Sistemi operativi

7. File system

Mauro Fiorentini

7. File system

File system

- Componente del sistema operativo che gestisce i file.
  - Generalmente riferito alla gestione di file su dispositivi ad accesso diretto: dischi magnetici, CD etc.
  - In molti sistemi permette un accesso uniforme a tutti i dispositivi di I/O.

Caratteristiche di un file system

- I file system si differenziano per come affrontano i vari problemi e per le interfacce che rendono disponibili.
  - Naturalmente differiscono per l'implementazione.
- Sebbene i dettagli siano normalmente nascosti al programmatore, hanno influenza:
  - sulle interfacce disponibili;
  - sulle prestazioni per molte operazioni.

Device e file system

- Rispetto al dispositivo fisico, i file system differiscono per i seguenti aspetti:
  - possibilità di essere fisicamente distribuiti su macchine diverse;
  - possibilità di comprendere più dispositivi fisici differenti;
  - possibilità per un singolo dispositivo di ospitare più file system indipendenti.

Differenze tra file system

- Altre importanti differenze riguardano:
  - identificazione dei file (nomi, versioni, tipi);
  - layout fisico;
  - layout logico:
    - struttura e directory;
    - possibilità di un file di appartenere a più directory;
  - metodi di accesso;
  - attributi e protezioni.

Nomi dei file (1)

- Servono **all'utente** per identificare i file.
- Ogni file deve avere un **nome unico**, almeno nella propria directory.
- I file system si caratterizzano per:
  - insieme dei caratteri ammessi:
    - alfabetici, numerici, speciali ecc.;
  - case sensitivity;
  - lunghezza massima ammessa per i nomi;
  - formato dei nomi e vincoli:
    - suddivisione in parti, suffissi, estensioni.

Nomi dei file (2)

- Generalmente maiuscole, minuscole e caratteri numerici sono sempre ammessi.
- Altri caratteri come lo spazio, gli apici, “$”, “@”, “|” possono essere ammessi o meno.
  - Gli utenti evitano di usarli, perché sono scomodi da gestire dalle shell.
- Alcuni file system distinguono maiuscole e minuscole (es. Unix), altri no (es. Windows).
- Alcuni caratteri possono fungere da separatori di directory o estensioni.

Nomi dei file (3)

- Alcuni file system hanno un limite di lunghezza e vincoli di formato per i file.
  - Es.: max. 12 caratteri nelle prime versioni di Unix, 8 caratteri + 3 di estensione in MS-DOS.
- E' preferibile non imporre vincoli e lasciare all'utente la più ampia libertà, anche se questo complica leggermente l'implementazione.

Versioni dei file

- Alcuni file system possono conservare la versione precedente ogni volta che un file viene modificato.
  - Caratteristica utile per tener traccia della “storia” di un file.
  - Praticamente implicita nel dispositivo nel caso di WORM (Write Once Read Many, es.: CD scrivibili).

Tipi dei file (1)

- I primi file system distinguevano i file in base al contenuto (eseguibili, sorgenti, dati binari, testo) e utilizzavano una partizione in directory rigida, basata sul tipo dei file.
- In alcuni sistemi primitivi (es. MS-DOS, Windows 95) una parte del nome indica alla shell o alle applicazioni il tipo di file, ma il file system non opera distinzioni.

Tipi dei file (2)

- I file system moderni trattano come file anche alcuni file speciali, che devono essere distinti, perché necessitano di un trattamento differente:
  - directory,
  - device drivers: possono essere a caratteri (es.: terminali) o a blocchi (es.: dischi magnetici).
- I file non sono distinti in base al contenuto (neppure per gli eseguibili o i file di testo) e l'utente ha ampia libertà per i nomi.

Tipi dei file (3)

- Se il sistema definisce e riconosce alcuni tipi di file, si hanno parecchi svantaggi:
  - minore possibilità di espansione;
  - vincoli più ristretti;
  - difficoltà nella realizzazione di automatismi.
- In compenso alcune parti del file system sono più semplici.

Layout fisico (1)

- E’ la collocazione fisica dei dati.
- E’ in parte imposto dal dispositivo.
  - Generalmente i file sono suddivisi in blocchi di dimensione uniforme, che costituiscono la minima unità di accesso (settori, nel caso di dischi).
  - Solo alcuni dispositivi ad accesso sequenziale (nastri) permettono di avere unità di accesso di dimensione variabile, addirittura per uno stesso file.

Layout fisico (2)

- Il file system stabilisce:
  - dimensione dei blocchi logici, multipli di quelli fisici.
  - come far corrispondere layout logico e fisico;
  - come allocare i blocchi per i file;
  - come tener traccia delle allocazioni;
  - se permettere o meno la condivisione di un blocco tra file diversi.
    - Aumenta la complessità, si sfrutta meglio la memoria nel caso di file piccoli.

Dimensione dei blocchi (1)

- Se troppo piccola, servono più accessi per leggere un file e peggiorano le prestazioni.
  - Serve anche più spazio per le strutture dati che tengono traccia della posizione dei blocchi appartenenti a uno stesso file, ma non è uno spreco rilevante.
  - Nel caso di nastri, vi è una separazione fisica tra i blocchi, che spreca spazio.
- Se troppo grande, si spreca spazio, soprattutto per i file piccoli.
  - In media si spreca sempre mezzo blocco per ogni file.
  - Servono inoltre buffer più grandi in memoria.

Dimensione dei blocchi (2)

- Un buon compromesso, nel caso di dischi magnetici, si raggiunge con dimensioni da 512 byte a 4 K, ma si trovano anche dimensioni maggiori.
  - Negli anni, aumentando lo spazio disponibile, la dimensione dei blocchi logici tende ad aumentare.

Layout logico

- Un file può essere visto:
  - come una sequenza di byte (byte stream);
  - come una sequenza di record di lunghezza uniforme;
  - come una sequenza di record di lunghezza variabile.
- Oggi la tendenza è fornire il byte stream, implementando eventualmente altri layout con librerie, esterne al sistema.

File con indici

- Il file è suddiviso in record.
  - In ogni record uno o più campi, in posizione fissa, fungono da chiavi di ricerca.
- Il file system deve gestire un sistema di indici, per poter ritrovare velocemente un record, data una chiave.
  - Di solito utilizza file separati.
- Per ogni file esiste un indice primario, associato all'ordinamento del file ed eventualmente indici secondari.

Indici

- Possono essere densi o sparsi.
- Indici densi:
  - il file system memorizza **tutti gli indici**.
- Indici sparsi:
  - il file system memorizza solo **alcuni indici**.
  - possibile solo in presenza di un criterio di ordinamento dei record.
- Gli indici primari possono essere sparsi; quelli secondari devono essere densi.

Layout logico con byte stream

- I file di testo non sono formati da record, ma implementati usando il byte stream.
- Il file system non li distingue dai file binari.
  - Si utilizza un carattere o un insieme di caratteri per indicare convenzionalmente la fine di una linea.
  - Il compito di operare le necessarie distinzioni spetta alle librerie di accesso.
- **Non esiste** un carattere convenzionale per indicare la fine di un file; il file system memorizza la lunghezza di ogni file.

Informazioni di sistema

- Il file system deve conservare alcune informazioni per ogni file:
  - nome,
  - tipo,
  - posizione dei dati sul disco (allocazione fisica),
  - dimensione.

Attributi

- Il file system può conservare informazioni varie per ogni file :
  - data e ora di creazione,
  - data e ora dell'ultimo accesso,
  - utente proprietario,
  - utente che ha eseguito l'ultimo accesso.
- Tali attributi sono accessibili con apposite system call.

Strutture dati

- Il file system gestisce tutte le strutture dati necessarie:
  - per gestire l'allocazione dei file;
  - per fornire i vari metodi di accesso.
- Le strutture dipendono dai vari casi (alberi, tabelle, indici).
  - Comportano l'allocazione separata di blocchi o file interi.

Metodi di accesso (1)

- A un file si può accedere:
  - sequenzialmente (sempre);
  - con accesso diretto, specificando il numero del byte desiderato (byte stream);
  - con accesso diretto, specificando il numero del record desiderato (file a record);
  - con accesso diretto, specificando una porzione di contenuto di un record (chiave), da ricercare nel file (record con indici).
- L'accesso diretto, con numero di record o chiave, è indispensabile per molte applicazioni.

Metodi di accesso (2)

- Dipendono dal layout logico.
  - In un file con chiave l'accesso diretto è inevitabile.
- Possono dipendere dal layout fisico.
  - Su un nastro gli accessi non sequenziali sono impossibili o molto lenti.
- Nei file system più antichi i metodi d'accesso andavano indicati alla creazione del file.
  - Il file system utilizzava differenti layout fisici e differenti strutture per i diversi tipi di file, sullo stesso dispositivo fisico.

Metodi di accesso (3)

- Oggi la tendenza (es. Unix, Windows) è fornire l'accesso sequenziale e diretto al byte, implementando eventualmente altri metodi con librerie o processi server, esterni al sistema.
- I data base, per esempio, utilizzano i byte stream per costruire le loro strutture.

