

Sistemi Operativi A Parte V - Il file system

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

Augusto Celentano, Sistemi Operativi A

1

Struttura dei 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

Augusto Celentano, Sistemi Operativi A

2

Attributi dei file (I)

- *Nome*
 - è l'unica informazione in forma umanamente leggibile
- *Identificatore*
 - etichetta univoca (numero) che identifica il file nel file system
- *Tipo*
 - informazione necessaria ai sistemi che gestiscono tipi di file diversi
- *Locazione*
 - puntatore al dispositivo e alla locazione del file in tale dispositivo
- *Dimensione*
 - dimensione corrente del file

Augusto Celentano, Sistemi Operativi A

3

Attributi dei file (2)

- *Protezione*
 - controlla chi può leggere, scrivere o far eseguire il file
- *Ora, data e identificazione dell'utente*
 - dati utili ai fini della protezione e del controllo di utilizzo
- Le informazioni sui file sono conservate nella struttura di directory, che risiede anch'essa nella memoria secondaria

Operazioni sui file

- Dipendono dal sistema specifico, ma includono un nucleo di operazioni essenziali cui possono aggiungersi operazioni altrimenti realizzabili con combinazioni di operazioni base
 - Creazione di un file (*create*)
 - Scrittura di un file (*write*)
 - Lettura di un file (*read*)
 - Posizionamento in un file (*seek*)
 - Cancellazione di un file (*delete, remove, unlink*)
 - Troncamento di un file (*truncate*)

Operazioni fondamentali

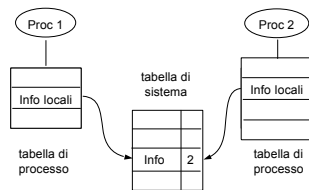
- Un file deve essere predisposto all'uso e rilasciato quando non è più utilizzato
 - *open(Fi)*: ricerca nella struttura di directory del disco l'elemento *Fi* (descrittore di file), e ne porta il contenuto in memoria
 - *close (Fi)*: rimuove il contenuto dell'elemento *Fi* dalla memoria e lo trascrive 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)*
 - *close (fd)*
 - } funzioni di basso livello (*file*)
 di Unix, type(fd) = int
 - *fd = fopen(nome, modo)*
 - *fclose(fd)*
 - } funzioni di alto livello (*stream*) della
 libreria I/O Unix, type(fd) = FILE

Gestione dei file aperti

- Per gestire i file aperti sono necessarie alcune informazioni specifiche
 - *file pointer*: puntatore all'ultima posizione letta/scritta
 - uno per ogni processo che usa il file
 - posizione del file su disco
 - *cache* degli accessi recenti
 - diritti di accesso
 - specifici per ogni processo
 - contatore di aperture: conta quante volte un file è aperto
 - serve per rimuovere le informazioni di gestione quando il file non è più utilizzato da nessuno

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)



Identificazione dei tipi di file

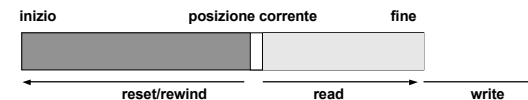
- Il tipo di un file può essere riconosciuto e gestito in modo personalizzato
 - Unix, Windows: supporto minimo da parte del sistema
 - sequenze di byte, formato definito dalle applicazioni
 - tipo identificato per convenzione sul nome (estensione) o sui primi caratteri del contenuto (*magic number*)
 - eseguibili e librerie hanno un formato definito dal sistema
 - MacOS: il sistema gestisce un livello di riconoscimento più avanzato
 - tipo/creatore del file, richiama automaticamente un'applicazione per la gestione
 - ripartizione in *resource fork* e *data fork*
 - RSX-11M, VMS: gestisce diversi tipi di formati con operazioni specializzate
 - ad accesso sequenziale / diretto
 - a record / a blocchi / stream

Metodi d'accesso

- Accesso sequenziale
 - read next
 - write next
 - reset, rewind
 - (rewrite)
- Accesso diretto
 - read n (n = numero relativo del record)
 - write n (rewrite n)
 - seek n
 - (read next)
 - (write next)

File ad accesso sequenziale

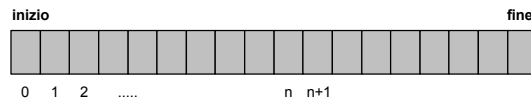
- Si ispira al modello dei nastri magnetici



