FILE SYSTEM

Visione dell'utente

- Esigenza di un ampio spazio per memorizzare informazioni a lungo termine ➔
  Uso di dispositivi di memoria di massa e memorizzazione delle informazioni come files

- I files hanno un nome (sempre)

- I files hanno un tipo (sovente): files normali <> directory
  ma anche files di testo, files eseguibili, files di immagini,....

- I files non fanno parte dello spazio di indirizzamento utente ➔
  si accede tramite primitive di sistema operativo ➔
  possono essere considerati un tipo di dato astratto su cui sono possibili operazioni come:
  - creazione
  - apertura
  - scrittura
  - lettura
  - posizionamento (***seek***)
  - cancellazione

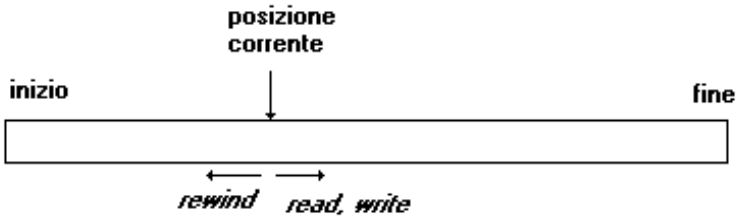
  ... ecc... ecc....

Metodi di accesso ai files

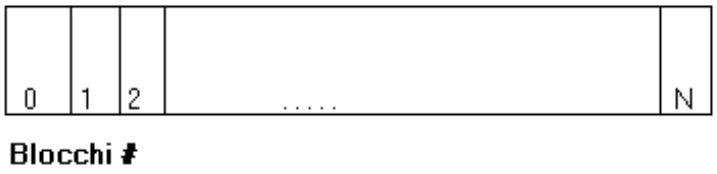
I files sono organizzati in modo diverso, consentendo quindi diverse modalita' di accesso:

- sequenziale
- diretto
- ad indice (indexed)

Accesso sequenziale:



Accesso diretto:



Si puo' simulare l'accesso sequenziale su files ad accesso diretto:

- `rewind` ➔ `current_pos =0;`
- `read` ➔ `read (current_pos); current_pos++;`
- `write` ➔ `write(current_pos); current_pos++;`

Accesso con indice:
si utilizza un indice associato ad una chiave ed eventualmente indici secondari.

Indipendenza dal device:

ottenuta in Unix: `cp abc /dev/tty`
operazione di `mount/umount`

meno chiara in altri sistemi:

Dos: `copy a:\pippo c:\windows\utenti\pippo`

Tipi di files:

in Unix: files normali
directories
files speciali (a blocchi o a caratteri)
pipes

Sui files normali e' possibile individuare con una estensione una particolare categoria di informazioni ivi contenute (es. file.c, file.o, file.exe, ...)

In Windows e' automaticamente associata ai files una applicazione per crearli/modificarli, che viene sempre attivata selezionando il corrispondente file:

- file.html➔explorer;
- file.doc➔word

ecc.

E' talvolta possibile usare altre informazioni per verifiche di consistenza

- data di creazione / modifica del file
- numero di versione

Directories

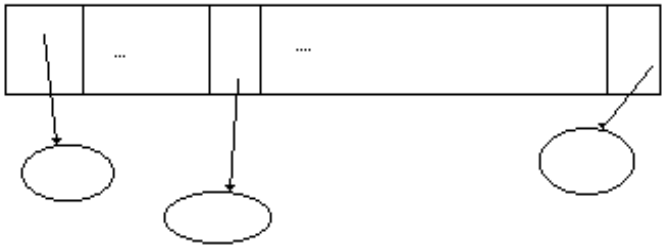
Le directories consentono di strutturare gerarchicamente il file system e quindi di rendere gestibili grandi quantita' di informazioni.
Una directory e' (generalmente) un file che contiene una entry per ogni file presente nella directory.
Tale entry puo' contenere diverse informazioni:

- file name
- file type
- file size
- owner
- protection
- creation date
- ...