Struttura di un file system (1)

- I file sono raggruppati in directory.
- Ogni directory è di solito implementata con un file, gestito dal file system in modo differente rispetto agli altri file.
- Nei primi file system erano possibili solo uno o due livelli di directory.
  - Una directory per ogni tipo di file.
  - Vi sono problemi di omonimia.
  - E' impossibile gestire un elevato numero di file.

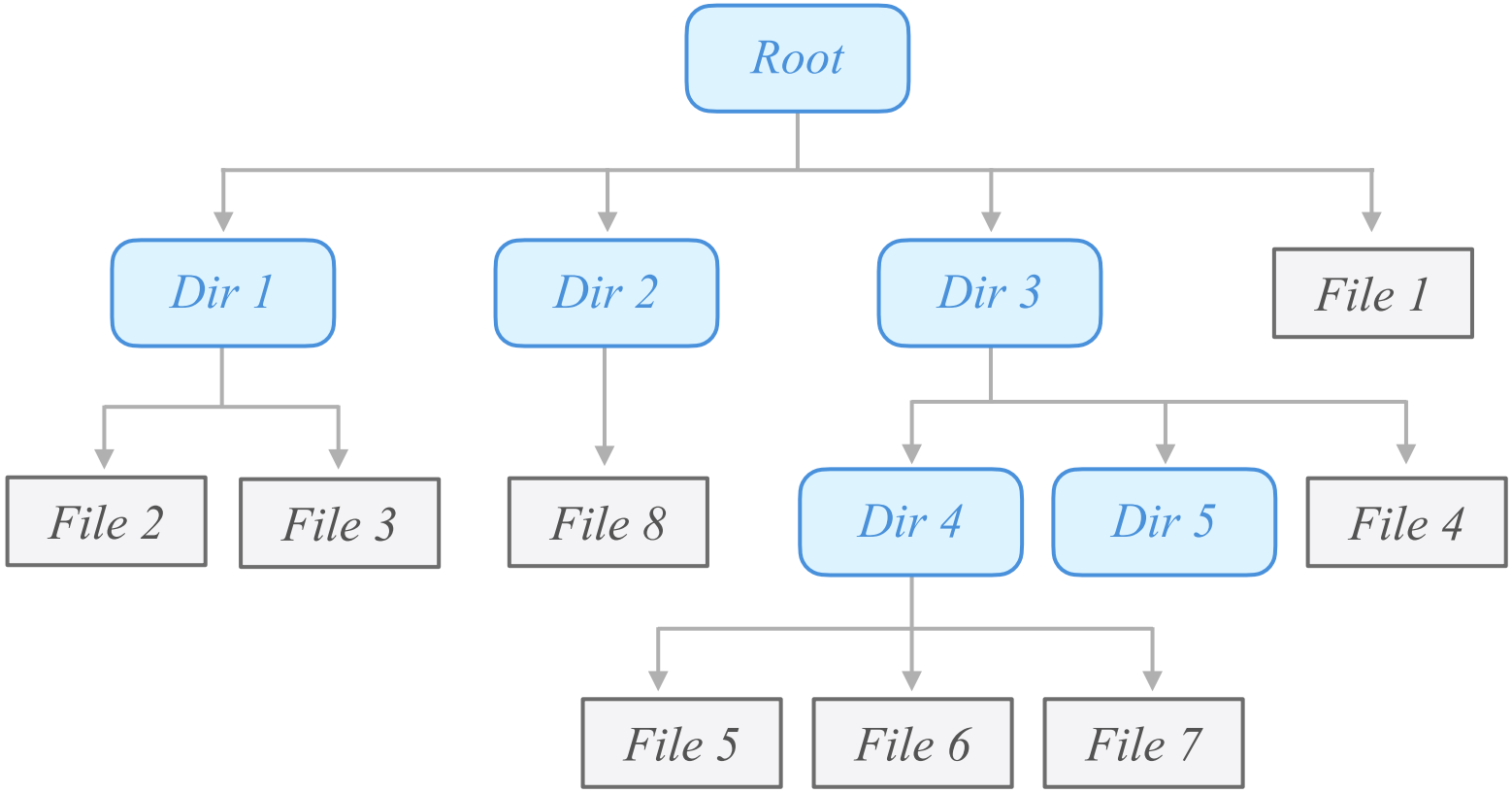
Struttura di un file system (2)

- Nei sistemi moderni ogni directory può contenerne altre, realizzando una struttura gerarchica ad albero, fino a un eventuale limite di profondità.
  - E' possibile gestire un numero virtualmente illimitato di file.
  - E' possibile tenere separati i file in base a vari criteri:
    - per utente;
    - per tipo di file;
    - per tipo di attività.

Directory

- Ogni directory contiene le informazioni relative a un gruppo di file:
  - nome, attributi, informazioni per accedere fisicamente al file;
  - oppure nome del file e indirizzo (o indice) di una struttura (i-node) contenente le informazioni.
    - Più flessibile, permette la condivisione di file tra directory differenti.
- Una directory iniziale (**root**) funge da punto di partenza delle ricerche.

Gerarchia di directory



Path name

- Si dice "**pathname**" il "percorso" di accesso a un file, a partire:
  - dalla root, se inizia con un carattere convenzionale (**pathname assoluto**);
  - da una directory considerata "corrente", altrimenti (**pathname relativo**);
- E' formato concatenando i nomi di tutte le directory attraversate e il nome del file.
  - Un separatore convenzionale serve a distinguere le varie componenti.

Localizzazione di dati di sistema e attributi

- Possono essere scritti:
  - nella directory (es.: MS-DOS);
  - in testa al file, lasciando nome e posizione nelle directory;
  - in strutture dati esterne a file e directory (es.: Unix).

Protezioni

- Un file system deve permettere un certo livello di protezione contro accessi indesiderati.
- Possono variare:
  - la granularità delle protezioni (singolo file, directory intere);
  - i diritti di accesso presi in considerazione (lettura, scrittura, esecuzione);
  - l'identificazione degli aventi diritto (un singolo utente, un gruppo di utenti).

Operazioni sui file

- Un file system deve supportare almeno le seguenti operazioni sui file:
  - creazione di un file vuoto,
  - cancellazione di un file,
  - cambio di nome,
  - lettura attributi,
  - modifica attributi.
- Sono possibili altre operazioni:
  - troncamento a una lunghezza fissata.

Operazioni di I/O sui file

- Un file system deve supportare almeno le seguenti operazioni di I/O:
  - apertura di un file,
  - chiusura di un file,
  - lettura di dati,
  - riscrittura di dati,
  - aggiunta di dati in coda,
  - posizionamento per la successiva operazione.

Apertura di un file

- Quando un file viene aperto, il file system:
  - scandisce il path name, leggendo le directory attraversate e cercando il nome di ogni componente;
  - verifica i diritti di accesso;
  - carica in memoria le informazioni (attributi e indirizzi) per l'accesso.

Chiusura di un file

- Quando un file viene chiuso, il file system:
  - aggiorna le informazioni nella directory;
  - scarica i buffer ancora in memoria;
    - può limitarsi ad accodare i buffer in una coda di buffer da scrivere appena possibile;
  - dealloca buffer e altre strutture interne.

Operazioni sulle directory

- Un file system deve supportare almeno le seguenti operazioni sulle directory, viste come file:
  - creazione di una directory vuota,
  - cancellazione di una directory vuota,
  - cambio di nome,
  - lettura attributi,
  - modifica attributi.

Operazioni di I/O sulle directory

- Un file system deve supportare almeno le seguenti operazioni sulle directory, dal punto di vista del contenuto:
  - apertura,
  - chiusura,
  - lettura,
  - aggiunta di file,
  - rimozione di file.
- La scrittura diretta deve essere **impedita** ai programmi utente, per garantire correttezza e coerenza del file system.

File memory mapped

- Introdotti da MULTICS.
- I file vengono visti come vettori in memoria.
- Si aggiungono due primitive:
  - `map`, che crea il mapping tra memoria e file;
  - `unmap` che lo annulla, dealloca le pagine e scrive le pagine modificate, ma non ancora scaricate.
- Il sistema alloca pagine di memoria virtuale, per le quali usa il file (o una sua copia) come area di swap.

Accesso ai file memory mapped

- Il programmatore riceve il puntatore dalla `map` e accede al file come a un vettore.
- I normali meccanismi di paginazione mantengono aggiornato il file quando vi si scrive.
  - Il programmatore ignora quale parte del file sia effettivamente caricata in ogni istante.
- Se un file è aperto contemporaneamente da più processi, per assicurarne la coerenza bisogna far condividere un segmento.

Vantaggi dei file memory mapped

- Gestione dell' I/O semplificata per il programmatore.
- Realizza una bufferizzazione molto efficiente a livello utente.
  - Il sistema interviene solo raramente.
- Particolarmente adatto ai sistemi segmentati.
  - Si utilizza un segmento per ogni file.

