SAPIENZA
UNIVERSITÀ DI ROMA

**Corso di laurea in Ingegneria dell'Informazione**
**Indirizzo Informatica**

Reti e sistemi operativi

# File system

# Introduzione

- Il file system è la parte più visibile del sistema operativo: esso fornisce i meccanismi e funzioni (system call) per il **salvataggio, l'accesso e la protezione di dati e programmi**.
- Vi sono vari dispositivi di salvataggio non volatile dei dati (HD, SSD, DVD, ecc...) → Il file system fornisce una **visione logica uniforme** delle informazioni salvate, nascondendo i dettagli implementativi specifici di ogni dispositivo
  - E un'**astrazione**, al pari della memoria virtuale
- Vi sono due livelli del file system:
  - Cosa vede l'utente (es. albero delle directory)
  - Com'è implementato per lo specifico device

# File

- **File è un contenitore di informazioni digitalizzate registrato in memoria secondaria**, può contenere:
  - Dati (es. alfanumerici o sequenze binarie)
  - Programmi
- Virtualmente è visto come uno spazio d'indirizzamento logico contiguo (sequenza ordinata di informazioni)
- **Qualsiasi accesso (lettura, scrittura) alle unità di memoria secondaria dovrà avvenire attraverso un file**.

# Struttura di un file

- Nessuna: sequenza di byte
- Struttura a “record” semplice
  - Linee
  - Campi a lunghezza fissa
  - Campi a lunghezza variabile
- Struttura complessa
  - Documento formattato (es. pdf, docx, ODF)
  - File rilocabili (es. file oggetto)
- Che decide e interpreta la struttura:
  - I programmi
  - Il sistema operativo (pochi casi fondamentali)

# Attributi di un file

→ Dipendono dal sistema operativo, solitamente:

- **Nome**: informazione leggibile, semplice “handle” per condividere il file.
- **Identificatore** univoco all'interno del S.O.
- **Tipo** del file.
- **Locazione**: dispositivo e posizione
- **Dimensione** attuale del file.
- Informazioni di **protezione** per controllare **chi** può effettuare gli accessi in lettura, scrittura ed esecuzione del file.
- **Utente** proprietario (opzionalmente, **gruppo**)
- Data e ora creazione, ultima modifica e accesso.

# Directory

- E' un'entità (un file “speciale”) che elenca (“contiene”) altre entità, ovvero file e altre directory.
- Le informazioni sui file presenti sono mantenute (o indicizzate) in una struttura associata alla directory, anch’essa conservata sul disco.
  - Ogni elemento di tale struttura consiste di un nome ed un identificatore, che a sua volta individua gli attributi del file.

# Operazioni sui file (1/2)

Un file è un tipo di dato astratto (ADT) su cui si possono eseguire le seguenti **operazioni fondamentali** (system call):

- **Creazione**: allocare spazio nel file system, registrare il file nella directory
- **Scrittura**: è necessario specificare nome del file e i dati da scrivere, il sistema operativo deve mantenere un “indicatore di posizione” di scrittura, che indica la posizione su cui effettuare la prossima scrittura.
- **Lettura**: è necessario specificare nome del file e il (puntatore al) buffer su cui memorizzare i dati letti, il sistema operativo deve mantenere un “indicatore di posizione” di lettura, che indica la posizione su cui effettuare la prossima lettura.

# Operazioni sui file (2/2)

- **Riposizionamento**: operazione di spostamento dell'indicatore di posizione di lettura/scrittura (non è un'operazione di I/O)
- **Cancellazione**: Rimuove il file, rilasciando lo spazio occupato in memoria secondaria ed eliminando il corrispondente elemento di directory
- **Troncamento**: “Svuota” il file, rilasciando lo spazio occupato in memoria secondaria, ma mantiene gli attributi originari ( a meno della dimensione)
- **Modifica attributi** (es. proprietario, accessi, ...)

→ Altre operazioni possono essere implementate come combinazione delle precedenti operazioni

  - Copia, rename, append, ...