- read next
- write next
- reset, rewind, rewrite
- Lettura e scrittura sono strettamente sequenziali
 - la scrittura nella posizione corrente imposta la fine del file
 - non è possibile leggere dopo l'ultimo dato scritto

File ad accesso diretto

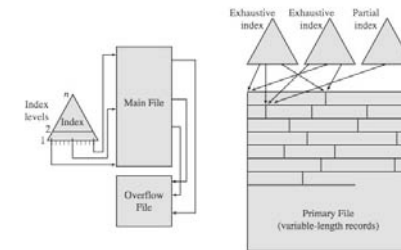
- Si ispira al modello dell'array
 - Il file è formato da blocchi logici (di lunghezza fissa), detti record, numerati progressivamente, ad accesso casuale
- E' la modalità di accesso ai dischi (ma la rotazione è sequenziale)



- read n (n = numero relativo del record)
- write n (rewrite n)
- seek n
- (read next)
- (write next)

File a indice

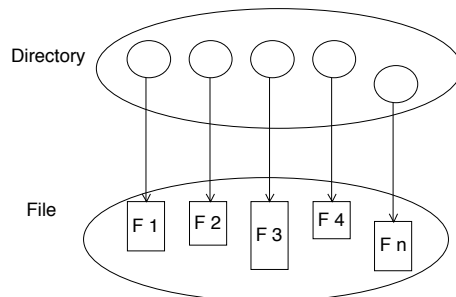
- L'accesso è associativo attraverso una o più chiavi di accesso
 - c'è sempre una chiave primaria univoca che identifica il record
 - l'ordine di accesso dipende dalle chiavi
 - la lettura con una chiave inesistente è erronea
 - la scrittura con una chiave esistente può essere un aggiornamento o un errore



Stallings, 2005

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

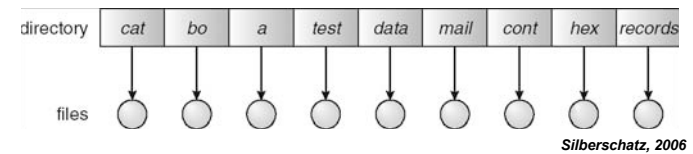
- Sono contenute in un descrittore del file
 - nome
 - tipo
 - indirizzo su disco
 - dimensione corrente
 - dimensione massima
 - data dell'ultimo accesso
 - data dell'ultimo aggiornamento
 - ID del proprietario
 - informazioni di protezione
 - ... etc ...

Operazioni eseguite su una directory

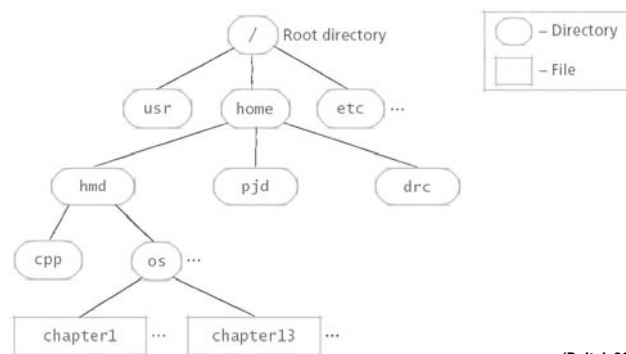
- Ricerca di un file
- Creazione di un file
- Cancellazione di un file
- Elencazione di una directory
- Ridenominazione di un file
- Attraversamento del file system

Directory a livello singolo

- Una singola directory per tutti gli utenti
 - confusione nei nomi dei file
 - problemi di gestione (nessun raggruppamento)



Directory ad albero (gerarchica)



(Deitel, 2006)

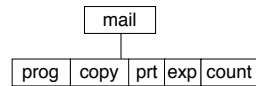
Directory ad albero

- 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" (working directory)

» `cd /spell/mail/prog`
» `type list`

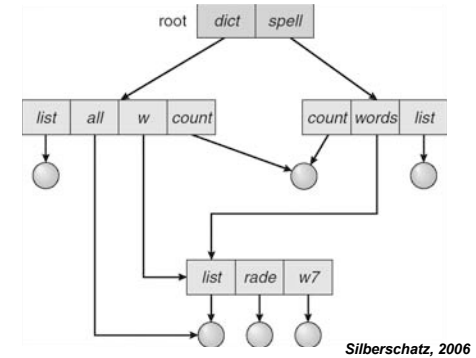
Directory ad albergo

- La creazione di un nuovo file può far riferimento alla directory corrente o al path name assoluto
 - `mkdir esami`
 - `mkdir /Users/auce/Auce/Soa2007`
 - Idem per la creazione di una nuova directory
 - `mkdir esami`
 - `mkdir /Users/auce/Auce/Soa2007`
 - 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
-
- ```
graph TD; mail[mail] --- row; row --- prog[prog]; row --- copy[copy]; row --- prt[prt]; row --- exp[exp]; row --- count[count];
```