Svantaggi dei file memory mapped

- Il sistema non può determinare quale sia l'ultimo byte scritto.
  - La lunghezza del file deve essere arrotondata in eccesso a un multiplo della lunghezza delle pagine.
  - Le applicazioni devono gestire la lunghezza o riconoscere un terminatore convenzionale.
- Le dimensioni dei file possono superare lo spazio di indirizzamento.
  - Bisogna fare il mapping di una parte per volta.

Clustering (1)

- Se possibile, blocchi che subiscono accessi consecutivi vanno tenuti vicini, per ridurre i tempi di attesa.
- Una possibile soluzione è il **clustering**, consistente nell'allocare non singoli settori, ma gruppi di settori consecutivi (di solito da 2 a 16).

Clustering (2)

- La soluzione è adattabile a qualsiasi tipo di file system.
- Il cluster diventa la minima unità di allocazione.
- Con cluster grandi si spreca molto spazio disco.
  - Un settore danneggiato fa scartare l'intero cluster cui appartiene.
  - Un file di 1 byte occupa un intero cluster.
  - Poco rilevante con dischi di grande capacità.

Esempi di file system su disco

- Allocazione contigua.
- File system a liste.
- File system con FAT.
  - Clustering.
  - MS-DOS.
- File system ad albero.
  - Unix.

Allocazione contigua

- Ogni file occupa settori consecutivi.
- Minimo overhead.
  - Nelle directory bisogna solo memorizzare settore di partenza e dimensioni.
- Ottime prestazioni.
  - Gli accessi a settori consecutivi sono molto veloci.
- Utilizzato solo nei sistemi più primitivi.
- Utilizzato di fatto in molti dispositivi a sola lettura (CD-ROM).

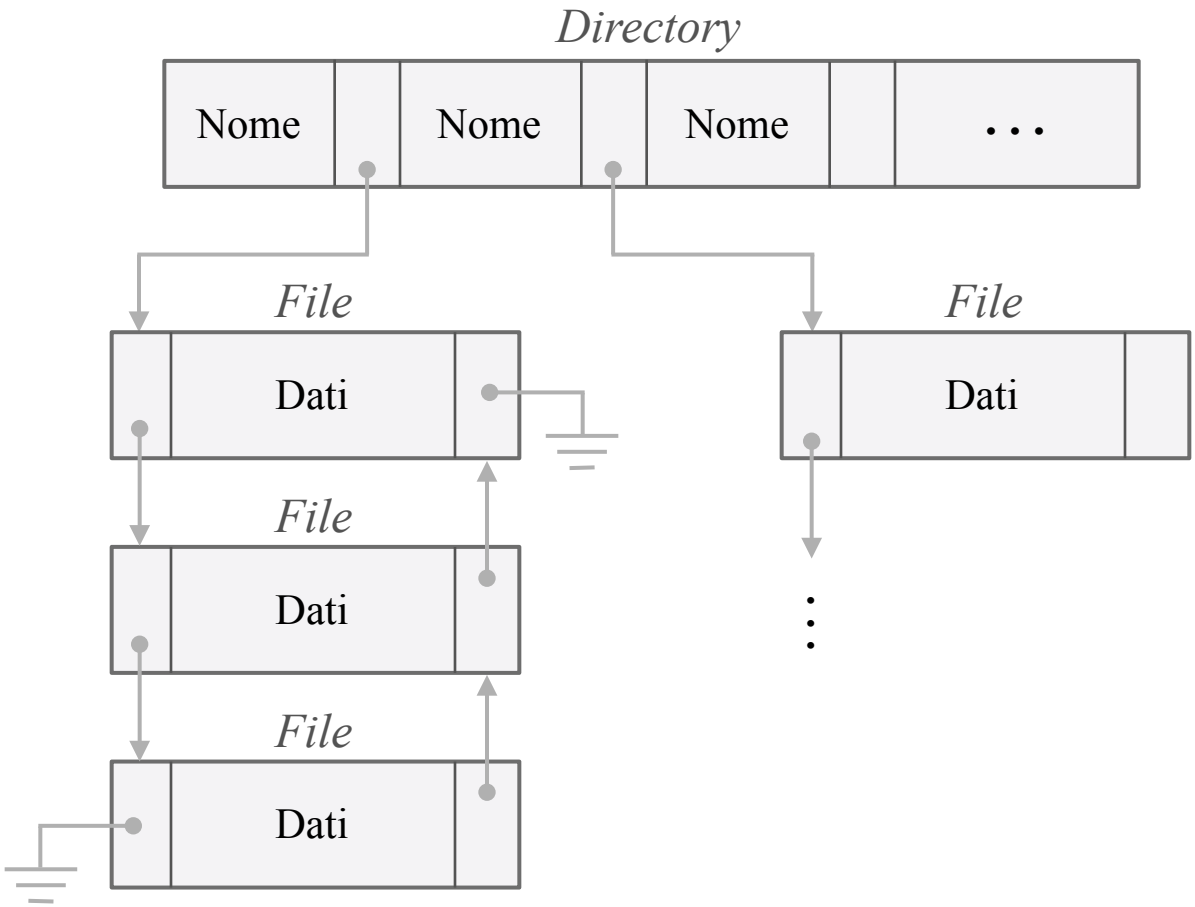
Svantaggi dell'allocazione contigua

- Poco flessibile.
  - In fase di creazione va specificata la dimensione massima dei file.
  - I file non possono crescere oltre la dimensione massima.
- Tende a produrre frammentazione.
  - Programmi di compattamento possono ridurre la frammentazione, girando di notte o a bassa priorità.

File system a liste (1)

- Ogni settore contiene, oltre ai dati, l'indirizzo dei settori successivo e precedente appartenenti allo stesso file.
  - Il settore diventa la minima unità di allocazione.
- Una lista raggruppa i settori liberi.
  - Il suo indirizzo è contenuto della directory principale o in un settore in posizione fissa.
- La directory principale inizia in un settore in posizione fissa.

File system a liste (2)



Vantaggi dei file system a liste

- Molto semplice.
- Overhead di disco trascurabile.
- Nessun problema di allocazione.
- Soluzione adeguata per file system molto piccoli (meno di un Mbyte).

Svantaggi dei file system a liste

- Gli accessi non sequenziali implicano numerosi accessi fisici al disco.
- E' molto vulnerabile a piccoli danni:
  - danneggiando un settore, si può perdere un file;
  - danneggiando una directory, si possono perdere tutti i file contenuti;
  - alterando un puntatore e continuando a operare si possono perdere tutti i file.
- L'informazione sull'allocazione del file è distribuita tra directory e dati.

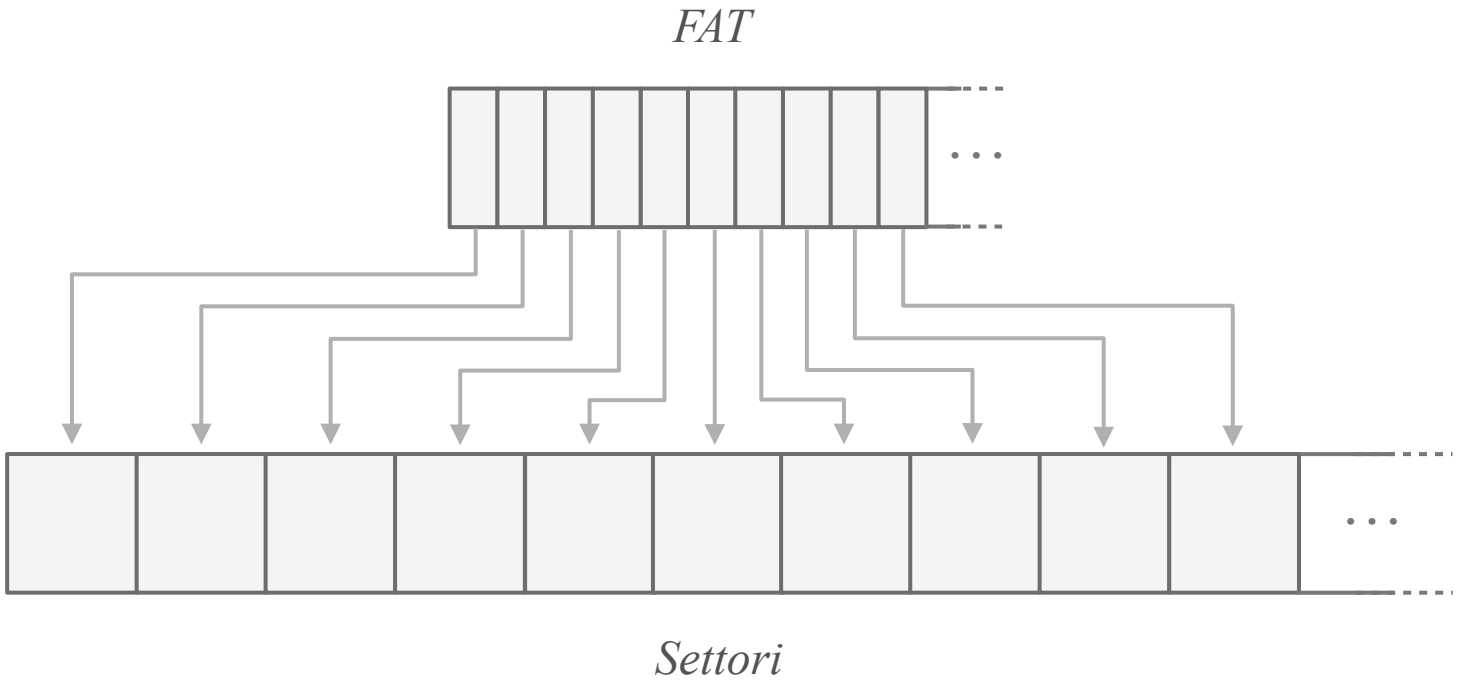
File system con FAT (1)

- Una **FAT** (File Allocation Table) tiene traccia dell'allocazione dei settori.
  - Si usa il numero di settore come indice per accedere agli elementi.
- I dati sono contenuti in una catena di settori.
- Per ogni settore, nella FAT vi è un numero che indica il successivo settore appartenente allo stesso file.
  - Valori convenzionali indicano l'ultimo settore e i settori danneggiati.

