2. FILE

Il File System è la parte più visibile del SO e fornisce i meccanismi per la memorizzazione e l'accesso ai dati e ai programmi del SO 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

Un file è una collezione di informazioni che sono correlate tra di loro e vengono conservate su una memoria non volatile a cui è stato assegnato un nome che lo identifica univocamente. Rappresentano un'unità logica di memorizzazione. I file contengono dati (numerici, caratteri, binari) e programmi (sorgenti, linkabili, eseguibili). La cosa che è importante da ricordare come descritto in precedenza è che sono conservati su memoria non volatile (dischi, memorie esterne, etc…). I file possono essere di vario tipo:

- **File di testo**: sequenza di caratteri, parole, linee, pagine.
- **File sorgente**: sequenza di subroutine e funzioni.
- **File oggetto**: sequenza di byte organizzate in blocchi che risultano comprensibili ai linker del SO.
- **File eseguibile**: serie di sezioni di codice binario che il caricatore porta in memoria ed esegue.

Per capire un file di che tipo è, s'inserisce alla fine del nome un'estensione. Ad esempio, se un file ha estensione .txt o .doc allora sappiamo che è un file di testo.

Mac OS X conserva, all'atto della creazione di un file, anche il nome del programma che ha creato il file (questo gli da modo di poter riconoscere il tipo del file). **Unix** memorizza un codice (numero magico) all'inizio di alcuni tipi di file in modo da indicarne in modo generico il tipo (eseguibile, script shell, PostScript). Ciascun file è identificato attraverso un nome naturalmente, però ha anche associato un insieme di **attributi** che vanno a specificare le caratteristiche di un file. Questi attributi sono:

- **Nome**: unica informazione in una forma leggibile dagli esseri umani.
- **Identificatore**: tag unico (spesso numerico) che identifica il file all'interno del file system.
- **Tipi**: nei sistemi che supportano differenti tipi di file.
- **Locazione**: la locazione può essere di due tipi:
  - *Posizione fisica:* puntatore alla locazione fisica del file nel dispositivo.
  - *Posizione logica:* il pathname del file, ossia la sequenza delle directory che bisogna percorrere per raggiungere il file (questa informazione NON è memorizzata esplicitamente da nessuna parte eccetto che in casi particolari).
- **Dimensione**: dimensione corrente del file.
- **Protezione:** determina chi può leggere, scrivere, eseguire.
- **Ora, data e identificativo dell'utente**: dati utili per la protezione, la sicurezza ed il monitoraggio d'uso.

Tutte queste informazioni appena elencate sono memorizzate insieme al file, naturalmente sul disco. Capiremo più avanti che gli attributi di un file andranno a formare il **File Control Block**. Vediamo ora quali sono le operazioni che il SO deve garantire sui file:

- **Creazione**: devono essere garantite delle system call che sono capaci di creare un file. Naturalmente il SO deve anche garantire che venga trovato dello spazio nel disco per la corretta creazione.
- **Scrittura / Lettura**: il SO mette a disposizione una opportuna system call per specificare il nome del file su cui operare. Il SO deve gestire il puntatore di scrittura/lettura al punto del file in cui si vuole scrivere/leggere (ad esempio una system call che ci permetta di posizionare il cursore in una qualsiasi posizione del file) e di occuparsi di trovare spazio sufficiente per ospitare l'eventuale espansione del file, in caso di scrittura.
- **Riposizionamento all'interno di un file**: per leggere o scrivere a partire dal punto specificato.
- **Cancellazione**: recupera lo spazio occupato dal file sul supporto di memoria secondaria e lo spazio occupato nella directory che lo conserva.
- **Troncamento**: cancella i dati memorizzati e recupera lo spazio occupato, ma mantiene tutti gli attributi del file.

Un processo, quando utilizza i file, deve tenere in uso una tabella, che viene chiamata **Tabella dei file aperti dal processo**: ogni processo, man mano che apre dei file per poterli utilizzare, annota in una tabella tutte le caratteristiche dei file che sono stati aperti. Questo descritto è relativo al singolo processo.

Il sistema, nella sua interezza deve avere anche lui una tabella dei file, che viene chiamata **Tabella dei file aperti**. Supponiamo che il Processo1 apre file1, file2 e file3 e che il Processo2 apre file4 e file5, nella Tabella dei file aperti ci stanno tutti e 5 questi file che sono stati aperti, in cui file1, file2 e file3 sono associati al Processo1 e file4 e file5 sono associati al Processo2.

