13. Interfaccia del File System

- Il concetto di File
- Modalità di accesso
- Struttura delle Directory
- Accesso rapido ai file
- Permessi di accesso ai file

13. Interfaccia del File System

- Per gli utenti di un computer, il **File System** è spesso la parte più visibile del Sistema Operativo di quel computer.

- Il File system fornisce infatti i meccanismi per la memorizzazione e l'accesso ai dati e agli applicativi del Sistema Operativo e degli utenti del sistema di calcolo.

- Un File System consiste di due parti:
  - un insieme di **file**
  - una **struttura di directory**, che permette di organizzare tutti i file del sistema

13.1 Il concetto di file

- **File**: unità logica di informazione memorizzata permanentemente (di solito) su un supporto di memoria secondaria e dotato di:
  - un nome
  - una posizione logica all'interno del File System
  - Alcuni attributi (dimensioni, diritti di accesso, date di creazione, accesso e modifica, etc…)

13.1 Il concetto di file

- I file contengono informazioni di diverso tipo:
  - **dati:**
    - numerici, caratteri, binari
  - **programmi:**
    - sorgenti, linkabili, eseguibili
  - **documenti:**
    - multimediali, omogenei

13.1 Il concetto di file

- e il sistema operativo e/o gli applicativi che li usano possono riconoscere una **struttura interna** ai file:
  - Un *file di testo* è formato da caratteri organizzati in righe
  - Un *programma sorgente* è suddiviso in procedure e dati
  - un *eseguibile* è spesso suddiviso in segmenti
  - ...
- (Naturalmente, in ultima istanza un file è solo una sequenza di bit...)

13.1.1 Attributi dei file

- A ciascun file sono associati degli **attributi**
- Tra le altre cose, gli attributi facilitano l'uso e le possibili operazioni che si possono fare su quel file:
- **Nome simbolico**, è l'unica informazione mantenuta in una forma adeguata per gli utenti umani
- **Tipo**: necessaria per quei sistemi operativi che supportano diversi tipi di file
- **Posizione fisica**: dove si trova il file sul supporto di memoria secondaria.

13.1.1 Attributi dei file

- **Posizione logica** del file all'interno del File system: il **pathname** del file
  - (occhio! questa informazione, per quanto fondamentale, **NON** è esplicitamente memorizzata da nessuna parte, eccetto che in alcuni casi particolari)

- **Dimensione** corrente del file. Il file può tuttavia occupare uno spazio maggiore o minore in memoria secondaria (provate a pensare perché... il secondo caso è più difficile)

13.1.1 Attributi dei file

- **Permessi di accesso/Protezione**: informazione di controllo dell'accesso che permette al sistema operativo di proteggere il file da usi non desiderati dal proprietario del file
- **Data, ora**: può essere la data della creazione, dell'ultima modifica, dell'ultimo accesso al file
- **identificazione del proprietario** del file: specifica l'utente proprietario del file (per sistemi multiutente) in modo da poter stabilire chi può fare cosa con il file all'interno del sistema.

13.1.1 Attributi dei file

- La memorizzazione degli attributi associati ad un file può richiedere anche più di un kilobyte di MS
- Gli attributi dei file sono memorizzati in opportune strutture dati accessibili attraverso il sistema di directory mantenuto in memoria secondaria (vedremo meglio più avanti)

13.1.2 Operazioni sui file

- Un file può essere visto come un **tipo di dato astratto** definito solo dalle operazioni che si possono compiere su di esso, rese disponibili dal sistema operativo:

- **Creazione di un file**: Richiede al SO di trovare spazio per il file, e poi di creare un accesso al file attraverso la directory che "contiene" il file, secondo le modalità di accesso stabilite per quel file

13.1.2 Operazioni sui file

- **Scrittura/Lettura di un file.** Il SO mette a disposizione una opportuna System Call per specificare il nome del file su cui si vuole operare. Il SO deve:
  - gestire il puntatore in scrittura/lettura al punto nel file in cui si vuole scrivere/leggere
  - occuparsi di trovare sull'HD spazio sufficiente per ospitare l'eventuale espansione del file, in caso di scrittura
- **Riposizionamento all'interno un file** nel punto desiderato per leggere o scrivere a partire dal punto specificato

13.1.2 Operazioni sui file

- **Rimozione di un file**: recupera lo spazio occupato dal file sul supporto di memoria secondaria e lo spazio occupato nella directory che lo "conteneva".

- **Troncamento di un file**: cancella i dati memorizzati e recupera lo spazio occupato, ma mantiene tutti gli altri attributi del file.

13.1.2 Altre Operazioni

- Oltre alle operazioni di base, di solito si può:
- **rinominare** il file
- **copiare** il contenuto di un file in un altro (il file di destinazione viene sovrascritto con il nuovo contenuto)
- **spostare** un file da una directory ad un'altra

13.2 Metodi d'accesso

- Un file può essere acceduto (letto o modificato) essenzialmente in due modalità:
- **Accesso sequenziale**: i dati del file (nel caso più semplice, i byte di cui è composto) vengono letti o modificati in modo sequenziale, a partire dall'inizio del file.
- **Accesso diretto**: si desidera leggere o modificare un dato posizionato in un punto ben preciso del file. Ad esempio, in un file di testo vogliamo poter leggere la 1000-esima riga del testo.

13.2 Metodi d'accesso

- Ovviamente, l'accesso diretto può essere simulato attraverso quello sequenziale: per leggere la 1000-esima riga del file possiamo incominciare a leggerlo dal primo carattere, contare le varie righe e fermarci quando abbiamo trovato la 1000-esima.
- Ma naturalmente questa soluzione è molto inefficiente. Come vedremo nel prossimo capitolo, i metodi di allocazione dei file in memoria secondaria vanno valutati anche da come permettono di implementare in modo più o meno efficiente l'accesso diretto ai file.

13.3 (Struttura del)le Directory (o cartelle, o folder)

- Un File System (FS) può essere molto grande: **decine di migliaia** di file, e occupare molto spazio: **centinaia di gigabyte**.
- Occorre una organizzazione che permetta di accedere a tutti questi dati in tempi ragionevoli
- In particolare, è fondamentale che i tempi di accesso ai singoli file (dati e attributi) **non crescano linearmente** con il numero dei file e con lo spazio occupato.