# Open e close (1/2)

- **Problema**: Per ogni operazione su file occorre cercare l'elemento file all'interno della directory (ricerca anche ricorsiva, es. /usr/local/share/file.txt)
- **Soluzione**: mantengo una tabella dei file aperti (**open-file table**) che contiene le informazioni circa tutti i file aperti → gli indici delle entry corrispondono ad esempio all'identificativo univoco.
- open-file table:
  - Globale ( tutti i file aperti)
  - Locale (file aperti da un processo)
- I file possono essere condivisi (“aperti”) da più processi → uno stesso file apparirà su più open-file table locali

# Open e close (2/2)

- Funzione **open()**:
  - Apre il file, inserendo nell' open-file table una entry che descrive il file (ovvero, gli attributi: posizione su disco, puntatore di lettura/scrittura, ecc.).
  - Oppure, se condiviso e già aperto, incrementa un contatore corrispondente.
  - Ritorna l'indice della entry nella tabella (file descriptor), da utilizzare per i successivi riferimenti al file
  - In Unix/Linux: int fd = open(“name”, mode);
- Funzione **close()**:
  - Chiude il file, completando le scritture bufferizzata, rimuovendo la entry corrispondente dalla tabella dei file aperti.
  - Oppure, se condiviso e ancora utilizzato da altri, decrementa il contatore corrispondente.
  - In Unix/Linux: close(fd);

# File aperti

- Open-file table globale: contiene per ogni file aperto:
  - Informazioni indipendenti dai processi (es.: posizione nel disco, data di creazione, proprietario, dimensione)
  - Contatore di aperture
- Open-file table locale: contiene per ogni file aperto dal processo:
  - Puntatore alla corrispondente entry nella tabella globale
  - Indicatore di posizione
  - Modalità di accesso (lettura/scrittura)

# Lock sui file aperti

- Alcuni sistemi operativi forniscono meccanismi di **sincronizzazione sull'accesso (lettura/scrittura) ai file** condivisi da diversi processi:
  - Può essere condiviso o esclusivo
  - Può essere “obbligatorio” (**mandatory**) o “consigliato” (**advisory**):
- Mandatory: una volta acquisito da un processo, il SO garantisce che nessun altro processo possa accedere al file (di default in Windows)
- Advisory: la sincronizzazione degli accessi è demandata ai programmatori (di default in Unix)
  - In Unix/Linux: flock(fd, operation);

# Tipi di file

- Sarebbe utile per il sistema operativo poter conoscere il tipo di file:
- MS-DOS usava l'estensione .COM, .EXE, .BAT.
- Mac OS ciascun file possiede un attributo di reazione contenente il nome e il programma che lo ha creato.
- UNIX a volte utilizza un codice memorizzato all'inizio del file per indicarne il tipo.
- **Uso dell'estensione: utile ma non sicura per conoscere il tipo.**

| file type | usual extension | function |
|---|---|---|
| executable | exe, com, bin or none | ready-to-run machine-language program |
| object | obj, o | compiled, machine language, not linked |
| source code | c, cc, java, pas, asm, a | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| text | txt, doc | textual data, documents |
| word processor | wp, tex, rtf, doc | various word-processor formats |
| library | lib, a, so, dll | libraries of routines for programmers |
| print or view | ps, pdf, jpg | ASCII or binary file in a format for printing or viewing |
| archive | arc, zip, tar | related files grouped into one file, sometimes com-pressed, for archiving or storage |
| multimedia | mpeg, mov, rm, mp3, avi | binary file containing audio or A/V information |

# Tipo e Struttura dei file

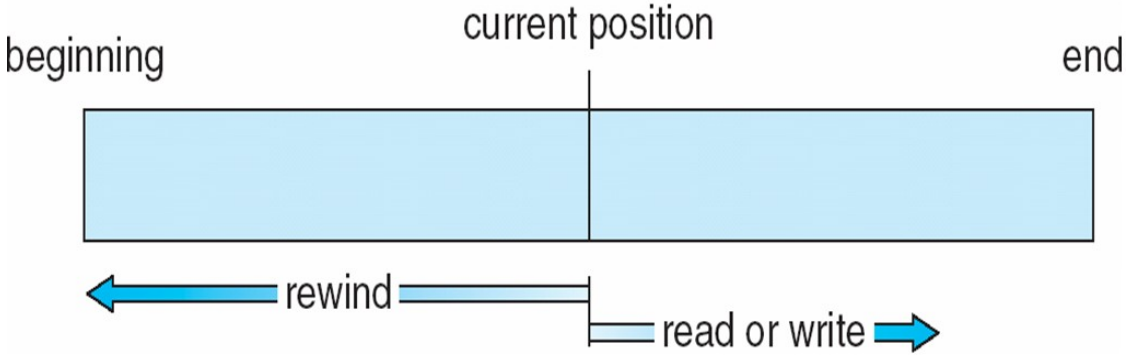
- Il tipo di un file e la corrispondente struttura logica possono essere riconosciute e gestiti in modi diversi in un sistema operativo
- In UNIX i file sono considerati semplici sequenze di byte terminate dal un carattere speciale (EOF, End of File) a cui s i accede sequenzialmente o con offset a partire dal primo byte → viene nascosto al programmatore il come sono effettivamente memorizzati.
  - **Solo i file eseguibili hanno un formato predefinito dal s.o.**