File system con FAT (2)

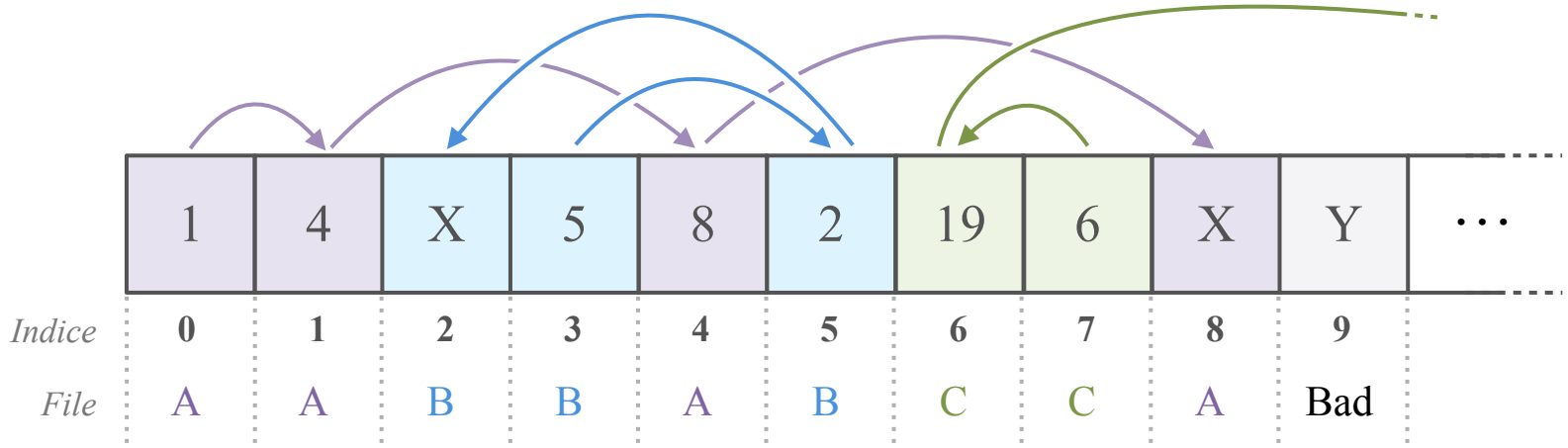
- Le directory contengono il numero del primo settore.
- La FAT ha posizione e dimensione fissa e si trova nei primi o ultimi settori del disco.
- I settori liberi sono riuniti in una catena oppure marcati con un valore convenzionale nella FAT.

FAT e settori



La corrispondenza è basata sull'indice: l'elemento *n* della FAT corrisponde al settore *n*.

FAT e file



Il numero del primo cluster di ogni file è nella directory:

- file A nei settori 0, 1, 4, 8;
- file B nei settori 3, 5, 2;
- file C nei settori 7, 6, 19, … .

Un valore convenzionale (X) indica l'ultimo cluster.

Un altro (Y) indica un cluster danneggiato.

Vantaggi della FAT

- Implementazione semplice.
- Minimo overhead (un indice per settore).
- L'accesso diretto è molto veloce, perché implica solo una scansione di liste in memoria.

Svantaggi della FAT

- La tabella dev'essere caricata in memoria per rendere efficiente l'implementazione.
  - Altrimenti gli accessi casuali possono implicare numerosi accessi fisici al disco.
- In caso di danni a un settore contenente la FAT, si possono perdere parecchi file.
- L'informazione sull'allocazione del file è distribuita tra directory (numero del primo cluster) e FAT.
- Metodo adatto a file system medio-piccoli.

FAT e clustering

- Per dischi grandi, la FAT può diventare enorme, rendendo impossibile caricarla in memoria.
- Per ridurre le dimensioni della FAT, si raggruppano settori consecutivi in “cluster”, di dimensione fissa.
  - Il cluster diventa la minima unità di allocazione.

Esempi di file system a cluster

- DOS 1.0, con FAT a 12 bit.
  - Solo floppy.
- DOS 2.0, con FAT a 16 bit.
  - Primi dischi, piccoli.
- DOS 4.0, con FAT a 16 bit e cluster variabili.
  - Dischi di maggior capacità.
- DOS 6.0, con FAT a 32 bit.

FAT a 12 bit

- Solo $2^{12} = 4096$ cluster.
  - Con cluster di 512 byte, 2 Mbyte per file system.
- Adatto ai floppy disk.
- 1.5 byte per elemento.
  - 6 Kbyte di occupazione della FAT.

FAT a 16 bit

- Solo $2^{16} = 65536$ cluster.
  - Con cluster di 512 byte, 30 Mbyte per file system.
- Adatto ai primi dischi rigidi.
- 2 byte per elemento.
  - Fino a 128 Kbyte di occupazione della FAT.

Fat a 16 bit con cluster variabili

- Cluster di dimensione variabile, in funzione della dimensione del disco.

Dimensione del disco	Settori per cluster	Cluster size
Meno di 128 M	4	2 K
Meno di 256 M	8	4 K
Meno di 512 M	16	8 K
Meno di 1 G	32	16 K
Meno di 2 G	64	32 K
Meno di 4 G	128	64 K

Problemi della FAT a 16 bit

- Molti vecchi programmi suppongono che la dimensione di un cluster (in byte) sia indicata da un intero di meno di 16 bit.
  - Limite a 2 Gbyte per disco.
- Il numero di settori per cluster è contenuto in un solo byte ed è una potenza di 2, quindi può essere al massimo 128.
  - Limite a 4 Gbyte per disco.

Fat a 32 bit con cluster variabili

- Cluster di dimensione variabile, in funzione della dimensione del disco.

Dimensione del disco	Settori per cluster	Cluster size
Meno di 8 G	8	4 K
Meno di 16 G	16	8 K
Meno di 32 G	32	16 K
32 G e oltre	64	32 K

FAT a 32 bit

- $2^{32} = 4294967296$ cluster.
  - 32 bit per il numero di settore.
  - Con settori di 512 byte, 2 Tbyte per file system.
- 4 byte per elemento.
  - Fino a 16 Gbyte di occupazione della FAT.
- Adatto agli attuali dischi rigidi.
- Se la velocità di crescita dei dischi rimarrà inalterata, diventerà obsoleto in pochi anni.