## Directory a grafo aciclico

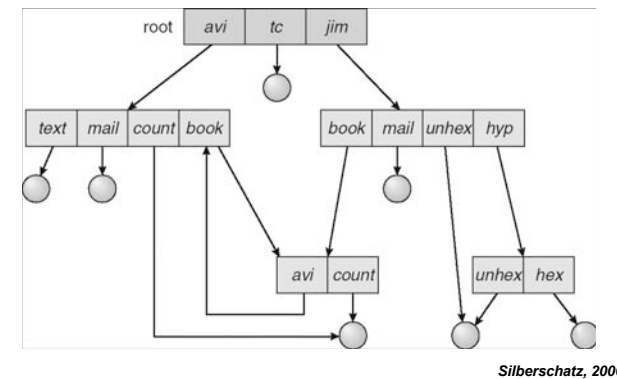
- Possono avere sotto-directory e file condivisi in più directory



## 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

## Directory a grafo generale



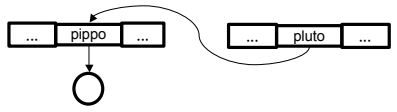
## Directory a grafo generale

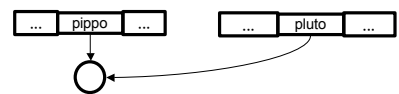
- L'esistenza di cicli è problematica
  - attraversamento infinito del file system
  - cicli isolati
- Come possiamo garantire di non avere cicli?
  - permettere solo link condivisi a file e non a directory
  - *garbage collection* per scoprire cicli isolati
  - per ogni nuovo link si controlla se crea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

## Link multipli ai file

- Diversi path possono riferire lo stesso file o directory
  - per condividere file tra utenti
    - es. Bianchi e Brunì 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 è implementata come:
  - link "simbolico" (alias)
    - elemento di directory speciale che punta al file attraverso il nome simbolico
- pluto contiene il pathname di pippo
- link "hard"
  - le informazioni relative al file sono duplicate nelle due directory
  - non vi è distinzione tra i due nomi (non ci sono originali e copie)



## 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)

## 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 (# 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 pippo
    - » cd pippo
    - » ln ../pippo errore (hard link a directory)
    - » ln -s ../pippo pippo
    - » ls -R pippo ok
    - » ls -RL ? (-L force dereference)

## Esempio

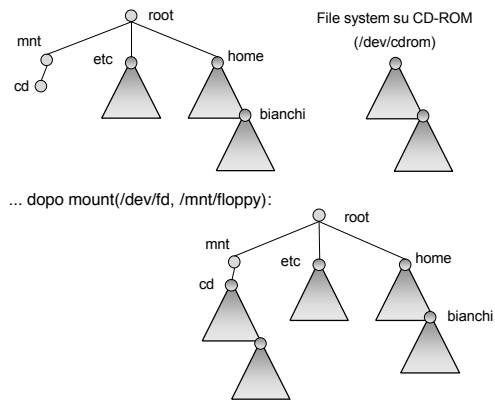
```
~ auce$ mkdir loop
~ auce$ cd loop
~/loop auce$ ln ../loop loop
ln: ../loop: Is a directory
~/loop auce$ ln -s ../loop loop
~/loop auce$ ls -Rl loop
lrwxr-xr-x 1 auce auce 8 15 Mar 10:18 loop -> ../loop
~/loop auce$ ls -RLl
lrwxr-xr-x 1 auce auce 8 15 Mar 10:18 loop -> ../loop
ls: loop: directory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

## Montaggio di un file system

- 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 volume rimovibile



## 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
```

## Protezione

- Il proprietario/creatore di un file deve essere in grado di controllare:
  - che cosa può essere fatto
  - da chi
- Tipi di accesso
  - lettura
  - scrittura
  - esecuzione
  - estensione
  - cancellazione
  - elencazione