# Accesso ai file

- L'accesso ai file deve essere efficiente e tale da minimizzare il verificarsi di fenomeni di frammentazione interna.
- La dimensione dei blocchi del dispositivo di massa è fissa → se il contenuto del file è minore di un multiplo di tale dimensione, vi è frammentazione interna.
- Di base, vi sono 2 modalità di accesso:
  - Sequenziale
  - Diretta

# Accesso sequenziale

- Basato su modello a nastro
- Le informazioni si elaborano un record dopo l'altro utilizzando chiamate:
  - **read next**: legge il prossimo elemento e aggiorna l'indicatore di posizione
  - **write next**: scrive il prossimo elemento e aggiorna l'indicatore di posizione
- Dopo una write non è più possibile leggere



# Accesso diretto (1/2)

- Si ispira alla modalità di accesso degli Hard Disk: il file è composta da una sequenza di blocchi di lunghezza fissa → **random access di ogni blocco**
- Metodo utile per accedere a grandi quantità di dati (es: basi di dati) → l'idea è che i blocchi logici siano della dimensione dei blocchi fisici (non sempre ciò è possibile)
- Il numero di blocco (0 oppure 1-based) è un parametro delle operazioni:
  - read(pos, buffer)
  - write(pos, buffer)

# Accesso diretto (2/2)

- Simulazione di accesso sequenziale

| sequential access | implementation for direct access |
|---|---|
| *reset* | $cp = 0;$ |
| *read next* | *read cp*;<br>$cp = cp + 1;$ |
| *write next* | *write cp*;<br>$cp = cp + 1;$ |

