. Il preamble è scritto quando il disco viene **formattato** e contiene svariate informazioni. Il compito del controllore del dispositivo è di convertire la sequenza di bit in un blocco di byte e di eseguire eventualmente la correzione degli errori.

Prima il blocco dei byte è ricostruito un bit alla volta in un buffer che si trova all'interno del controllore stesso, dopo che il suo checksum è stato controllato e che il blocco è stato dichiarato privo di errori, esso può essere copiato all'interno della memoria principale.

13.3.2 I/O mediante DMA

In questo tipo di trasferimento la CPU delega il DMA per effettuare il trasferimento dei dati. Inoltre, i dispositivi che effettuano questo tipo di trasferimento sono dei dispositivi collegati al DMA.

Visto che questi dispositivi devono effettuare operazioni di lettura o scrittura a velocità molto alte, ci possono essere dei problemi per la contesa del buffer del DMA.

Durante un'operazione di lettura (input), i dati vengono trasferiti dalla periferica ad un buffer del DMA, questi sono poi trasferiti dal buffer del DMA alla memoria al termine dell'operazione di I/O.

Durante un'operazione di scrittura (output), i dati vengono prima trasferiti dalla memoria al buffer del DMA, poi da questo alla periferica.

13.3.3 I/O mappato in memoria

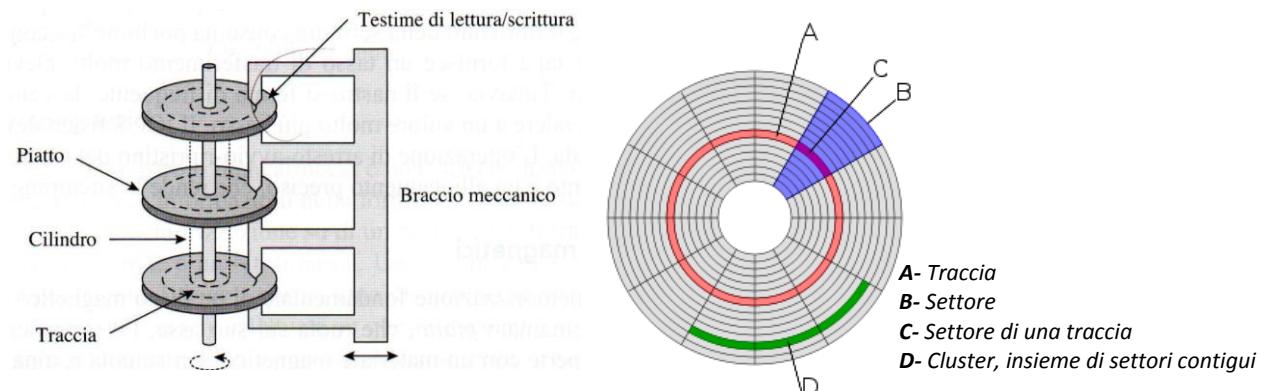
Il trasferimento dei dati tra la CPU e un dispositivo di I/O può essere realizzato utilizzando la tecnica dell'**I/O mappato in memoria**.

Ogni controllore possiede alcuni **registri** che sono usati per comunicare con la CPU. La CPU può comunicare con i registri in due modi.

Nel primo approccio, ogni registro del controllore è caratterizzato da un numero di **porta I/O**; in questo schema gli spazi di indirizzamento per l'I/O e la memoria sono diversi.

Nel secondo approccio, un insieme di indirizzi di memoria sono riservati a un dispositivo di I/O, cioè ad ogni registro del controllore viene assegnato un indirizzo di memoria unico. Questo sistema è chiamata **I/O mappato in memoria**. Quando la CPU scrive dei dati in una locazione di memoria con uno degli indirizzi riservati, i dati vengono scritti nel corrispondente registro del dispositivo. Analogamente, quando la CPU esegue un'istruzione di lettura da una locazione di memoria con uno degli indirizzi riservati, i dati vengono letti dal corrispondente registro del dispositivo di I/O. In questo modo il trasferimento dei dati avviene senza un DMA, ma non si impegnava troppo la CPU.

13.4 Dischi magnetici



I **dischi magnetici** sono costituiti da vari piatti, ognuno dei quali contiene tante tracce. Tutte le tracce uguali posizionate su piatti differenti formano un **cilindro**.

L'elemento di memorizzazione di un disco magnetico è un oggetto circolare chiamato **piatto**, che ruota sul suo asse ed è ricoperto da un materiale magnetico.

Ogni piatto memorizza i byte lungo delle **tracce circolari** sulla sua superficie. Ogni traccia è divisa in **settori** ed il numero di settori può variare a seconda delle tracce. I dischi moderni sono divisi in zone con più settori e zone con meno settori, ma grazie alla virtualizzazione, il SO vede tutte le zone come se avessero lo stesso **numero di settori**, dunque le richieste vengono fatte su quella base.

La **testina** è capace di leggere e scrivere sulla superficie del piatto. Essa è capace di "muoversi" sul piatto grazie al braccio in modo tale da scorrere le tracce.

Riassumiamo schematicamente le parti essenziali di un disco:

- **Piatto** – un disco rigido si compone di uno o più dischi paralleli, di cui ogni superficie, detta piatto, è identificata da un numero univoco ed è destinata alla memorizzazione dei dati
 - **Traccia** – Ogni piatto si compone di numerosi anelli concentrici numerati, detti tracce, ciascuna identificata da un numero univoco
 - **Cilindro** – L'insieme di tracce alla stessa distanza dal centro presenti su tutti i dischi è detto cilindro. Corrisponde a tutte le tracce aventi il medesimo numero, ma diverso piatto
 - **Settore** – Ogni piatto è suddiviso in settori circolari, ovvero in spicchi radiali uguali ciascuno identificato da un numero univoco
 - **Blocco** – L'insieme di settori posti nella stessa posizione in tutti i piatti
 - **Testina** – Su ogni piatto è presente una testina per accedere in scrittura o in lettura ai dati memorizzati sul piatto; la posizione di tale testina è solidale con tutte le altre sugli altri piatti. In altre parole, se una testina è posizionata sopra una traccia, tutte le testine saranno posizio-
- nate nel cilindro a cui la traccia appartiene

13.4.1 Parametri di prestazioni del disco

Analizziamo le prestazioni del disco. Uno dei parametri più importanti è il tempo di accesso, ovvero l'intervallo di tempo tra l'esecuzione di un comando di lettura o scrittura e l'inizio del trasferimento dei dati.

Il **tempo di accesso** per un record di un disco è dato da:

$$t_a = t_s + t_r + t_t$$

t_s è il **tempo di ricerca** (o *tempo di seek*), cioè il tempo necessario per muovere la testina sulla traccia desiderata (valori soliti 5-15ms).

t_r è la **latenza rotazionale**, cioè il tempo impiegato dal disco per ruotare portando il settore interessato sotto la testina richiesta; la latenza rotazionale media è il tempo impiegato per mezza rivoluzione del disco (valore soliti 3-4ms).

In pratica, la latenza rotazionale è data da $\frac{1}{r}$ dove r è la velocità di rotazione

Mentre la latenza rotazionale media è data da $\frac{1}{2r}$

t_l è il **tempo di trasferimento** (o *transfer rate*), cioè il tempo necessario per trasferire un numero di blocchi (o settori) da leggere/scrivere (valori soliti decine di megabyte al secondo).

Esso è dato da $T = \frac{b}{rN}$ dove b è il numero di blocchi da trasferire, r è la velocità di rotazione ed N è il numero di blocchi per traccia.

13.5 RAID

Gli utenti necessitano di dischi con maggiore capienza, accesso ai dati più veloce, tassi di trasferimento maggiori e maggiore affidabilità. Se a queste esigenze si aggiungere il fatto che col passare degli anni la forbice tra le prestazioni della CPU e degli hard disk tende ad aumentare sempre di più, ecco spiegati i motivi che hanno portato allo sviluppo della tecnologia **RAID** (*redundant array of inexpensive disks*).

Questa tecnologia usa un insieme di dischi rigidi per condividere o replicare i dati e ha come obiettivi principali l'aumento delle prestazioni e una migliore affidabilità. Per fare ciò distribuisce i dati coinvolti in un'operazione di I/O su diversi dischi ed esegue le operazioni di I/O su questi dischi in parallelo.

Questa caratteristica può fornire accessi veloci o alti tassi di trasferimento, in base alla configurazione adottata. L'alta affidabilità è ottenuta memorizzando informazioni ridondanti. L'accesso alle informazioni ridondanti non necessita di tempo di I/O aggiuntivo in quanto si può accedere in parallelo sia ai dati che alle informazioni ridondanti.

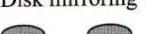
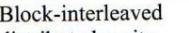
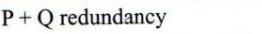
Da notare che la gestione di un RAID è più dispendiosa della gestione di un unico disco poiché vi sono più dischi da controllare.

Esistono diverse configurazioni RAID che utilizzano differenti tecniche di ridondanza e organizzazione. Queste tecniche sono chiamate **livelli RAID** e, ad oggi, vanno da 0 a 6. Esistono inoltre due configurazioni ibride basate sui livelli 0 e 1, che insieme al livello 5 risultano le configurazioni più usate.

