Sistemi Operativi - primo modulo
Parte V - Il file system e la memoria secondaria

Augusto Celentano
Università Ca' Foscari Venezia
Corso di Laurea in Informatica

Il concetto di file

- Un *file* è un insieme di informazioni, correlate e registrate nella memoria secondaria, cui è stato assegnato un nome
- I file sono organizzati in raggruppamenti logici (talvolta fisici) detti *cataloghi* (*directory*)
- Tipi fondamentali:
  - dati
    - numerici
    - alfabetici, alfanumerici
    - binari
    - etc.
  - programmi
    - eseguibili
    - librerie

Struttura di un file

- Nessuna: sequenza di bit, byte, word
- Strutture semplici a record
  - linee di testo
  - lunghezza fissa
  - lunghezza variabile
- Strutture complesse
  - es. documento formattato
  - es. file eseguibile
  - è possibile simulare una struttura su una semplice sequenza per mezzo di convenzioni e caratteri di controllo (es. Unix newline)
- Chi decide:
  - sistema operativo
  - programmi

Attributi dei file

- *Nome*: è l'unica informazione in un formato leggibile
- *Identificatore*: etichetta (di solito un numero) che identifica univocamente il file nel file system
- *Tipo*: informazione necessaria nei sistemi che gestiscono tipi di file diversi
- *Locazione*: puntatore al dispositivo e alla posizione del file
- *Dimensione*: dimensione corrente del file
- *Protezione*: controlla chi può leggere, scrivere o far eseguire il file
- *Ora, data, utente*: dati utili ai fini della protezione e del controllo di utilizzo

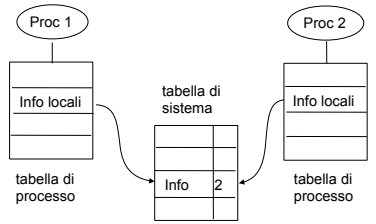
- Le informazioni sui file sono conservate in una o più strutture dati del *file system* nella memoria secondaria

Operazioni fondamentali

- Un file deve essere predisposto all'uso e rilasciato quando non è più utilizzato
  - *open*: ricerca nella struttura di directory del disco il descrittore del file, e ne porta il contenuto in memoria centrale
  - *close*: rimuove il descrittore del file dalla memoria centrale e lo trascrive nella struttura di directory del disco
- In un programma un file è identificato da un *descrittore* che contiene tutte le informazioni necessari per gestirlo
  - fd = open(nome, modo) } funzioni di basso livello (*file*)
  - close (fd) } di Unix, type(fd) = int
  - fd = fopen(nome, modo) } funzioni di alto livello (*stream*) della
  - fclose(fd) } libreria I/O Unix, type(fd) = FILE

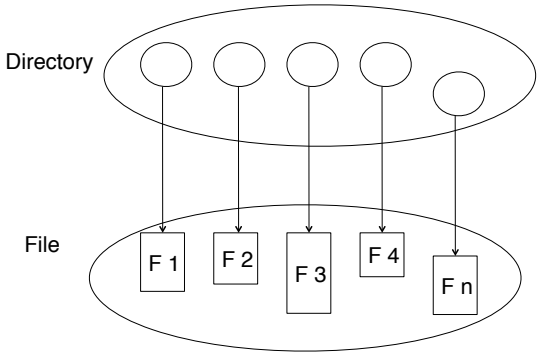
Gestione dei file in sistemi multiutente

- In un sistema multiutente/multitask servono due livelli di informazioni, contenute in tabelle
  - di sistema: indipendenti dai processi (+ contatore di aperture)
  - di processo: dipendenti dai processi
    - puntatore alla tabella di sistema
    - informazioni locali (posizione di lettura/scrittura, modalità di utilizzo)



Struttura di directory

- Un insieme di nodi contenenti informazioni sui file
  - sia la directory sia i file risiedono sul disco



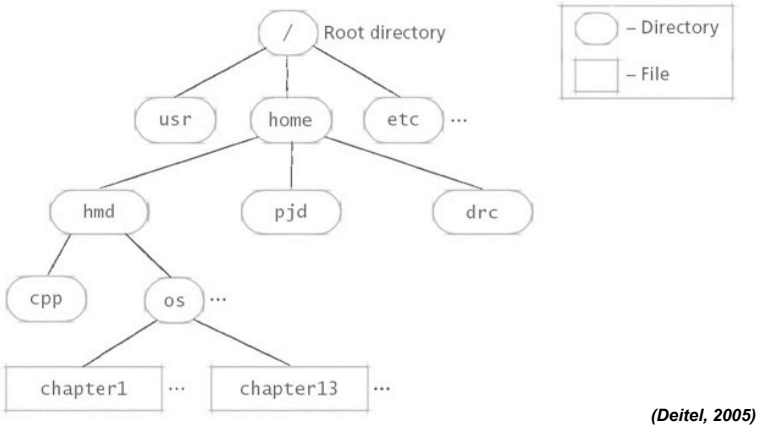
Informazioni sui file nella directory

- Sono contenute, per ogni file, in un *descrittore*
  - nome
  - tipo
  - indirizzo/i dei dati su disco
  - dimensione corrente
  - dimensione massima consentita o raggiunta
  - data di creazione
  - data dell'ultimo accesso
  - data dell'ultimo aggiornamento
  - ID del proprietario
  - informazioni di protezione
  - ... etc ...

Operazioni su directory

- Ricerca di un file
- Creazione di un file
- Cancellazione di un file
- Elenco del contenuto di una directory
- Ridenominazione di un file
- Attraversamento del file system

Directory gerarchica o ad albero (1)



(Deitel, 2005)

Directory gerarchica o ad albero (2)

- Ricerca efficiente
  - esplorazione di un solo sottoalbero
- Possibilità di raggruppamento secondo criteri imposti dall'utente
  - gerarchie arbitrarie
- Necessità di meccanismi di protezione
  - gerarchie separate per utenti diversi
- Path name relativo o assoluto
  - concetto di "posizione corrente" (*process working directory*)

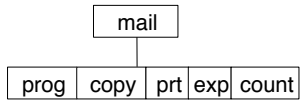
```
% cd /spell/mail/prog
% pwd
```


Directory gerarchica o ad albero (3)

- La creazione di un nuovo file può far riferimento alla directory corrente o al pathname assoluto
- Idem per la creazione di una nuova directory

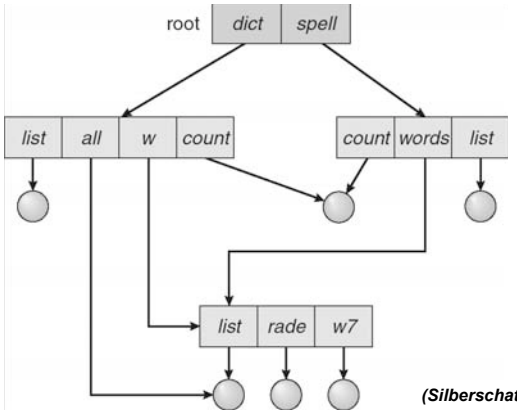
```
% mkdir esami
% mkdir /Users/auce/Auce/Sistemi\ operativi/soa2013
```

- Cancellazione di file e directory
  - se la directory non è vuota?
  - problemi di coerenza
  - es. cancellando mail si cancella (si cancellerebbe...) l'intero sottoalbero che ne discende