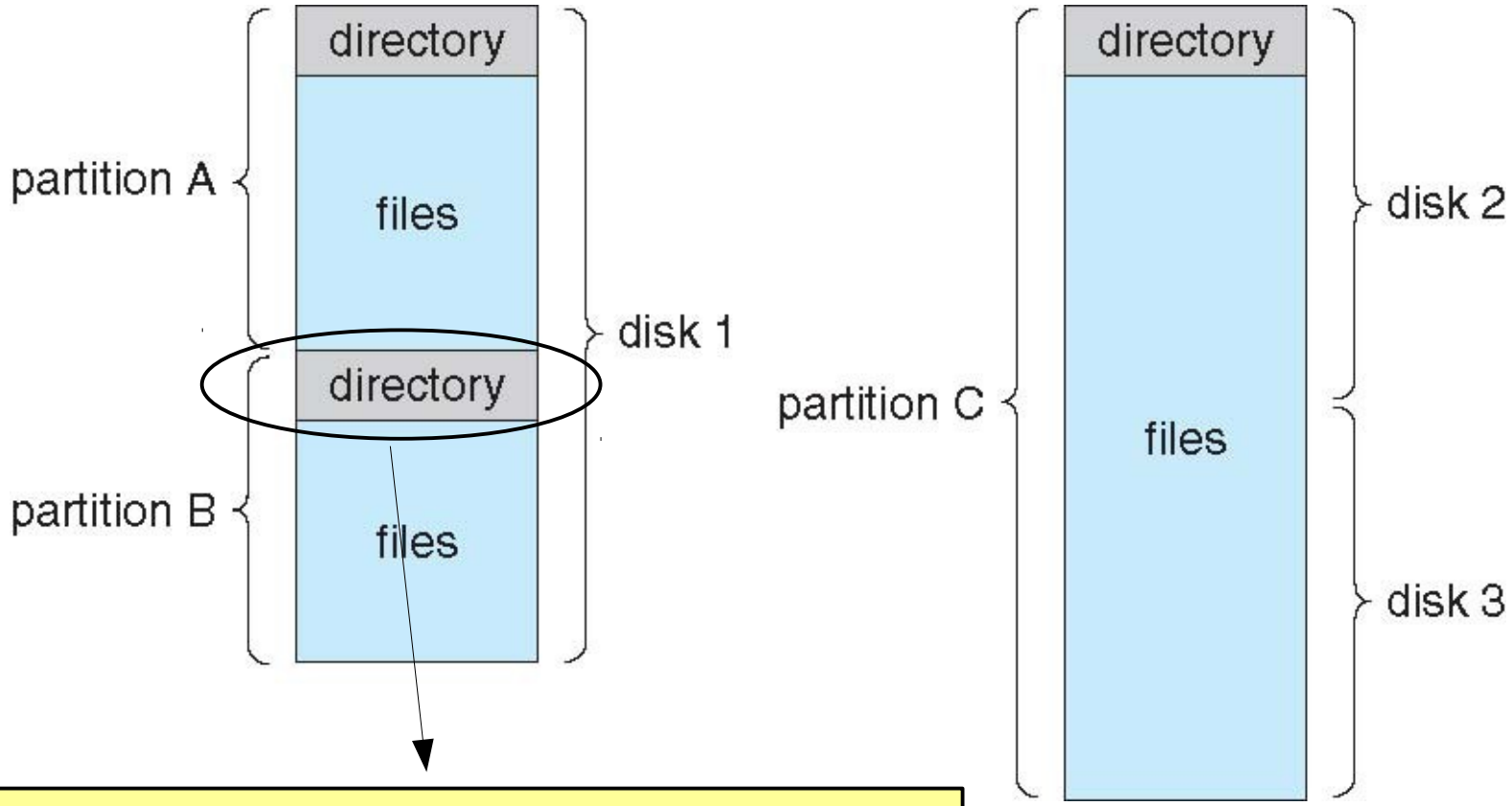
# Accesso indicizzato

- Definisce un **indice** (tabella chiave-posizione) per ogni file, memorizzato ad esempio all'inizio del file oppure in file indice correlati.
- Costruito a partire dal metodo di accesso diretto: si trova l'indice corrispondente al dato cercato (es. binary search tree) e si carica il blocco corrispondente che contiene il dato.

# Organizzazione di un file system

- Un dispositivo di archiviazione fisico può essere diviso in parti più piccole, chiamate **partizioni** (è una **suddivisione logica**).
- Viceversa, una partizione può contenere blocchi che appartengono a dispositivi fisici diversi (es. con organizzazione RAID, Redundant Array of Independent Disks).
- Il partizionamento dei dischi permette:
  - Coesistenza di più file system e sistemi operativi
  - Backup semplificato (es. separazione dati/programmi).
- Una partizione può contenere:
  - Un file system (anche detta **volume**)
  - Dati raw (es., partizione di swap)
- Solitamente la dimensione della partizioni è fissa, se possibile può essere modificata con appositi tool.

# Partizioni




Ciascuna partizione, se contiene un file system, ha una tabella detta **device directory** o **volume table of contents** che contiene informazioni su (o puntatori a) tutti i file presenti.

# Directory

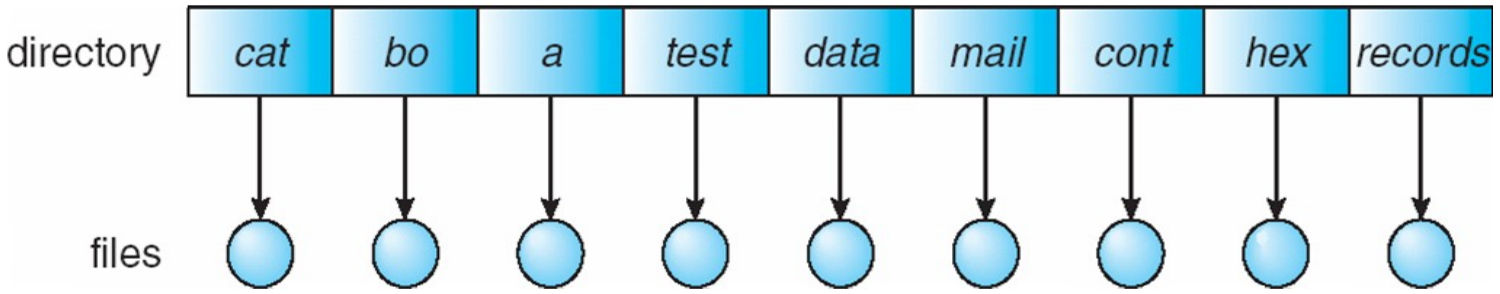
- La directory può essere vista come una tabella che associa nomi di file ad elementi della tabella stessa
- Operazioni su directory
  - Ricercare un file
  - Creare un file
  - Cancellare un file
  - Elencare i file contenuti in una directory
  - Rinominare un file
  - Attraversare il file system, ovvero accedere ad ogni sotto-directory per eseguire lì le operazioni elencate sopra

