

Argomenti

- FILE
 - Nomi, Struttura, Tipi, Accesso, Attributi, Operazioni
 - Un esempio UNIX di chiamata di sistema del file system
- DIRECTORY
 - Sistemi di directory, Path name, Operazioni
- REALIZZAZIONE DEL FILE SYSTEM
 - Layout del file system
 - Realizzazione dei file
 - Realizzazione delle directory

FILE SYSTEM

- Tutte le applicazioni informatiche hanno bisogno di memorizzare e recuperare le informazioni.
- Mentre un processo è in esecuzione, può memorizzare una quantità limitata di informazioni all'interno del proprio spazio di indirizzamento.
- **Primo problema:** per alcune applicazioni questo spazio è sufficiente, ma per gli altre (come ad esempio un sistema bancario) è troppo piccolo.
- **Secondo problema:** quando il processo termina, le informazioni scritte all'interno del suo spazio di indirizzamento sono perdute.
 - Per molte applicazioni le informazioni devono invece essere conservate per settimane, mesi o per sempre ... anche quando un crash del computer uccide il processo!

FILE SYSTEM

- I dischi magnetici sono stati utilizzati per anni per conservare i dati a lungo termine.
- Anche i nastri e i dischi ottici sono stati utilizzati, ma hanno prestazioni molto più basse.
- Un disco contiene una sequenza lineare di blocchi di dimensione fissa e supporta due operazioni:
 - 1) Leggi blocco k.
 - 2) Scrivi blocco k.
- Queste due operazioni non sono molto comode quando si ha a che fare con grandi ambienti che sono utilizzati da molte applicazioni e utenti concorrenti.

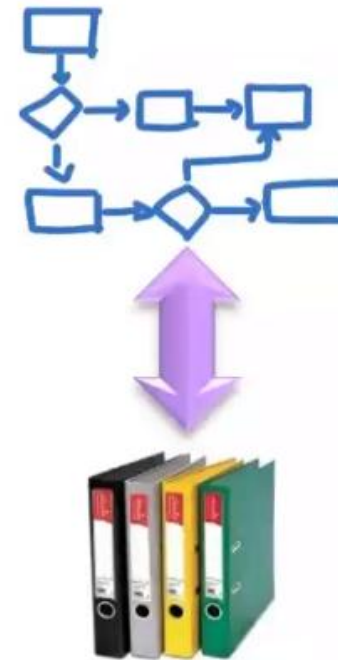
FILE SYSTEM

- Alcune delle domande:
 - Come si fa a trovare le informazioni?
 - Come si fa ad evitare che un utente legga i dati di un altro utente?
 - Come fai a sapere quali blocchi sono liberi?
- Il sistema operativo può risolvere questa complessità con una nuova astrazione: il **file**.
- In un sistema operativo le principali tre astrazioni sono:
 - 1) **Processi e Thread.**
 - 2) **Spazio di indirizzamento.**
 - 3) **File.**



FILE SYSTEM

- I file sono unità logiche di informazioni creato dai processi.
- I processi possono creare file e leggere quelli esistenti.
- Le informazioni memorizzate nei file sono persistenti e non dipendono dalla vita processo.
- Un file dovrebbe scomparire solo quando il suo proprietario lo rimuove.
- I file vengono gestiti dal sistema operativo.
- I principali argomenti nella progettazione di un sistema operativo sono le caratteristiche dei file: struttura, denominazione, accesso, utilizzo, protezione, realizzazione e gestione.



Nomi dei file

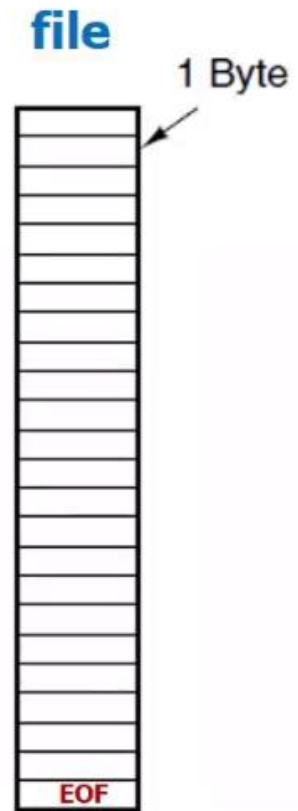
- Una delle caratteristiche più importanti di ogni meccanismo di astrazione è il modo in cui gli oggetti sono denominati.
- Le regole che devono essere adottate per i nomi dei file variano da sistema a sistema, ma tutti considerano nomi validi sequenze fino a 8 caratteri (ad esclusione dei meta caratteri usati dal file system: ~ " # % & * : < > ? / \ { | } ...).
- Frequentemente sono ammessi anche i caratteri speciali (!) o accentati (à è ì ò é ù ...).
- Molti file system supportano nomi fino al massimo di 255 caratteri, alcuni distinguono tra lettere maiuscole e minuscole (UNIX), altri no (MS-DOS).
- Molti sistemi operativi adottano la convenzione di separare il nome del file dal suo tipo (**estensione del file**).
- Nel MS-DOS, i nomi dei file erano lunghi al massimo 8 caratteri e l'estensione opzionale era al massimo di 3 caratteri.

Struttura dei file

- Le strutture di file più comuni sono:

1) Sequenza di byte (senza struttura)

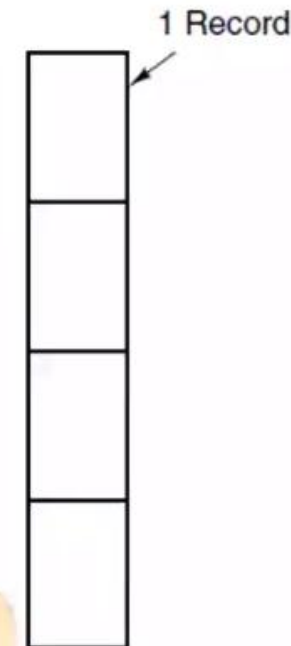
- Il sistema operativo non sa che cosa è contenuto nel **file**: vede solo una sequenza di byte.
- Massima flessibilità**: i programmi utente possono mettere tutto ciò che vogliono nei loro file.
- Tutte le versioni di UNIX, MS-DOS e Windows utilizzano questo modello di file.