Directory a grafo aciclico

- Possono avere sotto-directory e file condivisi in più directory

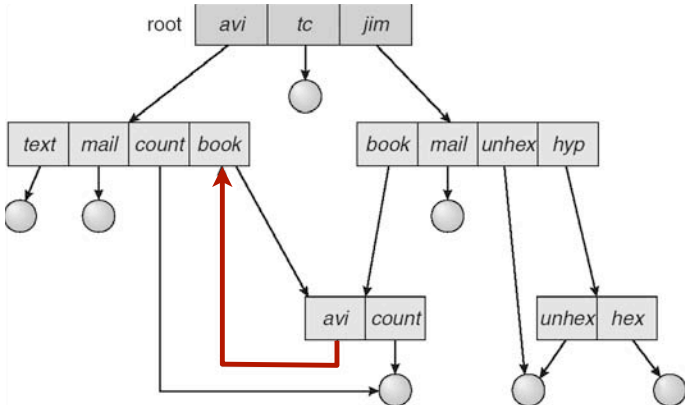


(Silberschatz, 2006)

Directory a grafo aciclico

- Più pathname per lo stesso file
  - *aliasing*
- Se si cancella un file condiviso restano riferimenti non risolti (*dangling reference*)
  - riferimenti da un file a tutte le directory in cui è contenuto
    - struttura a dimensione variabile
    - struttura a *daisy chain*
  - contatori di condivisione, il file si cancella solo se non più accessibile
- Creazione di nuovi elementi in directory
  - *link*, crea un altro nome per un file esistente
  - *alias*, crea un riferimento simbolico (indiretto) ad un file
  - per accedere al file si segue il link o l'alias
  - *link* e *alias* sono concetti differenti, implementati in modo diverso in sistemi operativi diversi

Directory a grafo generale



(Silberschatz, 2006)

Directory a grafo generale

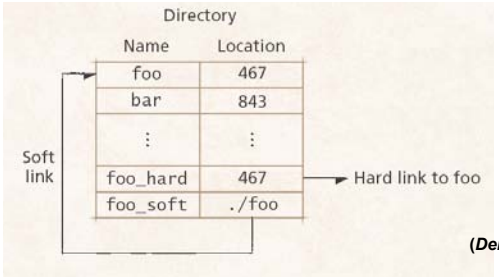
- L'esistenza di cicli è problematica
  - attraversamento infinito del file system
  - cicli isolati
- Come possiamo garantire di non avere cicli?
  - permettere solo link condivisi a file e non a directory
  - *garbage collection* per scoprire cicli isolati
  - per ogni nuovo link si controlla se crea un ciclo (oneroso)

Link multipli ai file

- Diversi path possono riferire lo stesso file o directory
  - per condividere file tra utenti
    - es. Bianchi e Bruni lavorano allo stesso progetto e possono avere i file di progetto nella propria home
  - diverse versioni di uno stesso programma
    - es gcc-ver1, gcc-ver2, /usr/bin/gcc è un link a una versione specifica
- Non ci sono più copie del file
  - il file è unico, ma esistono più riferimenti distinti
  - ogni modifica al file è visibile attraverso tutti i link

Link multipli ai file

- La struttura a grafo può essere implementata in due modi:
  - attraverso un link *simbolico* (*soft link*, *alias*)
    - elemento di directory speciale che punta al file attraverso il nome simbolico
  - attraverso un link *hard*
    - le informazioni relative al file sono duplicate nei due descrittori
    - non vi è distinzione tra i due nomi (non ci sono originali diversi dalle copie)



(*Deitel, 2005*)

Link multipli ai file

- Unix
  - Hard link
  - Symbolic link
- Windows
  - Nessun hard link
  - Collegamenti (link simbolici)
- Mac OSX
  - Link hard (a livello interno, Unix like)
  - Link simbolici (a livello interno, Unix like)
  - Alias (link simbolici, con riferimento più complesso al file originale, es. sopravvive al suo spostamento)

Link multipli ai file

- Attraversamento del file system
  - ad es. per effettuare il backup (non si vogliono salvare due copie dei file condivisi)
  - per funzioni statistiche o di accounting, non si deve accedere due volte allo stesso file / directory
  - soluzione Unix: non seguire i link simbolici (oppure marcare gli i-node visitati)
- Cancellazione
  - dopo la cancellazione di un file condiviso potrebbero rimanere dei puntatori ad un file che non esiste più
  - soluzione Unix
    - link simbolici: quando si rimuove il file i puntatori restano *dangling*; se si crea un nuovo file con lo stesso nome?
    - link hard: per ogni file si mantiene un contatore di condivisione (numero di riferimenti) e lo si elimina solo quando il contatore vale 0

Link multipli ai file

- Come fare per garantire una struttura di directory aciclica?
  - algoritmi che verificano l'aciclicità della struttura prima di aggiungere un link (costoso)
  - Si consentono link solo a file, non a directory (limitato)
- Unix adotta una soluzione di compromesso
  - sono permessi solo link simbolici (non hard) alle directory.
  - possono crearsi cicli, ma solo formati da link simbolici (non problematici)
  - esempio

% mkdir catalogo	
% cd catalogo	
% ln ../catalogo nuovo	errore (hard link a directory)
% ln -s ../catalogo nuovo	ok
% ls -R nuovo	ok
% ls -RL nuovo	(-L = *force dereference*)

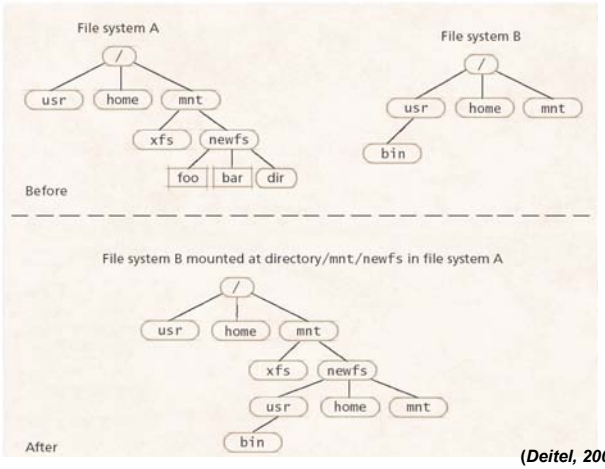
Esempio

```
~ auce$ mkdir loop
~ auce$ cd loop
~/loop auce$ ln ../loop loop
ln: ../loop: Is a directory
~/loop auce$ ln -s ../loop loop
~/loop auce$ ls -Rl loop
lrwxr-xr-x  1 auce  auce  8 15 Mar 10:18 loop -> ../loop
~/loop auce$ ls -RLl
lrwxr-xr-x  1 auce  auce  8 15 Mar 10:18 loop -> ../loop
ls: loop: directory causes a cycle
~/loop auce$ cd loop
~/loop/loop$ cd loop
~/loop/loop/loop$ pwd
/Users/auce/loop/loop/loop
~/loop/loop/loop$ cd ..
~/loop/loop$ pwd
/Users/auce/loop/loop
~/loop/loop/loop$ cd ..
~/loop/loop$ pwd
/Users/auce/loop
```


Montaggio di un file system (1)

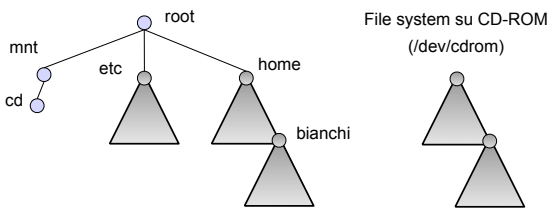
- Un file system deve essere "montato" (*mount*) prima di essere utilizzato
- Il montaggio consiste nell'associare la radice del file system con un *punto di montaggio*, solitamente foglia del file system principale
  - si crea una struttura gerarchica estensibile
  - si possono aggiungere volumi rimovibili
- Un file system montato può essere "smontato" (*unmount*)