# Organizzazione della directory

- La directory deve essere organizzata in modo tale da:
  - **Garantire efficienza** nel reperire i file
  - **Assegnazione dei nomi** conveniente per gli utenti
    - Due utenti possono dare lo stesso nome a file differenti
    - Lo stesso file può avere svariati nomi
  - **Raggruppare logicamente** varie tipologie di file (programmi utente, librerie, documenti, ...)

# Directory monolivello

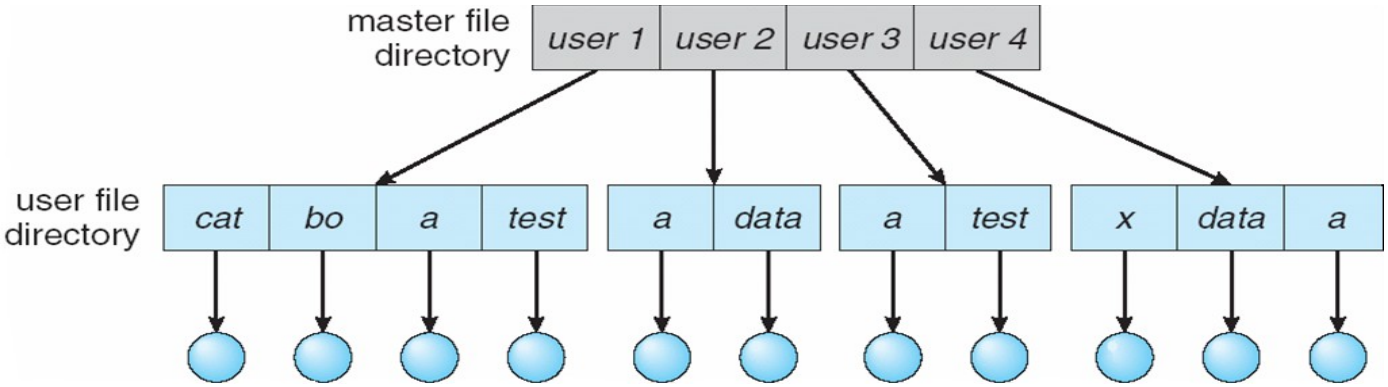
- Una directory unica per tutti gli utenti




- **Problemi**:
  - Possibili conflitti nei nomi dei file
  - Nessun raggruppamento logico