Struttura dei file

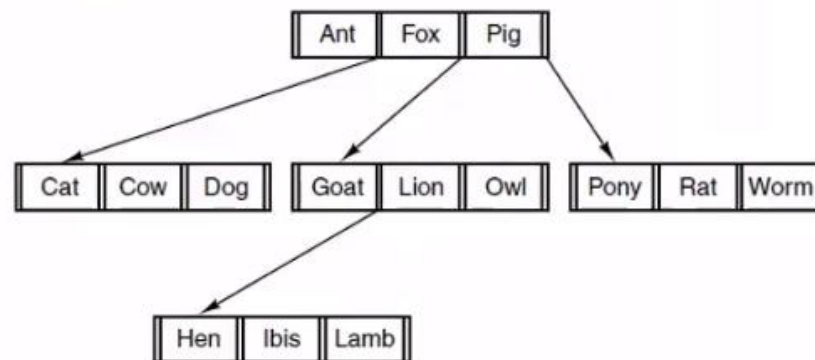
2) Sequenza di record.

- Il sistema operativo vede una sequenza di record di lunghezza fissa e con una struttura ben definita.
- Le operazioni di lettura/scrittura avvengono sui record.
- I primi mainframe utilizzavano record da 80 caratteri così come le 80 colonne di una scheda perforata:



Struttura dei file

- 3) **Albero di record**, non necessariamente tutti della stessa lunghezza, ciascuno contenente un **campo chiave** in posizione fissa nel record.
- L'albero è ordinato sul campo chiave, per velocizzare le ricerche sul campo chiave.
 - L'operazione base non è di ottenere il record successivo, ma rintracciare il record con una determinata chiave.
 - Questo modello è ancora utilizzato sui mainframe in alcune applicazioni commerciali.



Tipi di file

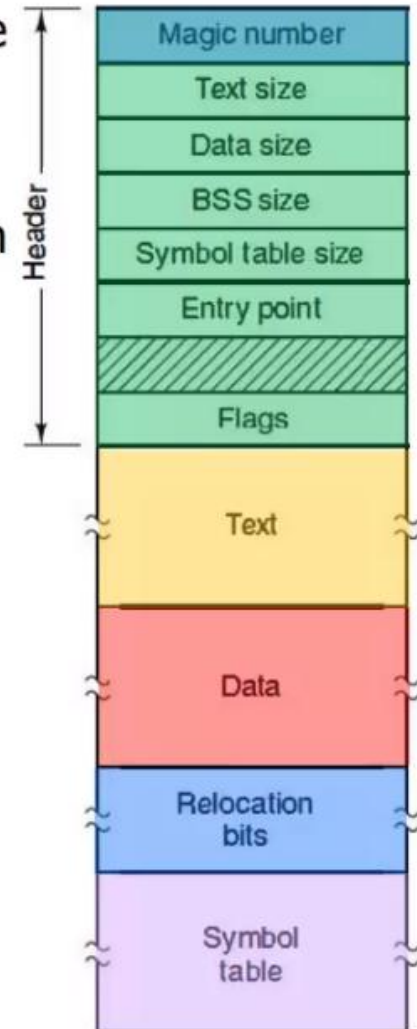
- Molti sistemi operativi supportano diversi tipi di file.
- UNIX e Windows supportano **file** e **directory normali**.
- UNIX ha anche **file speciali** a caratteri e a blocchi.
- I **file normali** sono quelli che contengono i dati degli utenti.
- Le **directory** sono file di sistema usati per mantenere la struttura del file system.
- I **file speciali a caratteri** sono legati all'input/output e sono utilizzati per modellare i dispositivi di I/O seriali come terminali, stampanti e reti.
- I **file speciali blocchi** sono utilizzati per modellare i dischi.

File normali

- I **file normali** sono generalmente ASCII o binari.
- I file ASCII consistono di righe di testo, non necessariamente della stessa lunghezza.
 - In alcuni sistemi ogni riga termina con un carattere speciale (Carriage Return=`chr(13)` e/o Line Feed`chr(10)`).
- I file ASCII sono facili da visualizzare e stampare e possono essere aggiornati con un semplice editor di testo.
- I programmi che utilizzano file ASCII per l'input/output possono essere interfacciati con facilità (es. pipe).
- I file binari sono sequenze di bit (non interpretati come caratteri) e hanno una struttura interna nota soltanto ai programmi che li utilizzano.
 - Non possono essere stampati e maneggiati da editor di testi.

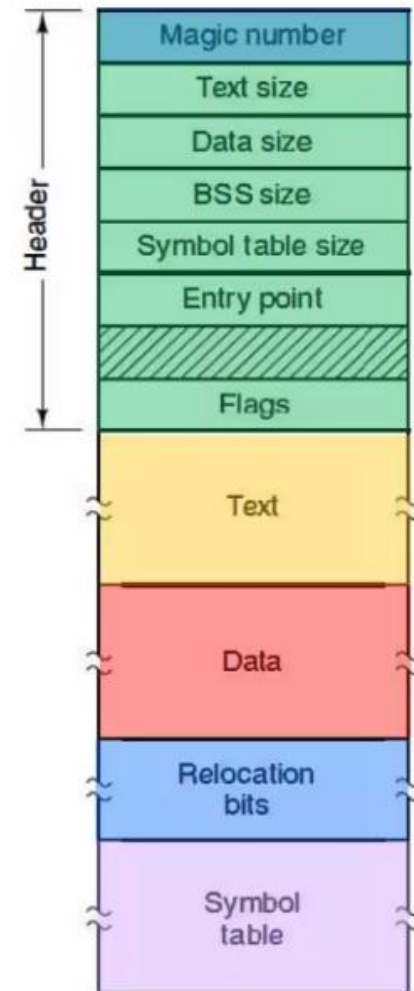
Un esempio di file binario eseguibile

- Questo esempio mostra il formato di un file binario eseguibile nelle prime versioni di UNIX.
- Il sistema operativo esegue solo i file con un formato specifico:
 - **Intestazione.**
 - **Testo.**
 - **Dati.**
 - **Bit di rilocalizzazione.**
 - **Tabella dei simboli.**



Un esempio di file binario eseguibile

- L'**Intestazione** inizia con un identificatore per il file eseguibile (**numero magico**) seguito da:
 - Le dimensioni dei segmenti che compongono il file (**Testo**, **Dati**, **BSS** **Tabella dei simboli**).
 - L'indirizzo di inizio del programma (**Entry point**).
 - Alcuni Flag.