Montaggio di un file system (2)

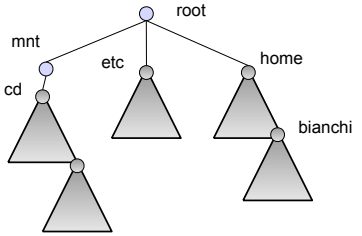


(*Deitel, 2005*)

Montaggio di un volume rimovibile



... dopo mount(/dev/cdrom, /mnt/cd):



Montaggio iniziale del file system

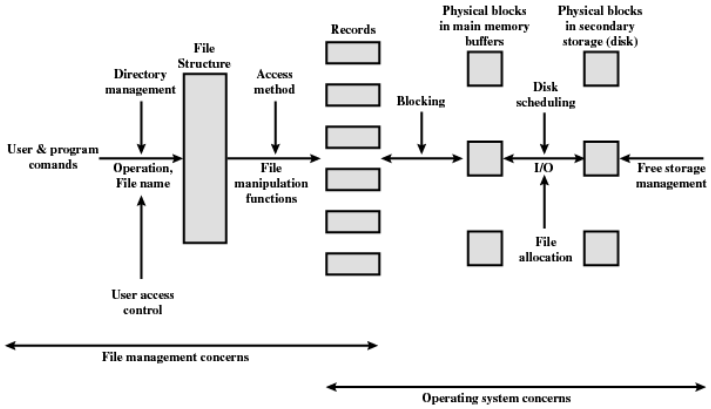
> more /etc/fstab

/etc/fstab: static file system information.

#

# <file system>	<mount point>	<type>	<options>
proc	/proc	proc	defaults
/dev/cciss/c0d0p1	/	ext3	defaults,errors=remount-ro
/dev/cciss/c0d0p11	/extra	ext3	defaults,quota
/dev/cciss/c0d0p9	/home	ext3	defaults,quota
/dev/cciss/c0d0p12	/cvsrep	ext3	defaults
/dev/cciss/c0d0p13	/scratch	ext3	noatime
/dev/cciss/c0d0p5	/usr	ext3	noatime
/dev/cciss/c0d0p6	/var	ext3	defaults
/dev/cciss/c0d0p10	/var/mail	ext3	defaults,quota
/dev/cciss/c0d0p7	none	swap	sw
/dev/cciss/c0d0p8	none	swap	sw
/dev/hda	/media/cdrom0	udf,iso9660	ro,user,noauto
/dev/fd0	/media/floppy0	auto	rw,user,noauto

Struttura del file system

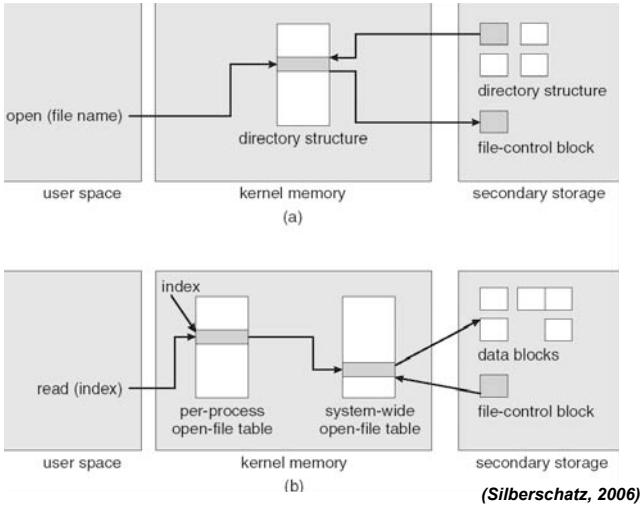


(Stallings, 2005)

Livelli di File System

- I comuni s.o. possono gestire più file system in modo non esclusivo
  - Unix/Linux: UFS, ext, ext2, ext3, msdos, ntfs, xenix, minix, ...
  - Windows: FAT, FAT32, NTFS
  - Mac OS X: HFS, HFS+
- I vari file system differiscono per il livello *file system logico*, mentre possono condividere:
  - controllo dell'I/O
  - file system di base
- Se sono presenti più file system è necessario un ulteriore livello che fornisca un'interfaccia comune
  - Linux: Virtual File System, VFS

Strutture del file system in memoria



(Silberschatz, 2006)

Metodi di allocazione

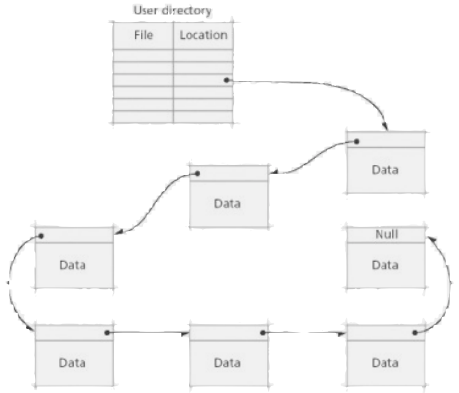
- Un *metodo di allocazione* indica il modo in cui i blocchi di disco vengono assegnati ai file
  - *allocazione contigua*: il file è allocato in blocchi contigui sulla memoria secondaria
  - *allocazione concatenata*: i blocchi del file sono collegati in sequenza attraverso puntatori interni o memorizzati in una tabella separata
  - *allocazione indicizzata*: i blocchi del file sono indicizzati in una struttura dati esterna al file

Allocazione contigua

- Ciascun file occupa un insieme di blocchi contigui nel disco
  - per ogni file è richiesto solo l'indirizzo del primo blocco (numero di blocco) e la lunghezza (in blocchi)
- Corrispondenza immediata tra indirizzo logico e posizione fisica
- E' una soluzione critica per l'impiego efficiente di spazio
  - ha gli stessi problemi della gestione della memoria centrale non paginata
  - genera frammentazione che può impedire l'allocazione di un file anche se la memoria complessivamente libera è elevata
- L'allocazione continua può consentire una elevata velocità di accesso sequenziale al file
  - es. CD audio, DVD video
  - es. caricamento in DMA di un file eseguibile (sistema RSX-11M, anni '80)

Allocazione concatenata (1)

- Ciascun file è composto da una lista concatenata di blocchi che possono essere sparsi in qualsiasi punto del disco



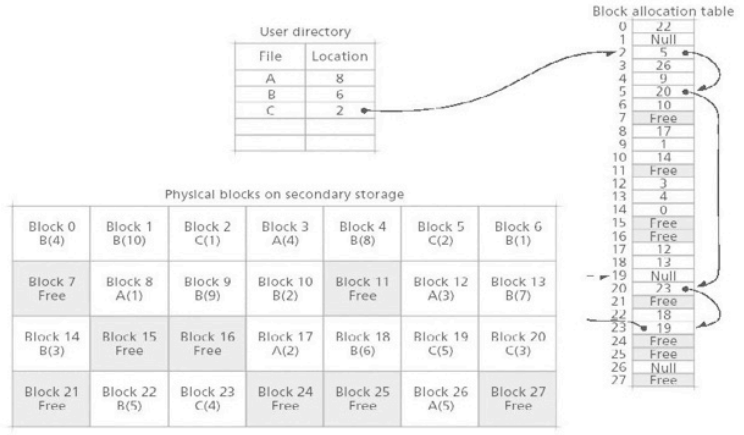
(Deitel, 2005)