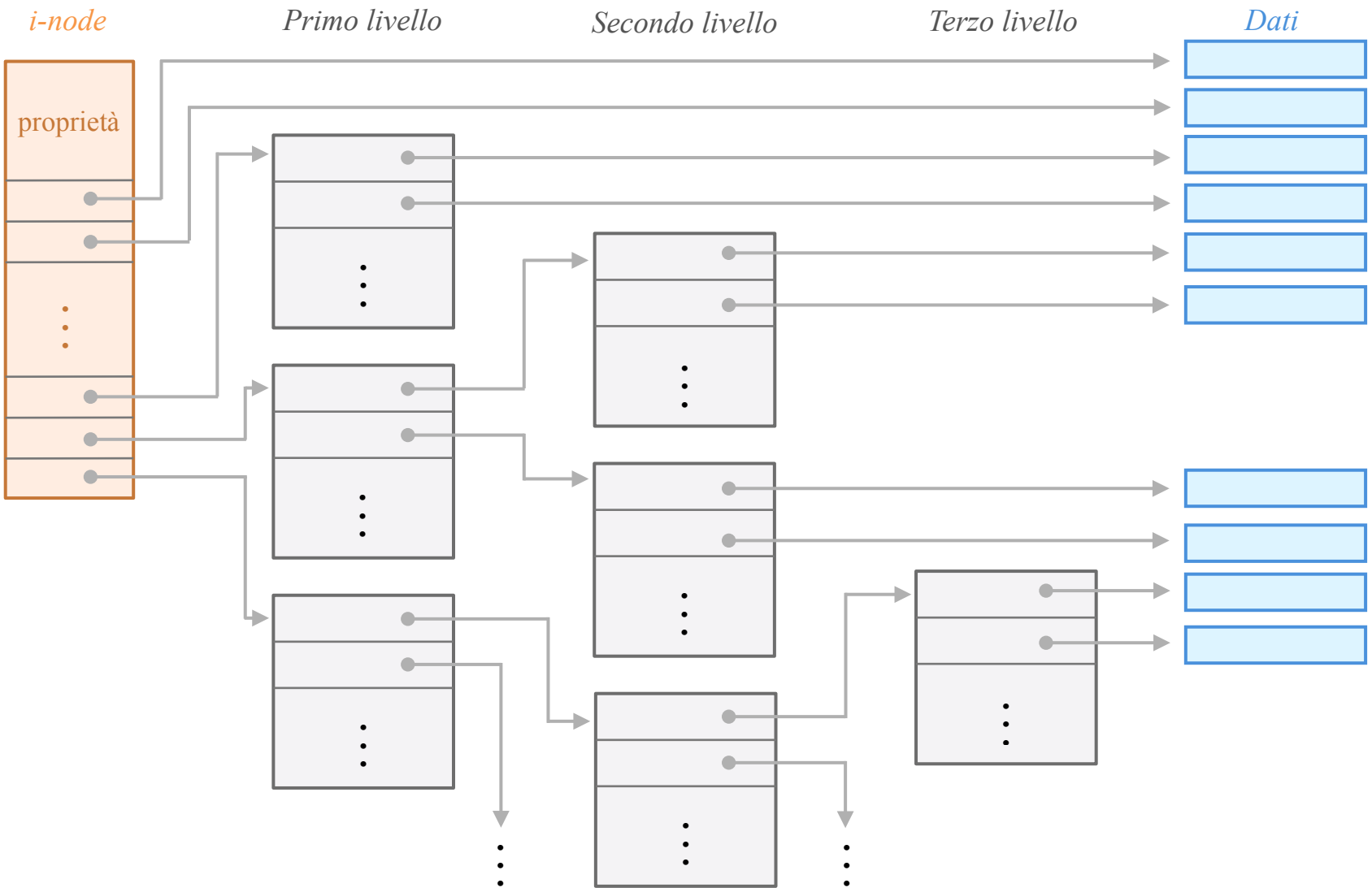
File system ad albero (1)

- Per ogni file si costruiscono uno o più alberi:
  - le foglie sono i settori contenenti i dati;
  - i nodi intermedi sono settori che contengono solo puntatori a un numero fisso di settori;
  - gli alberi sono di ordine elevato (128 o più);
  - il numero di livelli è molto limitato (generalmente non più di 3);
  - il numero di alberi per file è ridotto (di solito da uno a una dozzina).

File system ad albero (2)

- Per ogni file esiste una struttura (**i-node**, per "information-node") contenente le informazioni generali e i puntatori agli alberi relativi al file.
  - Gli i-node sono piccoli, di dimensione fissata, memorizzati in posizione fissa sul disco.
- Le directory contengono i nomi dei file e i numeri di i-node (**i-number**) associati.
- Una lista riunisce i settori liberi.

Esempio di file system ad albero



Posizione degli i-node

- Nelle prime versioni di Unix gli i-node erano in una serie dei settori consecutivi all'inizio del disco.
- In seguito tali settori sono stati talvolta spostati a metà del disco o addirittura sparpagliati in più aree, su cilindri differenti, per ridurre gli spostamenti della testina.

Vantaggi dei file system ad albero

- Ridotto overhead su disco.
  - Un puntatore per settore.
  - Qualche settore di puntatori non completamente utilizzato.
- Minima occupazione di memoria.
  - E' necessario caricare in memoria solo gli i-node dei file aperti.
- Molto flessibili.
  - Permettono file condivisi, directory e device driver visti come file.

Svantaggi dei file system ad albero (1)

- Implementazione leggermente complessa.
- In caso di danni a un settore contenente i-node, si possono perdere i dati di più file.
- L'accesso diretto comporta pochi accessi a disco (di solito 4 al massimo), ma più di quanti necessari (in media) con altre strutture, nel caso di file molto grandi.
- Nella forma base, metodo adatto solo per implementare byte-stream.

Svantaggi dei file system ad albero (2)

- Come nei file system a liste, se non si riorganizza periodicamente il file system, le prestazioni peggiorano progressivamente.
  - Allocazioni e deallocazioni finiscono con lo sparpagliare i blocchi liberi, e di conseguenza i file, sull'intero disco.

Esempio di file system ad albero: UNIX

- Ogni i-node contiene:
  - 10 puntatori a blocchi dati;
  - 1 puntatore a un albero con 1 livello intermedio;
  - 1 puntatore a un albero con 2 livello intermedi;
  - 1 puntatore a un albero con 3 livello intermedi.

File system UNIX

- Totale blocchi (da 512 byte) accessibili:
  - 10 direttamente (5 Kbyte);
  - 128 con un accesso supplementare (64 Kbyte);
  - 16384 con due accessi supplementari (8 Mbyte);
  - 2097152 con tre accessi supplementari (1 Gbyte).

Vantaggi del file system UNIX

- Accesso veloce ai file piccoli.
- Implementazione relativamente semplice.

Implementazione di directory (1)

- Comunemente mediante file.
- Se non esiste una struttura dati (i-node) separata, una directory deve contenere, per ogni file:
  - nome,
  - attributi e protezioni,
  - dimensione,
  - informazioni per l'accesso.

Implementazione di directory (2)

- Di solito si utilizzano elementi di lunghezza fissa.
  - Solo nome e informazioni d'accesso possono avere lunghezza variabile.
  - Se necessario si utilizzano più elementi, opportunamente marcati, per uno stesso file.
- Nei file system con allocazione contigua, a lista o a FAT le informazioni d'accesso si riducono all'indirizzo del primo blocco e alla lunghezza.

Esempi di directory entry

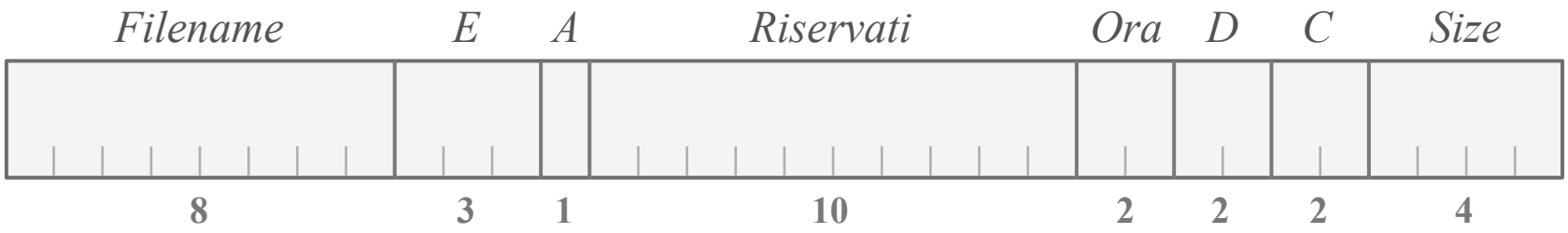
- CP/M.
- MS-DOS.
- Unix.

Directory entry CP/M

- In CP/M ogni entry di directory contiene fino a 16 indirizzi di blocchi.
  - I nomi dei file hanno lunghezza fissa.
  - Se il file occupa più di 16 blocchi si utilizzano più elementi consecutivi nella directory.
  - Adatto solo per file molto piccoli.
  - La directory non contiene la lunghezza, quindi sono sempre allocati interi settori.
    - Serve un terminatore convenzionale nel file.

Directory entry MS-DOS

Ogni entry occupa 32 byte.

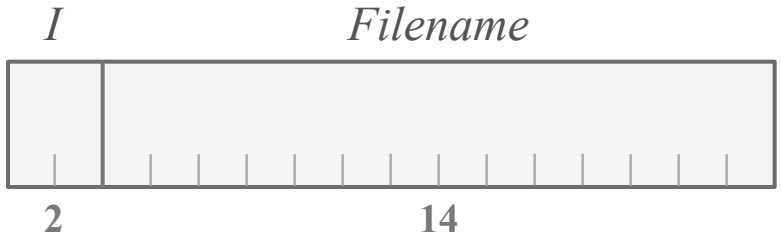


E estensione
A attributi
D data
C indice del primo cluster

Limiti imposti dalla struttura della directory entry DOS

- Nome di file di 8 + 3 caratteri.
- Tempo di scrittura in (secondi / 2) da mezzanotte.
- Dimensione massima 2^{32} = 4 Gbyte.
- Scarse possibilità di estensione.
- 2 dei 10 byte riservati sono usati per il numero del primo cluster con la FAT a 32 bit.

Directory entry Unix



I i-number

Nelle versioni recenti, la lunghezza del nome è stata portata da 14 a 60 caratteri, con la possibilità di concatenare più entry (con i-number convenzionale), per gestire nomi di lunghezza illimitata.

Ricerca di un file in Unix

- Il file system inizia la ricerca:
  - dalla root, se il nome inizia con “/”;
  - dalla directory corrente, altrimenti.
- Ogni componente del path viene ricercato nella directory attuale; se è a sua volta una directory (termina con “/”), si aggiorna la directory nella quale cercare e si prosegue.
- L’ultimo componente del path è il nome del file.