Accesso ai file

- I primi sistemi operativi avevano un solo tipo di accesso ai file: l'**accesso sequenziale**.
 - i byte vengono letti nell'ordine in cui sono scritti, cominciando dall'inizio.
- I file sequenziali erano perfetti quando l'archiviazione era su nastro magnetico piuttosto che su disco.
- Con i dischi è possibile leggere i byte o i record in qualsiasi ordine (da cui file ad accesso casuale) o accedere ai record tramite una chiave.
- Il metodo **read()** permette di leggere i dati a partire da una posizione specifica nel file.
- Il metodo **seek()** permette di impostare la posizione corrente nel file in modo da poterlo poi leggere sequenzialmente da quella posizione.

Attributi dei file

- Ogni file ha un nome, un contenuto (i dati) e alcuni **attributi (meta-dati)**: il timestamp di creazione, la dimensione, gli utenti autorizzati ad accedere al file, ...
- L'elenco degli attributi varia da sistema a sistema, di seguito una tabella dei più comuni:

Attributo	Significato
Protezione	Chi può leggere, eseguire e scrivere il file.
Password	Password necessaria per accedere al file.
Creator ID	IDentificatore dell'utente che ha creato il file.
Owner	Proprietario attuale.
Read-only flag	0 = Read/Write, 1 = Read-Only.
Hidden flag	0 = normale, 1 = non visualizzabile in elenco.
System flag	0 = file normale, 1 = file di sistema.
Archive flag	0 = è stato effettuato il backup, 1 = deve essere effettuato il backup.

Attributi dei file

Attributo	Significato
ASCII/binary flag	0 = file ASCII file, 1 = file binario.
Random access flag	0 = solo accesso sequenziale, 1 = accesso casuale.
Temporary flag	0 = normale, 1 = file rimosso con la terminazione del processo.
Lock flag	0 = unlocked, 1 = locked.
Lunghezza del record	Dimensione del record in byte.
Posizione della chiave	Offset della chiave all'interno di ciascun record.
Lunghezza della chiave	Dimensione della chiave in byte.
Tempo di creazione	Timestamp (data e orario) quando il file è stato creato.
Ultimo accesso	Timestamp (data e orario) relativo all'ultimo accesso al file.
Ultima modifica	Timestamp (data e orario) relativo all'ultima modifica del file.
Dimensione corrente	Dimensione del file espresso in byte.
Dimensione massima	Massimo numero di byte che il file può contenere.

Operazioni sui file

- I **file** nascono per memorizzare informazioni e per poterle rileggerle in un secondo momento.
- Esistono varie operazioni disponibili nei sistemi operativi, le chiamate di sistema più comuni sono:
 - **create()**, crea un file vuoto con alcuni attributi impostati.
 - **delete()**, rimuove il file e libera lo spazio sul disco.
 - **open()**, prima di poter utilizzare un file, un processo deve aprirlo. La chiamata di sistema memorizza in memoria gli attributi e l'elenco degli indirizzi sul disco per un accesso rapido nelle successive chiamate.
 - **close()**, chiude il file liberando la memoria dai riferimenti al file perché non verrà più utilizzato. Molti sistemi incoraggiano l'uso della chiamata imponendo un numero massimo di file aperti per i processi. Poiché il disco è scritto a blocchi, la chiusura di un file forza la scrittura del ultimo blocco del file.

Operazioni sui file

- **read()**, legge i dati dal file. Solitamente, i byte vengono letti dalla posizione corrente, occorre specificare il numero di dati da leggere e il buffer in memoria ove inserirli.
- **write()**, scrive i dati nel file, dalla posizione corrente.
- **append()**, aggiunge dati alla fine del file.
- **seek()**, cambia la posizione corrente nel file ad accesso casuale. Dopo questa chiamata di sistema, i dati possono essere letti o scritti a partire dalla nuova posizione.
- **stat()**, in UNIX viene utilizzato per leggere gli attributi dei file.
- **rename()**, che viene utilizzato per rinominare un file.

Un esempio di system call del file system

- Esamineremo un semplice programma UNIX che effettua la copia di un file.
- Il programma deve essere richiamato con il comando shell:
`copiaFile sorgente destinazione`
- Il programma `copiaFile` ricopia il file «`sorgente`» in quello «`destinazione`».
 - Se «`destinazione`» esiste già, sarà sovrascritto.
- Il programma deve essere richiamato con due parametri: nomi ammessi per i due file.
 - Il primo parametro è di input mentre il secondo è di output.

Un esempio di system call del file system

- Il programma può essere richiamato così:



- La variabile **argc** conta il numero degli argomenti sulla linea di comando incluso il nome del programma.
- argv** è un array di puntatori agli argomenti.
- Il programma utilizzerà:
 - una matrice di 4096 caratteri per leggere e scrivere un blocco di 4KB dal file (**buffer**).
 - Due riferimenti ai file (**descrittori dei file**) **in_fd**, per il file di input, e **out_fd**, per il file di output.
 - byte_letti** e **byte_scritti** memorizzano il numero di byte letti e scritti.