Allocazione concatenata (2)

- E' semplice: necessita solo dell'indirizzo di partenza
  - altre informazioni (es. indirizzo dell'ultimo blocco) rendono più veloci alcune operazioni ma non sono indispensabili
- Sistema di gestione dello spazio libero: non c'è spreco di spazio
  - ma ogni blocco contiene dati + puntatore al prossimo blocco
  - problemi nel caricamento DMA
  - dimensione dei dati non gestibile come multiplo della dimensione del blocco
  - la perdita di un blocco causa la perdita della restante parte di file
- Accesso casuale simulato con scansione

- Variante: tabella di allocazione dei file (FAT, *file allocation table*).
  - usato nei sistemi operativi Microsoft (MS-DOS e Windows) e per l'interoperabilità delle memorie rimovibili (es. USB)

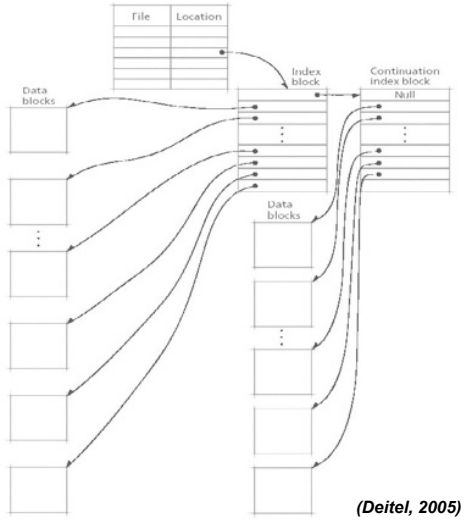
File-Allocation Table



(Deitel, 2005)

Allocazione indicizzata (1)

- Raggruppa tutti i puntatori in una sola locazione: l' indice (di uno o più blocchi)

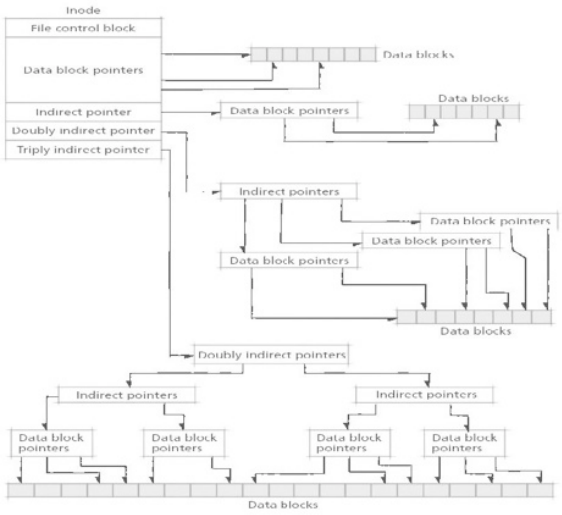


(Deitel, 2005)

Allocazione indicizzata (2)

- Necessita di una struttura supplementare per l'indice di ogni file
  - è auspicabile una dimensione piccola rispetto a quella del file
  - se l'indice è troppo piccolo non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni
- Accesso casuale
- Allocazione dinamica senza frammentazione esterna
- E' necessario un meccanismo per la gestione di file di dimensioni ragionevolmente grandi:
  - schema concatenato (dimensioni potenzialmente illimitate)
  - indice a più livelli (dimensioni limitate, accesso oneroso a file piccoli)
  - schema combinato (Unix, soluzione favorevole per file piccoli, consente file molto grandi)

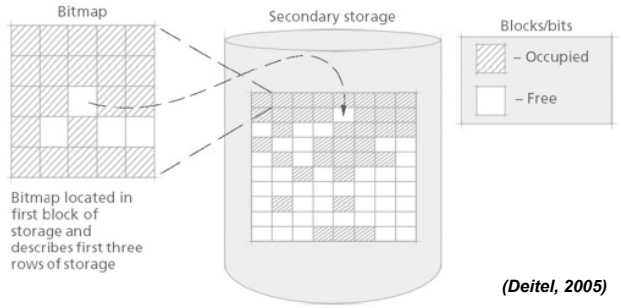
Schema combinato: UNIX



(Deitel, 2005)

Gestione dello spazio libero (1)

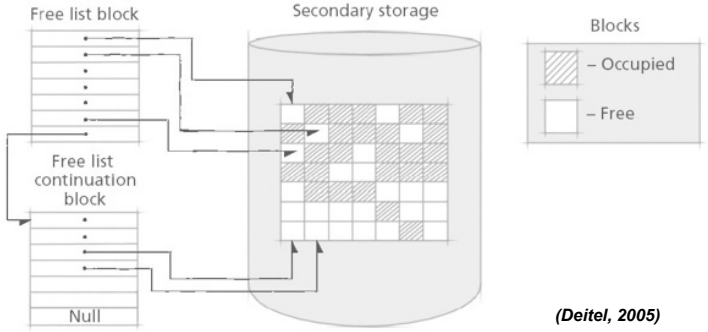
- Vettore di bit (*n* blocchi)
  - bit[i] = 0 se il blocco *i* è occupato
- Il numero del primo blocco libero è dato da:
  - (n. di bit per parola) x (n. di parole a 0) + offset del primo bit 1



(Deitel, 2005)

Gestione dello spazio libero (2)

- Lista concatenata di blocchi liberi
  - l'accesso al primo blocco libero è immediato
  - l'accesso a *n* blocchi consecutivi liberi è più complesso



(Deitel, 2005)

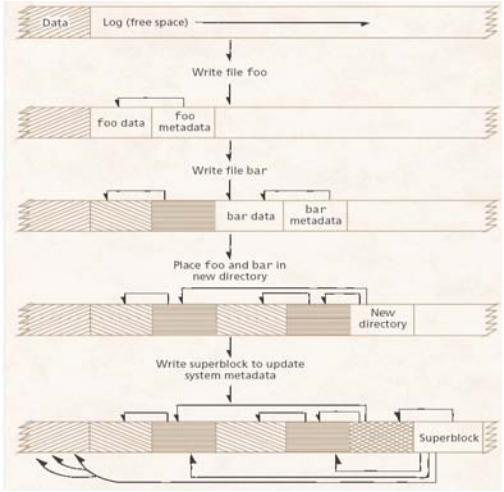
Gestione dello spazio libero (3)

- La bitmap richiede spazio per la sua memorizzazione
  - esempio:
    block size = 2^{12} bytes (4 Kbyte)
    disk size = 2^{36} bytes (64 Gbyte)
    n = $2^{36}/2^{12} = 2^{24}$ bits (2 Mbyte)
  - è facile consultarla per trovare ampio spazio contiguo
- La lista concatenata (FAT) non richiede spazio supplementare
  - i blocchi liberi vengono concatenati come se fossero un unico file
  - non è facile consultarla per trovare ampio spazio contiguo

Journaled File System (1)

- I file system con annotazione (*journaled file system*) tracciano gli aggiornamento nelle strutture del file system (metadati)
  - file system orientati alle transazioni (*log-based transaction-oriented file system*)
- Ogni insieme di operazioni che esegue uno specifico compito si chiama *transazione*
  - una transazione è un insieme di operazioni considerate atomiche (tutto o niente)
  - prima di eseguire una transazione, si esegue un'istruzione di inizio transazione
  - si eseguono le operazioni di modifica dei metadati
  - al termine si esegue un'istruzione di conferma transazione (*commit*)
  - il processo utente riprende l'esecuzione

