File System

Sommario

- Interfaccia del file system
- Concetto di File
- Metodi di accesso
- Struttura delle Directory
- Montaggio del File System
- Protezione

File system

- Per la maggior parte degli utenti il **file system** è l'aspetto più visibile del SO.
- Il file system rappresenta una astrazione del modo con cui i dati sono allocati e organizzati su un dispositivo memoria di massa.
- Attraverso il file system, Il SO offre una visione **logica** uniforme della memorizzazione delle informazioni sui diversi supporti (dischi, nastri, CD, ecc.).
- Elemento di base nella gestione a livello logico della memoria di massa è il **file**.

File

- Un **file** è un insieme di informazioni correlate e registrate nella memoria secondaria con un nome.
- Il file è la più piccola unità di memoria secondaria assegnabile all'utente che può scrivere sulla memoria secondaria solo registrando un file.
- Il file può avere una sua struttura interna, a seconda del formato o del tipo. Per esempio un file eseguibile è una sequenza di sezioni di codice che possono essere caricate ed eseguite.

Attributi

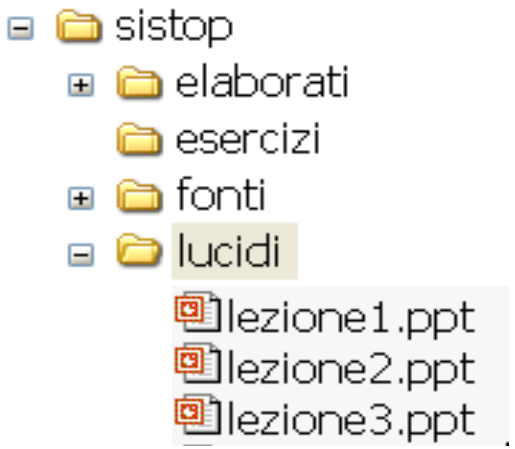
- Principali attributi del file sono:
  - **Nome:** è l'unica informazione mantenuta in un formato simbolico leggibile da una persona.
  - **Tipo**: usato nei sistemi che supportano più tipi diversi di file.
  - **Locazione**: puntatore al dispositivo e alla locazione di quel file sul dispositivo.
  - **Dimensione**.

Attributi

- Altri importanti attributi del file sono:
  - **Protezione**: informazioni di controllo che specificano chi può leggere, scrivere o eseguire il file.
  - **Ora, data e identificazione dell'utente**:informazioni usate per funzioni di protezione dei dati e di monitoring. Possono essere riferite a:
    - Creazione
    - Ultimo utilizzo
    - Ultima lettura

Directory

- Il file system può essere molto ampio (contenere molte migliaia di file) e tipicamente è organizzato in una struttura gerarchica che mantiene gli elementi terminali (i file) all'interno di contenitori detti **directory**.



Operazioni sui file

- La componente del SO che si occupa di gestire i file è detta **file system**
- Le operazioni messe a disposizione (mediante **system call**) dal file system per gestire i file, sono:
    - Create
    - Open
    - Close
    - Write
    - Read
    - Delete
    - Seek (Riposizionemento all'interno del file)
    - Truncate

Open e close

- **Open**: la maggior parte dei SO richiede che venga fatta una esplicita apertura del file prima delle operazioni di lettura e scrittura.
- **Close**: Analogamente viene richiesta una operazione di chiusura del file che indichi il termine del processo di lettura scrittura

Scrittura del file

- La **scrittura** su un file avviene per blocchi attraverso la chiamata ad una system call che specifica il nome del file e le informazioni che si vogliono inserire.
- Il SO:
  - Cerca il file nella directory
  - Mantiene un puntatore al punto in cui dovrà essere effettuata la prossima scrittura.