13.3 Le Directory

- Come possiamo tenere traccia dei file di un FS, e organizzarli in maniera conveniente?
- Mediante un sistema di **Directory**
- Una directory "contiene" dei file, nel senso che permette di risalire a tutte le informazioni relative ad un file (cioè i suoi dati e i suoi attributi) di quella directory a partire dal nome del file stesso

13.3 Le Directory

- Tipiche informazioni che devono essere recuperabili per i file di una directory sono:
  - **Dov'è memorizzato** il file in memoria secondaria
  - **Dimensioni correnti** del file
  - **Data dell'ultimo accesso** del file
  - **Data dell'ultima modifica** del file
  - **ID del proprietario** del file
  - **Protezioni e permessi di accesso** al file

13.3 Le Directory

- Tipiche operazioni che possono essere compiute su una directory sono:
  - **Ricerca** di un file nella directory
  - **Creazione/cancellazione** di un file nella directory
  - **Visualizzazione** del contenuto della directory
  - **Cambiamento** del nome di un file
  - **Spostamento** di un file in un'altra directory

13.3 Le Directory

- Le informazioni contenute nella directory sono vitali per poter accedere ai file, e quindi:
  - **perdere i dati della directory comporta quasi sempre la perdita di accesso ai file.**
- Le directory devono essere logicamente organizzate in modo da fornire un minimo di efficienza di recupero delle informazioni contenute

13.3 Le Directory

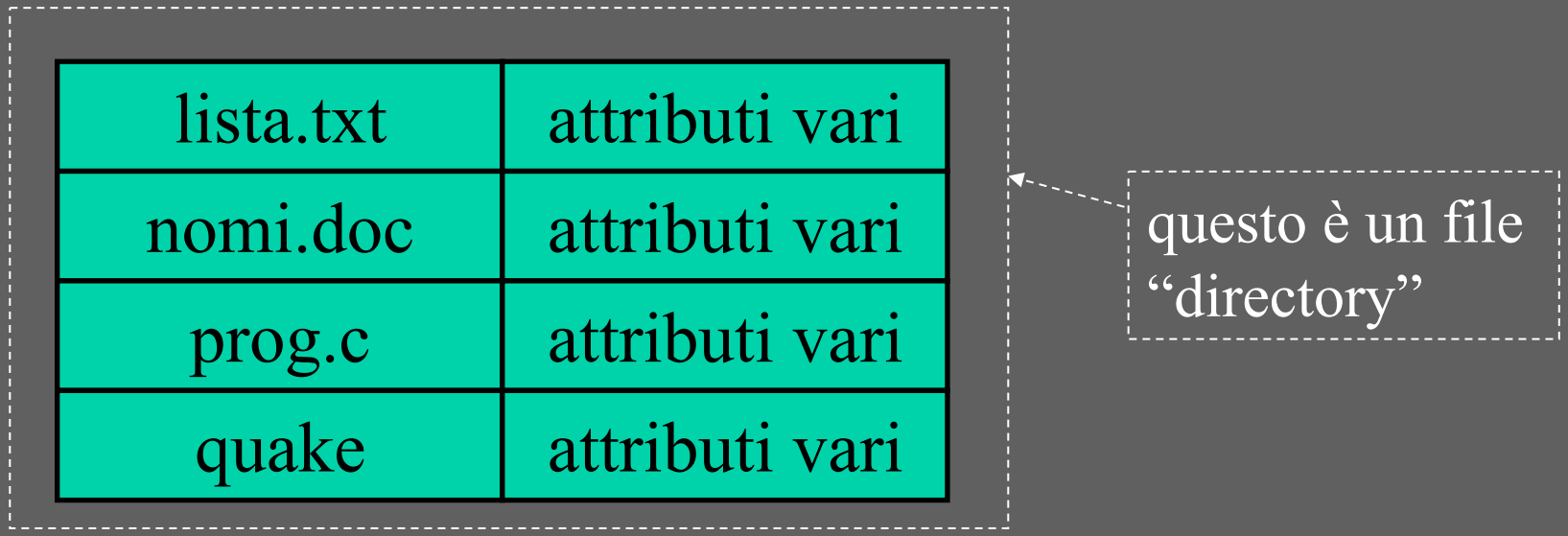
- Le directory sono esse stesse dei file, che però contengono informazioni relative ad altri file: i file “contenuti” in quella directory
- un file “directory” ha una struttura: contiene un certo numero di entry, una per ogni file di quella directory.
- Ogni entry contiene il nome di un file e una o più informazioni aggiuntive:
  gli attributi del file
  oppure
  un puntatore ad una struttura che li contiene

13.3 Le Directory

- Vi è una differenza fondamentale tra un le directory e gli altri file di un utente:
- Un generico file può essere aperto e modificato a piacere dal possessore del file (eventualmente anche facendo qualche "pasticcio")
- Al contrario, nemmeno il possessore di un file "directory" può modificarla a piacimento, ma solo attraverso le operazioni messe a disposizione dal sistema operativo
- Il SO deve infatti garantire l'integrità della struttura di ogni directory. In caso contrario, i file che "contiene" e i loro attributi potrebbero essere irrecuperabili.