Tutti i livelli RAID hanno però delle caratteristiche in comune:

- i vari hard disk configurati in RAID sono visti dal SO come un unico disco
- i dati sono divisi in strisce (*stripes*) distribuite sui vari dischi; una strip è memorizzata su più dischi nella stessa posizione in modo che, sincronizzati i dischi, è possibile leggere una striscia simultaneamente
- la capacità di ridondanza del disco è usata per memorizzare informazioni di parità che permettono il recupero dei dati in caso di guasti

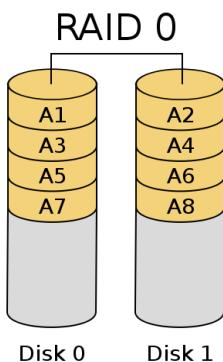
La tabella che segue riassume le proprietà dei vari livelli RAID.

| Livello | Tecnica | Descrizione |
|-----------|---|---|
| Livello 0 | Disk striping  ...  | I dati sono alternati su diversi dischi. Durante un'operazione di I/O, l'accesso ai dischi avviene in parallelo. Potenzialmente, utilizzando n dischi questa configurazione può garantire un incremento dell'ordine di n nel trasferimento dei dati. |
| Livello 1 | Disk mirroring  Disco 1 Disco 2 | Gli stessi dati sono memorizzati su due dischi. Durante la lettura dei dati, viene utilizzata la copia accessibile più velocemente. Una delle copie è accessibile anche dopo un guasto. Le operazioni di lettura possono essere eseguite in parallelo se non si verificano errori. |
| Livello 2 | Codici di correzione degli errori  D D P P | Le informazioni di ridondanza sono memorizzate per rilevare e correggere gli errori. Ogni bit di dati o di informazione ridondante è memorizzato su un disco differente e letto o scritto in parallelo. Garantisce tassi di trasferimento elevati. |
| Livello 3 | Bit-interleaved parity  D D P | Analogo al livello 2, fatta eccezione per il fatto che utilizza un singolo disco di parità per la correzione degli errori. Un errore che si verifica durante la lettura dei dati da un disco viene rilevato dal controller. Il bit di parità viene utilizzato per ripristinare i dati persi. |
| Livello 4 | Block-interleaved parity  D D P | Scrive un blocco di dati, ovvero byte di dati consecutivi, in una strip e calcola una singola strip di parità per le strip di una stripe. Garantisce tassi di trasferimento elevati per operazioni di lettura molto grandi. Le operazioni di lettura piccole hanno bassi tassi di trasferimento; tuttavia, molte di queste operazioni possono essere eseguite in parallelo. |
| Livello 5 | Block-interleaved distributed parity  ...  | Analogo al livello 4, fatta eccezione per il fatto che le informazioni sono distribuite su tutti i dischi. Consente di evitare che il disco di parità diventi un collo di bottiglia per l'I/O come nel livello 4. Inoltre garantisce migliori prestazioni in lettura rispetto al livello 4. |
| Livello 6 | P + Q redundancy  D D P P | Analogo al livello 5, fatta eccezione per il fatto che utilizza due schemi di parità distribuita indipendenti. Supporta il ripristino dal guasto di due dischi. |

Nel primo caso, deve essere disponibile almeno un controllore RAID; nei PC desktop, questo può essere una scheda PCI o può essere il controller integrato nella scheda madre.

Nel secondo caso, è il SO che gestisce l'insieme di dischi attraverso un normale controller. Questa opzione può essere più lenta di un RAID hardware, ma non richiede l'acquisto di componenti extra.

13.5.1 RAID di livello 0



Nel RAID 0 i dati vengono distribuiti su più dischi con la tecnica dello **striping**: i dati sono divisi equamente tra due o più dischi con nessuna informazione di parità o ridondanza. Quindi questa non è proprio una configurazione RAID poiché non usa nessuna memorizzazione ridondante dei dati.

Le operazioni sono svolte in parallelo sui vari dischi; ad esempio, se viene inviato un comando di lettura di un blocco di dati, tale comando si suddivide in più comandi, ognuno per ciascun disco interessato. In questo modo la lettura avviene in parallelo.

Il RAID 0 risulta indicato quando le esigenze di performance superano quelle dell'affidabilità.

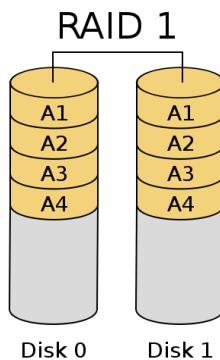
VANTAGGI:

Garantisce tassi di trasferimento elevati grazie agli accessi contemporanei ai vari dischi.

SVANTAGGI:

Ha una scarsa affidabilità. I dati diventano inaccessibili anche nel caso di un singolo disco spento. Inoltre la mancanza di ridondanza causa la perdita dei dati nel caso di malfunzionamento di un disco.

13.5.2 RAID di livello 1



Nel RAID 1 viene creata una copia esatta di tutti i dati su due o più dischi. Questa tecnica prende il nome di **disk mirroring**.

Se un dato viene aggiornato da un processo, una copia di questo dato viene copiata su ogni disco. Ciò aumenta l'affidabilità rispetto al sistema a disco singolo. Infatti, se si verifica un malfunzionamento ad uno dei dischi, un dato può essere recuperato dagli altri dischi.

Il RAID 1 non porta vantaggi nella scrittura ma porta notevoli miglioramenti nella lettura. Infatti, anche nel RAID 1 le operazioni sono svolte in parallelo aumentando, così, le prestazioni in lettura: durante una lettura, viene utilizzata la copia che può essere letta prima; inoltre è possibile leggere da un disco mentre l'altro è occupato.

Da notare il fatto che il sistema può avere una capacità massima pari a quella del disco più piccolo.

VANTAGGI:

Maggiori affidabilità visto che vengono create tante copie dei dati quanti sono i dischi.

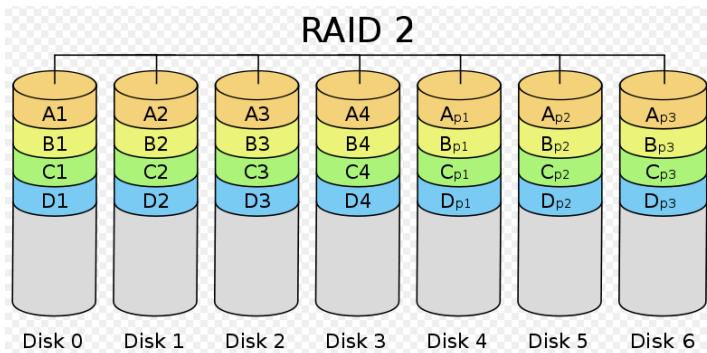
Migliori prestazioni in lettura, visto che il dato viene letto dal disco a cui si può accedere prima.

SVANTAGGI:

Viene prodotto molto overhead visto che ad ogni aggiornamento di un dato, devono essere aggiornati anche le sue copie presenti sugli altri dischi.

Il sistema vede un unico disco, ma con una capienza pari a quella del disco fisico più piccolo.

13.5.3 RAID di livello 2



Nel RAID 2 viene utilizzata la tecnica del **bit striping**: si memorizza ogni **bit** (invece che i blocchi) **di dati ridondanti su un disco differente**. Inoltre vengono salvati su altri dischi, i codici per la correzione degli errori. Solitamente si usa il codice di Hamming visto che esso è in grado di correggere errori su singoli bit e rilevare errori doppi.

In pratica, una parte dei dischi viene usata per salvare i dati, mentre la restante parte viene usata per salvare solo le informazioni per la correzione degli errori.

In sostanza, il RAID 2 è un RAID 0 maggiormente affidabile. Infatti questo sistema risulta molto efficiente negli ambienti in cui si verificano numerosi errori di lettura o scrittura.

VANTAGGI:

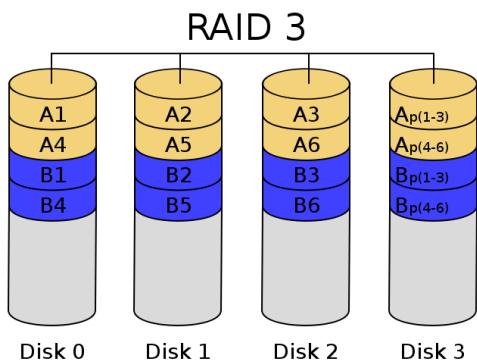
Affidabilità molto alta.

Velocità di lettura molto alta: visto che i dati sono spezzettati su molti dischi, la possibilità di leggere in parallelo i vari dischi, aumenta notevolmente la velocità di lettura.

SVANTAGGI:

Il numero di dischi utilizzati da questo sistema può essere molto alto.

13.5.4 RAID di livello 3



Il RAID 3 non è altro che una semplificazione del RAID 2. Infatti la differenza sostanziale tra i due livelli è che nel RAID 3 viene utilizzato un solo disco per il salvataggio dei codici per la correzione. Questo è possibile sostituendo il codice di Hamming con un nuovo metodo, il **bit di parità**.

Un sistema RAID 3 usa una divisione al livello di byte.

Ogni operazione di I/O richiede di utilizzare tutti i dischi. Questa caratteristica fornisce elevati tassi di trasferimento; tuttavia, non possono essere eseguite operazioni multiple contemporaneamente, ma solo un'operazione di I/O può essere eseguita in ogni istante. Nella figura in alto, una richiesta per il blocco A richiederà di cercare attraverso tutti i dischi; una richiesta simultanea per il blocco B rimarrà invece in attesa.

VANTAGGI:

Prestazioni e caratteristiche simili al RAID 2.

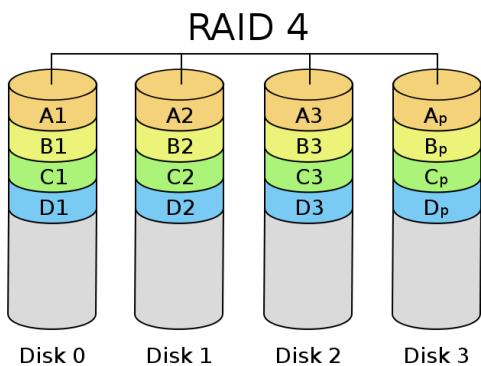
Elevati tassi di trasferimento.

Affidabilità molto alta.

Un solo disco viene utilizzato per memorizzare i codici per la correzione degli errori.

SVANTAGGI:

Possibilità di eseguire una sola operazione di I/O in ogni istante.

13.5.5 RAID di livello 4

Il RAID 4 è analogo al RAID 3 fatta eccezione per il fatto che usa una divisione a livello di blocchi con un disco dedicato alla memorizzazione dei codici di parità. La differenza sostanziale sta proprio nel modo in cui i dati vengono divisi: nel RAID 3 sono divisi a livello di byte, **nel RAID 4 sono divisi al livello di blocchi**.

Per le operazioni di lettura molto grandi, cioè quando sono richiesti più blocchi, il RAID 4 si comporta in modo analogo al RAID 3: l'operazione di I/O richiede di utilizzare tutti i dischi; ciò porta ad un alto tasso di trasferimento, ma anche al fatto di non poter eseguire più operazioni di I/O contemporaneamente.

Per le operazioni di lettura piccole, cioè quando è richiesto un solo blocco, il RAID 4 si comporta diversamente: l'operazione di I/O richiede di utilizzare un singolo disco; ciò porta a bassi tassi di trasferimento, ma anche la possibilità di eseguire più operazioni di I/O contemporaneamente. Nella figura, una richiesta per il blocco A1 potrebbe essere evasa dal disco 1; una richiesta simultanea al blocco B1 dovrebbe aspettare, ma una richiesta per il blocco B2 potrebbe essere servita allo stesso momento dal disco 2.

Un'operazione di scrittura necessita del calcolo delle informazioni di parità tenendo conto di tutti i blocchi memorizzati sui vari dischi. In pratica, quando si vuole scrivere, sono coinvolti tutti i dischi anche se la scrittura riguarda un solo blocco; ciò impedisce al sistema di effettuare operazioni in parallelo.