Lettura da file

- La **lettura** di un file avviene per blocchi attraverso la chiamata ad una system call che specifica il nome del file la locazione di memoria in cui deve essere trasferito il file.
- Il SO:
  - Cerca il file nella directory
  - Mantiene un puntatore al punto in cui dovrà essere effettuata la prossima lettura

File in uso

- Il sistema mantiene in memoria l'elenco dei file in uso in una apposita tabella detta **tabella dei file aperti**
- Quando viene iniziata una operazione sul file, il file viene inserito nella tabella in modo che ogni successiva operazione non comporti fasi di ricerca

Informazioni sui file aperti

- A ciascun file aperto sono associate diverse informazioni:
  - **Puntatore alla posizione corrente nel file**: per il supporto a read e write. Se più processi operano sul file esistono più puntatori alla posizione corrente.
  - **Contatore delle aperture**: se più processi possono aprire il file, il SO deve tenere conto delle aperture avvenute e quando sono tutte chiuse, rimuovere il file dalla tabella dei file aperti.
  - **Posizione su disco del file**: quando il file è aperto viene memorizzata la sua posizione sul dispositivo per evitare di doverlo cercare nuovamente ad ogni operazione.

Struttura dei file

- Tipicamente il SO non si occupa di gestire la **struttura interna** associata ai diversi tipi di file, ma lascia che siano le applicazioni a definirla e utilizzarla.
- Per il SO i file sono sostanzialmente insiemi ordinati di byte e le convenzioni che danno un significato specifico ai byte (cioè il formato dei file) sono gestiti a livello applicazione.
- I vantaggi nel delegare la gestione dei formati alle applicazioni sono diversi:
  - Dimensioni del codice del SO: per gestire tutti i formati il sistema dovrebbe essere enorme. Lasciando il controllo alle applicazioni si limita la dimensione del SO.
  - Flessibilità nella definizione di nuove strutture: quando occorre gestire nuovi formati non è necessario modificare il file system e il SO ma è sufficiente dotarsi delle applicazioni appropriate.
- Tutti i SO supportano almeno il formato dei file eseguibili, per poterli caricare in memoria ed eseguire.

Struttura interna dei file

- Cio' che l'HW vede di un file è una sequenza di accessi effettuati a dei **blocchi** (o **record fisici**) tutti di uguale dimensione.
- Dal punto di vista delle applicazioni, invece, il file è una sequenza di **record logici**, determinati dal formato del file, che hanno dimensioni variabili a seconda dell'esigenza dell'applicazione stessa.
- È improbabile che la dimensione del record logico corrisponda esattamente ad un record fisico e la localizzazione dei record logici all'interno del file può essere complessa.
- esempio per Unix
  - il file è un flusso di byte singoli e il record logico (che viene passato alle applicazioni attraverso system call) è costituito da un byte.
  - Lo scostamento del record logico è rappresentato dalla sua posizione nel file.
  - **Il SO impacchetta i record logici in blocchi fisici (se per esempio il blocco è di 512 byte, mette 512 record logici in ogni blocco).**

Struttura interna del file

- Si noti che :
  - lo spazio fisico è allocato per blocchi, è probabile che l'ultimo blocco di un file sia usato parzialmente, ovvero che una parte dell'ultimo blocco di un file vada sprecata.
  - Maggiore è la dimensione del blocco, maggiore è la frammentazione interna legata a questo fenomeno.

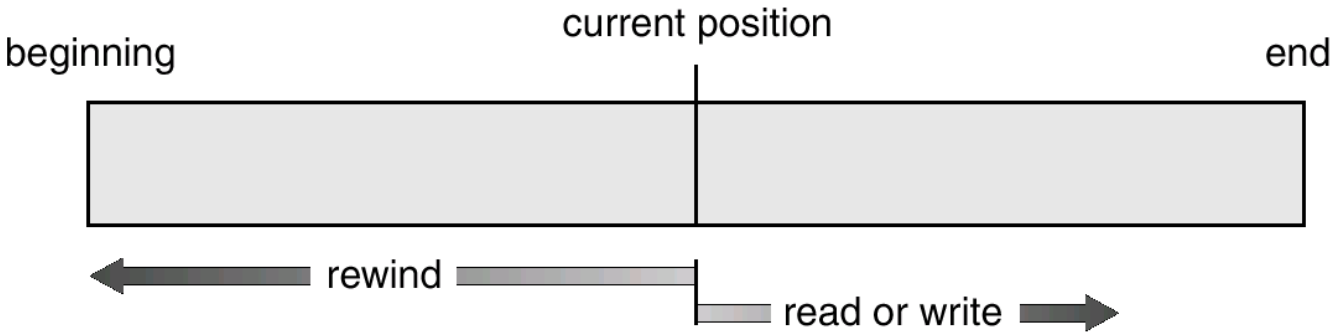
Metodi di accesso

- Il SO può fornire diversi metodi di accesso ai file:
  - **Sequenziale**: le informazioni del file vengono elaborate in ordine, un record dopo l'altro.
    - nastri
  - **Diretto**: il file è costituito da un insieme ordinato di blocchi a cui si accede direttamente.
    - dischi
  - **Attraverso indice**: al file vero e proprio è associato un file indice con lo scopo di velocizzare le ricerche.
    - Database