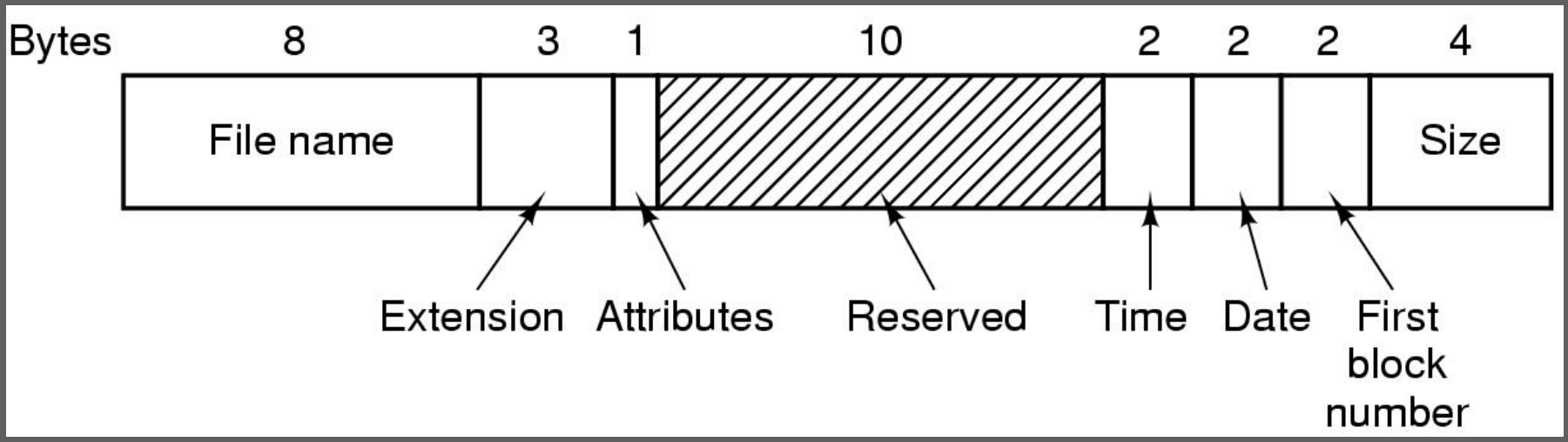
13.3 Le Directory: ms-dos

- Una prima possibilità è di inserire, a fianco del nome di ogni file, i suoi attributi (dimensioni, data di creazione/ accesso, tipo...) e informazioni sufficienti per sapere dove è memorizzato il file sul supporto di memoria secondaria: questa era la soluzione adottata dal ms-dos



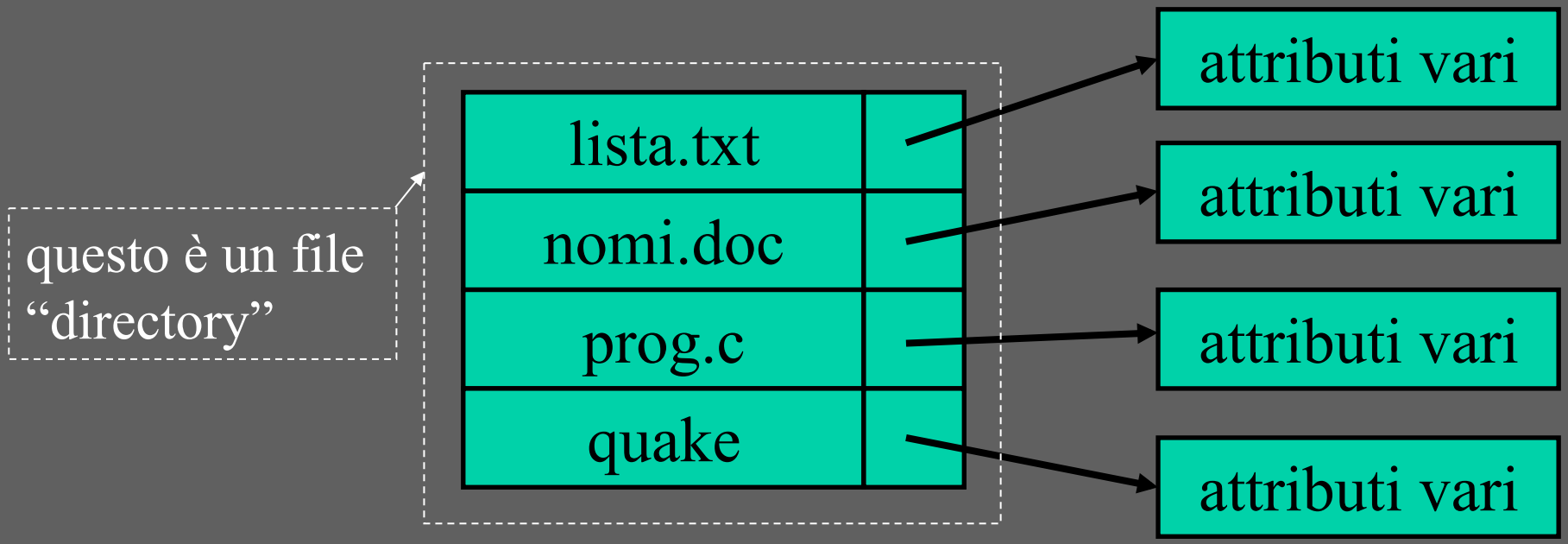
Le entry di una directory ms-dos

- In ms-dos un file directory è (era) fatto di una serie di entry di 32 byte ciascuna:



13.3 Le Directory: Unix

- Alternativamente, possiamo inserire, a fianco del nome di ogni file solo un puntatore ad una struttura interna, anch'essa memorizzata in memoria secondaria e gestita direttamente dal SO, in cui sono contenute tutte le informazioni su quel file: questa è la soluzione adottata dai sistemi Unix-like



13.3 Le Directory: NTFS

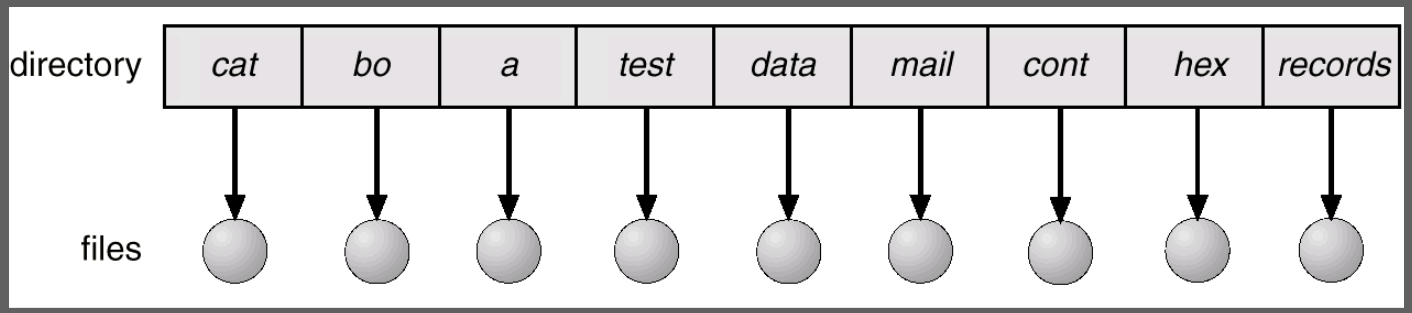
- Naturalmente sono possibili anche soluzioni alternative e "combinate".
- Ad esempio, nel File System NTFS (New Technology File System), adottato a partire da Windows XP, le entry di ciascuna directory non sono organizzate in modo lineare, come in ms-dos, ma in una struttura ad **albero di ricerca bilanciato**, in cui ogni foglia corrisponde ad un file "contenuto" nella directory.
- In questo modo il tempo richiesto per accedere alle informazioni di un qualsiasi file della directory è lo stesso, e non cambia a seconda della posizione in cui si trova l'entry di quel file nella directory.