Journaled File System (2)



(Deitel, 2005)

Journaled File System (3)

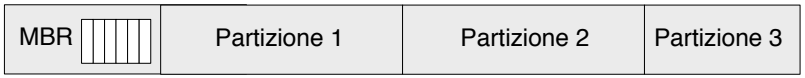
- Le modifiche effettuate sono riportate in modo asincrono sul file system al termine della transazione e rimosse dal file di log
- Il log può essere memorizzato in una sezione separata del file system, o anche in un disco doiverso.
- Se si verifica un'interruzione nel funzionamento del sistema, tutte le transazioni nel log devono ancora essere eseguite
- A fronte di maggiori garanzie di coerenza ci sono problemi di performance
  - il file di log è sequenziale e non consente accesso diretto
  - per limitare il carico si possono registrare solo le modifiche dei metadati e non quelle che riguardano il contenuto dei file

Organizzazione delle informazioni sul disco (1)

- Per usare un disco come contenitore di informazioni (sistema operativo, programmi e dati) il sistema operativo deve registrare le proprie strutture dati all'interno del disco
  - registrare le informazioni per l'avvio del sistema
  - suddividere il disco in una o più parti indipendenti (*partizioni*)
  - creare un file system (*formattazione logica*)
  - registrare le informazioni generali: es., indice dei file e delle directory, mappa dei blocchi liberi, etc.
- Il blocco d'avviamento (*boot block*) contiene il codice che inizializza il sistema
  - un piccolo caricatore d'avviamento (*bootstrap loader*) è memorizzato nella ROM.

Organizzazione delle informazioni sul disco (2)

- Un disco può essere diviso in una o più partizioni, porzioni indipendenti che possono ospitare file system distinti, nonché diversi sistemi operativi
- Il primo settore dei dischi è detto *master boot record* (MBR)
  - contiene il *boot loader*, utilizzato per il boot del sistema
  - contiene la *partition table* (tabella delle partizioni)
  - contiene l'indicazione della partizione attiva (*bootable*)
- All'avvio (*boot*), il MBR viene letto ed eseguito



Struttura generale di una partizione (1)

- Boot block
  - informazioni necessarie per avviare il s.o. da questa partizione
  - in NTFS: *partition boot sector*
- Superblock
  - informazioni sul tipo di file system e sui parametri fondamentali della sua organizzazione
    - numero e dimensione dei blocchi della partizione
    - dimensione massima di un file
    - posizione e dimensione delle tabelle per la gestione dello spazio libero / occupato
  - in NTFS: *master file table*

(Deitel, 2005)

Struttura generale di una partizione (2)

- Group descriptors
  - decrizione della posizione dei componenti di un block group
- Block allocation map
  - struttura dati per la gestione dello spazio libero (es. contatore blocchi liberi e relativi puntatori)
- I-node allocation map + i-node table
  - struttura dati per la gestione dello spazio allocato ai file
  - in NTFS: database relazionale nella MFT
- Data blocks
  - spazio dati per la struttura di directory e i file

(Deitel, 2005)

Boot del sistema

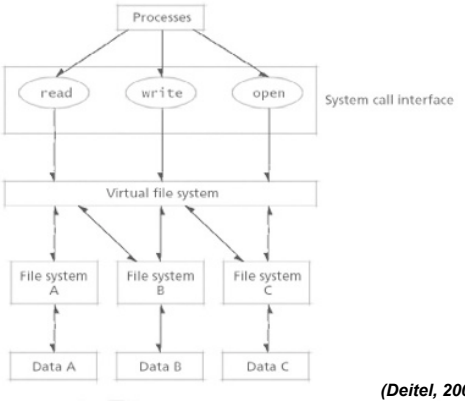
- All'avvio (*boot*) del sistema
  - si carica il MBR e lo si esegue
  - MBR carica il *boot block* della partizione attiva
    - MBR deve conoscere i diversi file system nel sistema
  - il boot block monta la *root partition* che contiene il nucleo del s.o. (ed altri file di sistema)
  - il s.o. monta le altre partizioni
    - automaticamente, nel caso di Windows, MacOSX
    - come indicato nel file /etc/fstab, o per esplicita richiesta di utenti / applicazioni, nel caso di Unix
- MBR e boot block sono in posizione e formato prestabiliti e non vi si accede tramite il file system (che entra in gioco più tardi)
  - *raw partition* (partizione senza file system)
  - *raw partition* usata anche per l'area di swap dei processi

Il File System di Unix

- Everything is a file
  - l'accesso ai dispositivi di I/O avviene attraverso un'interfaccia astratta che rende omogenee le operazioni assimilandole ad operazioni su file
- I tipi di file gestiti da Unix/Linux sono sette
  - regular file: un file secondo l'accezione comune
  - directory: un catalogo di file
  - symbolic link: un file che contiene un riferimento ad un altro file
  - char device: un file che identifica dispositiviìo di I/O ad accesso seriale a caratteri
  - block device: un file che identifica un dispositivo di I/O ad accesso a blocchi
  - fifo: un file che identifica un canale di comunicazione unidirezionale (pipe)
  - socket: un file che identifica un canale di comunicazione bidirezionale

Linux Virtual File System (1)

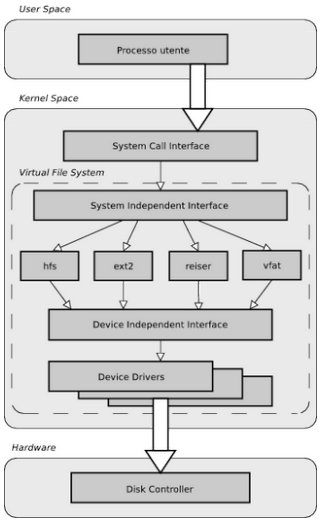
- In Linux la virtualizzazione dei tipi dei file è realizzata attraverso un livello di file system virtuale, il Virtual File System (VFS)



(Deitel, 2005)

Linux Virtual File System (2)

- Le system call che operano su file sono intercettate dal VFS
  - VFS esegue le operazioni che coinvolgono le strutture dati generiche del file system
  - VFS richiama le funzioni del file system specifico (non le implementa)
  - il file system specifico gestisce l'interfaccia verso le funzioni di I/O di basso livello
- Le operazioni sono divise su tre tipi di oggetti
  - file system, inode, file



Linux Virtual File System (3)

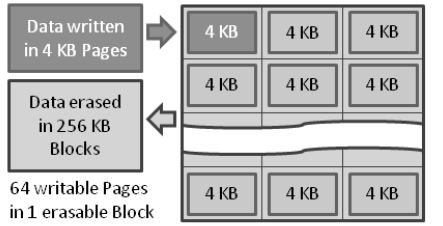
- VFS superblock
  - Contiene informazioni su un file system montato: tipo del file system, posizione su disco dell'inode root, informaiozni contabili, eetc.
  - Esiste solo in memoria centrale, è creato quando il file system viene montato
- VFS inode
  - descrive la posizione di ogni file (directory, link) in ogni file system disponibile
  - ogni file è identificato da una coppia (inode - file system)
- File descriptor
  - informazioni sull'inode
  - informazioni sulla posizione corrente nel file
  - flag di accesso (e.g. read/write, append-only)