VANTAGGI:

Possono essere effettuate più operazioni di I/O contemporaneamente se le operazioni di lettura sono piccole.

Il tasso di trasferimento è più alto se le operazioni di lettura sono grandi, poiché sono coinvolti più dischi.

Affidabilità molto alta.

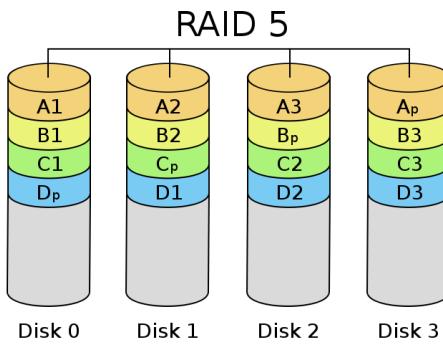
Un solo disco viene utilizzato per memorizzare i codici per la correzione degli errori.

SVANTAGGI:

C'è la possibilità di effettuare una sola operazione di scrittura alla volta, sia se l'operazione di scrittura è grande, sia se piccola.

Il disco di parità è un collo di bottiglia in quanto l'operazione di scrittura impedisce al sistema di effettuare operazioni in parallelo.

13.5.6 RAID di livello 5



Il RAID 5 è analogo al RAID 4 fatta eccezione per il fatto che i codici di parità sono distribuiti su tutti i dischi. In questo modo, se l'operazione di scrittura interessa un singolo blocco, il sistema ha comunque la possibilità di effettuare altre operazioni di scrittura in parallelo.

Per quanto riguarda la lettura, è tutto come nel RAID 3 e nel RAID 4:

- operazioni di lettura grandi portano ad un alto tasso di trasferimento ma all'impossibilità di effettuare operazioni in parallelo
- operazioni di lettura piccole portano a bassi tassi di trasferimento ma alla possibilità di effettuare operazioni in parallelo

VANTAGGI:

Possibilità di effettuare scritture in parallelo se queste richiedono un solo blocco, quindi il disco di parità non fa da collo di bottiglia.

Possibilità di effettuare letture in parallelo se queste richiedono solo un blocco.

Alti tassi di trasferimento se le operazioni di I/O richiedono più blocchi.

Affidabilità molto alta.

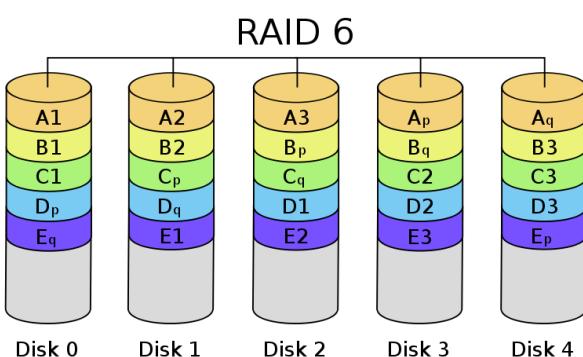
Un solo disco viene utilizzato per memorizzare i codici per la correzione degli errori.

SVANTAGGI:

Possibilità di eseguire una sola operazione di I/O in ogni istante se vengono richiesti più blocchi.

Difficile da implementare.

13.5.7 RAID di livello 6



Un RAID 6 usa una divisione a livello di blocchi con i dati di parità distribuiti *due volte* tra tutti i dischi.

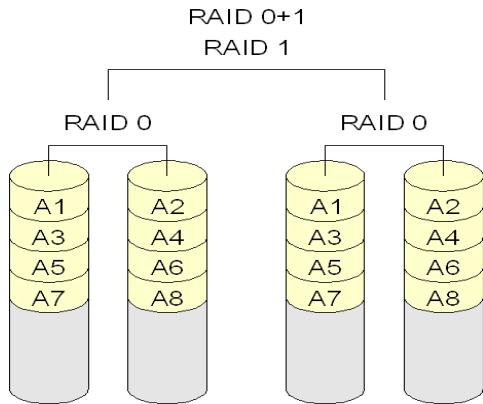
Risulta più ridondante del RAID 5 e il throughput è leggermente più alto a causa dell'esistenza di un disco in più rispetto al RAID 5.

13.5.8 RAID ibridi: RAID 0 + 1 e RAID 1 + 0

Le configurazioni ibride che usano le caratteristiche dei RAID 0 e 1 sono spesso usate in pratica per ottenere sia elevati tassi di trasferimento, come nel RAID 0, sia elevata affidabilità, come nel RAID 1.

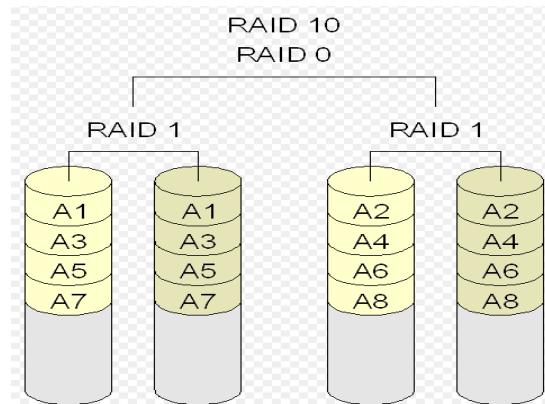
La differenza tra il RAID 0+1 e il RAID 1+0 è la diversa disposizione di ogni sistema RAID

RAID 0 + 1



Effettua prima lo striping, cioè alterna i dati su diversi dischi, come nel RAID 0; poi copia gli stessi dati su un altro disco, come nel RAID 1.

RAID 1 + 0



Copia prima gli stessi dati su un altro disco, come nel RAID 1; poi alterna i dati su dischi diversi.

13.6 Principi del software di I/O

Abbiamo detto che per effettuare l'I/O ci sono fondamentalmente tre tecniche:

- **I/O programmato**

Il processore genera un comando di I/O verso un modulo di I/O per conto di un processo; a questo punto il processo rimane in attesa attiva aspettando, prima di procedere, che l'operazione sia completata.

- **I/O Interrupt Driven (guidata dalle interruzioni)**

Il processore genera un comando di I/O per conto di un processo, continua ad eseguire le istruzioni seguenti, e viene interrotto dal modulo di I/O quando quest'ultimo ha completato il suo lavoro. Le istruzioni successive possono appartenere allo stesso processo, se non è necessario che esso aspetti il completamento dell'I/O, altrimenti, il processo è sospeso in attesa dell'interrupt e nel frattempo viene eseguito un altro lavoro.

- **DMA**

Un modulo di DMA controlla lo scambio di dati tra la memoria principale e un modulo di I/O; il processore invia una richiesta di trasferimento di un blocco di dati al modulo di DMA, e viene interrotto solo dopo che l'intero blocco è stato trasferito.

13.6.1 Scopi del software di I/O

Un concetto chiave nel progetto del software di I/O è l'**indipendenza dal dispositivo fisico**; ciò significa che dovrebbe essere possibile scrivere programmi dai quali si possa accedere ad ogni dispositivo, senza dover specificare in anticipo il dispositivo stesso.

Strettamente correlato a questo, è il problema della **denominazione uniforme**. Il nome di un file o di un dispositivo, dovrebbe essere semplicemente una stringa di caratteri o un numero intero, e non dovrebbe in alcuna maniera dipendere dal dispositivo. In Unix, tutti i dischi si possono montare sul **file system gerarchico in posti arbitrati**, così che l'utente non deve necessariamente sapere quale dispositivo corrisponda ad un certo nome.

Un altro concetto importante del software di I/O è il **trattamento degli errori**. In generale gli errori dovrebbero essere trattati il più vicino possibile all'hardware.

Un altro concetto chiave è il **trasferimento sincrono** piuttosto che asincrono (guidato dall'interruzione). La maggior parte dell'I/O è asincrono: la CPU fa partire il trasferimento e passa a fare qualcos'altro, finché non arriva l'interruzione; ma i programmi utente sono più facili da scrivere se le operazioni sono **sincrone**: dopo un comando di read, il programma viene automaticamente sospeso finché i dati non sono disponibili nel buffer.

Il concetto finale riguarda i **dispositivi dedicati** e quelli **condivisibili**; alcuni dispositivi come i dischi, possono essere usati da più utenti contemporaneamente; altri, come i dispositivi a nastro, devono essere dedicati a un singolo utente.

13.6.2 I/O programmato

Nell'I/O programmato tutto il lavoro è assegnato alla CPU.

In questo tipo di I/O, quando un programma utente vuole effettuare un'operazione di I/O deve effettuare una **system call** per richiamare l'attenzione del kernel, che eseguirà le operazioni al suo posto. Durante tutta l'esecuzione dell'operazione di I/O, il processore sarà occupato e non avrà la possibilità di eseguire altri processi.

Esempio:

Supponiamo sia la stampante il dispositivo coinvolto nell'operazione di I/O. Il processo utente genera prima una system call per acquisire il dispositivo, non appena è libero lo acquisirà. Dopodiché effettua un'altra system call per richiamare il kernel e per trasferire i dati da stampare dallo spazio utente allo spazio kernel. Il kernel, poi, si occuperà di trasferire una parte di questi dati alla volta alla stampante per stamparli. durante queste operazioni, la CPU è impegnata a controllare ripetutamente lo stato della stampante, che sarà occupato fino a che la stampa non sarà completata. Soltanto ora la CPU verificherà che lo stato risulta essere libero e potrà, quindi, effettuare altre operazioni.

Questo tipo di comportamento, cioè il fatto che la CPU controlla in continuazione lo stato della stampante anche se quest'ultimo risulta sempre occupato è chiamato **polling** o **attesa attiva** (*busy waiting*). Questo comporta un utilizzo a tempo pieno di CPU che durante l'elaborazione di un'operazione di I/O resta impegnata per tutto il tempo. L'attesa attiva è inefficiente in quanto spreca CPU soprattutto in sistemi complessi che richiedono molte operazioni di I/O.

13.6.3 I/O guidato dalle interruzioni

In questo tipo di I/O viene effettuata una system call dal programma utente per richiamare l'attenzione del kernel. Il processore è parzialmente coinvolto nell'operazione di I/O. Infatti ogni volta che il dispositivo di I/O coinvolto nell'operazione entra in funzione, viene generato un interrupt per permettere alla CPU di effettuare altre operazioni.

Esempio:

Prendiamo in considerazione la stampante. Viene effettuata una system call per acquisirla. Quindi viene effettuata un'altra system call per trasferire i dati da stampare dallo spazio utente allo spazio kernel.

Ogni volta che il kernel invia una parte di questi dati alla stampante, la CPU effettua altre operazioni mentre la stampante è impegnata nella stampa di questa parte di dati.