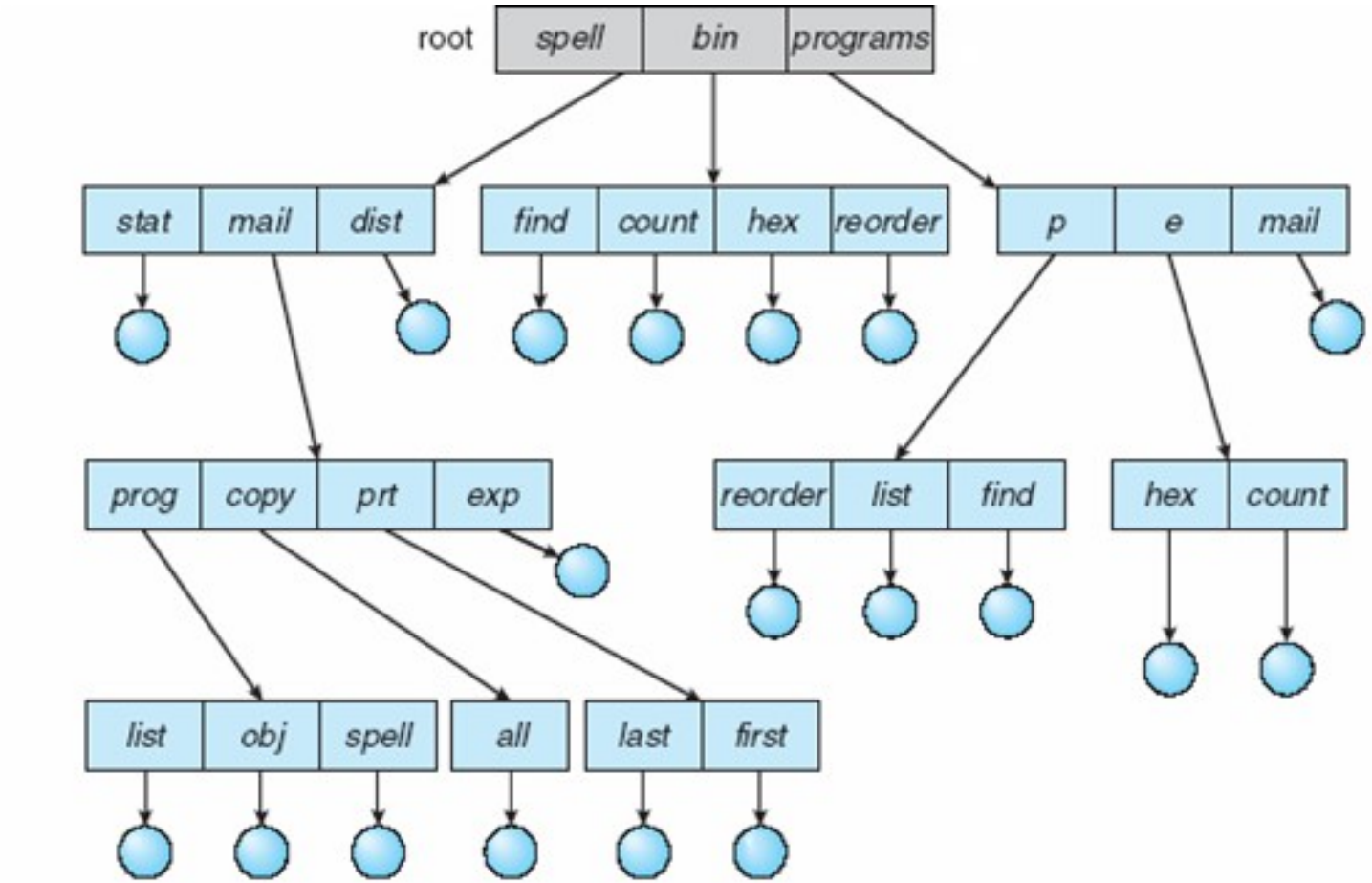
# Directory a due livelli

- Una directory per ogni utente




- Al momento del login, ogni utente viene “dirottato” all'interno della directory ad esso associata (UFD) → quando un utente si riferisce ad un file, la ricerca è circoscritta solamente alla propria UFD, spesso i file di sistema (es. binari) si trovano su una directory speciale (**search path**).
- Conflitti di nomi tra utenti risolti e ricerca più efficiente, possibili condivisioni di file tra utenti specificando il path completo (es. /userX/shared_file)
- **Problema**: nessun raggruppamento logico