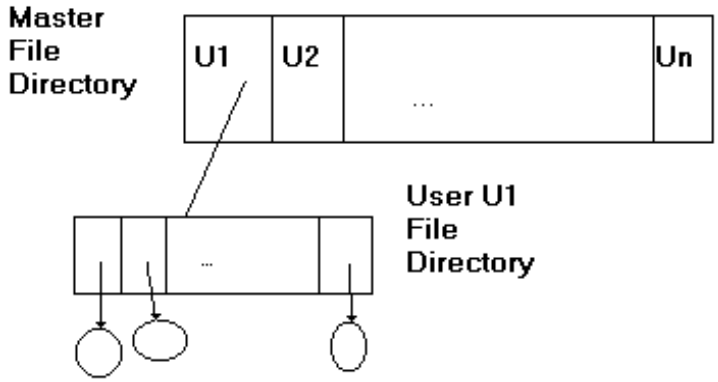
esempio: quelle che vengono listate con il comando `ls -l`
In piu conterra' informazioni necessarie per ricuperare il contenuto del file sul dispositivo periferico.

Strutturazione della directory:

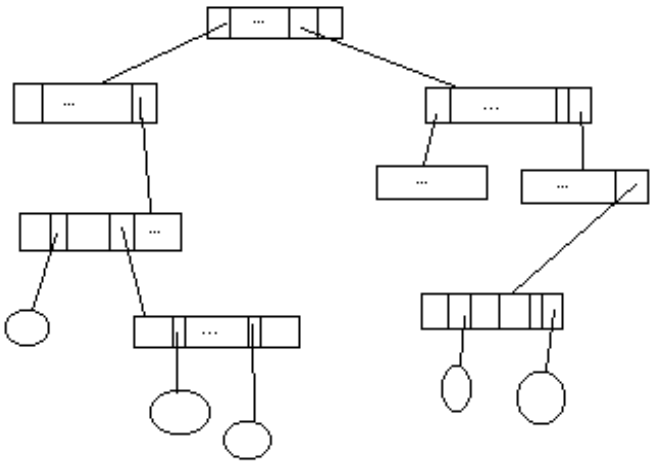
un solo livello



due livelli (una directory per ogni utente)



organizzazione ad albero



- root directory
- pathname assoluto e relativo
- working directory

organizzazione a grafo aciclico:
un file e' accessibile a partire da piu' directories: utilizzo di links

Progettazione di un file system: elementi fondamentali

- Gestione dello spazio su disco
  - blocchi di dimensione fissa (quale?)
  - come accedere allo spazio vuoto
- Memorizzazione dei files
  - contigua
  - con liste linkate
  - con indici
- Struttura delle directories
- Affidabilita' dei file systems
  - bad blocks
  - backups
  - consistenza
- Prestazioni
  - caching (tempo)
  - i-nodes (spazio)

Gestione dello spazio disco

Memorizzazione di un file di n bytes:

- allocazione di n bytes consecutivi

→ cosa succede se un altro file inizia al byte *n+1*esimo e uno dei due files deve "crescere?

- utilizzo di blocchi di dimensione fissa (non necessariamente consecutivi)

→ frammentazione interna (come per paginazione)

Dimensione di un blocco

- Cilindro
- Traccia
- Settore

Il cilindro potrebbe contenere 32KB → troppo grande!
Moltissimi files di Unix occupano 1K o meno: 31/32 di frammentazione (circa 97%)!!
Scegliere una unita' troppo piccola invece porta ad un *elevato tempo di accesso*:

Tempo necessario per leggere un blocco di K bytes (Ttras) =
SeekTime + RotationDelay + (K/ no.bytes per traccia) TransferDelay

Valori tipici :

- SeekTime 30 ms,
- RotationDelay <= 16.67 ms = TransferDelay
- no.bytes x traccia 32768

Quindi Ttras = 30 + 8.3 + (K/32768) 16.67 ms

Se K piccolo il data rate = K / Ttras risulta basso (vedi figura su Tanenbaum). Valori tipici per il blocco sono 512, 1K o 2K.

Elenco dei blocchi liberi: come organizzarlo

Come lista linkata:

- ogni blocco libero punta al successivo (poco sensato: obbligo di scandire la lista se si vuole ottenere parecchi blocchi)
- un blocco (due, tre…) contiene l'indirizzo degli altri blocchi liberi

Esempio:

blocchi di 1K e indirizzi di blocco lunghi 16 bit → un blocco contiene 511 indirizzi di blocchi (e il pt. al successivo) → in un disco da 20MB potrebbe occupare 20K indirizzi cioe' 40 blocchi!!

Come bitmap:

per ogni blocco mi serve un bit posto a 0 se libero, 1 se occupato.

Esempio:

per il disco da 20MB memorizzo 20K bit in soli tre blocchi!