Quando la stampante ha terminato di stampare questi dati, viene generato un interrupt che fa sì che la Cpu interrompa le operazioni che stava eseguendo per fornire un'altra parte di dati alla stampante.

Con questa modalità, la CPU dedica una parte del suo tempo a controllare l'evoluzione dell'operazione di I/O ed un'altra parte per eseguire altre operazioni.

13.6.4 I/O mediante DMA

Un ovvio svantaggio dell'I/O guidato dalle interruzioni è che si ha uno spreco di tempo di CPU ogni volta che viene generata un'interruzione. Nell'esempio della stampante, si richiede un'interruzione per ogni carattere, e ciascuna di queste interruzioni spreca una certa quantità di tempo della CPU.

Una soluzione è quella di usare il DMA (*Direct Memory Access*); l'idea è quella che il controllore del DMA emette i caratteri e li passi alla stampante uno alla volta, senza disturbare la CPU.

La tecnica del DMA funziona come segue: quando avviene una richiesta di I/O, la CPU avvia il modulo DMA indicandogli le seguenti informazioni:

- il tipo di operazione, cioè se si tratta di una lettura o una scrittura (comunicato usando la linea di controllo di lettura o scrittura tra processore e modulo di DMA)
- l'indirizzo di dispositivo di I/O coinvolto (comunicato sulle linee dati)
- la posizione iniziale in memoria da cui leggere o scrivere (comunicata sulle linee dati e memorizzata dal modulo DMA nel suo registro di indirizzi)
- il numero di parole da leggere o scrivere (ancora una volta comunicato attraverso le linee dati e memorizzato nel data count register)

A questo punto il processore può eseguire altre istruzioni, avendo delegato questa operazione di I/O al modulo DMA. In essenza, il DMA è I/O programmato, con la differenza che è il controllore del DMA a fare tutto il lavoro invece che la CPU. Il guadagno che si ottiene usando il DMA è quello di ridurre il numero di interruzioni, da una per ogni carattere da stampare a una per ogni buffer stampato: se ci sono molti caratteri e le interruzioni sono lente questo può essere un miglioramento sostanziale. D'altro canto il controllore del DMA è solitamente molto più lento della CPU e, se il controllore non è in grado di far andare il dispositivo a velocità massima, o se la CPU non ha molto da fare mentre sta aspettando l'interruzione dal DMA, allora l'I/O guidato dalle interruzioni o addirittura l'I/O programmato possono essere scelte migliori.

13.6.5 DMA breakpoint e interrupt

La CPU e il DMA competono per l'utilizzo del bus di sistema. Il DMA deve forzare la CPU a sospendere temporaneamente le operazioni, cioè gli ruba un ciclo di bus (*cycle stealing*).

Quando deve essere effettuata un'operazione di I/O, la Cpu passa al DMA le quattro informazioni necessarie ed è il DMA che la porta a termine, trasferendo un blocco alla volta. Quando l'operazione è conclusa, il DMA genera un interrupt per avvertire la CPU.

I DMA breakpoints sono dei punti dove la CPU può essere fermata per sottrarre un ciclo di bus. Sono 3 i momenti in cui può avvenire questa situazione:

- prima della fase di fetch
- dopo la fase di decode
- dopo la fase di execute

La differenza sostanziale tra DMA breakpoints e interrupt sta nel fatto che se viene generato un DMA breakpoints non viene salvato nessun contesto (cioè, invece, avviene per l'interrupt).

13.7 Driver di dispositivo

Ogni dispositivo collegato al computer ha bisogno di codice specifico che controlli il dispositivo stesso. Tale codice è il *driver del dispositivo*. E' detto driver l'insieme di procedure che permette ad un SO di utilizzare un dispositivo hardware senza sapere come esso funzioni.

Ogni dispositivo ha un suo driver anche se in realtà possono esistere unici driver per più di un dispositivo dello stesso tipo. Tale situazione resta comunque poco comune per la grande diversità che vi è tra un dispositivo e l'altro.

I driver sono caricati dalla procedura di boot del sistema in base alla classe dei dispositivi di I/O connessi al sistema. In alternativa, i driver possono essere caricati quando necessario durante il funzionamento del SO (ad esempio una penna USB); questa caratteristica è particolarmente utile per la funzione *plug-and-play*.

Solitamente i driver di dispositivo sono posizionati al di sotto del resto del SO. In realtà sarebbe anche possibile trattare i driver di dispositivo a livello utente utilizzando chiamate di sistema per interfacciarsi con il kernel; questa situazione porterebbe anche il vantaggio di isolare il kernel dai dispositivi evitando crash causati dai driver che in genere interferiscono con il kernel. Le moderne architetture, in ogni caso, utilizzano driver che sono installati all'interno del SO.

13.8 Scheduling del disco

Una delle funzioni dello IOCS e dei driver dei dispositivi è quella di utilizzare lo *scheduling del disco* per eseguire le operazioni di I/O in un ordine tale da ridurre il movimento delle testine e il tempo medio di attesa delle operazioni di I/O. Quindi al variare del tipo di accesso, si influenzano diversamente le prestazioni del sistema. Resta il fatto che l'obiettivo comune di tutte le politiche di scheduling è quello di servire il massimo numero di richieste nell'unità di tempo.

FIRST-COME, FIRST-SERVED (FCFS)

Seleziona l'operazione di I/O con tempo di richiesta inferiore, cioè le richieste vengono servite in modo sequenziale, in base all'ordine di arrivo.

Questa, solitamente, è la politica peggiore: si ottengono buone prestazioni solo se molte richieste riguardano settori ravvicinati.

SHORTEST SEEK TIME FIRST (SSTF)

Seleziona l'operazione di I/O con il più breve tempo di ricerca rispetto alla posizione corrente delle testine del disco, cioè seleziona la richiesta che richiede il minor movimento della testina dalla sua posizione corrente.

Può essere soggetta a starvation: se il disco viene usato molto frequentemente il braccio tenderà a rimanere sempre nella parte vicina alla traccia richiesta correntemente, quindi le richieste relative alle tracce più esterne dovranno aspettare molto tempo prima di essere servite.

SCAN

Questa politica è nota come *algoritmo dell'ascensore*. Muove le testine del disco da un estremo all'altro del piatto, servendo tutte le richieste che ci sono lungo la sua strada indipendentemente dal loro ordine di arrivo; una volta raggiunta l'ultima traccia, cioè l'altra estremità del piatto, la direzione di movimento viene invertita e le nuove richieste vengono servite nella scansione inversa. Una variante chiamata **LOOK** inverte la direzione delle testine del disco quando non ci sono più richieste di I/O nella direzione corrente.

Le prestazioni risultano buone se vi sono carichi elevati, infatti uno spostamento della testina pari a due volte il numero di tracce, permette di soddisfare qualunque coda di richieste.

CIRCULAR SCAN O C-SCAN

Questa politica esegue la scansione come nello scheduling SCAN. Tuttavia, non esegue mai la scansione inversa, quindi la scansione viene ridotta ad un'unica direzione.

Invece di procedere in direzione opposta, questa politica sposta le testine nella posizione di partenza sul piatto e inizia un'altra scansione. La variante *circular look* (che chiameremo **C-LOOK**) muove le testine solo finché ci sono richieste da eseguire prima di iniziare una nuova scansione.

Il vantaggio rispetto alla SCAN è che i tempi di attesa delle richieste sono più uniformi.

N-STEP-SCAN E F-SCAN

Sia con SSTF che con SCAN o C-SCAN può succedere che il braccio non si muova da un determinato settore per un lungo periodo di tempo, se, ad esempio, uno o più processi hanno alte frequenze di accesso ad una particolare traccia. Per evitare questo fenomeno si possono usare due approcci:

- **n-step-scan**: la coda viene suddivisa in sottocode sulle quali viene applicato lo SCAN fin quando non vengono svuotate. n è il numero di sottocode, dunque se $n=1$ allora l'algoritmo si trasforma in un semplice algoritmo di SCAN
- **f-scan**: la coda viene suddivisa in due sottocode. La prima coda viene servita con lo SCAN finché non si svuota, le richieste che arrivano successivamente vengono accodate nella seconda coda che verrà servita solo dopo che la prima è stata svuotata

CONSIDERAZIONI SUGLI ALGORITMI DI SCHEDULING DEL DISCO

L'algoritmo **SSTF** è il più comune ed è quello più "naturale".

SCAN e C-SCAN hanno le migliori prestazioni nei sistemi con un alto carico di richieste da soddisfare.

In generale le performance dipendono dal numero e dal tipo di richieste, inoltre anche il tipo di file system può influire sulle performance dello scheduling del disco.

L'algoritmo di scheduling del disco dovrebbe essere implementato come un modulo separato del sistema operativo in modo da poterlo rimpiazzare con un algoritmo differente se necessario. Sia SSTF che LOOK sono una scelta ragionevole come algoritmo di default.

13.9 Buffering e blocking dei record

Per migliorare le prestazioni dell'attività di elaborazione di un file all'interno di un processo, un *metodo di accesso* ricorre alle tecniche di **buffering** e **blocking** dei record.

La tecnica del **buffering** tenta di sovrapporre le attività di I/O e di CPU di un processo. Questo obiettivo è raggiunto in due modi:

- **prefetching** di un record di input in un buffer di I/O
- **postwriting** di un record di output da un buffer di I/O

Un *buffer* è un'area di memoria utilizzata per mantenere temporaneamente i dati coinvolti in un'operazione di I/O, cioè dei dati che, o devono essere letti da un dispositivo o devono essere scritti su di esso.

Per un file di input, la tecnica del buffering usa il *prefetching*: viene anticipata la lettura del prossimo dato per poi memorizzarlo nel buffer.

In pratica, mentre il processo è impegnato nell'elaborazione di un dato, viene avviata un'operazione che legge il dato successivo e lo memorizza in un buffer prima che questo sia richiesto dal processo.

Questa tecnica è capace, quindi, di ridurre o addirittura eliminare il tempo di attesa, in quanto vengono sovrapposte le due operazioni (elaborazione del dato corrente, lettura del dato successivo).

Per un file di output, la tecnica del buffering usa il *postwriting*: il record che deve essere scritto viene semplicemente copiato in un buffer quando il processo esegue un'operazione di scrittura in modo che l'esecuzione del processo possa proseguire. L'effettiva scrittura è eseguita dal buffer in un momento successivo.

La tecnica del *blocking* riduce il numero di operazioni di I/O da eseguire, leggendo o scrivendo molti record in una singola operazione di I/O. In pratica legge più dati da un dispositivo in una singola operazione di I/O rispetto a quelli richiesti da un processo.