## Controllo degli Accessi

- La tecnica più completa si basa sul concetto di Access Control List (ACL)
  - ad ogni file si associa un elenco di controllo degli accessi (ACL) che specifica
    - nomi (o classi) di utenti
    - tipi di accesso consentito
  - Esempio:
 

| File  | ACL                                   |
|-------|---------------------------------------|
| pippo | (rw, {bianchi, rossi }) (rx, {verdi}) |
| pluto | (r , {verdi, lolli}) (x, {bianchi})   |
- Consente un controllo preciso degli accessi, ma
  - la ACL può essere lunga e problematica da mantenere
  - l'elemento di directory che descrive un file non ha più dimensione fissa

## Gruppi

- Per esprimere la ACL in modo condensato molti sistemi suddividono gli utenti in tre classi
  - proprietario
    - creatore del file (default, può essere modificato)
  - gruppo
    - gruppo di utenti che condividono il file (codice di gruppo)
  - universo
    - tutti gli altri utenti
- Ad ogni file sono attribuiti dei diritti di accesso specifici per ogni classe

## Controllo degli accessi in Unix

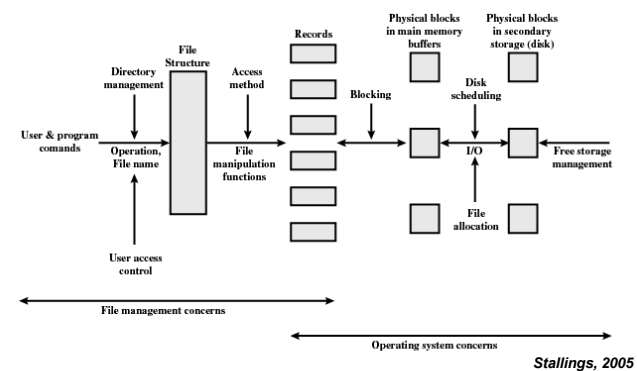
- Modalità di accesso: lettura (r), scrittura (w), esecuzione (x)
- Tre classi di utenti

|              |   |   |   |   |   |
|--------------|---|---|---|---|---|
|              |   |   | r | w | x |
| proprietario | 7 | ⇒ | 1 | 1 | 1 |
| gruppo       | 6 | ⇒ | 1 | 1 | 0 |
| universo     | 1 | ⇒ | 0 | 0 | 1 |

## Altri metodi di protezione: password

- Alternativamente la protezione di file e directory può essere assicurata tramite password
- Questo metodo non può essere applicato in generale (troppe password!)
- Può essere associato a directory e a volumi
  - MacOSX permette di associare password a immagini di volumi virtuali e a volumi rimovibili
  - Un volume rimovibile Iomega Zip può essere protetto con password (gestita dal driver attraverso una codifica ottica)

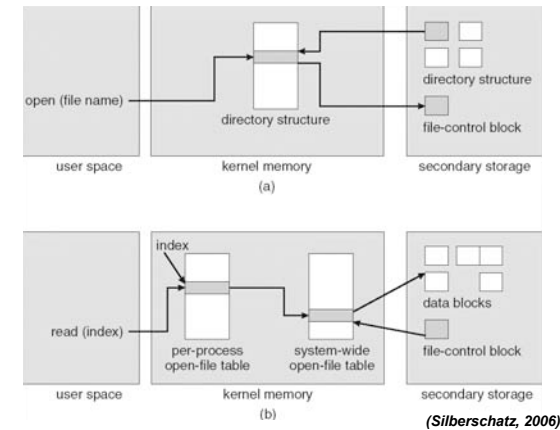
## Struttura del file system



## Livelli di File System

- I comuni s.o. gestiscono vari file system
  - 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 "FS 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



## Metodi di allocazione

- Un *metodo di allocazione* indica il modo in cui i blocchi di disco vengono assegnati ai file:
  - allocazione contigua
  - allocazione concatenata
  - allocazione indicizzata

## Allocazione contigua (I)

- Ciascun file occupa un insieme di blocchi contigui nel disco
  - è una soluzione semplice
  - 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

$$\begin{array}{l} \text{Indirizzo Logico} \\ \text{Dimensione cluster} \end{array} \begin{array}{l} \nearrow Q \\ \searrow R \end{array}$$

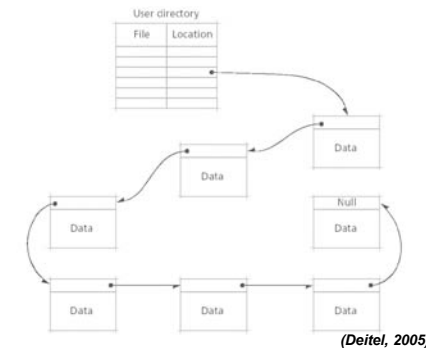
Blocco di accesso =  $Q + \text{indirizzo del primo blocco}$   
 Offset nel blocco =  $R$