Accesso sequenziale

- Le informazioni del file vengono elaborate in ordine, un record dopo l'altro. E' il più utilizzato (ES: compilatori).
- Operazioni:
  - **Lettura**: **read**, legge la prossima porzione del file e avanza il puntatore di posizione.
  - **Scrittura**: **write**, scrive i nuovi dati in coda al file e avanza l'end of file.
  - **Reset**: riposiziona il puntatore di posizione a inizio file.

Accesso sequenziale



Accesso diretto

- Il modello i riferimento per l'accesso sequenziale sono i dispositivi come il nastro che consentono solo accessi in sequenza.
- I dispositivi ad accesso casuale (nel senso di non sequenziale) sono modello per un altro metodo di accesso al file: **l'accesso diretto** (o **accesso relativo**) in cui il file è costituito da record logici di lunghezza fissa e viene considerato come un insieme ordinato di blocchi (record) numerati che possono essere acceduti in modo arbitrario.
- Il **numero di blocco** è tipicamente assegnato in modo **relativo** (ciascun file inizia con il suo blocco 0).

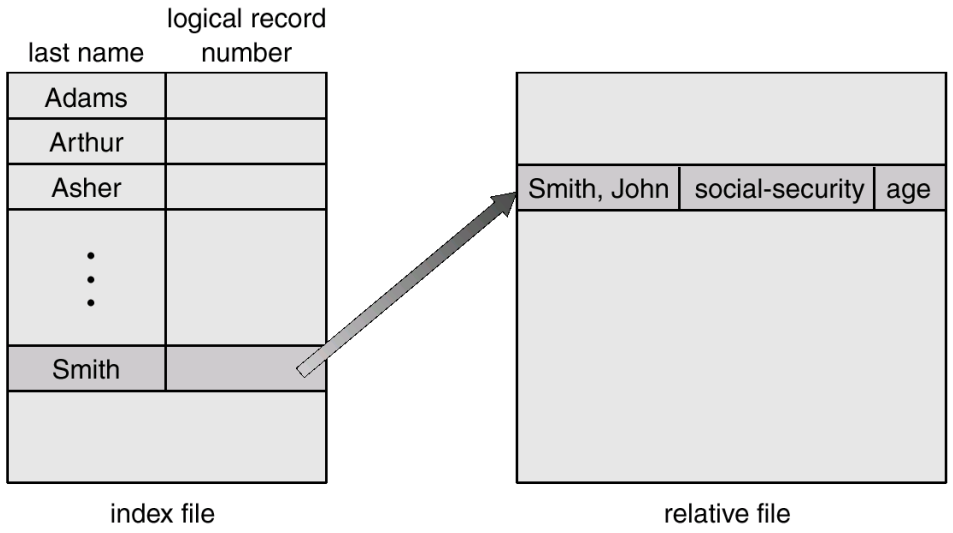
Accesso diretto

- Le istruzioni di lettura e scrittura devono quindi specificare su quale blocco devono essere eseguite:
  - **Lettura**: **read(n)**, legge il blocco con posizione relativa n.
  - **Scrittura: write(n)** scrive il blocco con posizione relativa n.
  - **Riposizionamento**: **seek(n)**, si posiziona sul blocco
- L'accesso sequenziale è facilmente realizzabile se si ha a disposizione l'accesso diretto (cp means current position):

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;