13.3 Le Directory: NTFS

- Ciascuna foglia dell'albero di ricerca contiene il nome di un file e un puntatore (detto **file reference**) alla struttura interna memorizzata in memoria secondaria che contiene tutte le informazioni associate al file (quindi, una soluzione simile a quella adottata da Unix)
- Tuttavia, per ragioni di efficienza, alcuni degli attributi del file (dimensione corrente, data dell'ultimo aggiornamento) vengono anche replicati nella foglia dell'albero che contiene il nome del file e il suo file reference (quindi, una soluzione simile a quella adottata da ms-dos)

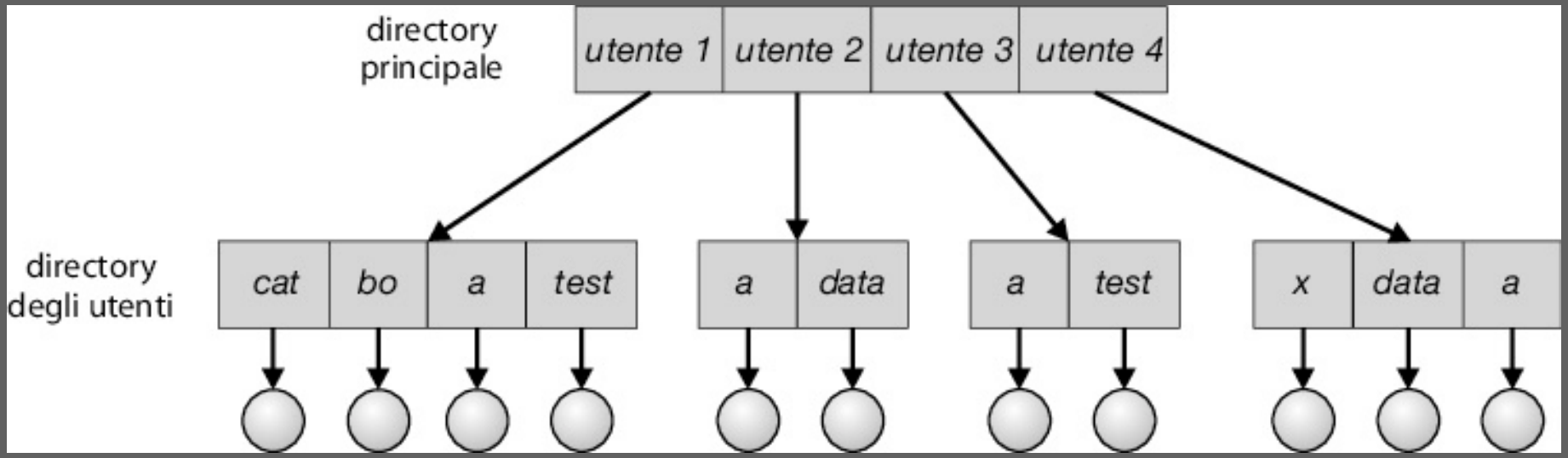
13.3.1 Directory ad un solo livello

- Come possiamo organizzare i file di un FS mediante le directory? Nel caso più semplice, un'unica directory "contiene" tutti i file del FS (fig. 13.7). E' la soluzione più facile da implementare, ma:
  - file di utenti diversi non possono avere lo stesso nome
  - i file non possono essere raggruppati separatamente
  - la ricerca di un file può essere molto inefficiente



13.3.2 Directory a due livelli

- Un ovvio miglioramento consiste nell'avere una directory per ciascun utente e, una directory principale che "contiene" le directory degli utenti (fig. 13.8):