Quando più record vengono letti o scritti insieme, è necessario differenziare l'accesso e l'elaborazione dei dati e come vengono scritti sul dispositivo di I/O:

- un *record logico* è l'unità di dati utilizzata in un processo per l'accesso e l'elaborazione
- un *record fisico*, chiamato anche *blocco*, è l'unità dei dati utilizzata per il trasferimento da e per un dispositivo di I/O

Il fattore di *blocking* di un file è il numero di record logici in un record fisico. Si dice che un file adotta il *blocking* dei record se il fattore di blocking è maggiore di 1.

Le azioni necessarie per l'estrazione di un record logico da un blocco, in modo che possa essere utilizzato in un processo, sono collettivamente chiamate *azioni di deblocking*.

13.10 Cache del disco e dei file

Una tecnica generale per velocizzare l'accesso ai dati consiste nell'utilizzare una gerarchia di memoria composta da una parte della memoria e dai file memorizzati sul disco.

Il *caching* è la tecnica di mantenere alcuni file in memoria, in modo che vi si possa accedere senza dover eseguire un'operazione di I/O. Il caching riduce il numero di operazioni di I/O necessarie per accedere ai dati memorizzati in un file, migliorando le prestazioni delle attività di elaborazione dei file nei processi e migliorando inoltre le prestazioni del sistema.

L'IOCS fisico implementa una *cache del disco* per ridurre il numero di operazioni di I/O per accedere ai file memorizzati sul disco. Un metodo di accesso implementa una *cache di file* per ridurre il numero di operazioni di I/O eseguite durante l'elaborazione di un file.

CAPITOLO 14. Sicurezza e protezione

I SO adottano misure di sicurezza e protezione per evitare che un utente non autorizzato possa utilizzare, illegalmente, le risorse di un sistema o interferire con esse in qualsiasi modo.

Le misure di sicurezza riguardano le minacce provenienti dall'esterno di un sistema.

Le misure di protezione si occupano delle minacce interne.

Il principale strumento di sicurezza sono le password, che ostacolano i tentativi, da parte di utenti non autorizzati, di fingersi l'entità legittima. La confidenzialità delle password è mantenuta tramite la cifratura.

Per quanto riguarda la protezione dei dati memorizzati su un PC, un primo aiuto è dato dalla possibilità di impostare i privilegi di accesso a tali dati per ciascun utente della macchina. La funzione di protezione del SO assicura, allora, che tutti gli accessi al file avverranno, rigorosamente, secondo i privilegi di accesso specificati.

In questo capitolo verranno trattati i tipi di violazione della sicurezza come i *cavalli di Troia*, i *virus*, i *worm* e il *buffer overflow*. In seguito verrà discussa la *cifratura*. Infine verranno descritte le tre strutture di protezione più note: le *liste di controllo degli accessi*, la *capability list* e i *domini di protezione*.

14.1 Sicurezza e protezione: introduzione

Uno dei tre obiettivi fondamentali di un SO è assicura la protezione e la sicurezza di una risorsa, dove con risorsa si può intendere un file, un dispositivo hardware, un servizio offerto dal SO, ecc.

Durante il funzionamento di un sistema si possono verificare diversi tipi di interferenza; chiameremo ognuna di esse *minaccia*.

Una delle minacce più comuni è l'accesso non autorizzato alle risorse. Questa minaccia riguarda sia il tentato accesso al sistema degli utenti non registrati ad esso, sia l'accesso alle risorse del sistema da parte di utenti registrati ma che non hanno l'autorizzazione per accedervi.

Una minaccia meno comune consiste nell'impedire l'accesso legittimo alle risorse agli utenti autorizzati. Questa minaccia è meglio nota come *DOS – Denial Of Service* (negazione di servizio).

Le tecniche implementate nei SO per rispondere alle minacce possono essere divise in due categorie:

- le misure di *sicurezza* proteggono i dati e i programmi di un utente dall'interferenza da parte di persone o programmi al di fuori del SO; ci riferiamo, in senso lato, a tali persone e ai loro programmi come *non utenti*
- le misure di *protezione* proteggono i dati e i programmi di un utente dall'interferenza da parte degli altri utenti del sistema

| Termino | Descrizione |
|----------------|--|
| Autenticazione | L'autenticazione è la fase di verifica dell'identità dell'utente. I sistemi operativi, molto spesso, eseguono l'autenticazione <i>per conoscenza</i> . Ad una persona, che afferma di essere l'utente X, viene richiesto di provare qualche conoscenza condivisa solo tra il SO e l'utente X, come, per esempio una password. |
| Autorizzazione | L'autorizzazione ha due aspetti: <ol style="list-style-type: none"> 1. assegnare un insieme di privilegi di accesso a un utente; per esempio, a un utente sono stati concessi i privilegi di lettura e scrittura su un file, mentre ad altri sono stati concessi quelli di sola lettura; 2. verificare un diritto di accesso dell'utente a una risorsa con determinate modalità. |

La tabella descrive due metodi usati dai SO per l'implementazione della sicurezza e della protezione.

L'autenticazione ha come obiettivo la sicurezza e consiste nel verificare l'identità di una persona.

Si parla di autenticazione per conoscenza quando il sistema si basa sulla seguente assunzione: una persona è l'utente che dice di essere se sa qualcosa che solo il SO e l'utente sanno, come una password.

Si parla di autenticazione biometrica quando il sistema si basa su cose che solo l'utente ha; ad esempio le impronte digitali o l'iride.

L'autorizzazione ha come obiettivo l'implementazione della protezione. Consiste (1) nell'assegnare a un utente un privilegio di accesso ad una risorsa, e (2) nel determinare se un utente ha il diritto di accesso alla risorsa con le modalità specifiche.

MECCANISMI E POLITICHE

| | |
|------------|--|
| Sicurezza | <ul style="list-style-type: none"> <i>Politica:</i> una persona può o non può diventare un utente del sistema. L'amministratore di sistema impiega una politica nel registrare nuovi utenti. <i>Meccanismi:</i> aggiungere o cancellare utenti, verificare se una persona è un utente registrato (per esempio, eseguire l'autenticazione), utilizzare la cifratura per assicurare la confidentialità delle password. |
| Protezione | <ul style="list-style-type: none"> <i>Politica:</i> il proprietario del file specifica la politica di autorizzazione per un file. Decide quali utenti possono accedere a un file e in quale modo. <i>Meccanismi:</i> impostare o modificare le informazioni di autorizzazione per un file. Controllare se una richiesta di elaborazione file è compatibile con i privilegi utente. |

La tabella descrive i meccanismi e le politiche in ambito sicurezza e protezione.

14.1.1 Obiettivi di sicurezza e protezione

| Obiettivo | Descrizione |
|-------------|---|
| Segretezza | Solo gli utenti autorizzati hanno accesso alle informazioni. Questo obiettivo è anche detto <i>confidenzialità</i> . |
| Privatezza | Le informazioni sono usate solo per gli scopi per i quali erano state intese e condivise. |
| Autenticità | È possibile appurare la sorgente o il mittente dell'informazione e verificare inoltre che l'informazione è stata conservata nella forma in cui era stata creata od inviata. |
| Integrità | Non è possibile distruggere o alterare le informazioni, per esempio cancellando un disco. |

La tabella descrive i 4 obiettivi di sicurezza e protezione, cioè *segretezza*, *privatezza*, *autenticità* e *integrità* dell'informazione. Dei quattro obiettivi, solo la *privatezza* è un elemento esclusivamente legato alla protezione, mentre *segretezza*, *autenticità* e *integrità* riguardano sia la protezione che la sicurezza.

14.1.2 Minacce alla sicurezza e alla protezione

Consideriamo un SO tradizionale. Le sue procedure di autenticazione assicurano che solo gli utenti registrati possono collegarsi al sistema e attivare processi; di conseguenza, il SO sa quale utente ha iniziato uno specifico processo e quindi, può prontamente controllare se, al processo, è consentito usare una specifica risorsa. Quando i processi comunicano con altri processi, anche le azioni del SO relative alla comunicazione sono confinate sullo stesso sistema, quindi un accesso illegale a una risorsa sono, più che altro, minacce alla protezione piuttosto che alla sicurezza.

Se invece consideriamo un sistema collegato ad internet, le minacce sono perlopiù minacce alla sicurezza. In questi sistemi il verificarsi di minacce alla sicurezza è molto più facile.

14.2 Attacchi alla sicurezza

I tentativi di infrangere la sicurezza di un sistema sono detti *attacchi alla sicurezza* e la persona o il programma che effettua l'attacco è detto *intruso*.

Gli attacchi alla sicurezza più comuni sono due:

- **impersonificazione (masquerading)**: assumere l'identità di un utente registrato del sistema attraverso strumenti illegittimi
- **denial of service (DoS)**: impedire che gli utenti registrati del sistema accedano alle risorse per cui posseggono i privilegi di accesso

In un attacco *masquerading*, concluso con successo, l'intruso ottiene l'accesso alle risorse alle quali è autorizzato l'utente che sta impersonando, di conseguenza può modificare o distruggere programmi e dati appartenenti ad esso. Consiste essenzialmente nell'attivazione di programmi, come *trojan*, *virus* e *worm*, costituiti da codice "infetto".

Un attacco di tipo **DoS**, è lanciato sfruttando i bug del SO, in particolare i bug riguardanti la sua progettazione e il suo funzionamento. In pratica è una tecnica per tempestare di richieste un singolo servizio al fine di farlo collassare.

Si può lanciare un attacco DoS tramite diversi mezzi; molti degli strumenti utilizzati in questi attacchi sono legittimi, il che rende difficile la loro individuazione per il SO.

Ad esempio, un attacco DoS può essere lanciato sovraccaricando una risorsa con richieste fittizie in modo che non possa essere utilizzata dagli utenti legittimi.

Un attacco DoS può essere lanciato modificando un programma che offre un servizio, o distruggendo qualche informazione di configurazione nel kernel, per esempio l'uso di un dispositivo di I/O può essere negato cambiando la sua entry nella tabella dei dispositivi del kernel.

Un attacco Dos di rete può essere lanciato inondando la rete con messaggi diretti ad un particolare server, in modo tale da saturargli la banda, e quindi impedire al server di gestire i messaggi "genuini".

14.2.1 Trojan, virus e worm

I *trojan*, i *virus* e i *worm* sono programmi contenenti codice in grado di lanciare un attacco alla sicurezza quando sono attivati.

| Minaccia | Descrizione |
|------------------|--|
| Cavallo di Troia | Un programma che esegue una funzione autorizzata, nota a un SO o ai suoi utenti e che, inoltre, ha una componente nascosta che può essere usata successivamente per scopi illegali come attacchi alla sicurezza dei messaggi o attacchi di masquerading. |
| Virus | Un pezzo di codice che si unisce ad altri programmi del sistema e si propaga ad altri sistemi quando tali programmi vengono copiati o trasferiti. |
| Worm | Un programma che si propaga ad altri sistemi sfruttando i punti deboli della sicurezza nel SO come la debolezza negli strumenti per la creazione di processi remoti. |

La tabella sintetizza le loro caratteristiche.