File system per memorie a stato solido

- Le memorie a stato solido (SSD, flash memory) presentano alcuni problemi specifici che richiedono soluzioni ad hoc nella gestione dei dati memorizzati
  - le tecnologie utilizzate presentano una diversa granularità di accesso per lettura/scrittura/cancellazione
  - l'affidabilità diminuisce con l'aumentare delle operazioni di scrittura
- I dispositivi di memoria commerciali hanno normalmente a bordo un controller che si occupa di questi problemi
  - il file system può ignorarli e comportarsi come se operasse un disco magnetico
  - le prestazioni tuttavia rimangono molto variabili in funzione del tipo di uso

Limitazioni operative delle memorie a stato solido (1)

- Le memorie flash presentano una granularità diversa per le operazioni di lettura/scrittura/cancellazione
  - possono essere lette o scritte inizialmente partendo da uno stato "zero" (*programmate*) indirizzando le celle in modo random con accesso al byte
  - posso essere cancellate solo "a blocchi"
  - la cancellazione pone tutti i bit del blocco a 1
  - un bit a 1 può essere scritto con 0 con un'indirizzamento singolo
  - un bit a 0 può essere riportato a 1 solo cancellando l'intero blocco



Typical NAND Flash Pages and Blocks

Limitazioni operative delle memorie a stato solido (2)

- L'affidabilità diminuisce con l'aumentare delle operazioni di scrittura
  - si verifica un fenomeno di "usura" del materiale che non garantisce affidabilità dopo un certo numero di cicli di cancellazione
  - normalmente da 5.000 a 50.000-100.000 cicli in funzione della tecnologia e della destinazione d'uso
  - il problema viene parzialmente risolto adottando politiche di rotazione dei blocchi e di garbage collection
- L'usura è tipica anche delle memorie ottiche riscrivibili, con un numero di cicli di scrittura molto più limitato (~1.000 cicli)

Write amplification

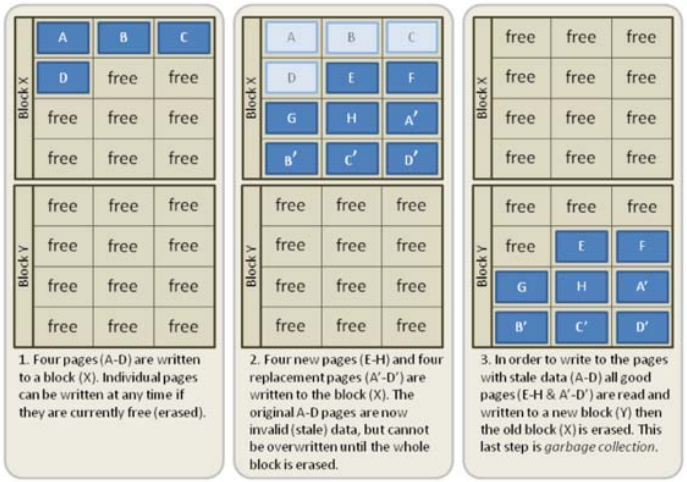
- Con il termine *write amplification* si indica l'aumento effettivo delle operazioni fisiche di scrittura su una memoria a stato solido rispetto alle operazioni comandate dal file system
  - deriva dalla necessità di cancellare e riscrivere un intero blocco a fronte di modifiche
  - è influenzato anche dalle tecniche di rotazione dei blocchi usati per uniformare l'usura ("wear leveling")

$$WA = \frac{\text{dati scritti sulla memoria}}{\text{dati inviati dall'host}}$$

Wear leveling

- E' una tecnica di rotazione dell'uso dei blocchi di una memoria a stato solido per distribuire in modo uniforme l'usura e prolungare il ciclo di vita del dispositivo
  - la modifica di una pagina non viene attuata *in loco* ma scrivendo l'intero contenuto della pagina modificata in un'altra pagina e marcando la prima pagina come inutilizzabile
  - quando un intero blocco è stato utilizzato il suo contenuto viene scritto in un nuovo blocco e il primo viene cancellato per essere riutilizzato
- Si hanno due implementazioni
  - *leveling dinamico*: solo i blocchi che vengono modificati sono periodicamente riscritti altrove
  - l'usura del dipositivo è variabile da blocco a blocco in funzione dell'uso
  - *leveling statico*: anche i blocchi che non modificano il loro contenuto sono periodicamente riscritti altrove per uniformarne l'usura

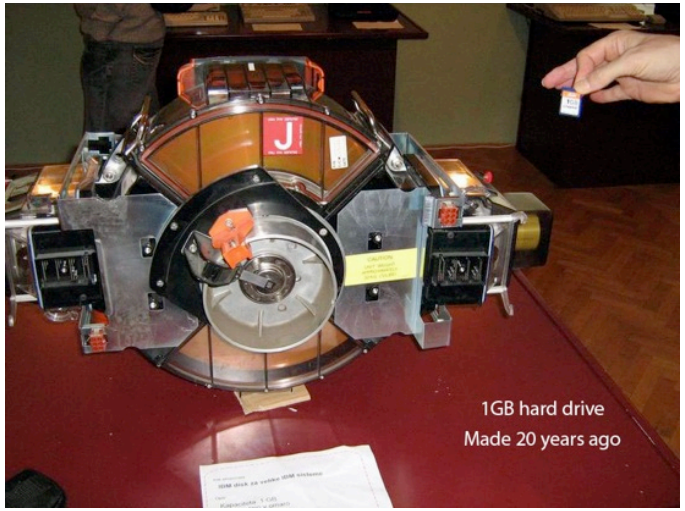
Garbage collection



Problemi a livello file system

- Nonostante i dispositivi SSD usino dei controller per la gestione dei problemi tipici della loro tecnologia, i normali file system non consentono l'ottimizzazione del loro utilizzo rispetto ai dischi magnetici
  - le operazioni di scrittura e modifica comportano scritture frequenti di poche informazioni per volta (metadati, descrittori)
- I file system orientati al *journaling* funzionano meglio perché accumulano le modifiche in una struttura dati sequenziale e le applicano alle strutture permanenti in tempi più dilazionati
- Sono stati progettati alcuni file system specifici che trovano impiego nei sistemi embedded
  - normalmente di tipo "log oriented"
- I problemi di sicurezza diventano più critici
  - la "cancellazione" dei dati non li elimina fisicamente

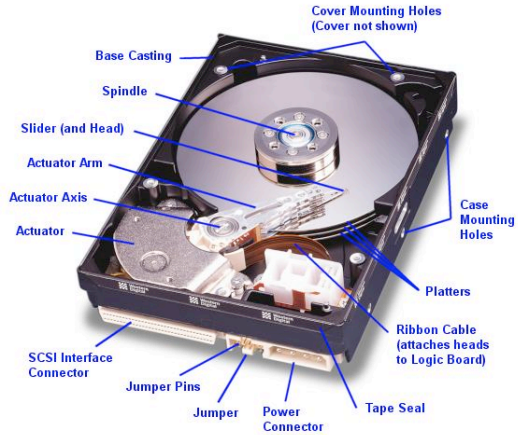
Evoluzione delle memorie secondarie

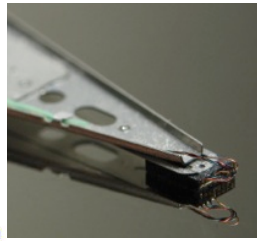


Dischi magnetici

- Proprietà principali e parametri
  - Velocità di rotazione più comuni: 4200, 5400, 7200, 10000 giri/minuto
  - Velocità di trasferimento: istantanea e a regime
  - Tempo di posizionamento: comprende il tempo necessario per muovere il braccio portatestina sul cilindro richiesto (seek time) e il tempo di rotazione necessario per portare il settore richiesto sotto la testina di lettura (rotational latency)
  - Fissi o rimovibili, a sola lettura o R/W
- Sono collegati attraverso un'interfaccia di I/O
  - Diverse tecnologie: ATA/IDE, EIDE, Serial ATA, USB, FW, SCSI
  - La gestione è curata da un controller logicamente diviso in due parti: verso l'host e verso l'unità a disco