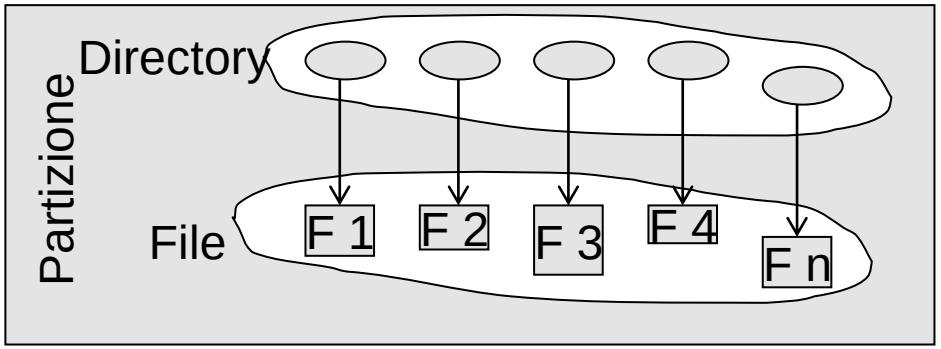
Indici

- Un altro metodo di accesso, che offre supporto alla ricerca veloce di occorrenze in un file, è quello che prevede la costruzione di un **indice**.
- L'indice contiene una o più chiavi di ricerca a cui sono associati i puntatori ai diversi blocchi del file.
- Il file contenente l'indice viene quindi associato ad un file di contenuti.



Directory

- La directory si può considerare come una tabella in cui sono memorizzate le informazioni relative al file e necessarie a:
  - Individuare la posizione del file
  - Gestirlo (copiarlo, cancellarlo, eseguirlo…)
- La directory si può organizzare in molti modi



Informazioni nella directory

- Informazioni tipicamente contenute nella device directory sono:
  - Nome del file
  - Tipo
  - Indirizzo
  - **Lunghezza attuale**
  - Lunghezza massima
  - Data ultimo accesso
  - Data ultimo aggiornamento
  - **Proprietario del file**
  - **Informazioni sulla protezione**

Operazioni sulla directory

- Il file system deve fornire le seguenti operazioni sulla directory:
  - **Ricerca** di un file
  - **Creazione** di una directory
  - **Cancellazione** di una directory
  - **List** della directory
  - **Attraversamento del file system** (visita a tutto il contenuto del file system, particolarmente utile per il back-up). Usato anche dal comando *find*.

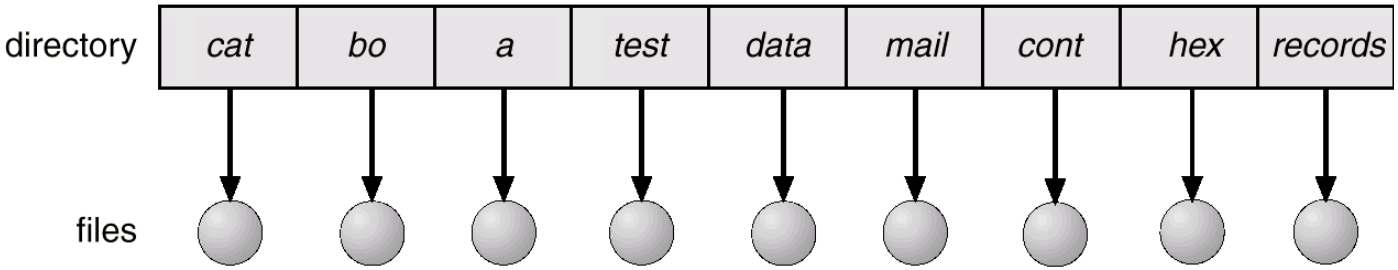
Directory

- **Obiettivi nella realizzazione della directory:**
  - **Efficienza,** nel localizzare un file velocemente.
  - **Gestione dei nomi (Naming)** conveniente per l'utente:
    - Due utenti devono poter dare lo stesso nome a due file diversi
    - Lo stesso file deve poter essere associato a più nomi
  - **Grouping:** i file devono poter essere raggruppati in base alle loro caratteristiche (tutti programmi java, tutti i file word,…) a seconda delle scelte dell'utente, che strutturera' le directory a seconda di come vuole raggruppare i propri files.

Single-Level Directory

- L'organizzazione della device directory più semplice è quella a **livello singolo** (**Single-Level Directory**).
- Tutti i file della partizione sono gestititi mediante una struttura lineare.
- E' particolarmente inefficace quando:
  - Ci sono più utenti (che devono usare nomi diversi per i file!).
  - Ci sono molti file (e anche un solo utente si confonde).