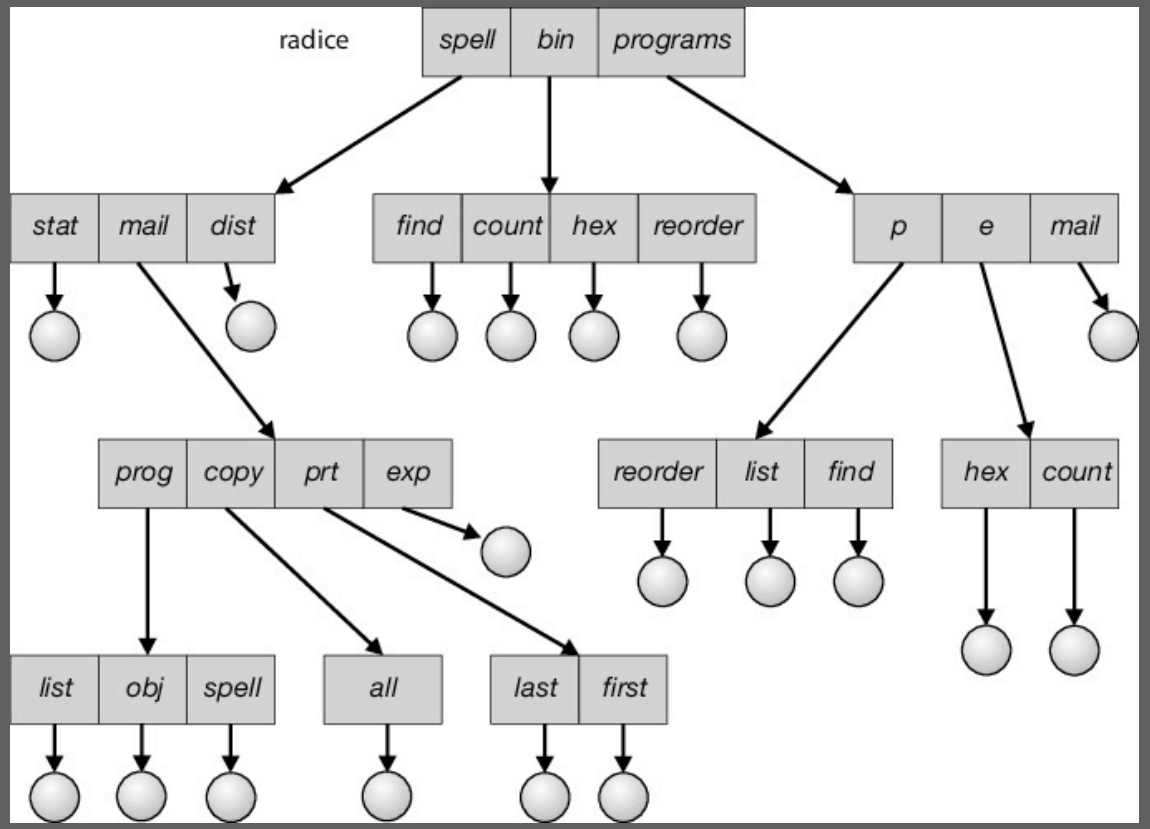


13.3.2 Directory a due livelli

- Emerge il concetto di **pathname** dei file, il percorso che si deve compiere a partire dalla directory principale per raggiungere un file all'interno di una delle cartelle degli utenti
- I file di utenti diversi sono raggruppati separatamente, per cui anche la ricerca di un file è più efficiente
- ma tutti i file di ciascun utente sono ancora tenuti insieme

13.3.3 Directory con struttura ad albero

- Naturale generalizzazione del concetto di directory a due livelli sono le directory con struttura ad albero, in cui ogni directory può contenere file normali **e altre directory**, e così via ricorsivamente (fig. 13.9)



13.3.3 Directory con struttura ad albero

- Per ovvie ragioni, la directory a partire dalla quale si dirama un File System è comunemente nota come **Root (o radice)** del File System.
- Ciascun utente di un computer ha a disposizione una porzione di File System che può modellare a piacere: la porzione di FS che si dirama a partire dalla cosiddetta **Home Directory** (di quell'utente/account)
- Quando un utente inizia una sessione di lavoro (si collega al SO col proprio account), viene posizionato automaticamente all'interno della home directory associata a quell'account

13.3.3 Directory con struttura ad albero

- Naturalmente, l'utente può navigare liberamente all'interno della propria porzione di File System, usando opportuni comandi (come "cd") o usando l'interfaccia grafica.

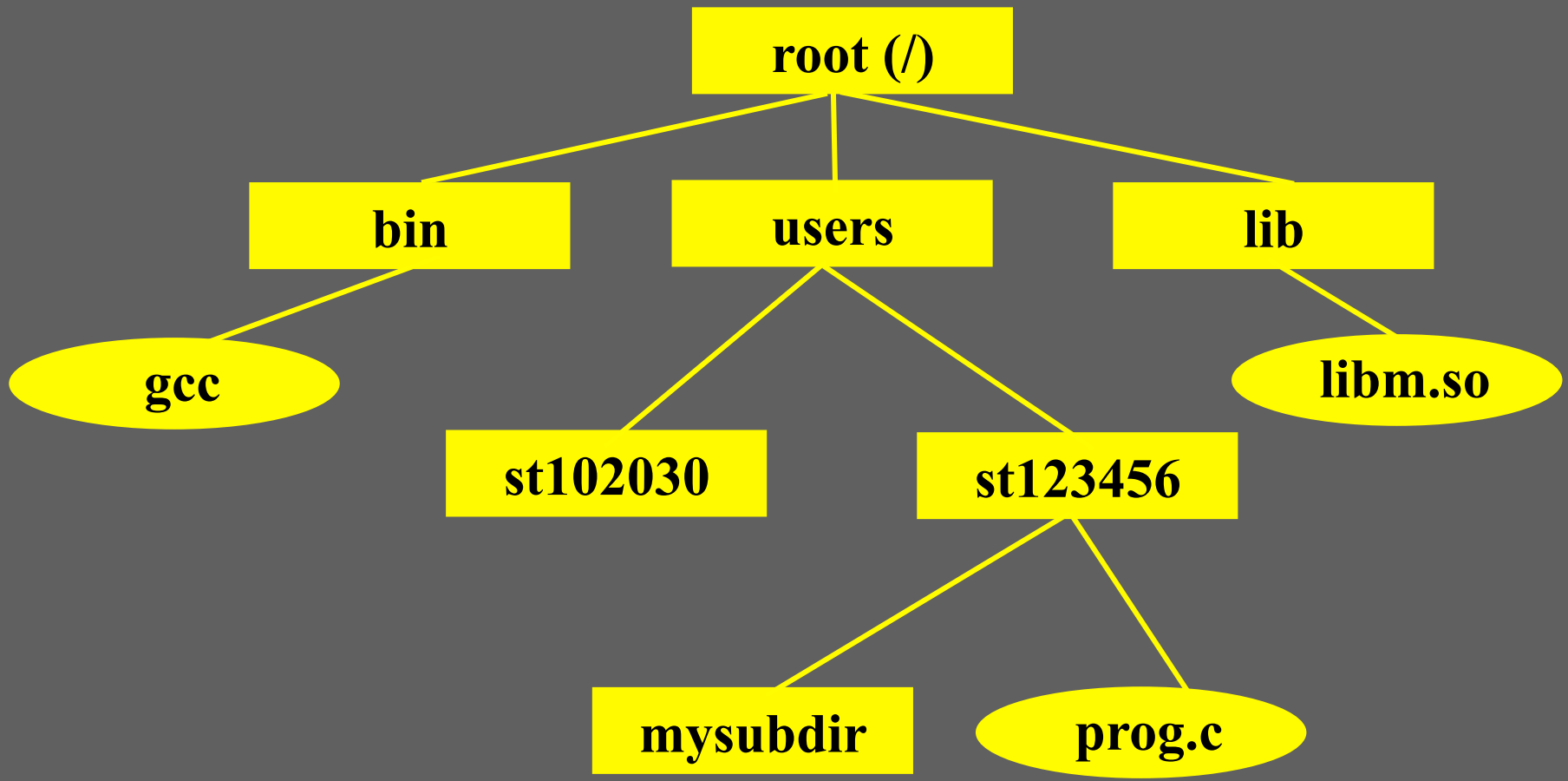
- Durante l'uso del sistema dunque, ogni utente è in ogni momento "posizionato" su una delle sue directory: la **current** o **working directory** (domanda: qual è la current directory in un sistema con interfaccia a finestre?)