## Allocazione contigua (2)

- E' una soluzione critica per l'impiego efficiente di spazio
  - è simile al problema dell'assegnazione dinamica della memoria
  - 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 intero file eseguibile nel sistema RSX-11M

## Allocazione concatenata (1)

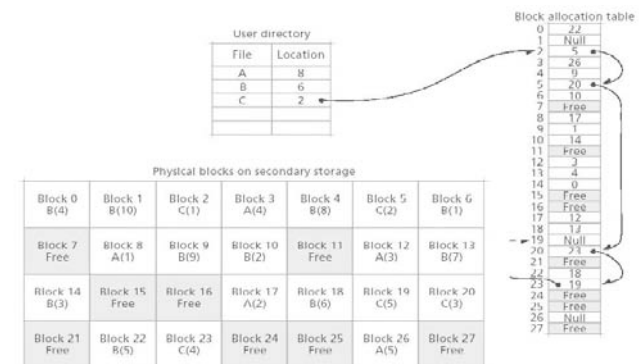
- Ciascun file è composto da una lista concatenata di blocchi che possono essere sparsi in qualsiasi punto del disco



## Allocazione concatenata (2)

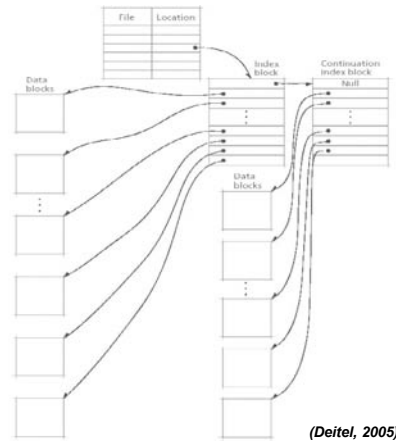
- E' semplice: necessita solo dell'indirizzo di partenza
- Sistema di gestione dello spazio libero: non c'è spreco di spazio
  - ma ogni blocco contiene dati + puntatore al prossimo blocco
  - problemi nel caricamento DMA
  - dimensione dei dati non gestibile come multiplo della dimensione del blocco
- Accesso casuale simulato con scansione
- Variante :tabella di allocazione dei file (FAT, *file allocation table*).
  - usato nei sistemi operativi MS-DOS e Windows

## File-Allocation Table



### Allocazione indicizzata (1)

- Raggruppa tutti i puntatori in una sola locazione: il blocco indice.



### Allocazione indicizzata (2)

- Necessita di una struttura supplementare per l'indice di ogni file
  - è auspicabile una piccola dimensione
  - se 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)
  - schema combinato (Unix, soluzione favorevole per file piccoli, consente file molto grandi)

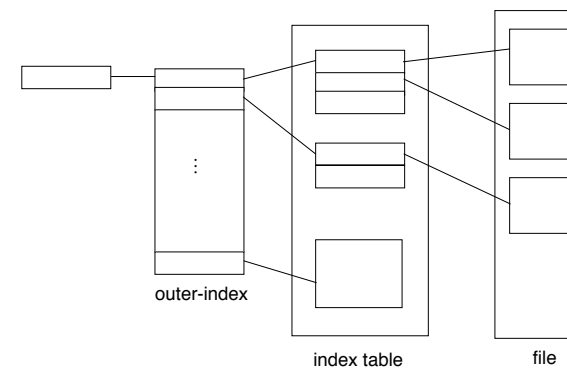
### Allocazione indicizzata (2)

- Corrispondenza tra indirizzo logico e indirizzo fisico
  - dipende dalle dimensioni del blocco di indice e dalle dimensioni massime del file
- Esempio
  - mapping da indirizzo logico a indirizzo fisico per un file di dimensione massima di 512K byte, 1024 byte per blocco fisico, 2 byte per puntatore
  - un blocco di indice (= blocco fisico) contiene 512 puntatori per indirizzare 512 blocchi