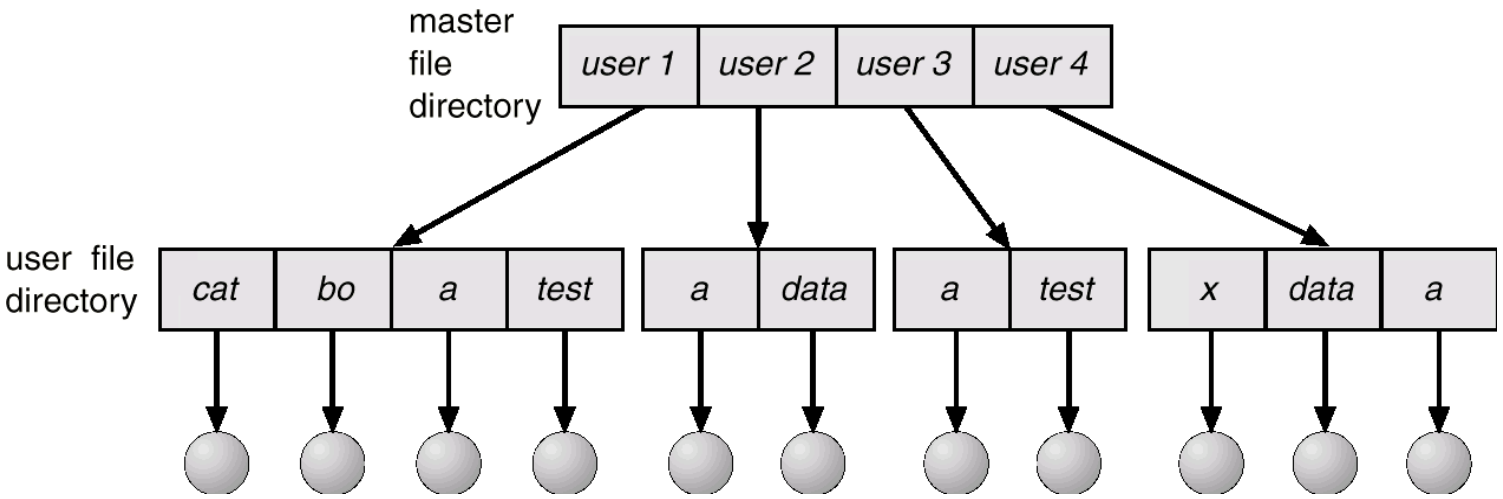
Single-Level Directory



Two-Level Directory

- Il principale problema nella device directory a livello singolo è la gestione dei nomi condivisa tra più utenti.
- Questo problema può essere risolto con una device directory a **doppio livello** (**Two-Level Directory**) in cui ciascun utente ha una sottodirectory separata (detta **UFD**, **user file directory**) e definisce quindi un suo spazio dei nomi.
- Un caso particolare di questo tipo di gestione riguarda gli eseguibili di sistema che l'utente vuole eseguire semplicemente digitando il nome del comando.
  - Il s.o. gestisce una UFD speciale per i comandi e ad ogni richiesta di accesso a un file controlla prima l'UFD dell'utente e poi l'UFD dei comandi (vi ricordate la PATH ?). Il s.o. implementa cioe' dei **percorsi di ricerca** (**search path**).