- A fianco del concetto di **path name assoluto** di un file (definito a partire dalla radice del file system) emerge il concetto di **pathname relativo di un file**, perché è definito rispetto alla current directory.

13.3.3 Pathname assoluti

- Il pathname assoluto di un file **inizia sempre** con la directory radice (o root) dell'albero, la quale viene indicata con un carattere speciale:
  - / "slash" = la radice di un File System unix
  - \ "back slash" = la radice di un volume ms-dos/windows
- lo stesso simbolo si usa per separare i nomi delle varie directory intermedie del pathname
- Nel caso di ms-dos/windows, la radice può ulteriormente essere preceduta dal nome del volume a cui si fa riferimento (**C:** oppure **A:** e così via)

13.3.3 pathname assoluto di “prog.c”

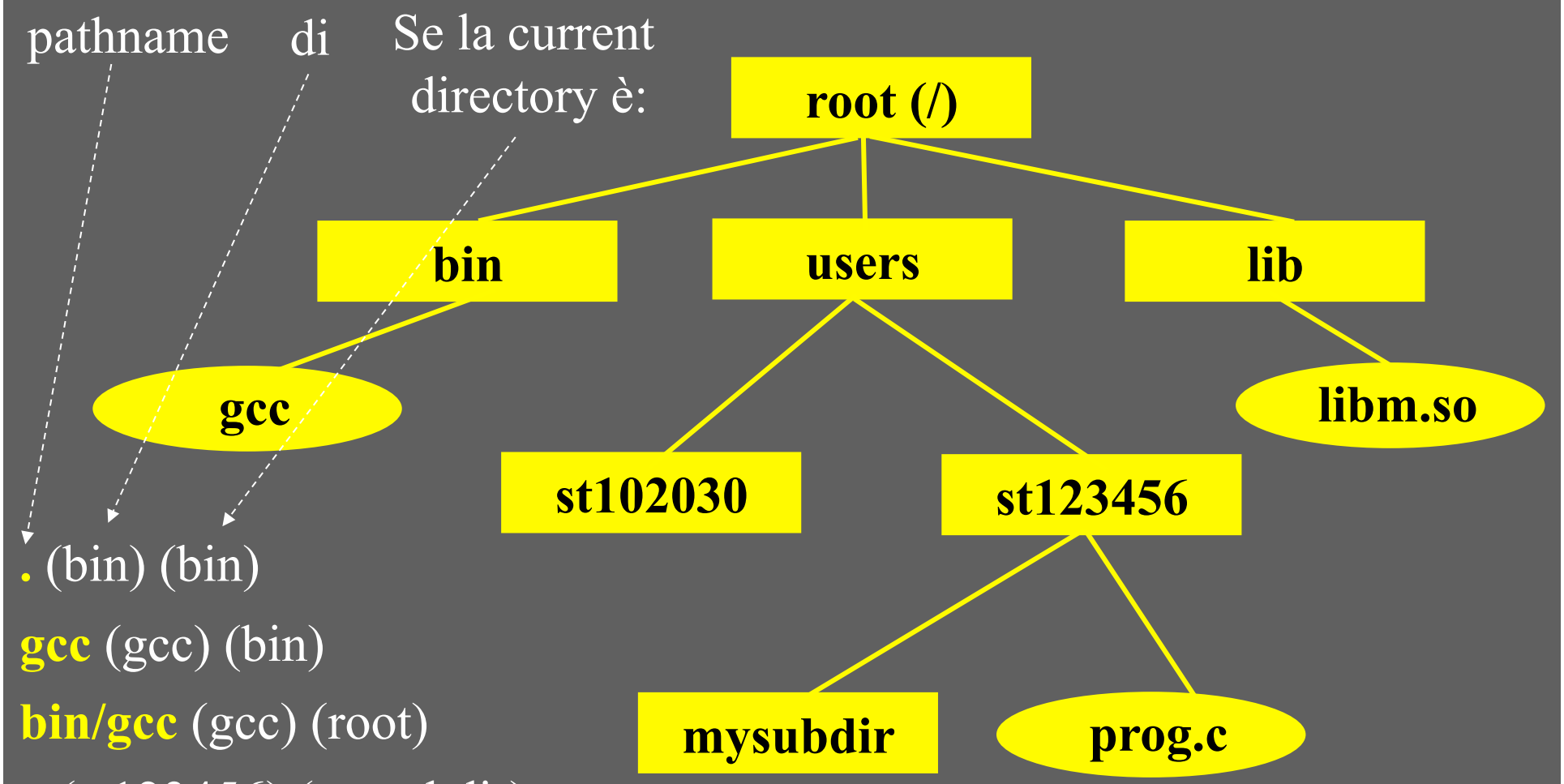


- /users/st123456/prog.c (unix)
- C:\users\st123456\prog.c (ms-dos/windows)

13.3.3 Pathname relativi

- Un pathaname relativo non inizia mai con / o \, ma con il nome di una directory diversa dalla radice
- convenzione adottata in qualsiasi FS:
  - . (punto) è un sinonimo della current directory
  - .. (punto punto) è un sinonimo della "parent" current directory (directory padre, o directory genitrice)
- **Data una qualsiasi current directory e un qualsiasi file del FS, esiste sempre un pathname relativo che permette di specificare la posizione del file rispetto alla current directory**

13.3.3 pathname relativi



13.3.3 Pathname

- Il pathname (relativo o assoluto) può essere usato come argomento di un comando o di una system call. Ad esempio, in UNIX:
  - ls -l ../users/st13456/prog.c
  - fopen (“/users/st13456/prog.c”, “w”);
- I pathname si usano anche nei sistemi a finestre: un eseguibile che debba lavorare su un file, deve comunque fare riferimento al file usando il suo pathname