# Directory con struttura ad albero (1/3)



# Directory con struttura ad albero (2/3)

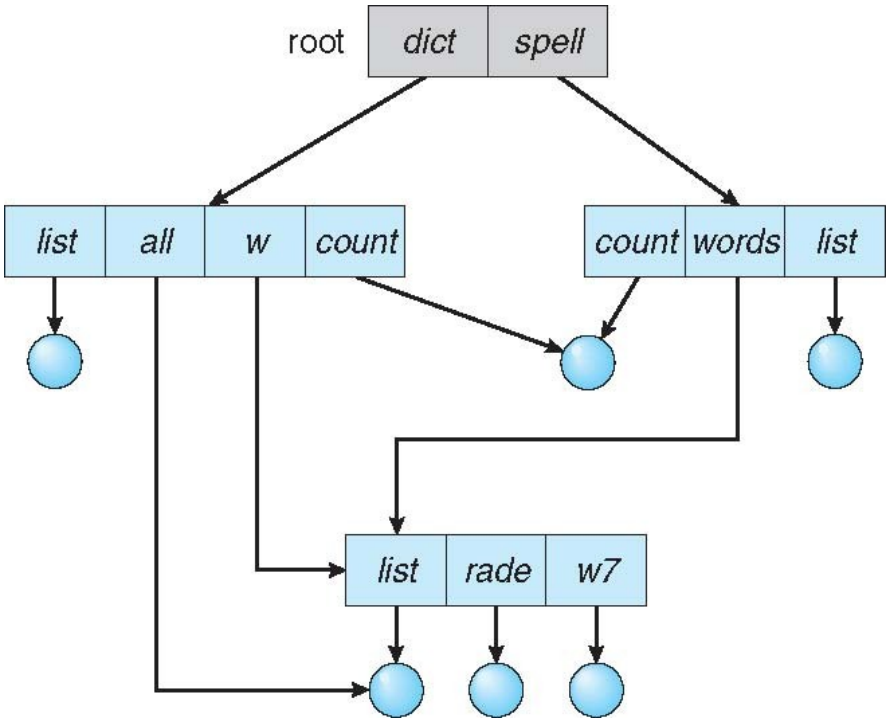
- Ricerca efficiente
- Raggruppamento logico
- File specificabili con:
  - Percorso **relativo** (es: file.txt, dir1/dir2/file.txt
  - Percorso **assoluto** (es. /home/user/filte.txt)
- Le directory sono **file** trattati in modo speciale
- Ogni processo ha una sua directory corrente (**working directory**) in cui vengono cercati i file a cui si riferisce con **percorso relativo**, modificabile tramite chiamata di sistema (utilizzata ad esempio dal comando cd)

# Directory con struttura ad albero (3/3)

- Una directory per ogni utente (**home**) specificata nel file di configurazione di login e copiata in una specifica variabile d'ambiente
- Politica di cancellazione di una directory
  - Impossibile se la directory contiene file
  - Eliminazione ricorsiva di tutti i file e le sottodirectory
- In Unix/Linux:
  - Creazione di una directory: mkdir dir
  - Eliminazione directory: rm directory, rm -r se contiene file o sottodirectory

# Struttura a grafo aciclico (1/2)

- Grafo aciclico (albero) → non contiene cicli
- Idea di base → **Aliasing**: Due o più differenti nomi/locazioni per identificare lo stesso file o directory
- Permette la condivisione di file e sottodirectory
- Si può realizzare prevedendo un tipo di file speciale→ **link**: puntatore ad altro file o directory
- Per **risolvere** il link, semplicemente si segui il percorso puntato