I *trojan* sono genericamente software malevoli (malware) nascosti all'interno di programmi apparentemente utili, e che dunque l'utente esegue volontariamente. Il tipo di software malevolo che verrà silenziosamente eseguito in seguito all'esecuzione del file da parte dell'utente può essere sia un virus che un qualunque tipo di minaccia informatica poiché permette al cracker che ha infettato il PC di risalire all'indirizzo IP della vittima. L'intruso può quindi controllare il PC infettato da remoto, può "spiarlo", può negargli servizi, ecc.

I *virus* sono software, appartenenti alla categoria dei malware, in grado, una volta eseguiti, di infettare dei file in modo da riprodursi facendo copie di sé stessi, generalmente senza farsi rilevare dall'utente.

Un virus è composto da un insieme molto piccolo di istruzioni, che viene aggiunto alla istruzione di un programma eseguibile presente sulla macchina. È solitamente composto da un numero molto ridotto di

istruzioni in modo da rendersi il più possibile invisibile. Caratteristica principale di un virus è quella di riprodursi e quindi diffondersi nel computer ricoppiando il codice infetto anche in altri file eseguibili presenti nel sistema.

I virus possono essere o non essere direttamente dannosi per il SO che li ospita: solitamente un virus danneggia solo il software della macchina che lo ospita, anche se esso può potenzialmente provocare danni anche all'hardware, ad esempio causando il surriscaldamento della CPU mediante overclocking, oppure fermando la ventola di raffreddamento.

Un virus tipicamente configura una **backdoor**, o *porte di servizio*, che consentono all'intruso di superare in parte o in tutto le procedure di sicurezza attivate in un sistema informatico. Con l'attivazione di una backdoor un utente esterno può prendere il controllo remoto della macchina senza l'autorizzazione del proprietario.

Un **worm** è una particolare categoria di malware in grado di autoreplicarsi. È simile ad un virus, ma a differenza di questo non necessita di legarsi ad altri eseguibili per diffondersi. E' molto più difficile tracciarlo rispetto a un virus proprio a causa della sua natura autoreplicante. I worm sono noti per il tasso inimmaginabilmente alto di replicazione.

Gli attacchi alla sicurezza lanciate mediante trojan, virus o worm possono essere contrastati attraverso le seguenti misure:

- installare sulla macchina solo software "affidabile"
- usare buoni antivirus
- risolvendo i bug e le falte del SO attraverso l'installazione delle patch (non è compito degli utenti ma dei fornitori e produttori del SO)

14.3 Cifratura

La **cifratura** è l'applicazione di una trasformazione algoritmica ai dati in modo tale da proteggere i dati stessi

La **crittografia** è quella branca della crittologia che si occupa di *tecniche di cripitura*.

L'obiettivo è quello di prendere un messaggio o un file, chiamato **messaggio in chiaro**, e crittarlo in **messaggio cifrato** in modo tale che soltanto un utente, che conosce come ripristinare la forma originale del dato, lo può utilizzare. Questa caratteristica aiuta a mantenere la **riservatezza** del dato.

La **segretezza** del dato dipende dai parametri dell'algoritmo, cioè dalle **chiavi di cripitura**: l'efficacia della cripitura dipende dal fatto che un intruso possa determinare la chiave di cripitura attraverso tentativi ed errori. In pratica, il **messaggio cifrato** è ottenuto usando un algoritmo di crittazione (noto), un **messaggio in chiaro** ed una chiave di cripitura (non nota).

| Termino | Descrizione |
|-------------------------------------|--|
| Cifratura | La cifratura è l'applicazione di una trasformazione algoritmica E_k ai dati, dove E è un <i>algoritmo di cifratura</i> e k è una <i>chiave di cifratura</i> . È usato per proteggere la riservatezza dei dati. Il dato originale è ricostruito applicando una trasformazione $D_{k'}$, dove D è un <i>algoritmo di decifratura</i> e k' è una <i>chiave di decifratura</i> . Uno schema che usa $k = k'$ è detto <i>cifratura simmetrica</i> , e uno che usa $k \neq k'$ è detto <i>cifratura asimmetrica</i> . |
| Plaintext (testo in chiaro) | Il dato da cifrare. |
| Ciphertext (testo cifrato) | La forma cifrata del testo in chiaro. |
| Confusione | Il principio di Shannon della confusione richiede che i cambiamenti causati in un testo cifrato dovuti a una modifica del testo in chiaro non saranno facili da individuare. |
| Diffusione | Il principio di Shannon della diffusione richiede che l'effetto di una piccola sottostringa nel testo in chiaro sia ripartito su molte cifre nel testo cifrato. |
| Attacchi ai sistemi di crittografia | Un <i>attacco</i> è una serie di tentativi, da parte di un intruso, di trovare una funzione di decifratura D_k . In un attacco <i>solo testo cifrato (ciphertext only)</i> , l'intruso può esaminare solo un insieme di testi cifrati per determinare D_k . In un attacco <i>testo in chiaro conosciuto (known plaintext)</i> , l'intruso ha l'opportunità di esaminare la forma in chiaro e quella cifrata dello stesso dato, mentre in un attacco <i>testo in chiaro selezionato (chosen plaintext)</i> l'intruso può scegliere un testo in chiaro e ottenere la sua forma cifrata per eseguire l'attacco. |
| Funzione one-way | Una funzione, il calcolo della cui inversa è talmente oneroso da essere considerato impraticabile. Il suo utilizzo come funzione di cifratura rende difficili gli attacchi di crittografia. |
| Cifratura a blocchi | La tecnica della cifratura a blocchi (block cipher) consiste nel sostituire blocchi di misura fissa del testo in chiaro con blocchi del testo cifrato. Ciò introduce un po' di confusione, ma non introduce sufficiente diffusione. |
| Cifratura a flusso | Sia il testo in chiaro che la chiave di cifratura sono considerati come flusso di bit (bit stream). I bit nel testo in chiaro vengono cifrati usando un ugual numero di bit della chiave di cifratura. Una cifratura a flusso (stream cipher) non introduce confusione e introduce una limitata diffusione; tuttavia, qualche sua variante può introdurre un elevato livello di diffusione. |
| DES | Il Data Encryption Standard del National Bureau of Standards, adottato nel 1976, usa una tecnica basata sulla cifratura a blocchi e prevede, come opzione, la cifratura a blocchi in cascata. Esso effettua 16 iterazioni, che eseguono trasformazioni complesse sul testo in chiaro o sul testo cifrato intermedio. |
| AES | L'Advanced Encryption Standard è il nuovo standard adottato dal National Institute of Standards and Technology (formalmente conosciuto come National Bureau of Standards) nel 2001. Esso esegue tra 10 e 14 cicli di operazioni, ognuna effettua solo sostituzioni e permutazioni, sui blocchi del testo in chiaro di 128, 192 o 256 bit. |

Usiamo la seguente notazione:

- d dato
- P_d messaggio in chiaro del dato d
- C_d messaggio cifrato del dato d
- E algoritmo di cifratura
- D algoritmo di decifratura
- k chiave di cifratura
- k^l chiave di decifratura

N.B. Nella *cifratura simmetrica*, la decifratura è eseguita utilizzando la stessa chiave k ; nella *cifratura asimmetrica*, la decifratura è eseguita utilizzando una chiave differente k^l .

Gli algoritmi di cifratura e decifratura sono noti, sono le chiavi ad essere segrete.

14.3.1 Attacchi ai sistemi di crittografia

Un attacco a un sistema di crittografia è composta da una serie di tentativi per individuare la funzione di decifratura D_k . La qualità della cifratura è migliore se il calcolo di questa funzione richiede una quantità non perseguitabile di tentativi e di tempo.

Nell'*attacco esaustivo (brute force)* si prova ripetutamente la funzione D_k finché non si riesce a trovare quella giusta. In pratica, in questo attacco, si provano tutte le possibilità per trovare D_k . Questo tipo di attacco prevede un elevato numero di tentativi. Per esempio, per infrangere una cifratura che usa una chiave a 56 bit potrebbero essere necessari 2^{56} tentativi. Si pensava che con l'aumentare dei bit nella chiave, la cifratura migliorasse. Invece, con l'utilizzo di tecniche matematiche avanzate, si può calare notevolmente il numero di tentativi per ottenere la funzione D_k .

Nell'*attacco solo testo cifrato*, un intruso ha accesso soltanto a una collezione di testi cifrati. Conseguentemente, per rendere l'attacco più efficace rispetto a quello esaustivo, l'intruso si basa sugli indizi dall'analisi delle stringhe nei testi cifrati e dalle caratteristiche dei testi in chiaro, per esempio se sono formati solo da parole del dizionario.

Nell'*attacco testo in chiaro conosciuto*, l'intruso conosce il testo in chiaro corrispondente a un testo cifrato. Questo attacco è applicabile se un intruso riesce a ottenere una posizione all'interno del SO da cui può osservare sia il testo in chiaro che il corrispondente testo cifrato. Recuperare un numero sufficiente di coppie di testo in chiaro – testo cifrato fornisce indici per determinare D_k .

Nell'*attacco testo in chiaro scelto*, un intruso è in grado di fornire un testo in chiaro e osservare la sua forma cifrata. Ciò consente all'intruso di creare sistematicamente una collezione di coppie testo in chiaro – testo cifrato a supporto dei tentativi e del raffinamento dei tentativi durante l'attacco.

14.3.2 Tecniche di cifratura

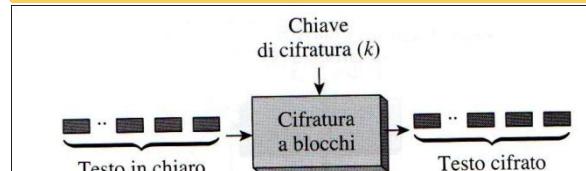
Le tecniche di cifratura si differenziano nel modo in cui tentano di ostacolare i tentativi dell'intruso nel trovare D_k . L'approccio fondamentale è mascherare le caratteristiche di un testo in chiaro.

La tecnica di cifratura più semplice è la classica *sostituzione del blocco*, o anche detta *cifratura per sostituzione*, che sostituisce ogni lettera in un testo in chiaro con un'altra lettera dell'alfabeto. Tale tecnica non maschera molto bene le caratteristiche di un testo in chiaro.