13.3.3 Pathname

- La presenza di directory e sottodirectory implica la necessità di opportuni comandi per la gestione del FS:
  - **mkdir** [pathname]*nomedir* (crea una nuova directory)
  - **rmdir** [pathname]*nomedir* (rimuove una directory)
  - **cd** *pathname* (riposiziona la current dir. a *pathname*)
- Nelle interfacce a finestre, analoghe operazioni possono essere fatte via menù (in ogni caso, comandi e operazioni via menù sono implementati con opportune system call)

13.3.4 Directory con struttura a grafo aciclico

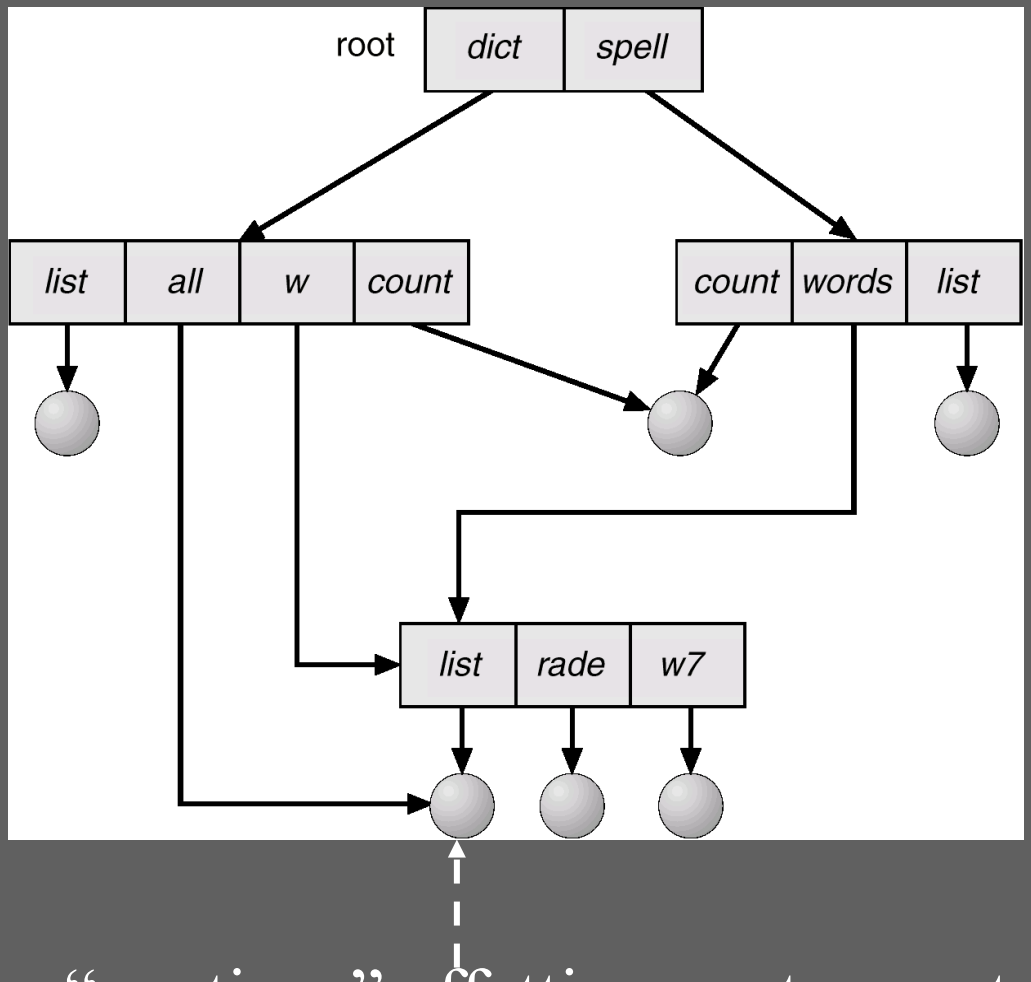
- La struttura ad albero non permette di condividere file o directory con nomi diversi. Questo è un grosso limite alla condivisione e alla cooperazione.

- Se lo si desidera, lo stesso file dovrebbe poter essere visto da directory diverse, eventualmente anche usando nomi diversi nelle diverse directory.

13.3.4 Directory con struttura a grafo aciclico

- I diversi collegamenti ad un file o ad una directory prendono il nome di **link**
- Diversi SO realizzano i link in maniera diversa, ottenendo risultati diversi.
- Vedremo in dettaglio come possono essere implementati i link nel capitolo 14.

13.3.4 Directory con struttura a grafo aciclico



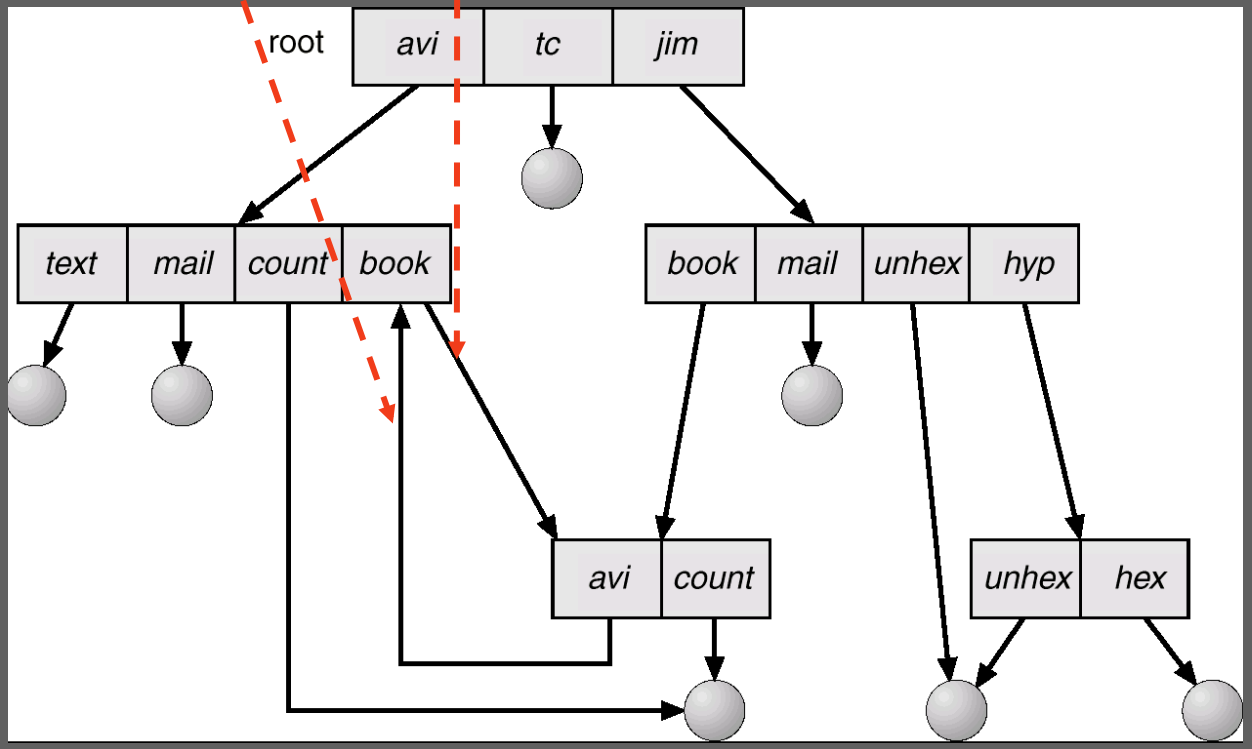
quale directory “contiene” effettivamente questo file, e qual è il suo pathname (fig 13.10)?