2.1 FILE IN UNIX

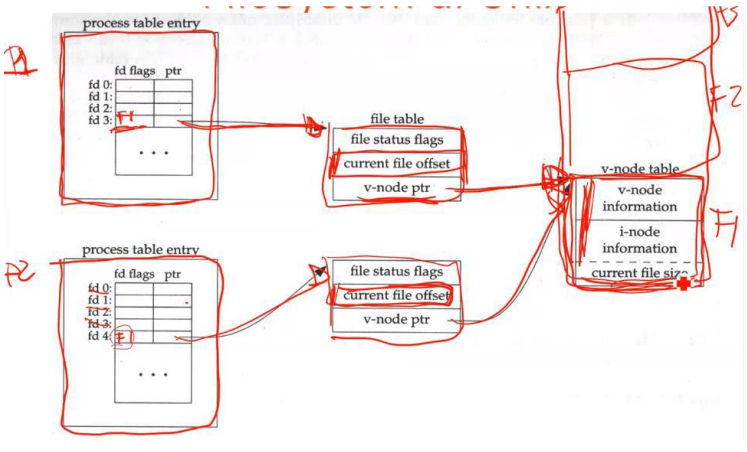
Dal punto di vista di Unix quello che succede quando 2 processi utilizzano lo stesso file è questo:

Supponiamo di eseguire un primo processo P1. Per quanto descritto fino ad ora, quello che succede è che viene messo a disposizione del processo P1 uno spazio in memoria principale. In questo spazio è sicuramente presente il codice del processo P1. Supponiamo che durante l'esecuzione di P1 si incominciano ad aprire dei file (file1, file2, …), allora nello spazio messo a disposizione in memoria principale per P1, esisterà una tabella che è la tabella dei file aperti dal processo, e al suo interno ci sarà una entry per file1, file2, file3 e così via quanti file verranno aperti. Il SO anche lui nel suo spazio avrà una tabella di tutti i file aperti nel sistema e scriverà tutti i file che i processi apriranno.

Supponiamo di eseguire un nuovo processo P2, al quale verrà naturalmente dedicato dello spazio in memoria principale e avrà la propria tabella dei file aperti dal processo. Supponiamo che P2 apre file1 (lo stesso aperto da P1). Dal punto di vista del SO, poiché file1 è stato già salvato nella propria tabella (perché già P1 lo ha aperto precedentemente), dovrà solamente tenere in considerazione che da questo momento sia P1 che P2 utilizzano quel file (tutto ciò mediante contatore).

Osservando l'immagine, associamo al primo process table entry, la tabella dei file aperti dal processo di P1, e la seconda di P2.

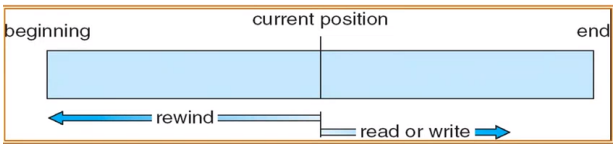
P1 ha aperto, nell'immagine, 4 file e a ciascuno di questi file viene dato un numeretto locale alla tabella, che viene chiamato **F**ile **D**escriptor. P2 ha aperto, nell'immagine, 5 file e anch'esso ha associato ai file un file descriptor.



Unix, per ciascuno dei file aperti nel processo, ha una sorta di puntatore agli attributi generali del file (nell'immagine **file table**), per esempio *file status flags*, *current file offset*. All'interno degli attributi del file, c'è anche un **v-node ptr**, che rappresenta un puntatore alla **v-node table** che sarebbe la tabella dei file aperti dal sistema. Nell'immagine si vede che fd3 del processo P1 e fd4 del processo P2 hanno entrambi un puntatore v-node ptr allo stesso campo nella v-node table: questo perché contemporaneamente hanno aperto lo stesso file, cioè F1.