Directory in Unix

- Ogni directory contiene sempre due entry:
  - una con nome “.” e l’ i-number della directory stessa;
  - una con nome “..” e l’i-number della directory superiore.
- Solo la root non ha l’entry corrispondente alla directory superiore.
- In questo modo le ricerche che comportano risalite lungo l’albero sono gestite esattamente come le altre.

File speciali

- In molti sistemi operativi esistono file speciali, corrispondenti alle periferiche fisiche.
  - Contengono il codice delle funzioni di basso livello che controllano le periferiche.
  - Leggere e scrivere su tali file significa leggere e scrivere direttamente sulle periferiche.
    - Operazione normalmente vietata agli utenti.

File speciali in Unix

- In Unix esistono due tipi di file speciali (normalmente situati nella directory “/dev”):
  - dispositivi a carattere, corrispondenti alle periferiche che gestiscono i dati un byte alla volta;
  - dispositivi a blocchi, corrispondenti alle periferiche che gestiscono i dati a blocchi.
- Un caso particolare è un file, di solito chiamato “/dev/mem”, corrispondente alla memoria fisica della macchina, vista come un file.

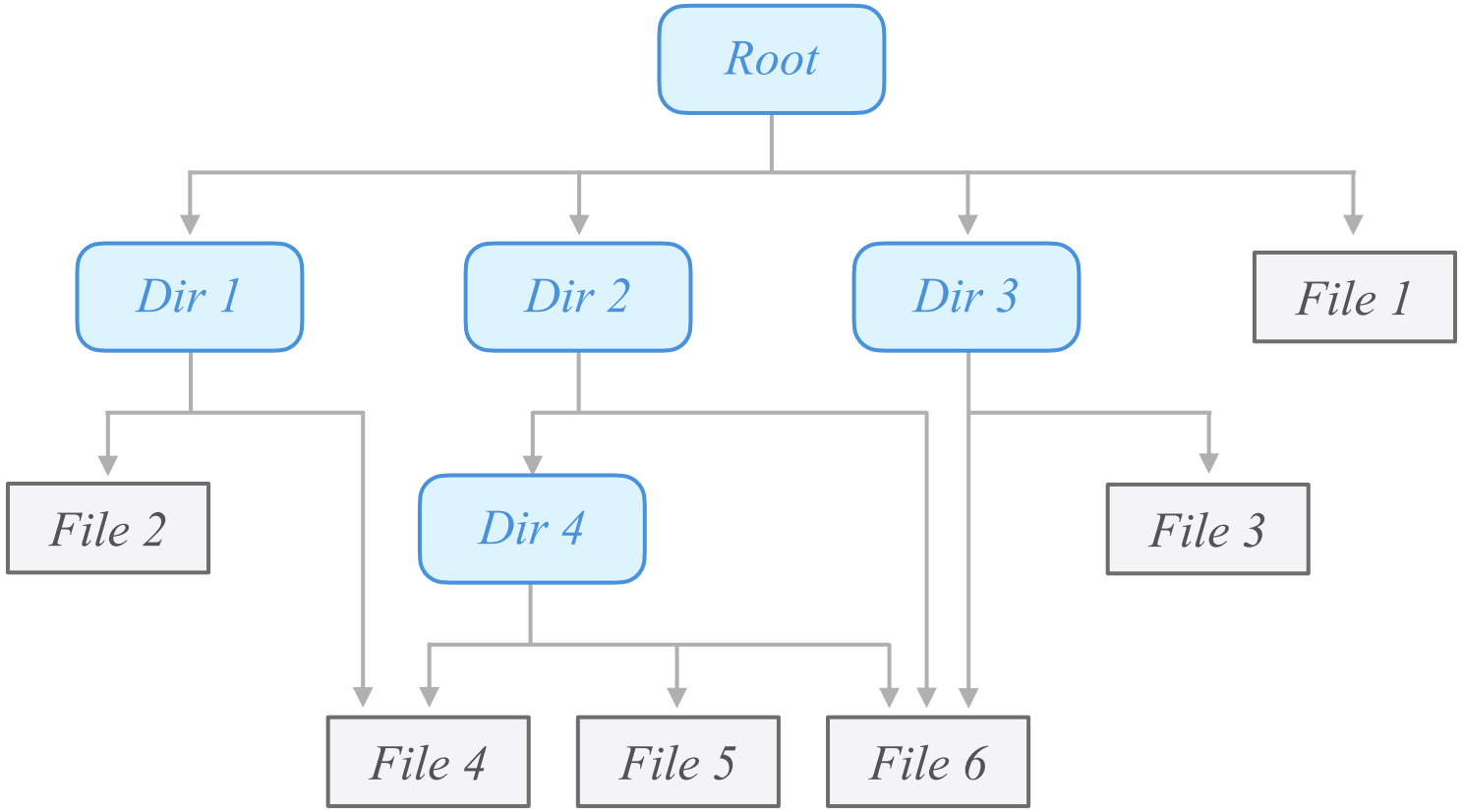
Link tra file

- In molti sistemi è possibile condividere lo stesso file in directory differenti.
  - Se le directory sono gestite come file, possono essere a loro volta condivise.
- Il file condiviso ha più nomi ed è riferibile con più path, ma mantiene uniche le altre proprietà: proprietario, permessi di accesso, lunghezza, date di accesso e dati contenuti.
- Il file system non costituisce più un albero, ma diventa un **grafo orientato aciclico**.

Link fisici

- I link fisici consistono semplicemente nel condividere lo stesso i-number in più punti del file system.
  - Il file resta unico, ma è riferibile con più nomi.

Link fisici in Unix



Implementazione dei link fisici in Unix (1)

- Ogni file ha un **unico i-node**, che lo identifica e conserva tutte le proprietà.
- Lo stesso i-number può comparire in più directory diverse, associato a nomi differenti.

Implementazione dei link fisici in Unix (2)

- Nell'i-node viene mantenuto il numero di link (numero di occorrenze dell'i-number nelle directory).
  - Quando si cancella un file, questo contatore viene semplicemente decrementato.
  - Se diventa zero, l'i-node e tutti i dati possono essere rimossi.

Link simbolici (1)

- I link fisici richiedono la presenza di una struttura dati (i-node) che descrive il file.
  - Le informazioni fondamentali, come indirizzi di disco e lunghezza non possono essere scritte nelle directory.
- Un metodo alternativo per realizzare i link è costituito dai **link simbolici**: file speciali, che contengono semplicemente il path name del file reale.

Implementazione dei link simbolici

- Inevitabili nei sistemi privi di i-node.
- Comunque possibili in tutti i sistemi.
- Il file system deve riconoscerli e reindirizzare ogni operazione (apertura, interrogazione sulle proprietà ecc.) sul file reale.
- Un link simbolico è una sorta di “puntatore”, che punta a un file o a un altro link.

Differenze tra link fisici e link simbolici (1)

- In presenza di link fisici, tutti i link al file sono simmetrici: non esiste una versione privilegiata.
- I link simbolici costituiscono un meccanismo asimmetrico: esiste una versione "base" del file.
  - Se l'originale viene rimosso o spostato, i link simbolici riferiscono un file inesistente e non sono più validi.

Differenze tra link fisici e link simbolici (2)

- I link fisici non occupano spazio aggiuntivo, a parte il loro entry nelle directory.
- I link simbolici occupano più spazio, sia per il file costituente il link (di solito un settore di disco), sia per il relativo i-node, se esiste.

Differenze tra link fisici e link simbolici (3)

- L'utilizzo di link fisici non richiede accessi aggiuntivi.
- Per agire su di un file tramite link simbolico sono necessari più accessi al disco, per leggere l'i-node del link e il link stesso.
  - Si può avere una lunga catena di link simbolici.

Differenze tra link fisici e link simbolici (4)

- I link fisici possono essere utilizzati solo all'interno dello stesso file system.
  - Tutti i link devono far parte dello stesso file system, quindi generalmente sullo stesso disco.
- I link simbolici possono riferire anche file su una macchina diversa e lontana.

Problemi creati dai link

- I programmi che utilizzano il path name per localizzare i file (per esempio, alcuni programmi per produrre backup o copiare un disco su nastro) tendono a “vedere” e trattare più copie distinte dello stesso file.
  - Ricaricando un backup, i link non sarebbero ricostruiti e si creerebbero più copie del file.