Un esempio di system call del file system

```
#include <sys/types.h>
#include <fcntl.h>
#include <Stdlib.h>
#include <Unistd.h>
int main (int argc, char *argv[]);
#define BUF_SIZE 4096
#define OUTPUT_MODE 0700
int main (int argc, char *argv[]) {
    int in_fd, out_fd, byte_letti, byte_scritti;
    char buffer[BUF_SIZE];
    if (argc != 3) exit(1);
    in_fd = open (argv[1], O_RDONLY);
    if ( in_fd < 0 ) exit(2);
    ou_fd = create (argv[2], OUTPUT_MODE);
    if ( out_fd < 0 ) exit(3);
    while (1) {
        byte_letti = read(in_fd, buffer, BUF_SIZE);
        if (byte_letti <= 0) break;
        byte_scritti = write (out_fd, buffer, byte_letti);
        if (byte_scritti <= 0) exit(4);
    }
    close(in_fd);
    close(ou_fd);
    if (byte_letti == 0)    exit(0);
    else                  exit(5);
}
```

```
/* Include i file di intestazione necessari */

/* prototipo ANSI del main() */
/* Utilizza un buffer di 4096 byte */
/* protezione per il file di output 111000000 */

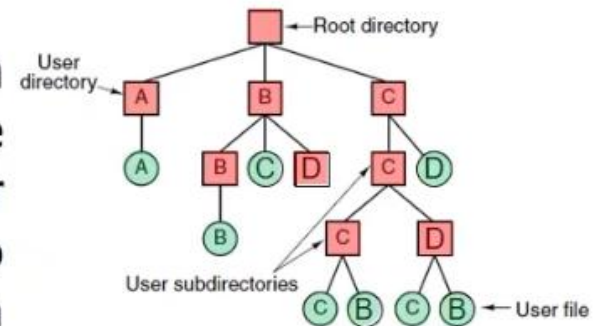
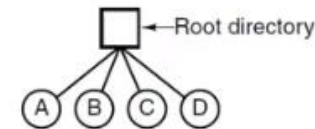
/* Errore di sintassi se non argv è diverso da 3 */
/* Apre il file sorgente */
/* Se non può essere aperto, uscita con errore */
/* Crea il file di destinazione */
/* Se non può essere creato, uscita con errore */
/* Ciclo per la copia */
/* Prova a leggere un blocco di 4KB */
/* Non è riuscito a leggere byte, esce dal ciclo */
/* Ora prova a scrivere i byte letti sul file */
/* Non ha scritto alcun byte, errore */

/* Chiude i due file */

/* l'ultima lettura è ok, uscita senza errori */
/* Errore nell'ultima lettura, uscita con errore */
```

DIRECTORY

- I file system normalmente organizzano i file in contenitori denominati **directory** o cartelle.
- La forma più semplice è la **directory principale** (o **root directory**): una sola directory (chiamata root) contenente tutti i file.
- L'adozione di una definizione ricorsiva in cui ogni directory può contenere altre directory (o file), con il vincolo che per ciascuna di esse esista sempre un solo contenitore (o genitore), conduce ad una struttura gerarchica (i file sono le foglie dell'albero).
- Gli utenti possono avere una directory di inizio (**home**) della propria gerarchia.

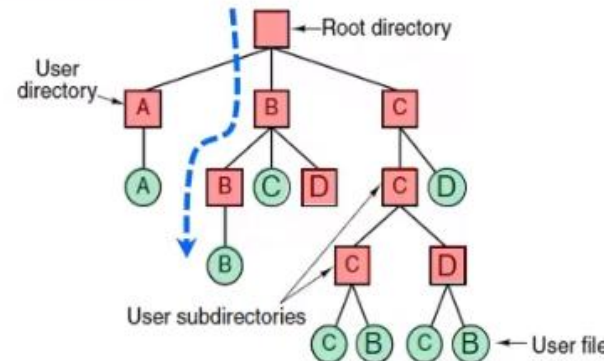


Path name

- Quando il file system è organizzato come un albero per identificare in modo univoco ogni oggetti della struttura (directory e file) sono utilizzati due metodi:
- **path name assoluto**: ogni file (o directory) viene identificato dal percorso necessario per raggiungerlo, a partire dalla directory principale. Il percorso è composto dalla sequenza ordinata di directory, separate da un carattere speciale: «/» in UNIX, «\» in Windows, «>» in MULTICS.

- Esempi:

UNIX	/B/B/B
Windows	\B\B\B
MULTICS	>B>B>B



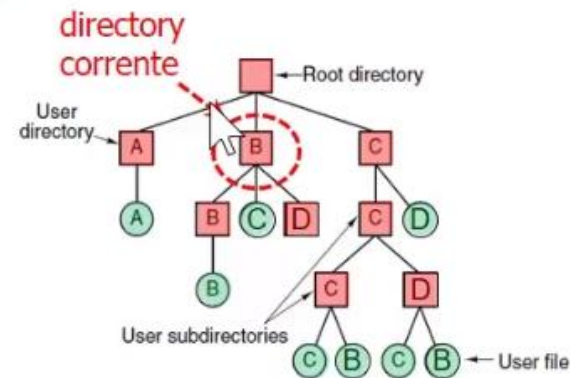
- Se il primo carattere del nome percorso è il separatore, allora il riferimento è assoluto.

Path names

- **path name relativo**: il file o la directory viene individuato utilizzando un punto relativo sull'albero (la **directory corrente**, la **home** dell'utente, ...).

- Esempi:

UNIX B/B
Windows B\B
MULTICS B>B

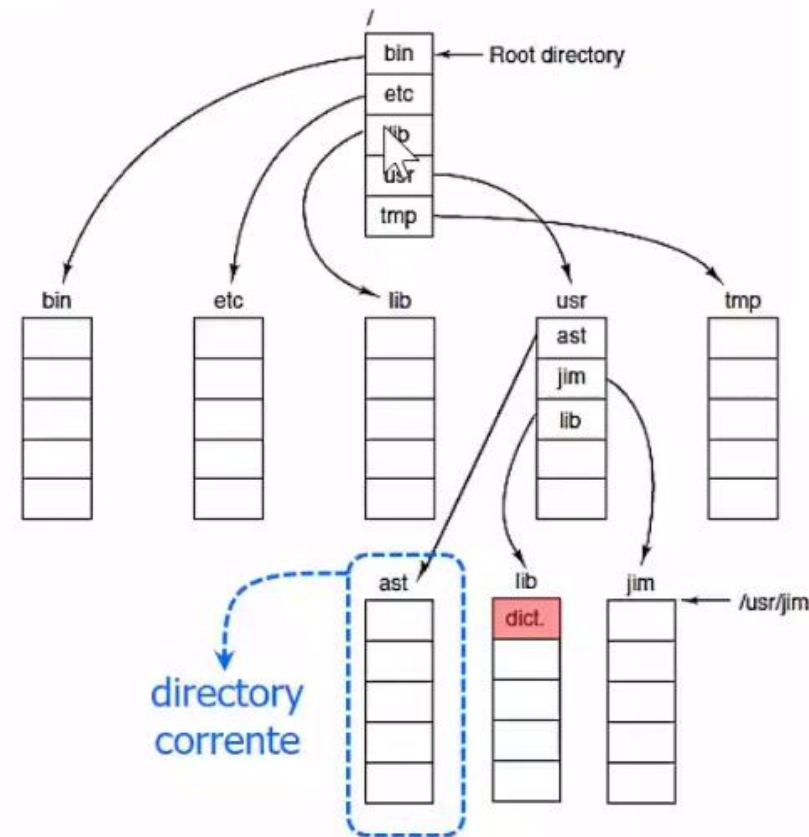


- La maggior parte dei sistemi operativi hanno in ciascuna directory due file speciali:
 - «.» è la directory corrente.
 - «..» è il suo genitore (tranne per root che fa riferimento a se stesso).