La bitmap e' ancora piu' vantaggiosa se il sistema puo' mantenerla in memoria centrale (difficilmente possibile con la lista)

Memorizzazione dei files

Memorizzazione sequenziale dei blocchi: occorre trovare un "buco" sul disco in cui piazzare il file.

- Problema generale: frammentazione esterna
- Strategie di soluzione: *first/best/worst fit*. In questo caso worst fit e' la strategia peggiore, le altre sono equivalenti rispetto alla frammentazione e first e' anche leggermente piu' veloce di best.
- Problema ulteriore: necessita' di conoscere a priori la dimensione max dei files o, in alternativa, di *compattare* il disco (`DEFRAG`)
- Buone prestazioni per accesso diretto ai files

Linked list di blocchi: ogni blocco contiene, ad esempio,

- 1022 bytes di dati
- 2 bytes di indirizzo del blocco successivo

In questo caso la dimensione del blocco non e' piu' una potenza di 2, e per accesso diretto ad un generico byte occorre accedere comunque anche a tutti i blocchi precedenti

Esempio:

per accedere al byte 32768 occorre calcolare 32768/1022 = 33 quindi per avere l'indirizzo del 33-mo blocco si devono scandire i primi 32 blocchi.

Metodo alternativo per memorizzare una linked list di blocchi:

la FAT del DOS.

FAT= File Allocation Table

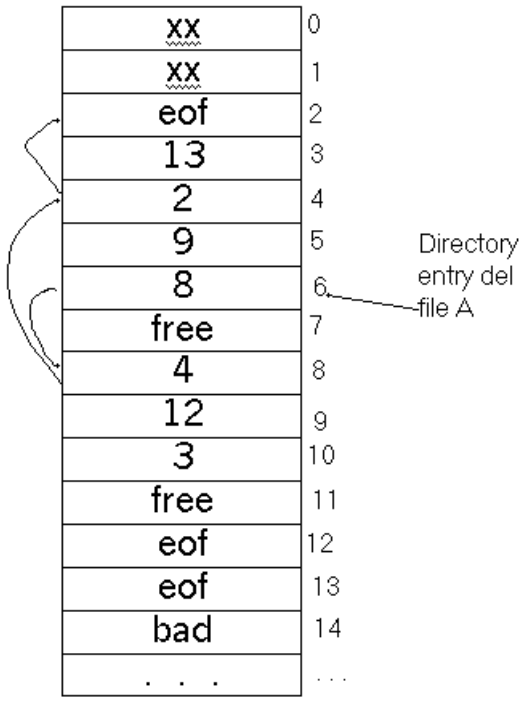
- ogni disco ne ha una, e se il disco e' utilizzato viene anche ricopiata in memoria
- per ogni file la directory contiene l'indirizzo del *primo* blocco del file
- ogni blocco del disco ha una entry nella FAT, che in generale contiene l'indirizzo del blocco *successivo*.

Esempio:

La directory corrente contiene tre files, A, B, C.
Supponiamo che A occupi nell'ordine i blocchi 6,8,4,2;
B occupi 5,9,12;
C occupi 10,3,13.
La directory sara' cosi' organizzata (semplificando):

Nome del file	*Primo blocco*
A	6
B	5
C	10

La FAT sara':



Quanto occupa una FAT? Troppo?!

Esempio 1:

floppy da 320 KB, blocchi di 1KB, indirizzi di blocco a 12 bit

➔ la FAT occupa 480 bytes per 320 entries quindi 1 settore del floppy

Esempio 2:

floppy da 360 KB ➔ la FAT occupa 540 bytes quindi 2 settori

Esempio 3:

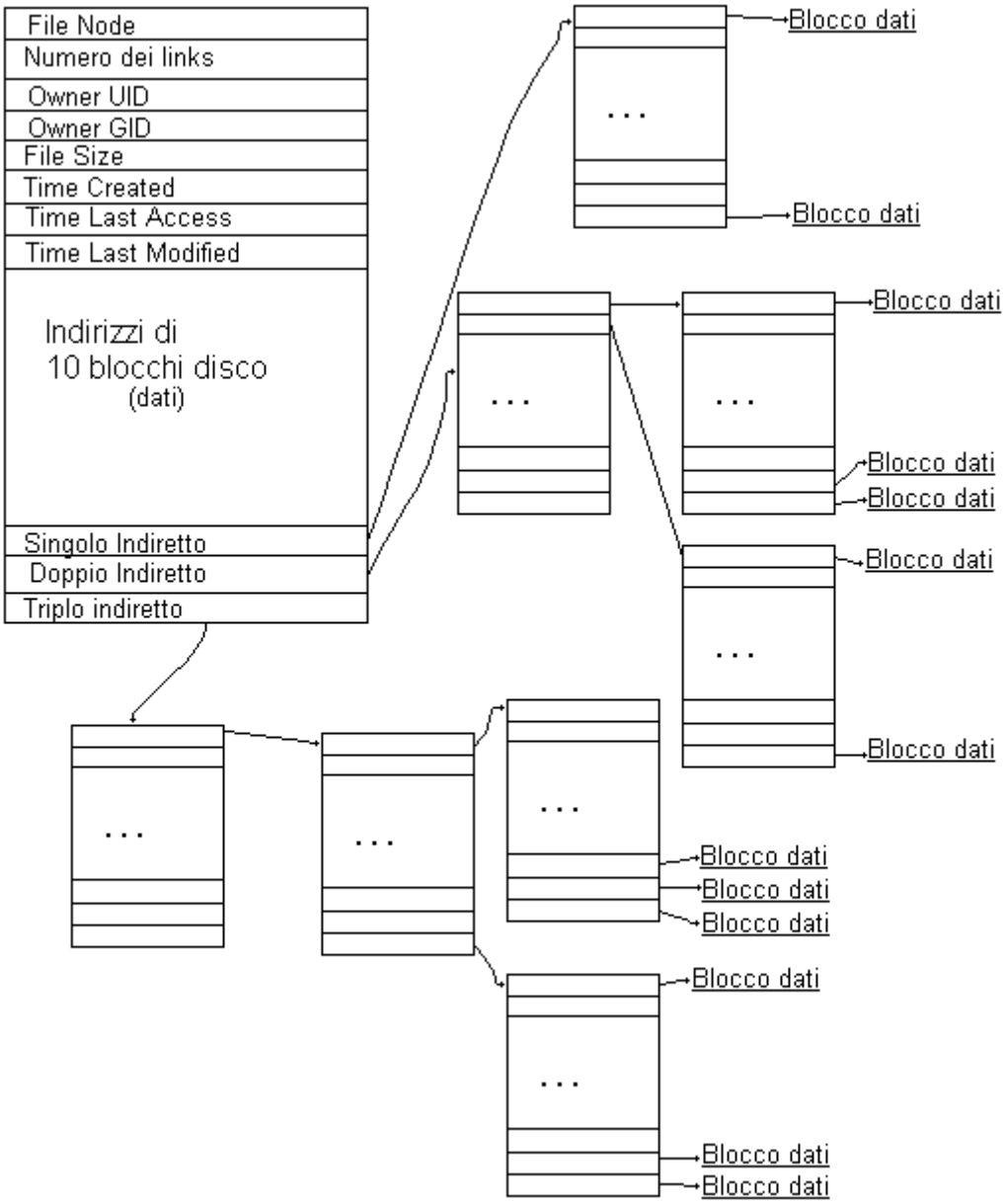
hard disk di piu' di 4096 blocchi ➔ indirizzi di blocco a 16 bit, per 64 MB di blocchi di 1K (cioe' 64K blocchi) la FAT occupa 128 KB

Esempio 4:

hard disk da 70MB ➔ indirizzo a piu' di 16 bit, la dimensione della FAT aumenta ancora!

Il problema e' dovuto al fatto che si mantengono tutte assieme le informazioni di tutti i files.

UNIX mantiene invece una struttura di *indice* associata ad ogni file, detta *I-node:*



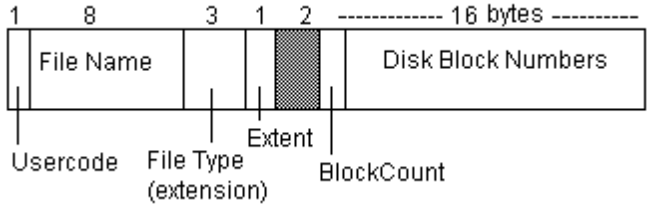
L'I-node indirizza direttamente i primi 10 blocchi componenti il file. Per files piu' lunghi si usano gli ultimi tre puntatori:

- con il *single indirect* si indirizzano altri 256 blocchi (totale 10+256=266)
- con il *double indirect* si indirizzano altri 256 * 256 blocchi (totale 65802)
- con il *triple indirect* si arriva fino a 16 gigabytes circa.