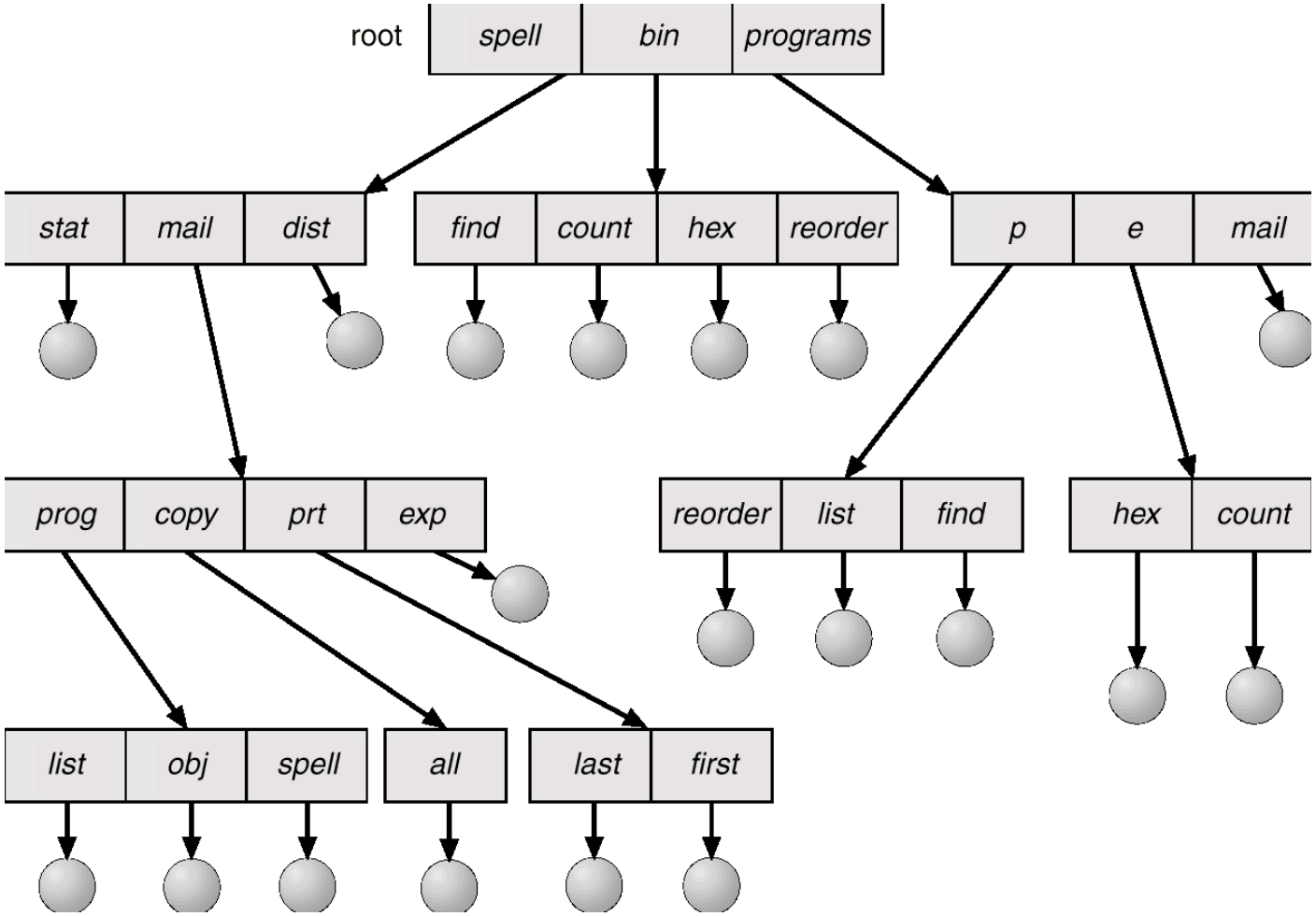
Two-Level Directory



Directory ad albero

- La directory a 2 livelli può essere considerata come un albero a due soli livelli.
- E' naturale estendere il concetto ad alberi in generale, definendo così le **directory ad albero** come strutture ad albero di altezza arbitraria.
- Questa generalizzazione consente agli utenti di creare le proprie sottodirectory e di organizzare a piacimento i file.

Directory ad albero



Directory ad albero

- Quindi:
  - L'albero ha una radice che è detta **root directory**.
  - Una directory (o una sottodirectory) possono contenere file o sottodirectory.
- Un **path name** univoco descrive il percorso che deve essere compiuto dalla root directory, attraverso le sottodirectory, per raggiungere il file.

Directory ad albero

- Per evitare di dover usare sempre l'intero path name del file viene definita una directory corrente rispetto alla quale viene effettuata la ricerca dei file.
- La directory corrente può essere cambiata dell'utente (tipicamente con il comando **cd**, **change directory**).
- I **path name** possono essere:
  - **Assoluti**: partono dalla root
  - **Relativi**: partono dalla directory corrente

STOP

Basta cosi'

Passiamo direttamente alla

Struttura del File System