Esempi di comandi UNIX

- Con il seguente comando di shell il file `/usr/lib/dictionary` è copiato nella directory corrente (`/usr/ast`):

```
cp ../lib/dictionary .
```



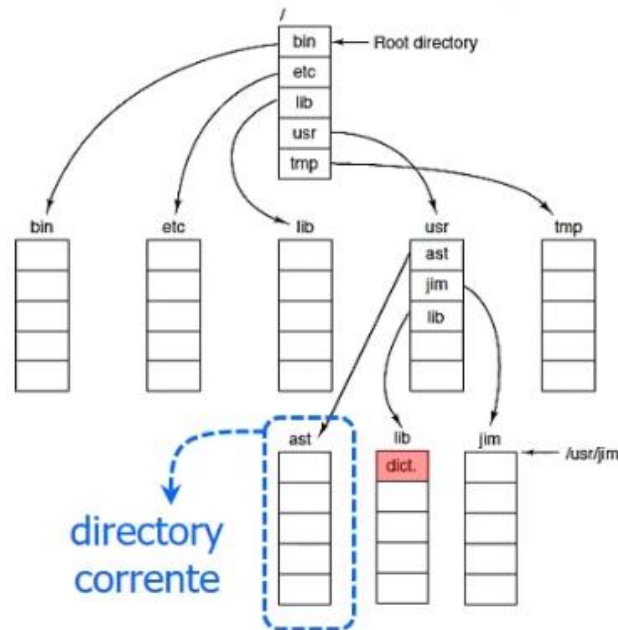
Esempi di comandi UNIX

- Lo stesso risultato può essere ottenuto attraverso i comandi:


`cp /usr/lib/dictionary .`

`cp /usr/lib/dictionary dictionary`

`cp /usr/lib/dictionary /usr/ast/dictionary` 



Operazioni sulle directory

- Le chiamate di sistema per la gestione delle directory hanno qualche variante da sistema a sistema rispetto a quelle per la gestione dei file.
- In UNIX:
 - **mkdir()**, crea una directory vuota (ad eccezione di «.» e «..» che sono automaticamente inseriti). 
 - **rmdir()**, rimuove una directory vuota (deve contenere esclusivamente «.» e «..»).
 - **opendir()**, carica in memoria tutti i riferimenti di localizzazione sul disco prima della lettura (così come avviene per i file).
 - **readdir()**, restituisce la successiva voce di una directory aperta.
 - **closedir()**, chiude la directory al termine della lettura e libera il corrispondente spazio in memoria.

Operazioni sulle directory

- **rename()**, rinomina una directory esistente.
- **link()**, crea un **link hard** tra un file esistente e il pathname indicato.

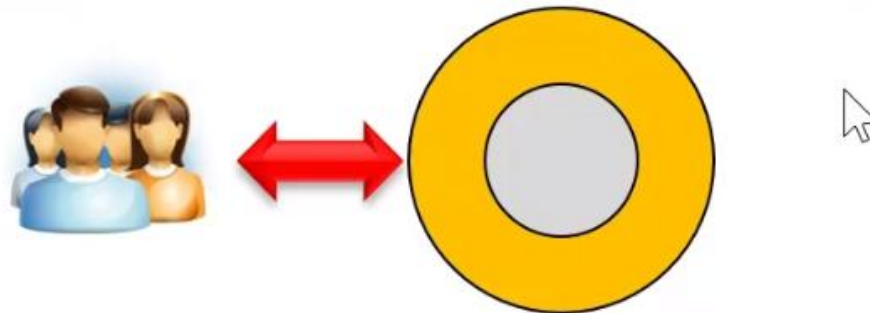
È una tecnica che permette ad un file di trovarsi in più di una directory. Si tratta di un **link hard** poiché il file system incrementa il contatore nel **i-node** del file (in questo modo si tiene traccia del numero di directory che contengono il file).

I **link simbolici** possono invece attraversare i confini del disco (computer remoti) ma sono meno efficienti dei **link hard**.

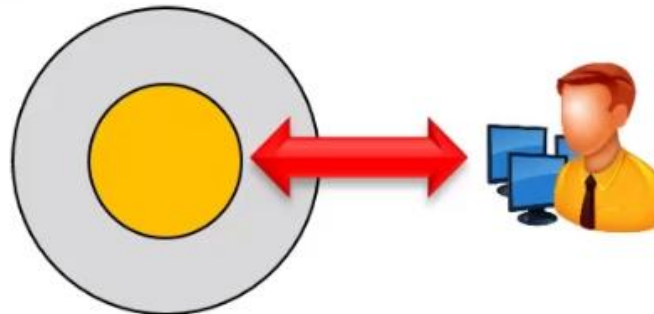
- **unlink()**, rimuove il collegamento nella directory, se il file è unico è cancellato dal file system altrimenti è rimosso solo un collegamento.

REALIZZAZIONE DEL FILE SYSTEM

- Agli utenti interessa solo il modo con cui richiamare i file, le operazioni disponibili, com'è fatto l'albero delle directory, e altri problemi simili che hanno a che fare con l'interfaccia.