File system montati (1)

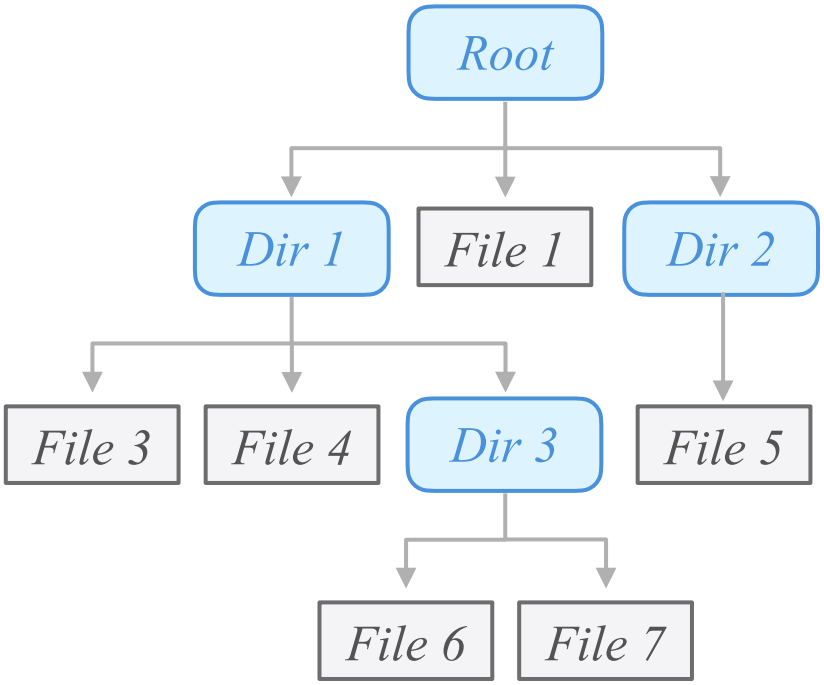
- In Unix è possibile “montare” un file system entro quello principale.
  - Il file system montato è identificato attraverso un file speciale, corrispondente a un dispositivo a blocchi.
  - La radice del file system montato rimpiazza una directory nel file system principale.
  - La directory rimpiazzata non viene cancellata, ma resta inaccessibile finché il file system è montato.
  - Tutti gli accessi che passano attraverso tale directory vengono dirottati sul file system montato, tramite il file speciale collegato.

File system montati (2)

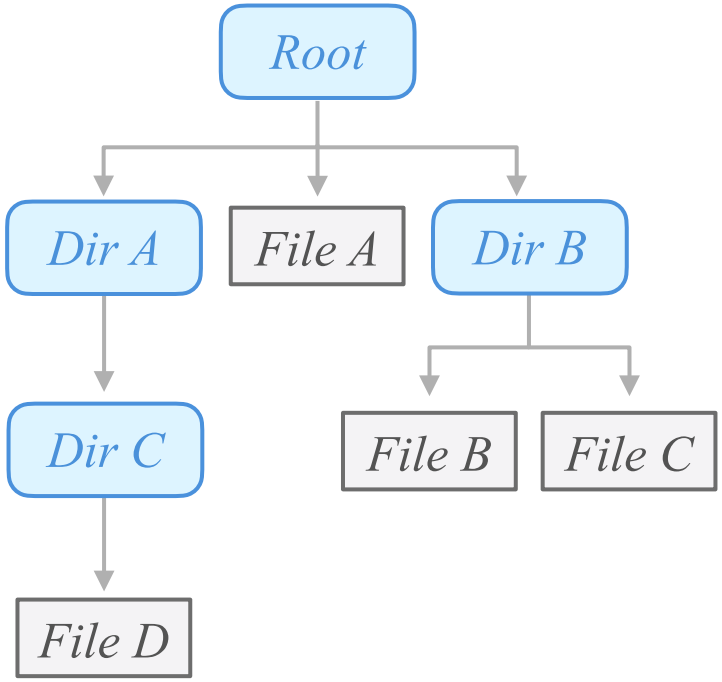
- In questo modo l'intero file system resta un unico albero, anche se comprende più dispositivi fisici, anche di tipo diverso.
  - Si possono montare dischi magnetici, CD, DVD, dischi esterni, chiavette di memoria.
  - Alcuni automatismi rendono le operazioni di montaggio e smontaggio trasparenti all'utente: inserendo un DVD nel lettore o collegando un disco esterno, questo viene montato in posizioni predefinite.

File system montati (3)

File system principale

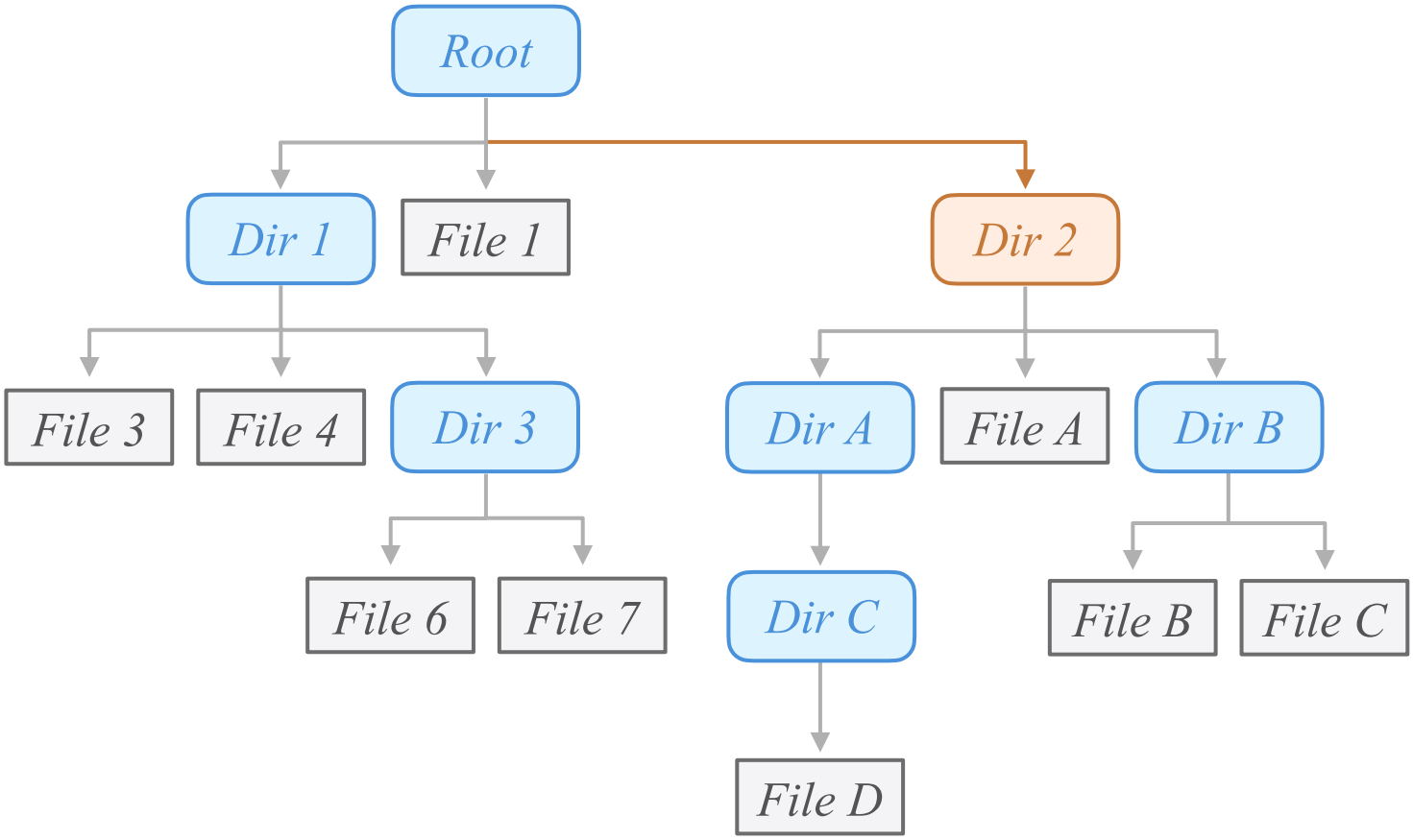


File system da montare



File system montati (4)

File system montato



Gestione dei settori liberi

- Il file system deve sapere quanti e quali settori sono liberi.
- Sono possibili vari metodi:
  - liste,
  - liste di alberi,
  - bitmap.

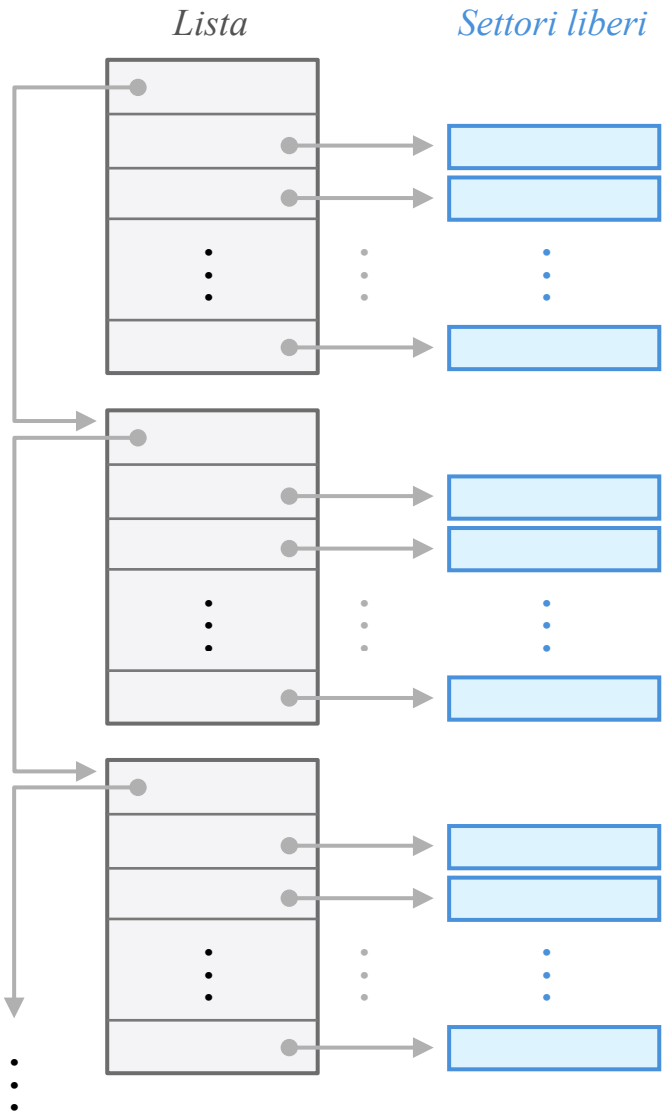
Settori liberi con liste

- I settori liberi sono concatenati in una lista: ogni settore contiene l'indirizzo del successivo.
  - Non si occupa spazio aggiuntivo.
  - Molto semplice.
  - Poco efficiente: serve un accesso al disco ogni volta che si libera o occupa un settore.
  - Poco affidabile: in caso di danni a un settore, si perdono tutti i successivi.