Organizzazione delle directories

Tale organizzazione deve consentire di accedere al meglio a tutti i blocchi componenti i files una volta noto il nome (o il path name).

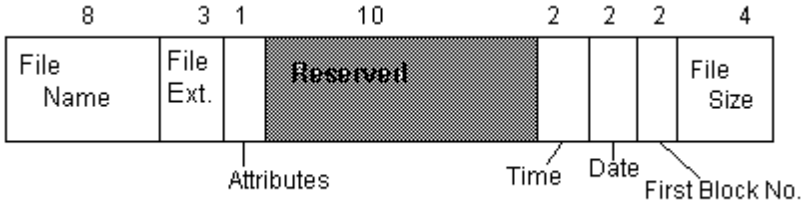
CP/M: sistema piu' semplice: *una sola directory* che contiene una entry per ogni file cosi` fatta:



- Usercode: codice dell'utente
- Extent: se il file e' piu' lungo di 16 blocchi, e' l'indirizzo della entry usata per memorizzare gli indirizzi dei blocchi successivi
- Block Count: numero di blocchi del file

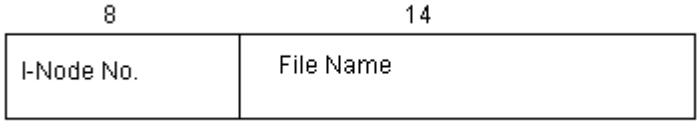
Il numero max di files e' 256 ma anche di meno, se si usano files lunghi che richiedono un extent.

DOS: organizzazione gerarchica delle directory, ciascuna entry e' di 32 bytes ed e' cosi' organizzata:



- First Block No. e` il puntatore alla FAT che individua il primo blocco del file
- La *root directory* ha dimensione fissa per ogni periferico (per floppi da 360K sono 112 entries)
- Le directories sotto root sono files di dimensione arbitraria e quindi possono contenere anche un maggior numero di files

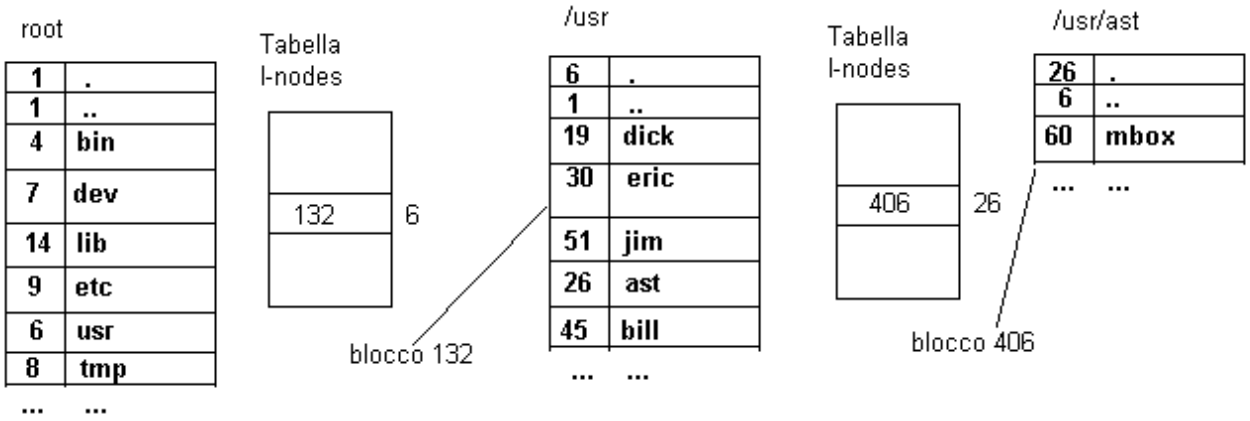
Unix: organizzazione gerarchica di directories, ciascuna di esse e' un file contenente un numero arbitrario di entries.
Ogni entry e' cosi' fatta:



E' una organizzazione particolarmente semplice perche' tutte le informazioni sul file sono contenute nell'I-Node.

Risoluzione di un path name gerarchico (Unix)

Si consideri il path `usr/ast/mbox`, si vuole individuare l'I-Node di tale file (e quindi il suo contenuto) secondo il seguente percorso:



Dalla directory `/usr/ast` si ricava che l'I-Node di `mbox` e' il numero 60.