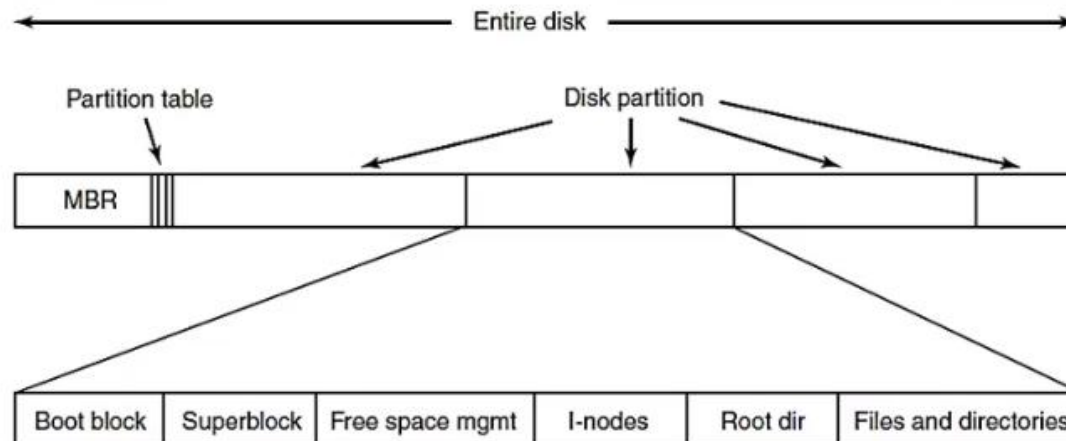


- Agli sviluppatori interessa il modo in cui file e directory sono memorizzati e come è gestito lo spazio sul disco.



Layout del file system

- I file system sono memorizzati sui dischi.
- La maggior parte dei dischi possono essere divisi in una o più partizioni, con file system indipendenti su ogni partizione.
- **Settore 0** del disco è chiamato **MBR** (**Master Boot Record**) e viene utilizzato per avviare il computer.
- La fine del **MBR** contiene la **tabella delle partizioni**: dove sono indicati l'inizio e la fine di ogni partizione.
- Nella tabella una sola partizione è contrassegnata come attiva.

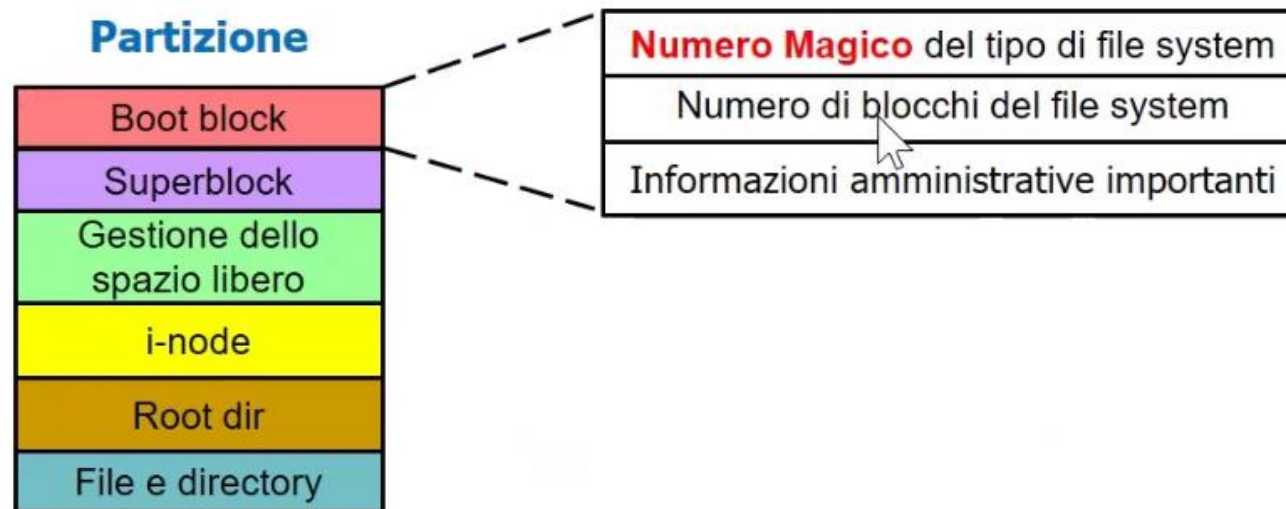


Layout del file system

- Quando il computer è avviato, il BIOS legge ed esegue il **MBR**.
- **MBR** localizza la partizione attiva, legge ed esegue il primo blocco (**blocco di boot** o **boot block**) sulla partizione.
- Il programma nel **blocco di boot** carica il sistema operativo contenuto nella partizione.
- Ogni partizione inizia con un **blocco di boot**, anche se non contiene un sistema operativo avviabile perché potrebbe contenere uno in futuro.

Il contenuto della partizione

- Anche se in generale il contenuto delle partizioni cambia da file system a file system, uno schema ricorrente è:



- Il **blocco di boot** contiene tutti i parametri fondamentali relativi al file system, viene letto in memoria quando il computer è avviato o il file system viene usato per la prima volta.

Realizzazione dei file: allocazione contigua

- Una questione importante nella memorizzazione di un file è tenere traccia di quali blocchi del disco sono associati a ciascun file.
- Esistono vari metodi utilizzati in diversi sistemi operativi.
- Lo schema di allocazione più semplice è quello di memorizzare ogni file come blocchi adiacenti del disco (**allocazione contigua**).
- Poiché la dimensione del file sarà difficilmente multiplo della dimensione del blocco, si crea un piccolo spreco di spazio interno all'ultimo blocco (così come accadeva per le pagine di memoria).

Allocazione continua



Svantaggi

- Piccolo spreco di spazio nell'ultimo blocco.
- Frammentazione a causa della rimozione dei file (gli spazi vuoti tra file non riescono ad essere impiegati per i nuovi).