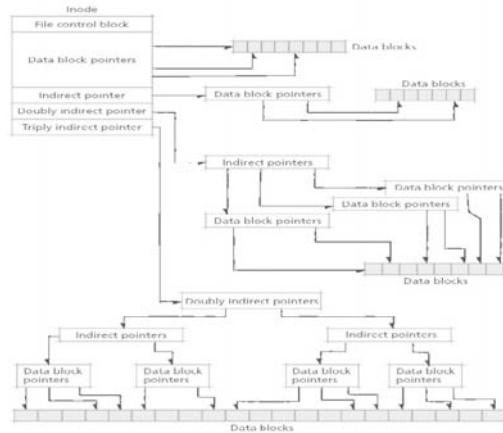
$$\frac{\text{Indirizzo Logico}}{\text{Dimensione blocco}} \begin{matrix} \nearrow Q \\ \searrow R \end{matrix}$$

Cluster di accesso = Q  
Offset nel blocco = R

### Allocazione indicizzata multilivello



### Schema combinato: UNIX

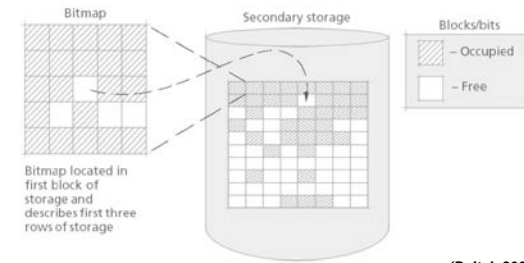


Augusto Celentano, Sistemi Operativi A

52

### Gestione dello spazio libero (1)

- Vettore di bit (n blocchi)
  - $bit[i] = 1$  se il blocco  $i$  è occupato
- Il numero del primo blocco libero è dato da:  
 $(\text{numero di bit per parola}) \times (\text{numero di parole a 0}) + \text{offset del primo bit 1}$



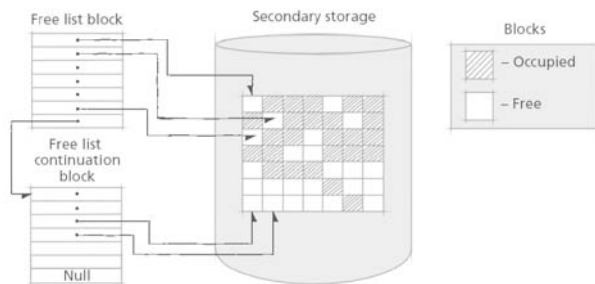
(Deitel, 2005)

Augusto Celentano, Sistemi Operativi A

53

### 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)

Augusto Celentano, Sistemi Operativi A

54

### Gestione dello spazio libero (3)

- La bitmap richiede spazio per la sua memorizzazione
  - esempio:
    - $\text{block size} = 2^{12} \text{ bytes (4Kbyte)}$
    - $\text{disk size} = 2^{36} \text{ bytes (64 Gbyte)}$
    - $n = 2^{36}/2^{12} = 2^{24} \text{ bits (2 Mbyte)}$
  - è facile trovare ampio spazio contiguo
- La lista concatenata (FAT) non richiede spazio supplementare
  - ma non è facile consultarla per trovare ampio spazio contiguo

Augusto Celentano, Sistemi Operativi A

55

## Tecniche per garantire la coerenza

- I meccanismi di caching possono causare inconsistenze nel file system
  - ad esempio, a causa di interruzioni di corrente
  - il problema è particolarmente critico per zone del disco contenenti strutture dati per la gestione: FAT, bitmap, i-node, directory
- Due possibili soluzioni:
  - curare (file system checker)
    - fsck, scandisk
  - prevenire (journaling file system)
    - ext3, ntfs, hfs+

## Journalled File System

- I file system con annotazione (*journaled file system*) tracciano gli aggiornamenti 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

## Journalled File System

- Le modifiche effettuate sono riportate in modo asincrono sul file system al termine della transazione e rimosse dal file di log
- Il giornale delle modifiche si potrebbe mantenere in una sezione separata del file system, o anche in un disco separato.
- Se si verifica un'interruzione nel funzionamento del sistema, tutte le transazioni nel log devono ancora essere eseguite

## Organizzazione dei dati sul disco

- Le strutture dati memorizzate sul disco devono contenere informazioni su
  - organizzazione logica del disco (partizioni)
  - eventuale avviamento del sistema operativo memorizzato nel disco
  - numero complessivo dei blocchi, numero e locazione dei blocchi liberi
  - struttura delle directory e dei file