Meccanica di un disco magnetico

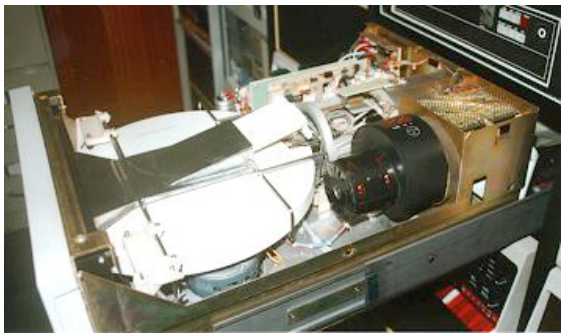




Evoluzione dei dischi magnetici

IBM RAMAC, 50x100Kbyte, 1956

Evoluzione dei dischi magnetici



Disco rimovibile DEC RK02, 2.5Mb, 1975

- The disk pack with rear door open (for head access)
- The spindle drive motor is just visible below the disk pack
- Head voice coil positioner
- The black device in the foreground is the blower fan. This draws air through the logic box to the right, and then expels is through an absolute air filter (below) and then into the disk pack.
- The power driver for the head positioner is at the rear right. The power supply (not visible) is below.

Evoluzione dei dischi magnetici

DEC RL02, 5Mbyte, rimovibile, 1980

Evoluzione dei dischi magnetici

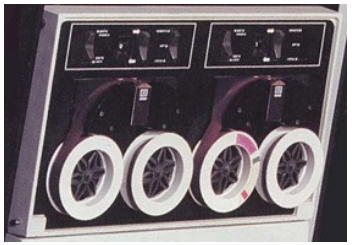
Seagate ST506, 5Mbyte, 1979

Toshiba, 0.85” 4 Gbyte, 2004

Nastri magnetici

- Proprietà principali
  - Sono stati i primi supporti per la memorizzazione esterna di dati
  - Grande capacità, lunghi tempi di conservazione (~)
  - Accesso sequenziale, bassa velocità di posizionamento, velocità di trasferimento accettabile
  - Usati per copie di archivio e per trasferimento dati tra sistemi
  - Dimensioni tipiche 20-200GB per nastro
  - Costo relativo molto basso, poco pratici
  - Cartucce, sistemi robotizzati

Nastri magnetici

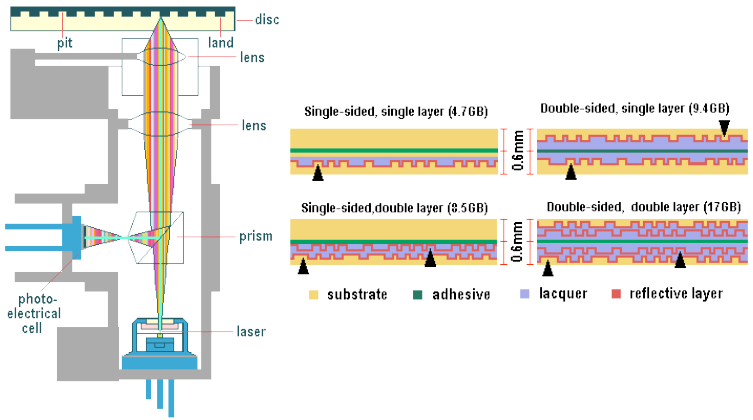


Sistemi robotizzati

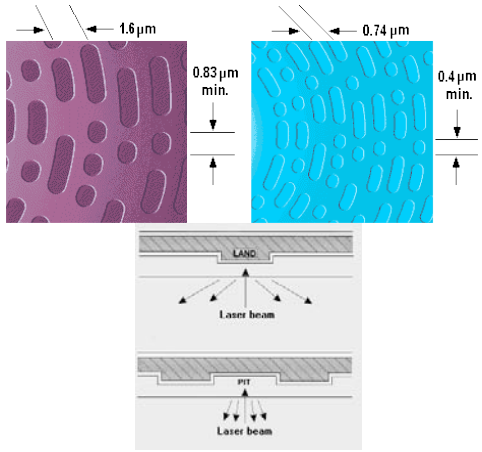
Dischi ottici

- CD-DVD
  - Nati per applicazioni audio-video, accesso prevalentemente sequenziale
  - Tecnologia di lettura e scrittura laser, capacità variabile in funzione della lunghezza d'onda del laser
  - Costo molto basso, velocità elevata in caso di accesso sequenziale
  - Durata abbastanza elevata (20-100 anni?)
  - Utilizzo prevalente per distribuzione di informazioni e archivio personale
  - Supporti non riutilizzabili (eccetto dispositivi RW)

Dischi ottici (CD-DVD)



Dischi ottici (CD-DVD)

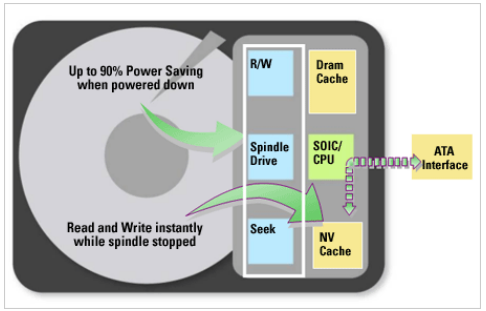


Memorie di massa a semiconduttore

- Memorie flash
  - stanno sostituendo gli hard disk di notebook e desktop
  - i costi sono in sensibile calo ma ancora elevati
  - non sono ancora chiare l'affidabilità e la durata effettiva

Memorie ibride

- Disco magnetico + memoria cache non volatile



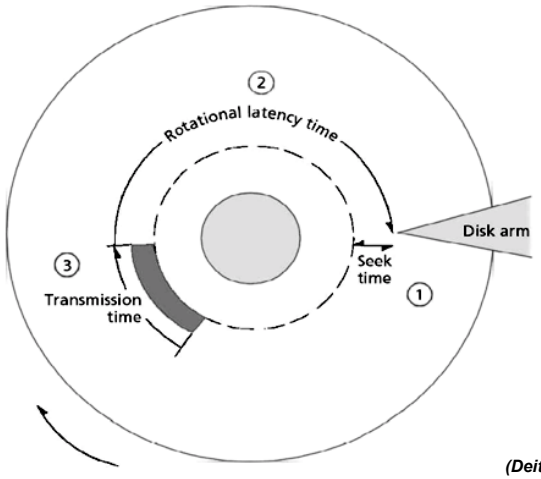
Struttura logica dei dischi magnetici

- I dischi sono considerati un grande vettore monodimensionale di blocchi logici, dove un blocco logico è la minima unità di trasferimento.