Di seguito descriveremo quattro schemi di cifratura.

CIFRATURA A BLOCCHI

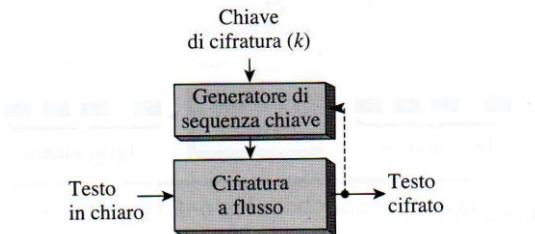
La cifratura a blocchi è un'estensione della cifratura per sostituzione. Esegue una sostituzione di blocchi di dimensione fissa di un testo in chiaro con blocchi del testo cifrato di uguale dimensione.



Blocchi identici in un testo in chiaro producono blocchi identici nel testo cifrato. Questa caratteristica rende l'approccio vulnerabile a un attacco testo in chiaro conosciuto o testo in chiaro scelto. Per rendere vita dura agli attacchi, è possibile creare blocchi più grandi.

CIFRATURA A FLUSSO

Una cifratura a flusso considera che un testo in chiaro, e la chiave di cifratura siano un flusso di bit. La cifratura è eseguita usando una trasformazione che prende alcuni bit del testo in chiaro e un egual numero di bit della chiave di cifratura. Questa tecnica risulta più veloce della cifratura a blocchi.



Esistono molte varianti di questa tecnica come la *cifratura vernam*, la *ciphertext autokey*, la *RC4*, ecc.

14.3.3 DES – Data Encryption Standard

Il DES è un algoritmo di cifratura scelto come standard per il governo degli Stati Uniti e in seguito diventato di utilizzo internazionale.

Si basa su un algoritmo a chiave simmetrica con chiave a 56 bit per cifrare blocchi di dati a 64 bit ed è pertanto un cifrario a blocchi. Il blocco di 64 bit è suddiviso in 8 sottoblocchi di 8 bit ciascuno; l'ultimo bit di ogni sottoblocco è di controllo, ecco perché i bit liberi che costituiscono la chiave sono 56.

Inoltre **fornisce una modalità di cifratura a blocchi a cascata (CBC).**

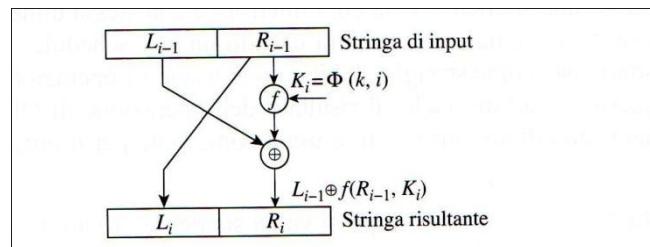
Il testo da cifrare viene suddiviso in blocchi di 64 bit ciascuno e vengono cifrati uno dopo l'altro in successione con uguale procedimento, che consta di tre passi: il passo iniziale di permutazione, il passo di trasformazione e il passo finale di permutazione.

Il primo blocco del testo è combinato con un vettore iniziale tramite l'operazione di XOR e poi viene cifrato; il testo cifrato risultante viene poi combinato con un secondo blocco del testo tramite l'operazione di XOR, poi viene cifrato, e così via.

Questo algoritmo accetta una stringa di input di 64 bit per fornire una stringa risultante della stessa dimensione.

FUNZIONAMENTO:

Alla prima iterazione, la stringa di input è il primo blocco del testo. In tutte le altre iterazioni, la stringa di input è l'output della precedente iterazione.



La stringa di input è divisa in due metà di 32 bit ciascuna. La metà di destra diventa la metà di sinistra della stringa risultante; viene, poi, eseguita una trasformazione complessa che coinvolge entrambe le metà (destra e sinistra) per ottenere la metà di destra della stringa risultante. Questa trasformazione è costituita da 16 passi, in ogni passo vengono effettuate delle trasposizioni e delle sostituzioni usando delle sottochiavi diverse per ogni passo. Le sottochiavi sono, comunque, ricavate dalla chiave originale.

Indichiamo con:

- $T(i)$ il risultato dell' i -esimo passo
- $L(i)$ il blocco sinistro
- $R(i)$ il blocco destro
- $K(i)$ la sottochiave di ogni passaggio

Avremo che:

- $T(i) = L(i) R(i)$ il risultato dell' i -esimo passo è l'unione dei due blocchi
- $L(i) = R(i-1)$ il blocco di destra diventa quello di sinistra
- $R(i) = L(i-1) \oplus f[R(i-1), K(i)]$

Vediamo come opera questa funzione:

- 1) il blocco di destra viene esteso da 32 a 48 bit con un modulo di espansione; indichiamo il blocco espanso con $E[R(i-1)]$
- 2) il blocco E viene combinato con la chiave K_i usando un'operazione di XOR (funzione f in figura)
 - la chiave K_i deriva dalla chiave di cifratura k , ma è diversa per ciascun passo i
- 3) il risultato del passo 2 è diviso in otto gruppi di 6 bit ciascuno, contenenti rispettivamente i bit 1-6, 7-12, 13-18 e così via; ciascuno di questi gruppi è l'input per una S-box che sostituisce i 6 bit in input con 4 bit in output; i risultati della sostituzione sono concatenati per ottenere una stringa a 32 bit
- 4) la stringa a 32 bit è permutata per ottenere un'altra stringa a 32 bit
- 5) questa stringa viene unita alla metà di sinistra della stringa di input mediante una XOR, per ottenere la metà di destra della stringa risultato

N.B. Le S-box sono delle matrici prefissate

Le S-box utilizzate al passo 3 introducono confusione, la permutazione del passo 4 introduce confusione, la XOR del passo 5 introduce confusione.

CONSIDERAZIONI:

Il DES effettua sia cifratura che decifratura utilizzando la stessa sequenza di passi, a parte il fatto che le chiavi sono utilizzate nell'ordine inverso durante la decifratura.

Il DES utilizza una chiave a 56 bit, che ai giorni d'oggi, sono crackabili in poche ore; la lunghezza della chiave rende il DES vulnerabile agli attacchi.

Negli anni 90, per sopperire a questo difetto, fu pubblicato un nuovo standard, il *Triple DES*, che effettuava tre iterazioni, ciascuna delle quali applicava l'algoritmo DES, usando una chiave diversa derivata dalla chiave di cifratura. Anche questo nuovo algoritmo, che poteva utilizzare chiavi di lunghezza fino a 168 bit, fu sostituito nel 2001 da un nuovo standard, l'*AES*.

14.3.4 AES – Advanced Encryption Standard

Questo algoritmo usa 128 bit (16 byte) come dimensione del blocco e chiavi a 128, 192 o 256 bit.

Un blocco di testo in chiaro di 16 byte è trattato come una matrice 4x4 byte detta *state*. Questa matrice è cifrata mediante molti cicli di operazioni, dove il numero di cicli dipende dalla lunghezza della chiave: sono necessaria 10 cicli per chiavi a 128 bit, 12 cicli per chiavi a 192 bit, 14 cicli per chiavi a 256 bit.

Ogni ciclo è composto dalle seguenti operazioni:

- 1) *Sostituzione di byte*: ogni byte della state è soggetto a una trasformazione non lineare applicata da una S-box
- 2) *Shifting di righe*: le righe della state sono shiftate ciclicamente rispettivamente di 0, 1, 2 e 3
- 3) *Mixing di colonne*: i 4 byte in una colonna sono sostituiti in modo tale che ogni byte risultante sia funzione di tutti e 4 i byte nella colonna
- 4) *Key addition*: una sottochiave, la cui dimensione è la stessa dimensione della state, è ricavata dalla chiave di cifratura usando un key schedule. La sottochiave e lo state sono viste come stringhe di bit e unite usando la XOR. Se questo è l'ultimo ciclo, il risultato della XOR è un blocco del testo cifrato; altrimenti, è usata come state per il prossimo ciclo di cifratura

Per far sì che la stessa sequenza di passi valga sia per la cifratura che per la decifratura, prima di cominciare il primo ciclo, viene eseguita una key addition e, nell'ultimo ciclo, viene saltato il passo mixing di colonne.

14.3.4 RSA – Rivest-Shamir-Adleman

RSA è un sistema a chiave pubblica, diversamente da DES ed AES che sono sistemi a chiave privata.

La crittografia a chiave pubblica risulta molto più lenta ed è strutturata nel modo seguente: si prende una coppia di chiavi (chiave pubblica, chiave privata) e si rende nota quella pubblica; la chiave pubblica rappresenta la chiave di crittazione, mentre quella privata è la chiave di decrittazione. Pertanto per spedire un messaggio segreto ad un utente, il mittente cripta il messaggio con la chiave pubblica del ricevente, e poiché solo tale persona possiede la chiave privata, sarà il solo a poter decriptare il messaggio.

La prima applicazione pratica basata sulle tecniche di crittografia a doppia chiave fu sviluppata nel 1978 da tre professori: Ronald Rivest, Adi Shamir e Leonard Adleman, che realizzarono una procedura di calcoli matematici che prenderà il nome di “**algoritmo RSA**”, dalle iniziali dei suoi inventori.

Il codice RSA si basa su un procedimento che utilizza i numeri primi e funzioni matematiche che è quasi impossibile invertire. Dati due numeri primi, è molto facile stabilire il loro prodotto, mentre è molto più difficile determinare, a partire da un determinato numero, quali numeri primi hanno prodotto quel risultato dopo essere stati moltiplicati tra loro. In questo modo si garantisce la sicurezza.

GENERARE LE CHIAVI:

Si considerano due numeri primi molto grandi, P e Q.

Calcolo $N = P * Q$

Calcolo $Z = (P-1) * (Q-1)$ sarà sicuramente un numero pari poiché prodotto di numeri pari

Scelgo un valore E, che sarà l'*esponente pubblico*, tale che sia primo rispetto a Z e minore di Z

Calcolo un valore D, che sarà l'*esponente privato*, tale che $(D^E) \bmod Z = 1$

Gli elementi pubblici saranno N ed E, quelli segreti saranno P, Q, Z e D.

CIFRATURA:

Il testo in chiaro viene visto come una sequenza di bit. Questa sequenza viene divisa in blocchi di K bit, dove K è il più grande intero tale da soddisfare la disequazione $2^K < N$.

Il blocco M sarà cifrato in questo modo:

$$M_c = M^E \bmod N \quad (\text{dove } M_c \text{ è il messaggio cifrato})$$

Il blocco MC sarà decifrato in questo modo:

$$M = M_c^D \bmod N \quad (\text{dove } M \text{ è il messaggio in chiaro})$$

Quindi per la cifratura del messaggio sono necessari E ed N (chiave pubblica: E, N), mentre per la decifratura del messaggio sono necessari D ed N (chiave privata: D, N).