## Gestione dell'unità a disco

- Un disco magnetico deve essere diviso in settori che possano essere letti o scritti dal controllore
- Per usare un disco come contenitore di informazioni, il sistema operativo deve registrare le proprie strutture dati all'interno del disco
  - suddividere il disco in uno o più gruppi (partizioni)
  - creare un file system (formattazione logica)
- 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 dei dati sul disco

- 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 il cosiddetto 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

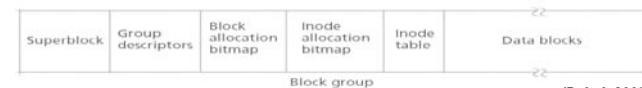
- Boot block
  - informazioni necessarie per avviare il s.o. da questa partizione
  - in NTFS: *partition boot sector*
- Superblock
  - informazioni sul tipo di file system e sui parametri fondamentali della sua organizzazione
    - numero e dimensione dei blocchi della partizione
    - dimensione massima 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

- Group descriptors
  - descrizione 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)
- Inode allocation map + inode 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; non vi si accede, ovviamente, tramite il FS
  - raw partition (partizione senza file system)
  - raw partition usata anche per l'area di swap dei processi

## Il File System di Unix (I)

- *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 dispositivi 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

## Il File System di Unix (2)

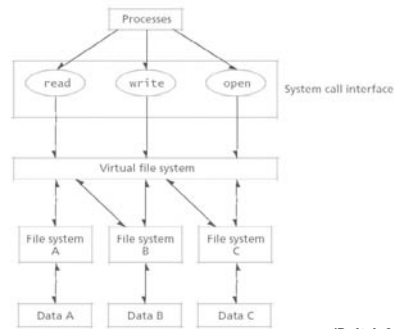
- In Unix tutti i file di dati sono omogenei
  - sequenze ordinate di byte
  - il formato logico è interpretato da programma (es. file di testo vs. file binari)
  - sono definite alcune convenzioni standard cui si appoggiano le funzioni e le librerie di I/O; es. I file di testo sono sequenze di linee terminate da un carattere *line feed*
  - il tipo di file system definisce l'organizzazione delle strutture dati del file system e non il formato dei file
    - ufs, ext, ext2, ext3
  - Unix può elaborare dati di altri file system attraverso appositi driver
    - msdos, ntfs, etc.

## Unix vs. Windows vs. VMS

- In Windows file binari e file di testo sono differenziati
  - i file di testo sono sequenze di record di lunghezza variabile terminati da una coppia di caratteri CR LF
  - il testo può essere codificato secondo diversi standard: ANSI, Unicode, etc.
- In VMS i file possono essere di tre tipi
  - ad accesso sequenziale, con record di lunghezza variabile: ogni record è preceduto da una coppia di byte che contengono la lunghezza
  - ad accesso sequenziale o diretto, con record di lunghezza fissa
  - ad accesso sequenziale o diretto al byte (stream): sono simili ai file Unix

## 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)



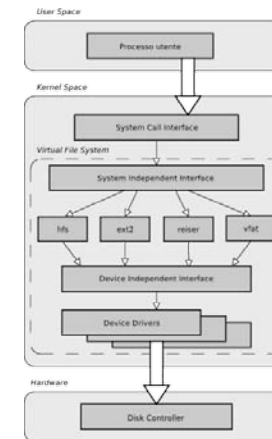
(Deitel, 2005)

Augusto Celentano, Sistemi Operativi A

68

## 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



Augusto Celentano, Sistemi Operativi A

69

## Linux Virtual File System (3)

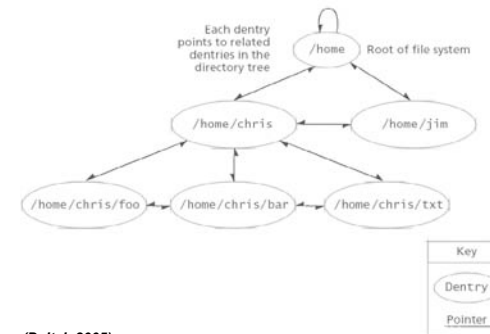
- VFS superblock
  - Contiene informazioni su un file system montato: tipo del file system, posizione su disco dell'inode root, informazioni contabili, eetc.
  - Esiste solo in memoria centrale, è creato quando il file system viene montato
- VFS inode
  - descrive la posizione di ogni file (directory, 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)