- Il vettore corrisponde in modo sequenziale ai settori del disco:
  - il settore 0 è il primo settore della prima traccia sul cilindro più esterno
  - la corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via di cilindro in cilindro, dall'esterno verso l'interno
  - eccezione: tracce di riserva

Scheduling del disco

- Il sistema operativo è responsabile di una gestione efficiente delle risorse fisiche
  - tempi d'accesso contenuti e ampiezze di banda elevate
- Il tempo d'accesso ha due componenti principali:
  - il tempo di ricerca (seek time) è il tempo necessario affinché il braccio dell'unità a disco sposti le testine fino al cilindro contenente il settore desiderato
  - la latenza di rotazione (rotational latency) è il tempo aggiuntivo necessario perché il disco ruoti finché il settore desiderato si trovi sotto la testina
  - minimizzare il tempo d'accesso ~= minimizzare la distanza percorsa
- L'ampiezza di banda del disco (disk bandwidth) è il numero totale di byte trasferiti diviso il tempo totale intercorso fra la prima richiesta e il completamento dell'ultimo trasferimento



(Deitel, 2005)

Calcolo del tempo di accesso

- Il seek time dipende dalla distanza tra le tracce

$S \cong a + bN$

~ 1 mS fra tracce adiacenti
~ 50 mS per l'intero disco (~10^4 tracce)

- La latenza di rotazione dipende dalla velocità di rotazione del disco

$L_{mS} = 1/2 \times 60/v_{rpm} \times 1000$

4200 rpm	7,14mS
5400 rpm	5,56mS
7200 rpm	4,17mS
10000 rpm	3 mS

- Il seek time è dominante

Strutture RAID

- Le *batterie ridondanti di dischi* (RAID, *redundant array of independent/inexpensive disk*) hanno lo scopo di affrontare i problemi di prestazioni e affidabilità.

- La tecnica RAID è organizzata su diversi livelli (0-6)

Dischi RAID (1)

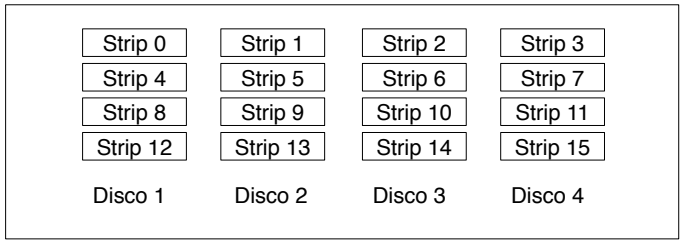
- L'evoluzione tecnologica ha reso le unità a disco progressivamente più piccole e meno costose tanto che oggi è possibile, senza eccessivi sforzi economici, equipaggiare un sistema di calcolo con molti dischi
- La presenza di più dischi, qualora si possano usare in parallelo, rende possibile l'aumento della frequenza alla quale i dati si possono leggere o scrivere
- Gli schemi RAID migliorano l'affidabilità della memoria secondaria poiché diventa possibile memorizzare le informazioni in più dischi in modo ridondante o più sicuro
  - La copiatura speculare (*mirroring* o *shadowing*) mantiene un duplicato di ciascun disco
  - L'organizzazione con blocchi intercalati a parità distribuita utilizza meno la ridondanza dei dati e più la sicurezza data dai codici di parità

Dischi RAID (2)

- Performance
  - il data path dai dischi alla memoria (controller, bus, ecc) deve essere in grado di sostenere le maggiori performance
  - l'obiettivo è quello di
    - Ridurre il tempo di accesso per accessi a grandi quantità di dati
    - Ridurre il tempo di risposta (tramite bilanciamento del carico) per accessi multipli indipendenti a piccole porzioni di dati
- Affidabilità
  - la presenza di più dischi aumenta le probabilità di guasto
  - per compensare questa riduzione di affidabilità, RAID utilizza la ridondanza nella memorizzazione (mirroring, meccanismi di parità)

RAID 0 (striping)

- Descrizione
  - il sistema RAID viene visto come un disco logico
  - i dati nel disco logico vengono suddivisi in strip (e.g., settori, blocchi, byte, bit oppure altro)
  - strip consecutive sono distribuiti su dischi diversi, aumentando le performance nell'accesso ai dati

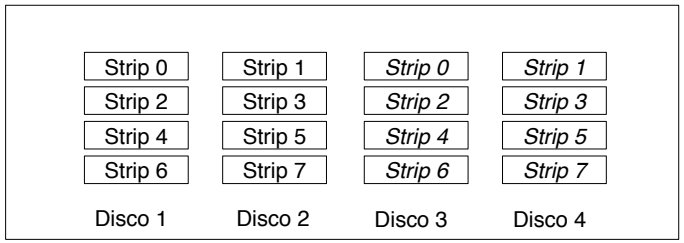


RAID 0 (striping)

- Vantaggi
  - più richieste possono essere servite in parallelo
  - stripes a livello di blocco: se due richieste di I/O riguardano blocchi indipendenti di dati, è possibile che i blocchi siano su dischi differenti
  - stripes a livello < blocco: una richiesta di un blocco viene servita in tempo minore (blocco più grande nello stesso tempo)
- Ma...
  - Non è un membro "a tutti gli effetti" della famiglia RAID, perchè non utilizza meccanismi di ridondanza
  - Può essere utilizzato per applicazioni in cui l'affidabilità non è un grosso problema, ma lo sono la velocità e il basso costo

RAID 1 (mirroring)

- Descrizione
  - Adotta uno stile di ridondanza semplice: mirroring (shadowing)
  - I dati di ogni disco sono copiati in modo speculare su un altro disco di un secondo insieme
  - Come prima, il sistema è basato su striping, ma questa volta ogni strip viene mappato su due dischi diversi



RAID 1

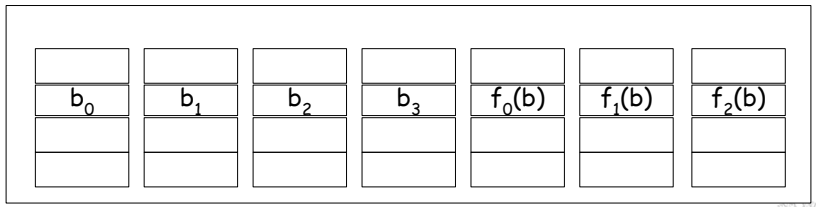
- Performance
  - ogni richiesta di lettura può essere servita da uno qualsiasi dei due dischi che ospitano il dato
    - si può scegliere quello con tempo di seek minore
  - una richiesta di scrittura deve essere portata a termine su ambedue i dischi
    - è legato alla più lenta delle due scritture
- Ridondanza
  - il recovery è molto semplice
    - se un disco si guasta, i dati possono essere recuperati dalla sua copia speculare
    - è quindi necessario sostituire il disco con la copia
- Il costo per unità di memorizzazione raddoppia

RAID 2-3 (accesso parallelo)

- Accesso parallelo
  - tutti i dischi partecipano all'esecuzione di ogni richiesta di I/O
  - i dischi sono sincronizzati in modo che le testine di lettura siano nella stessa posizione allo stesso istante
  - suddivisione fra dischi dati e dischi parità
    - un codice di correzione di errore o di parità viene calcolato a partire dai bit corrispondenti dei dischi dati
    - questo codice viene memorizzato nei dischi parità
  - si utilizza data striping, con stripes molto piccoli (bit, byte, word)

RAID 2

- Descrizione
  - ECC (error correction code) è basato tipicamente sul codice di Hamming, con distanza 3
    - permette di correggere errori fino a un bit (e di rilevare errori fino a due bit)
  - il numero di dischi di parità è proporzionale al logaritmo del numero di dischi di dati
  - è costoso



RAID 3

- Descrizione
  - il codice calcolato è un semplice bit di parità
  - meno costoso: è richiesto un solo disco di parità
  - idea:
    - i dischi hanno già dei meccanismi interni di controllo degli errori
    - una lettura errata viene segnalata dal disco interessato
    - il bit di parità consente quindi di correggere l'errore