Vantaggi

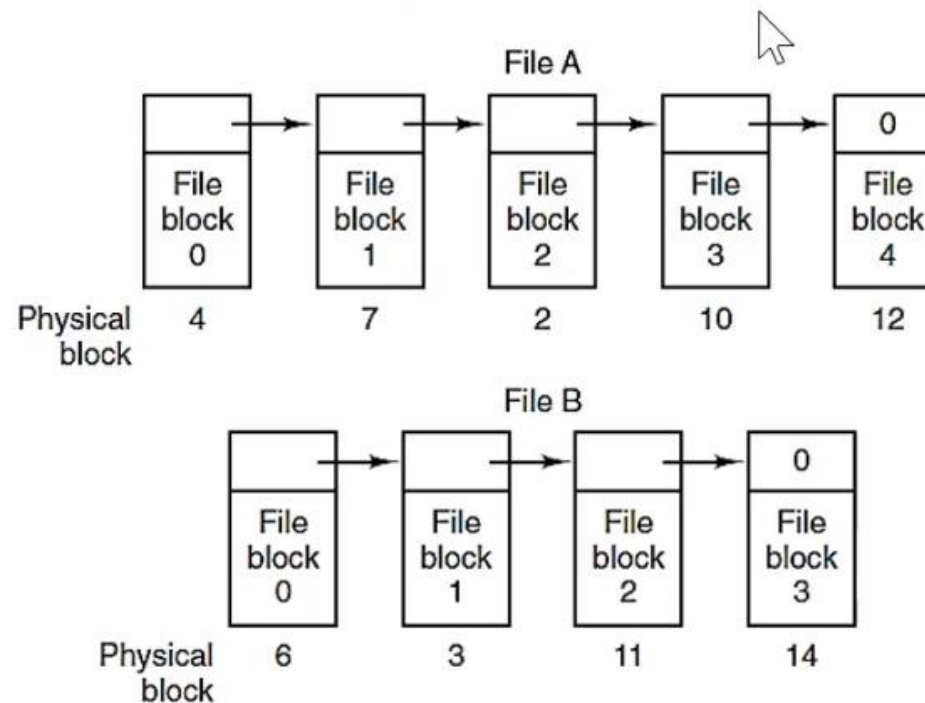
- Semplice da realizzare: è sufficiente conoscere l'indirizzo del disco del primo blocco e il numero di blocchi nel file.
- Alte prestazioni: l'intero file può essere letto dal disco in una sola operazione (il tempo di seek e la latenza domina il tempo di accesso del disco).

Allocazione continua

- L'allocazione contigua è stata utilizzata inizialmente con i dischi magnetici in ragione alle loro caratteristiche.
- Il sistema è stato poi abbandonato perché all'atto della creazione del file era necessario specificarne la dimensione massima (non sempre nota in anticipo).
- Con l'avvento dei CD-ROM, dei DVD e degli altri supporti ottici riscrivibili, l'allocazione contigua improvvisamente viene nuovamente utilizzata.
- In questi casi si parla dei «*cicli e ricicli della storia dell'informatica*» per indicare che anche con l'incremento tecnologico, in modo bizzarro e inaspettato, algoritmi o metodi divenuti ormai obsoleti ritornino ad essere attuali.

Allocazione con liste collegate

- Il secondo metodo per memorizzare file è di utilizzare una lista concatenata di blocchi del disco (**non necessariamente adiacenti**).
- La prima parola di ogni blocco viene usato come un puntatore alla successiva. Il resto del blocco è per i dati.



Allocazione con liste collegate



Svantaggi

- Piccolo spreco nell'ultimo blocco (come nell'altro metodo).
- L'accesso casuale è **estremamente lento**: prima di arrivare ad un blocco occorre accedere ai puntatori dei blocchi che lo precedono.
- Poiché la dimensione del blocco dati non è una potenza di due, a causa dello spazio riservato al puntatore (pochi byte), ci potrebbe essere una inefficienza dovuta ai programmi che leggono/scrivono quantità di dati espressi in 2^N .

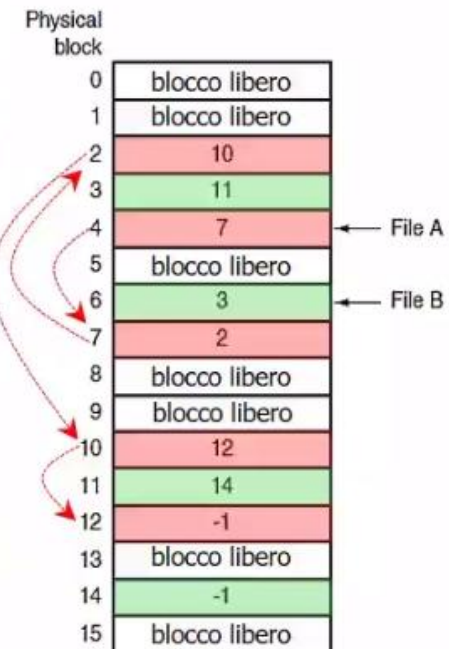


Vantaggi

- Possono essere utilizzati tutti i blocco del disco (non c'è spazio libero inutilizzato tra i file).
- Per ciascuna entry nelle directory è sufficiente memorizzare solo l'indirizzo del primo blocco del disco.
- La lettura dei file in modo sequenziale è facile ma non può essere svolta in un sol colpo come prima.

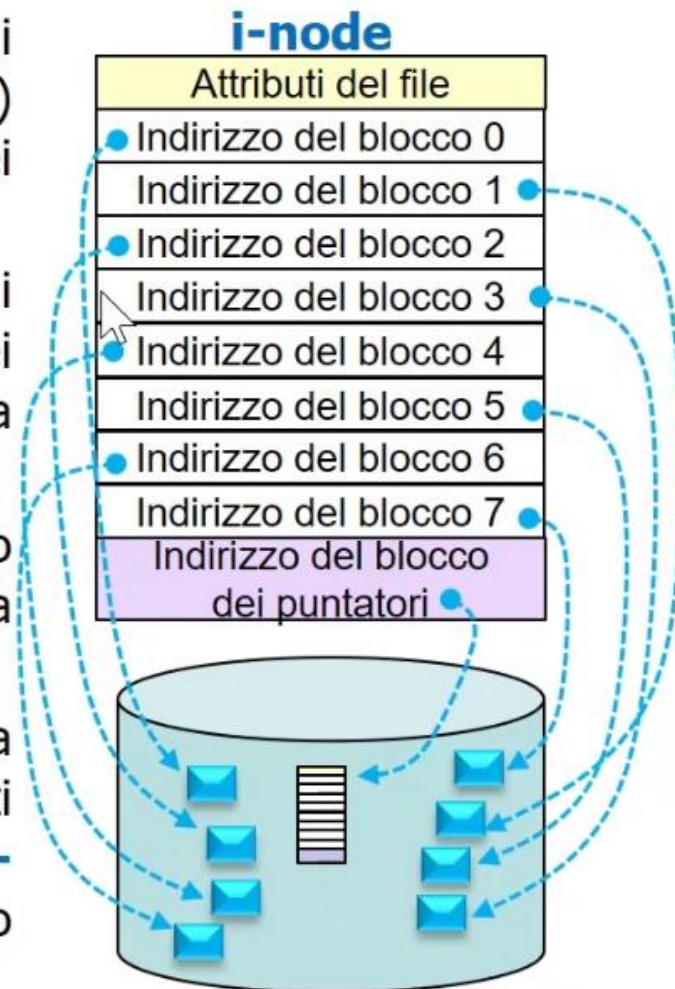
Allocazione con tabella in memoria

- Entrambi gli svantaggi dell'allocazione a lista collegata possono essere eliminati utilizzando una tabella in memoria dei puntatori di ogni blocco del disco (**File Allocation Table** o **FAT**).
- Le catene dei dati sono terminate con uno speciale terminatore (-1).
- L'intero blocco è disponibile per i dati e accesso casuale è facile.
- La entry nella directory mantiene il numero di blocco iniziale (un intero).
- L'intera tabella deve essere in memoria (con un disco da 200GB e blocchi da 1 KB, la tabella occupa quasi 600 MB di RAM!).
- L'idea della FAT non scala bene con dischi di grandi dimensioni.