Settori liberi con liste di alberi (1)

- I settori liberi formano una lista di alberi, a un solo livello: ciascun settore del livello superiore contiene l'indirizzo del successivo e di tanti settori liberi quanti ve ne stanno.
  - Non si occupa spazio aggiuntivo.
  - Abbastanza semplice.
  - Con un accesso si possono ottenere anche centinaia di indirizzi di settori liberi.
  - Poco affidabile: in caso di danni a un settore, si perdono tutti i successivi.
- Utilizzati in Unix.

Settori liberi con liste di alberi (2)



Settori liberi con bitmap

- Si usa una bitmap, con un bit per settore.
  - Occupa poco spazio aggiuntivo.
    - Però lo spazio occupato è costante, anche quando il disco è quasi pieno.
  - Molto semplice.
  - Molto efficiente.
    - Se la bitmap può essere caricata interamente in memoria, non serve accedere al disco per allocare o liberare settori.
  - E' facile ottimizzare le allocazioni, allocando blocchi vicini per lo stesso file.
- Utilizzata nel "fast file system" per Unix.

Settori difettosi

- Capita frequentemente che alcuni settori del disco siano difettosi (bad block).
  - In un settore difettoso non sempre si riesce a scrivere e rileggere i dati.
  - Esistono programmi che identificano i settori difettosi, eseguendo ripetutamente operazioni di scrittura e lettura.
  - Il difetto può essere già all'origine o essere stato prodotto con l'utilizzo.
    - I dischi nuovi possono essere venduti già con alcuni settori difettosi.

Gestione hardware dei settori difettosi (1)

- Effettuata dal controller, all'insaputa del file system.
  - Per ogni settore difettoso il controller seleziona un sostituto in un'area di riserva (uno o più cilindri appositi).
  - Si può gestire un numero limitato di settori difettosi.

Gestione hardware dei settori difettosi (2)

- La lista delle sostituzioni viene scritta sul disco, in un'area riservata, e letta all'accensione.
  - Il controller aggiorna la lista di settori difettosi quando ne identifica uno.
- Ogni operazione richiesta su un settore difettoso viene dirottata sul sostituto.

Gestione software dei settori difettosi

- Effettuata dal file system, indipendentemente dal controller.
  - I settori difettosi vengono riuniti in un file, che non viene mai cancellato.
    - Molti sistemi marcano questo file, in modo da riconoscerlo e vietare qualsiasi operazione su di esso.
  - Essendo perennemente occupati, i settori non verranno mai utilizzati per i dati.
  - Non vi è limite al numero di settori difettosi gestibili in questo modo.

Quote di disco (1)

- Nei sistemi che servono molti utenti, bisogna limitare lo spazio su disco che ogni utente può occupare.
- In alcuni casi si fa pagare all'utente un costo, che dipende dal numero di settori e dalla durata dell'occupazione.

Quote di disco (2)

- In altri casi si pone un limite (hard limit) per ogni utente al numero di settori che possono essere allocati e al numero di file che può possedere.
  - Se uno dei limiti viene superato, l'operazione fallisce, anche se sul disco rimane spazio disponibile.
  - Spesso esiste un secondo limite (soft limit), inferiore.
    - All'utente viene notificato un avviso quando lo supera.

Backup

- Normalmente i dati sono **molto più preziosi** dei dischi che li contengono.
  - Bisogna assicurarne la sopravvivenza in caso di danni, anche gravi, ai dischi.
- La strategia più semplice è effettuare copie integrali (backup) periodicamente.
  - Con grandi archivi si impiega molto tempo.

Backup a due dischi

- Si utilizzando due file system e due dischi, ciascuno contenente un file system e copia dell'altro.
  - La copia viene aggiornata automaticamente ogni notte.
  - Si spreca la metà dello spazio disco disponibile.
  - Il ripristino dei file perduti è velocissimo.

Backup incrementale

- Consiste nel salvare periodicamente (p. es. ogni giorno) tutti e soli i file modificati a partire dall'ultimo backup.
  - Serve comunque un backup integrale periodico (p. es. ogni mese).
  - Ogni backup incrementale va effettuato **senza sovrascrivere** i precedenti.
  - Per ripristinare un file bisogna cercarlo all'indietro, fino a trovarne l'ultima modifica.
  - Esistono programmi che rendono automatica la maggior parte delle operazioni.

Consistenza

- Un file system è una struttura dati complessa.
- Se si verifica un guasto (caduta di tensione, malfunzionamento) può trovarsi in uno stato inconsistente.
- Per questo esistono programmi, lanciati periodicamente o a ogni accensione, che verificano la consistenza del file system ed eventualmente lo "riparano".

Verifica di consistenza in Unix

- Il programma **fsck** considera, uno alla volta, tutti i file system disponibili ed esegue alcune verifiche su ognuno:
  - verifica dei settori,
  - verifica dei file,
  - formato e contenuto delle directory.
- Se intraprende azioni correttive, avverte con appositi messaggi.

Verifica dei settori in Unix (1)

- Si costruisce una tabella di contatori, con un elemento per settore.
- Si identificano i settori appartenenti alla free list e a tutti i file (rintracciati partendo dagli i-node), incrementandone il contatore.
- Se alla fine qualche contatore è 0, il settore viene messo nella free-list.

Verifica dei settori in Unix (2)

- Se alla fine il contatore è maggiore di 1, vi sono 3 casi:
  - settori agganciati più volte alla free-list, che vengono riagganciati alla stessa una sola volta;
  - settori che appartengono alla free list e a un file, che vengono attribuiti a quest'ultimo;
  - settori che appartengono a più file, che vengono duplicati, inserendo una copia in ogni file.
    - I file sono probabilmente danneggiati, ma il file system ritorna consistente.

Verifica dei file in Unix

- Si costruisce una tabella di contatori, con un elemento per i-node.
- Si identificano i link nelle directory, incrementando il contatore di ogni file trovato.
- Alla fine i contatori vengono copiati negli i-node.
- Eventuali file descritti da i-node non riferiti in alcuna directory vengono messi nella root, con nome convenzionale.

Prestazioni

- Gli accessi al disco sono da 10000 a 100000 volte più lenti degli accessi in memoria.
- Per migliorare le prestazioni, riducendo gli accessi, si utilizza spesso una cache, contenente gli ultimi settori letti o scritti su un disco.
  - Un primo livello di cache è spesso integrato nel controller del disco.
- Il meccanismo è simile a quello di paginazione.

Differenze tra paginazione e caching di disco

- Il page fault viene scoperto dall'hardware, la presenza di un settore in cache dal software.
- Gli accessi ai file sono molto meno frequenti.

Aggiornamento del disco

- Non è prudente tenere un settore di dati modificato nella cache per tempi lunghi, senza aggiornare la copia su disco.
  - In caso di guasto, i dati contenuti sarebbero persi.
  - In Unix esiste un comando, `sync`, che forza lo scaricamento della cache.
  - Spesso esiste un processo che periodicamente forza l'aggiornamento (p.es., ogni ora).

Cache di disco

- Normalmente write-back, per ridurre gli accessi.
- In MS-DOS è write-through, per consentire la rimozione di un floppy in qualunque momento senza perdita di dati.
  - Molto meno efficiente, in caso di scritture multiple sullo stesso settore.
- Spesso è write-through nei dispositivi rimovibili.

Algoritmi di sostituzione della cache di disco

- Si possono utilizzare gli stessi algoritmi della paginazione (FIFO, LRU ecc.), ma è bene modificarli, dividendo i settori in categorie.
  - I settori vitali per la consistenza del file system (es.: settori contenenti i-node e directory) vanno scritti il più presto possibile.
  - I settori che presumibilmente saranno necessari tra breve (es.: settori parzialmente scritti di file aperti), vanno tenuti nella cache.