Augusto Celentano, Sistemi Operativi A

70

## Linux Virtual File System (4)

- Dentry (directory entry)
  - crea una corrispondenza tra descrittori e inode



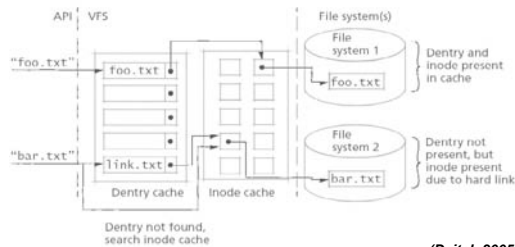
(Deitel, 2005)

Augusto Celentano, Sistemi Operativi A

71

## Linux Virtual File System (5)

- Dcache (directory entry cache)
  - cache di *dentry* corrispondenti a directory accedute recentemente
- Inode cache
  - contiene gli inode corrispondenti alle *dentry* in dcache
- Consentono di tradurre velocemente un pathname in un inode



(Deitel, 2005)

Augusto Celentano, Sistemi Operativi A

72

## Il file system /proc (1)

- Contiene informazioni aggiornate in tempo reale sullo stato del kernel e dei processi
  - consente di ottenere informazioni dettagliate su hardware, risorse e network attraverso operazioni su file
  - esiste solo in memoria centrale
  - le funzioni read e write possono accedere a dati di kernel
  - consente ad un utente di inviare dati al kernel

Augusto Celentano, Sistemi Operativi A

73

## Il file system /proc (2)

```
root> ls /proc
1 20535 20656 751 978 interrupts pci
10 20538 20657 792 acpi iomem self
137 20539 20658 8 asound ioports slabinfo
19902 20540 20696 811 buddyinfo irq stat
2 20572 20697 829 bus kcore swaps
20473 20576 20750 883 cmdline kmsg sys
20484 20577 3 9 cpuinfo ksyms sysvipc
20485 20578 4 919 crypto loadavg tty
20489 20579 469 940 devices locks uptime
20505 20581 5 960 dma meminfo version
20507 20583 536 961 dri misc vmstat
20522 20586 541 962 driver modules
20525 20587 561 963 execdomains mounts
20527 20591 589 964 filesystems mtrr
20529 20621 6 965 fs net
20534 20624 7 966 ide partitions
```

(Deitel, 2005)

Augusto Celentano, Sistemi Operativi A

74

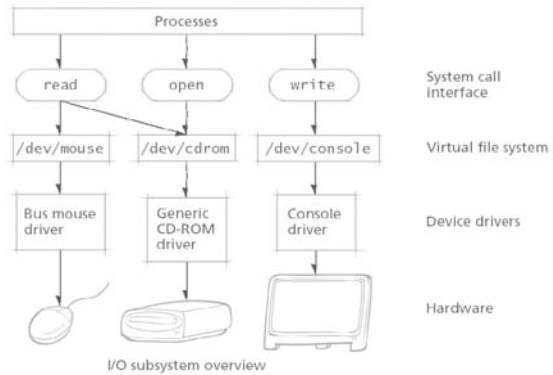
## Linux Device Driver (1)

- I dispositivi fisici sono organizzati in classi
  - i membri di ogni classe svolgono funzioni simili, ma ogni dispositivo può essere controllato in modo ottimo per quanto riguarda le prestazioni
- I device driver sono l'interfaccia software tra le system call e un dispositivo
- Device special files
  - la maggior parte dei dispositivi è implementata come file (speciale)
  - la directory /dev contiene i file corrispondenti ai dispositivi previsti
  - la directory /proc/devices contiene informazioni dettagliate sui dispositivi effettivi

Augusto Celentano, Sistemi Operativi A

75

## Linux Device Driver (2)



(Deitel, 2005)

## Linux Device Driver (2)

```
root> cat /proc/devices
Character devices:
 1 mem Physical memory access
 2 pty BSD-style terminal (TTY) devices
 3 tty Virtual console
 4 vc/%d Multiplexor for AT&T-style terminal (TTY) devices
 5 ptmx Parallel printer
 6 lp Virtual console capture devices
 7 vcs Non-serial mice, other devices
10 misc Input core (typically contains a mouse)
13 input Audio device
14 sound Advanced Linux Sound Driver
116 alsa AT&T-style terminal (TTY) devices
128 ptm USB device
136 pts Direct Rendering Manager (video card)
180 usb
226 drm

Block devices:
 2 fd Floppy disk drive
 3 ide0 Primary IDE channel
22 ide1 Secondary IDE channel
root>
```

(Deitel, 2005)