i-nodes

- Per ogni file viene creata una struttura di dati denominata **i-node** (**index-node**) che contiene gli attributi e gli indirizzi dei blocchi del disco.
- Poiché l'**i-node** ha un numero fisso di posizioni, per consentire l'incremento dei blocchi del file, l'ultima cella è riservata ad un altro blocco simile.
- L'**i-node** è caricato in memoria solo quando il file viene aperto (la sua dimensione è più piccola di una FAT).
- Mentre la FAT è proporzionale alla dimensione del disco (tante righe quanti i blocchi del disco), la dimensione dell'**i-node** dipende solo dal numero massimo di file che possono essere aperti.

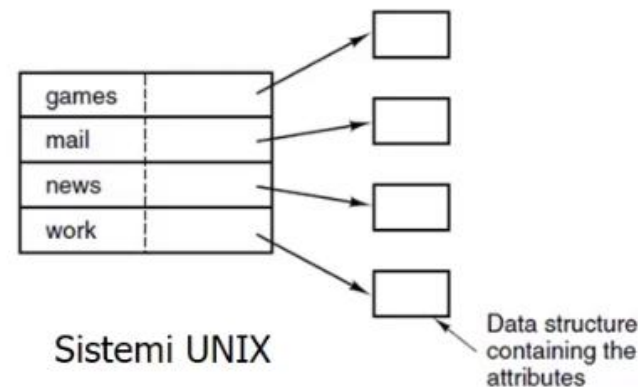


Realizzazione delle directory

- Durante l'apertura del file, il sistema operativo utilizza il pathname per individuare la voce nella directory che fornisce:
 - l'indirizzo del primo blocco del disco (allocazione contigua e gli schemi lista collegata), oppure
 - il numero di **i-node**.
- La directory associa il nome del file, gli attributi e il puntatore ai dati.
- Nel caso dell'**i-node** gli attributi possono essere memorizzati anche nell'**i-node** stesso.

games	attributes
mail	attributes
news	attributes
work	attributes


Sistemi Windows



Sistemi UNIX

Realizzazione delle directory

- Tutti i sistemi operativi moderni supportano nomi di file di lunghezza variabile.
- Si può definire una lunghezza massima (in genere 255 caratteri) all'interno di ciascuna voce nello spazio delle directory.



games	attributes
mail	attributes
news	attributes
work	attributes

Questa soluzione è semplice da realizzare ma spreca spazio nella directory, poiché solo alcuni file hanno nomi molto lunghi.

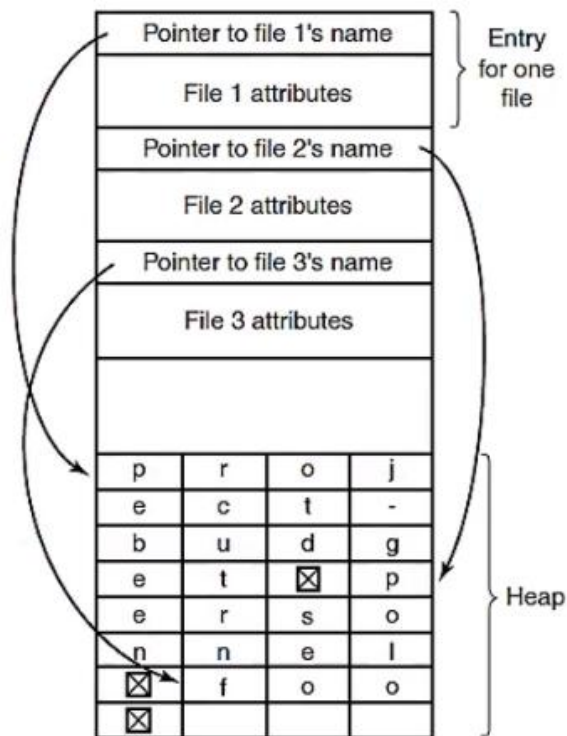
Realizzazione delle directory

Entry for one file

File 1 entry length			
File 1 attributes			
p	r	o	j
e	c	t	-
b	u	d	g
e	t	☒	
File 2 entry length			
File 2 attributes			
p	e	r	s
o	n	n	e
l	☒		
File 3 entry length			
File 3 attributes			
f	o	o	☒
⋮			

- Le voci della tabella delle directory può contenere due parti:
 - Una di lunghezza fissa per gli attributi del file.
 - Una di lunghezza variabile per il nome del file.
- Problemi:**
 - Quando un file viene rimosso, si cre un vuoto di dimensioni variabile (frammentazione), lo spazio si può compattare perché è interamente in memoria.
 - Se una voce della directory è nella congiunzione di due pagine, la lettura del nome del file può causare un **page-fault**.

Realizzazione delle directory



Un'altra soluzione è che tutte le voci della directory hanno lunghezza fissa e i nomi dei file sono mantenuti in memoria heap alla fine dello spazio per le directory.

L'eliminazione di un file crea uno spazio utile all'inserimento di un altro poiché le voci hanno tutte la stessa dimensione.

Gli errori di pagina possono ancora verificarsi durante l'accesso ai nomi di file.

Per velocizzare la ricerca all'interno directory estese (più di 100 file) si può utilizzare una tabella hash per ciascuna directory.

- Le tabella hash hanno tempi di accesso velocissimi, ma lo svantaggio di una maggiore complessità nella gestione. Un differente approccio per accelerare le ricerche è di utilizzare una memoria cache.