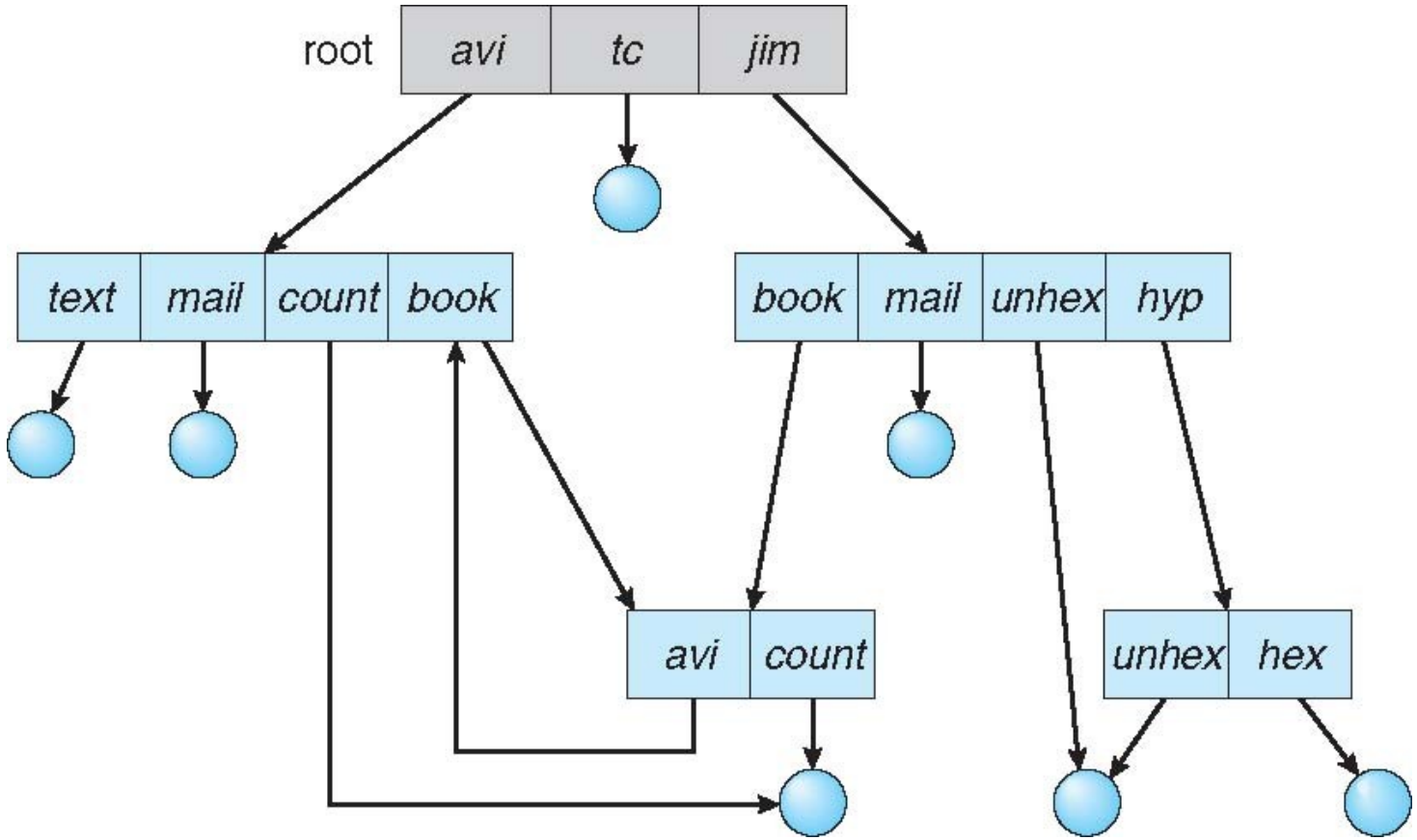

In Unix/Linux implementati come una stringa che rappresenta il percorso assoluto o relativo

# Struttura a grafo aciclico (2/2)

- In caso di eliminazione di un link, di solito viene eliminato solo il link lasciando inalterato il file puntato (es.symbolic links in Unix/Linux)
- In caso di eliminazione di un file puntato da un link, le possibili politiche sono:
  - Si lascia a disco il link, se si prova ad accedervi il file puntato non verrà più trovato (symbolic link in Unix/Linux)
  - Si mantiene una lista dei riferimenti a file, eliminando tutti i link di conseguenza (**problema**: campo a lunghezza variabile. Possibile soluzione: organizzazione a *daisy chain* dei link, ovvero in serie)
  - Si conserva il file fino a quando non esistono più link (attraverso un contatore di link, quando il contatore è 0 il file può essere cancellato, es. hard link in Unix/Linux).

# Directory a grafo generale (1/2)

- **Problema:** i cicli all'interno di una struttura di directory possono portare ad esempio a cicli infiniti durante l'attraversamento della struttura.



# Directory a grafo generale (2/2)

- Come garantire l'assenza di cicli nella struttura?
  - Si ammettono solo link a file
  - Si utilizza un algoritmo di identificazione dei cicli ogni volta che si aggiunge un collegamento → alto costo computazionale
- In caso di cicli si può evitare i loop infiniti durante l'attraversamento semplicemente non seguendo i link a directory
- In caso di self reference, si usa un garbage collector: si attraversa il file system, marcando i file accessibili. In un secondo passaggio si rimuovono i file non marcati → alto costo computazionale.

# Montaggio di un file system

- Prima di accedere ad un file system, esso va montato. Il montaggio è un concetto simile al concetto di apertura di un file (open()) prima di poter effettuare qualsiasi operazione sul file stesso.
- In alcuni sistemi operativi , è possibile definire il punto di montaggio (mounting point), che può essere qualsiasi directory del filesystem logico. L'accesso al nuovo filesystem si effettuerà in maniera totalmente trasparente.
- Il contenuto precedente, di solito, diventa inaccessibile.
- In Unix/Linux:
  - Montaggio: **sudo mount /dev/nome_disco -t auto directory**
  - Smontaggio: **umount directory**