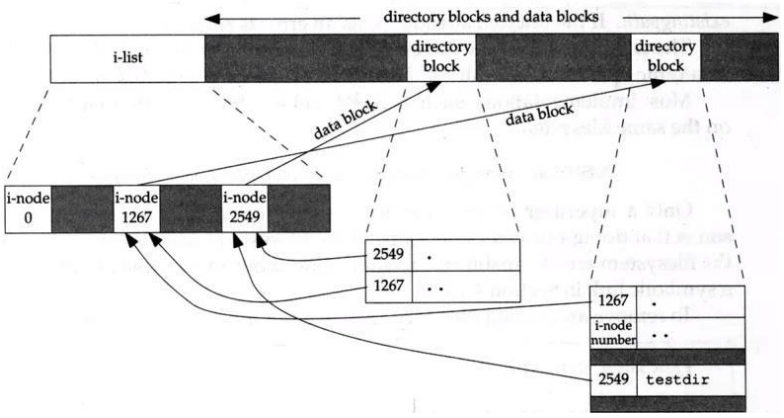
Quando si accede ad un file, ci possono essere due modalità di accesso:

- **Accesso sequenziale**: si accede ai byte presenti in un file uno dopo l'altro in modo sequenziale (un po' come viene fatto per le vecchie cassette). Non si può saltare in una posizione specifica del file, ma bisogna leggere tutti i byte precedenti fino a che non si raggiunge quello x.
- **Accesso diretto**: se l'utente vuole accedere al blocco x, l'accesso avviene direttamente, senza passare per i precedenti.



2.2 ORGANIZZAZIONE DI UN FILE SYSTEM IN UNIX

La struttura dell'organizzazione di un file system di Unix è la seguente:



Supponiamo che tutto il primo rettangolo sia lo spazio a disposizione messo sul disco per il file system. Il file system è partizionato nel seguente modo: c'è un **i-list** che rappresenta tutta la lista degli attributi dei file che vengono riferiti mediante i-node (campo presente nella v-node table visto precedentemente). L'i-node dunque contiene gli attributi dei file (i-node 0 contiene gli attributi del file 0 e così via…).

In Unix, tutti gli attributi dei file vengono posizionati all'inizio del file system dunque. Fisicamente i file sono poi memorizzati all'interno di **directory block**, che come suggerisce il nome, sono delle directory che contengono al loro interno i file stessi. Ad esempio, nell'immagine, all'interno della prima directory block sono presenti due file identificati con 2549 e 1267 (che sono gli **i-node number**). Su 2549 è presente un solo puntino e questo indica la directory del file è la corrente, mentre per 1267 ci sono due punti è questo indica la directory padre.

Ogni file contenuto nelle directory è identificato mediante i-node number, e questi punteranno all'i-node contenuto nell'i-list, il quale contiene al suo interno tutti gli attributi di un file.

Organizzare logicamente i file in directory permette di ottenere:

- **Efficienza**: localizzazione rapida dei file.
- **Assegnazione di nomi conveniente per gli utenti**: due utenti possono scegliere lo stesso nome per file differenti.
- **Raggruppamento**: raggruppamento logico di file in base alle proprietà

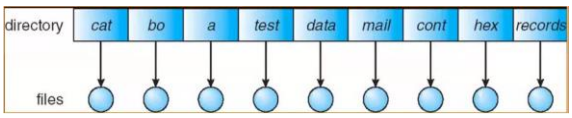
2.3 STRUTTURE DIRECTORY

Partiamo dal caso di gestione banale fino ad arrivare a quelli più complessi ed utilizzati nei nostri sistemi:

DIRECTORY A SINGOLO LIVELLO:

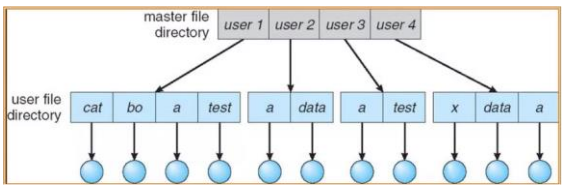
Tutti i dati sono contenuti in una sola directory per tutti gli utenti. Di conseguenza non abbiamo la possibilità di crearne altre. Tutti i file che abbiamo sono quindi aggregati.

Banalmente, questa gestione "estrema" crea molti problemi perché siamo costretti ad utilizzare nomi diversi per ciascun file, non si possono differenziare se non per il nome. I file non possono essere raggruppati in maniera logica.
Non viene mai usato, viene mostrato solo per definizione.



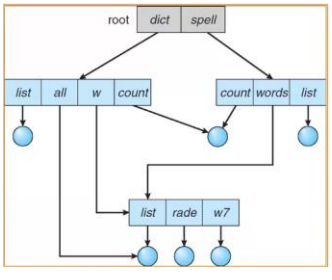
DIRECTORY A DUE LIVELLI:

In questo caso si crea un primo strato di directory (per esempio una per ciascun utente), e i file degli utenti vengono dunque separati. In questo caso c'è un minimo di miglioramento rispetto al caso precedente, è possibile applicare una ricerca efficiente.