13.3.5 Directory con struttura a grafo generale

- In un File System a grafo generale, una directory può "contenere" il nome di una directory padre (e più in generale una directory "antenata").
- Questa situazione è pericolosa: se un programma visita ricorsivamente una directory e le sue sottodirectory, potrebbe non accorgersi di essere entrato in un loop.
- Inoltre, che succede se cerchiamo di cancellare il contenuto di una directory che contiene la directory padre?

13.3.5 Directory con struttura a grafo generale

- Per queste ragioni, di solito i sistemi operativi proibiscono l'insorgere di situazioni di questo tipo (fig. 13.11)



13.1.2 Accesso rapido ai file

- L'accesso ad un file attraverso il suo pathname è estremamente inefficiente, in quanto può richiedere più accessi alla Memoria Secondaria, su cui i file, i loro attributi, e le directory sono memorizzati.
- Consideriamo un programma che debba scrivere più volte all'interno di un file, e supponiamo che per farlo debba ogni volta specificare la posizione del file all'interno del File System. Ad esempio:
- fprintf("/users/john/subdir/myfile", "%d", myvar);

Attenzione: la fprintf
NON si usa così

13.1.2 Accesso rapido ai file

- Per ogni scrittura in “myfile” il codice che implementa la fprintf dovrebbe percorrere il pathname specificato fino ad arrivare a myfile, il che vorrebbe dire (rispetto alla precedente printf):
  - preleva dalla memoria secondaria le informazioni della cartella “/”, e vedi se contiene il nome di una cartella di nome “users”
  - se si, preleva dalla memoria secondaria le informazioni della cartella “users”, e vedi se contiene la cartella “john”
  - se si, preleva dalla memoria secondaria le informazioni della cartella “john”…
- e così via, fino a raggiungere “myfile”.

13.1.2 Accesso rapido ai file

- Per evitare questo modo inefficiente di accedere ai file, di solito, il SO richiede ai programmi di aprire (system call **open**) i file che vogliono usare (leggere, scrivere).
- le informazioni relative ad un file che è stato aperto vengono copiate in MP, in una **open file table.**
- Dopo la open del file, ogni accesso al file non passerà più dal file system, ma dalle informazioni relative a quel file contenute nella open file table in RAM
- Quando un programma chiude (system call **close**) un file su cui ha terminato di operare, le informazioni del file nella open file table possono essere rimosse.

13.1.2 Accesso rapido ai file

- Ed ecco allora come, correttamente, si può scrivere in un file (usiamo il sistema di I/O della libreria C dello Unix, che a sua volta sfrutta le system call del sistema)

```
File *fp;

fp = fopen("/users/john/subdir/myfile", "W"); //usa la "open"

fprintf(fp,"%d", myvar);

…

fprintf(fp,"%s", "hello");

fclose(fp);    //usa la system call "close"
```

13.1.2 Accesso rapido ai file

- Il “file pointer” **fp** dell’esempio del lucido precedente svolge quindi il ruolo di “scorciatoia” per accedere a tutte le informazioni sul file, che sono state copiate in MP durante l’esecuzione della open (o della fopen, nel nostro caso) nella **open file table**.

- Infatti, **fp** punta alla entry della open file table che contiene le informazioni sul file che è stato aperto con:

  fopen(“/users/john/subdir/myfile”, “W”);

13.1.2 Accesso rapido ai file

- Ma quali informazioni relative ad un file vengono copiate in MP durante la “open” del file? Come minimo, una copia di tutti gli attributi del file, così leggerli (ed eventualmente aggiornarli) ripetutamente è più veloce
- Ma di solito, anche la porzione di un file aperto su cui un processo sta operando viene copiata in MP, in modo che tutte le operazioni sui dati del file vengono fatte sulla copia in MP.
- Il SO si occupa poi di ricopiare in MS le eventuali modifiche apportate alla copia del file e ai suoi attributi che stanno in MP, periodicamente o, come minimo, quando il file viene chiuso mediante la “close”

13.4 protezione

- Il proprietario/creatore di un file deve poter controllare
  - **che cosa** si può fare sul suo file
  - **chi** può fare qualcosa sul suo file
- Che cosa si può fare:
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List properties

13.4 protezione

- Come potremmo specificare chi può fare qualcosa?
- mediante **lista d'accesso**: associare ad ogni file la lista degli utenti che hanno un qualche diritto sul file specificando per ciascuno l'insieme di operazioni permesse. Oppure:
- mediante una **capability list**: associare ad ogni utente la lista dei file su cui hanno un qualche diritto, specificando l'insieme di operazioni permesse.
- Entrambe le scelte sono molto costose e inefficienti da implementare, e di solito si ricorre a versioni semplificate di uno dei due approcci

13.4 Permessi di accesso in Unix

- E' una variante delle liste di accesso.
- **Esistono solo tre "classi" di utenti:**
  - il possessore del file
  - il/i gruppo/i a cui appartiene il possessore del file
  - tutti gli altri utenti del sistema
- **Esistono solo tre tipi di protezione:**
  - in lettura
  - in scrittura
  - in esecuzione