Il codice RSA viene considerato sicuro perché non è ancora stato trovato il modo per fattorizzare numeri primi molto grandi, che nel nostro caso significa riuscire a trovare p e q conoscendo n. Nel corso degli anni l'algoritmo RSA ha più volte dimostrato la sua robustezza: in un esperimento del 1994, coordinato da Arjen Lenstra dei laboratori Bellcore, per “rompere” una chiave RSA di 129 cifre, svelando il meccanismo con cui quella chiave generava messaggi critografati, sono stati necessari 8 mesi di lavoro coordinato effettuato da 600 gruppi di ricerca sparsi in 25 paesi, che hanno messo a disposizione 1600 macchine da calcolo, facendole lavorare in parallelo collegate tra loro attraverso Internet.

Data la mole delle risorse necessarie per rompere la barriera di sicurezza dell'algoritmo RSA, è chiaro come un attacco alla privacy di un sistema a doppia chiave non sia praticamente realizzabile. Inoltre, nell'esperimento era stata utilizzata una chiave di 129 cifre mentre i programmi di crittografia attualmente a disposizione prevedono chiavi private con una “robustezza” che raggiunge e supera i 2048 bit, risultando quindi praticamente inattaccabili, visto anche che l'ordine di grandezza dei tempi necessari alla rottura di chiavi di questo tipo è esponenziale e passa in fretta da qualche giorno a qualche centinaia di anni.

14.3.5 Algoritmo dello zaino

E' un algoritmo a chiave asimmetrica e pubblica.

CALCOLO CHIAVE PRIVATA:

Si calcola una serie di valori $W = \{n_1, n_2, \dots, n_m\}$ tale che l' n -esimo valore sia maggiore della somma degli $n-1$ valori precedenti.

Poi si calcola la somma degli n valori di W e si considera un valore $Q >$ somma.

Infine si sceglie un valore R coprimo a Q e minore di Q .

Chiave privata: W, Q, R

CALCOLO CHIAVE PUBBLICA:

Si genera una sequenza β , dove ogni valore è dato da $(W_i * R) \bmod Q$

Chiave pubblica: β

CIFRATURA MESSAGGIO IN CHIARO:

Si prende il messaggio e lo si converte in binario.

Ogni bit viene moltiplicato per β_i

Si sommano tutti questi valori per ottenere il messaggio cifrato M_c

CIFRATURA MESSAGGIO CIFRATO:

Il messaggio M_c deve essere moltiplicato per $R^{-1} \bmod Q$.

$$M = M_c R^{-1} \bmod Q$$

14.4 Strutture di protezione

Per implementare la protezione dei file in un file system vengono utilizzati i privilegi di accesso al file.

Una struttura di protezione è un database che contiene le informazioni su quali utenti possono accedere, a quali file e con quali privilegi.

Un privilegio di accesso a un file è il diritto ad effettuare una specifica forma di accesso al file, come una lettura o una scrittura. Un utente può avere uno o più privilegi di accesso ad un file. Solitamente si usano le notazioni r, w e x per indicare rispettivamente la lettura, la scrittura e l'esecuzione sui dati o sui programmi.

Un descrittore di accesso è una rappresentazione di un insieme di privilegi di accesso ad un file. Ad esempio l'insieme $\{r, w\}$ indica che l'utente ha i privilegi di lettura e di scrittura. I sistemi moderni non usano i descrittori di accesso perché sono costosi in termini di memoria, ma utilizzano una rappresentazione a bit, cioè una stringa di bit, in cui ogni bit indica la presenza o l'assenza di uno specifico privilegio di accesso.

14.4.1 Granularità della protezione

Col termine granularità della protezione viene rappresentato il grado di discrezionalità che il proprietario di un file può esercitare in relazione alla protezione dei file.

| Granularità | Descrizione |
|---------------------------|--|
| Protezione a grana grossa | Privilegi di accesso a un file specificabili solo per gruppi di utenti. Ogni utente in un gruppo ha identici privilegi di accesso al file. |
| Protezione a gran media | Privilegi di accesso a un file specificabili individualmente per ciascun utente nel sistema. |
| Protezione a grana fine | Privilegi di accesso a un file specificabili per un processo o per una fase dell'esecuzione di un processo. |

I SO moderni ricorrono alla protezione a grana grossa perché, una protezione a grana fine o media comporta strutture di protezione di grandi dimensioni.

14.4.2 Matrice di controllo degli accessi - ACM

Una **matrice di controllo degli accessi** (ACM) è una struttura di protezione che fornisce accesso efficiente sia ai privilegi di accesso degli utenti ai vari file sia alle informazioni per il controllo degli accessi ai file. Ogni elemento dell'ACM contiene i privilegi di accesso di un utente a un file.

Ogni utente ha una riga nell'ACM, mentre ogni file ha una colonna. In questo modo, una riga nell'ACM descrive i privilegi di accesso di un utente per tutti i file nel sistema, ed ogni colonna descrive le informazioni per il controllo degli accessi a un file.

| | File → | | | |
|----------|----------|---------|-------|-----|
| ↓ Utenti | alpha | beta | gamma | |
| | Jay ↓ | {r} | {r,w} | |
| | Anita | {r,w,x} | | {r} |
| | Sheila | | | {r} |

↑
Informazioni di controllo
degli accessi alpha

Privilegi di accesso
di Anita

La figura mostra un'ACM.

L'ACM fornisce protezione a grana media. La matrice risulta essere di grandi dimensioni perché solitamente un SO contiene molti utenti e un numero elevato di file. Per questo motivo, un'ACM richiede un ampio spazio in memoria durante il funzionamento del sistema.

Inoltre, bisogna considerare anche che la maggior parte degli elementi in un'ACM contengono elementi nulli

14.4.3 Liste di controllo degli accessi (ACL)

La **lista di controllo degli accessi** (ACL) di un file è una rappresentazione delle informazioni per il controllo degli accessi; contiene gli elementi non nulli relativi alla colonna del file nell'ACM.

Una ACL è memorizzata come una lista di coppie del tipo (*ID_utente, privilegi_di_accesso*).

| Nome del file | Lista di controllo degli accessi (ACL) |
|------------------|---|
| alpha | <{(Jay, {r}), (Anita, {r, w, x})}} |
| beta | <{(Jay, {r, w})}} |
| gamma | <{(Anita, {r}), (Sheila, {r})}} |

L'ACL elimina la necessità di memorizzare i privilegi di accesso nulli, ma nonostante ciò, la presenza di un gran numero di utenti in un sistema comporta ACL di grandi dimensioni, e pertanto uno spreco di spazio.

La soluzione migliore è quella di utilizzare una protezione a grana grossa, raggruppando gli utenti in gruppi

14.4.4 Capability list (C-list)

Una **capability list** (C-list) rappresenta i privilegi di accesso di un utente a vari file nel sistema; essa contiene le entry, non nulle, che avrebbe contenuto la riga dell'utente nell'ACM. Ogni entry nella C-list è una **capability**, che rappresenta i privilegi di accesso ad un file; è costituita da una coppia del tipo (*file_ID, privilegi_di_accesso*).

| |
|--------------------|
| (alpha, {r, w, x}) |
| (gamma, {r}) |
| |

La figura mostra una C-list relativa all'utente *Anita*.

Le C-list sono solitamente di piccole dimensioni.

14.4.5 Dominio di protezione

La matrice di controllo degli accessi, la lista di controllo degli accessi o la capability list servono a conferire privilegi di accesso agli utenti. Questa soluzione è utile per la *segretezza*, obiettivo della sicurezza e della protezione in quanto solo gli utenti autorizzati possono accedere a un file. Ma da sole non riescono a soddisfare anche il requisito di riservatezza, che senza altri meccanismi potrebbe essere violato.

Ma bisogna considerare anche il requisito della *riservatezza*, uno degli obiettivi della sicurezza e della protezione: essa richiede che l'informazione sia usata solo per gli scopi stabiliti.

Per assicurare tale requisito, cioè evitare violazioni della riservatezza, entra in gioco il concetto di *dominio di protezione*. Questo specifica le risorse cui il processo può accedere. Ogni dominio definisce un insieme di oggetti e tipi di operazioni che si possono compiere su ciascun oggetto.

Si può pensare ad un dominio di protezione come a un "ambiente di esecuzione" teorico: un processo che agisce all'interno di un dominio di protezione, può accedere ai file per i quali il dominio di protezione possiede i privilegi di accesso. In pratica, ad un processo viene consentito di agire all'interno di un dominio di protezione solo se ha bisogno di accedere ai file per i quali il dominio di protezione ha i privilegi di accesso.

La riservatezza può essere migliorata consentendo a un processo l'accesso ad alcune risorse solo durante specifiche fasi della sua esecuzione. Per fare ciò, ad un processo è consentito cambiare il suo dominio di protezione durante l'elaborazione.

14.5 Unix

In Unix ogni utente ha un ID unico nel sistema. L'amministratore di sistema crea gruppi di utenti disgiunti e assegna un group_ID unico a ogni gruppo.

Le credenziali di un utente sono composte dalla sua user_ID e il suo group_ID e servono per l'autenticazione dell'utente.

Unix definisce tre classi di utente, proprietario del file, user group e altri utenti, e fornisce solo tre diritti di accesso, lettura (r), scrittura (w) ed esecuzione (x).

Per ogni classe di utente viene utilizzato un descrittore di accesso bit-encoded a 3 bit e l'ACL contiene i descrittori di accesso per le tre classi di utente nella sequenza proprietario, user group, altri utenti. In questo modo, l'ACL richiede solo 9 bit ed è memorizzata nell'i-node di un file. L'identità del proprietario del file è memorizzata in un altro campo dell'i-node del file.

| | |
|-------------------------------|-------|
| <code>r w - r - - r --</code> | sigma |
| <code>r - - r - - r --</code> | delta |
| <code>r w x - - - - -</code> | phi |
| ` ` ` | |
| proprietario user altri | |
| del file group utenti | |

Unix cambia il dominio di protezione di un processo sotto due condizioni: quando il processo effettua una chiamata di sistema e quando è usata la proprietà *setuid* o *setgid*.

La chiamata *setuid* può essere effettuata in due modi.

Un processo può effettuare una chiamata di sistema *setuid <id>* per cambiare la sua uid in *<id>*, e un'altra chiamata di sistema *setuid* con il suo ID per ritornare al suo stato originario.

In alternativa, l'uid può essere cambiato implicitamente quando un processo esegue una *exec* per eseguire un programma.