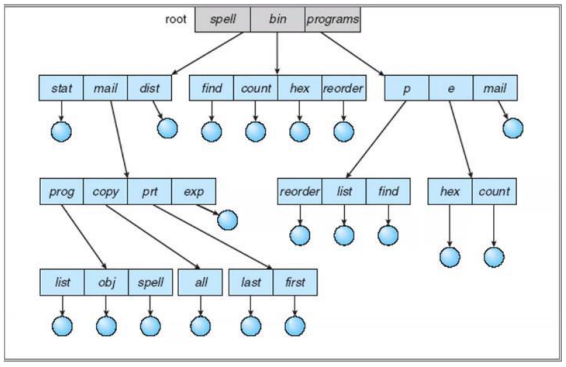
STRUTTURA A GRAFO ACICLICO:

Rappresenta un ulteriore avanzamento della struttura ad albero in cui ci sono file o sottodirectory condivise. Per esempio, osservando l'immagine, si consideri la directory list, questo si trova all'interno di tre directory. Viene implementata sotto UNIX con gli hard link.



STRUTTURA AD ALBERO:

Rappresenta la struttura a cui noi siamo abituati, in cui c'è una root iniziale dell'albero che ha le sue directory, ogni directory contiene i suoi file con eventuali sottodirectory poste all'interno. In questo modo la ricerca è efficiente, c'è alta capacità di raggruppamento, etc…



2.4 MONTAGGIO DI UN FILE SYSTEM

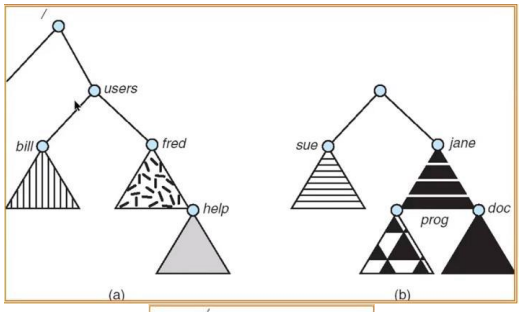
Avviene un montaggio di un file system quando si vuole agganciare un file system nuovo ad un file system esistente. Un po' come quando si inserisce una penna usb.

Sotto Windows, quando s'inserisce una penna usb, viene assegnata una porta con una lettera al dispositivo esterno (per esempio "E:"). In questo modo il file system che era presente prima dell'inserimento della penna usb, rimane staccato da quello che viene inserito esternamente.

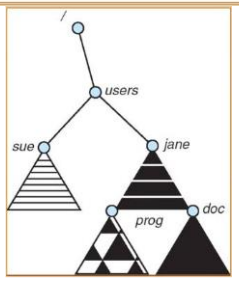
Sotto UNIX, invece, la penna usb viene agganciata ad una delle directory che sono già presenti all'interno del file system. Il file system della penna usb verrà visto come parte integrante dell'intero file system del sistema operativo. In UNIX si ha la libera possibilità di scegliere la cartella sotto la quale agganciare il nuovo file system. A quel punto cosa succede però? Assumiamo di fare l'aggancio sotto ad una directory, ma se in quella directory era presente qualcosa (file, directory, …), che fine fanno queste informazioni? Vengono oscurate. Non si vedranno più le informazioni che c'erano sotto quelle directory e vengono momentaneamente sostituite (non perse) da quelle del dispositivo esterno appena inserito. Nel momento in cui si rimuove il dispositivo esterno, si rivedranno le informazioni che erano state precedentemente oscurate.

Facciamo un esempio →

Osservano la figura (a), che rappresenta il file system esistente, mentre la figura (b) rappresenta il file system di una penna USB. Supponiamo di voler effettuare il mount di (b) all'interno della cartella users della figura (a). Quello che succede, come descritto precedentemente, è che il contenuto di users (bill, fred, help) viene temporaneamente oscurato e si vede sostituito da quello del file system della penna USB.



Il risultato è dunque il seguente →



Per proteggere file e directory, viene assegnato a ciascuno di essi un **User ID** e un **Group ID**.

I permessi di accesso di un file/directory sono: lettura, scrittura, esecuzione. Ci sono tre classi di utenti:

		RWX
▪ Accesso proprietario 7	→	1 1 1
▪ Accesso gruppo 6	→	1 1 0
▪ Accesso pubblico 1	→	0 0 1