Capitolo 13 : File System

Table of contents

Un **File** è uno spazio indirizzi **logico continuo**; in un file sono presenti dei **dati** che possono essere sotto forma **numerica, carattere** (usando la stessa codifica binaria per le variabili usate nella CPU).

Infatti è molto diverso scrivere "3.14" come **caratteri** e scrivere "3.14" come **floating point**.

In un file potrebbe essere presente un **programma**, nei vari formati:

- Sorgente
- Oggetto
- Eseguibile

In generale un file è un oggetto **astratto** i cui contenuti sono definiti dal creatore; solo il creatore del file, infatti, è a conoscenza di quello che è presente all'internod del file.

Quali sono gli attributi associati ad un file?

associati ad un file possono essere presenti una serie di informazioni:

- **Nome** in chiaro del file, che è l'unica informazione del file mantenuta in **human readable** , ovvero il nome del file, leggibile da un umano, che internamente è molto più astratto.
- **Identificatore** è diverso per ogni file che lo identifica univocamente.
- **Tipo** alcuni sistemi antichi prevedevano una serie di tipi di file predefiniti. Ai nostri giorni i SO continuano a vedere i file come una sequenza di byte, successivamente a seconda dell'estensione il SO capisce che tipo di file si ritrova a dover gestire. Una volta capito che tipo di file si tratta, il SO lo manda al gestore adatto (ad esempio .txt si apre un editor di testo).
- **Posizione - locazione** ci dice dove il file è locato nel disco, in modo da permettere al SO di accedere alla locazione di memoria dove il file è situato.

- **Grandezza** , ovvero la grandezza **corrente** del file.
- **Informazioni di protezione** , ovvero delle informazioni che consentono di evitare che il file venga manipolato, modificato da utenti che non sono autorizzati.
- **Tempo data ed identificazione utente**

Tutte queste informazioni sono presenti **nel contenuto della struttura di directory**; esiste una directory che non è altro un contenitore contenente tutte queste informazioni.

La posizione di queste informazioni varia da SO a SO, ma in ogni caso esse **devono essere presenti.**

Dimensioni correnti - approfondimento

Quando leggiamo: `Grandezza: 100 bytes (115 KB sul disco)` cosa vuol dire?

Fondamentalmente lo spazio sul disco viene allocato a **blocchi**, che nella loro grandezza minima sono di 512bytes. Questo significa che c'è un minimo di **frammentazione interna**, e quindi viene assegnato più spazio di quello necessario.

Operazioni sui files

Sui files si ragiona in termini di dato astratto, nel senso che sono state definite le operazioni indipendentemente da come esse vengono implementate.

- Create
- Write - Alla posizione **del puntatore write**
- Read - Alla posizione **del puntatore read**
- Riposizionare tra il file

- Delete
- Truncate
- Open(Fi)
- Close(Fi)

La lettura e scrittura avvengono sempre a partire dalla posizione **corrente** all'interno del file; il SO gestisce una posizione corrente all'interno del file, che è la posizione alla quale avverrà la prossima lettura o scrittura.

Questo ci permette, quando leggiamo/scriviamo sequenzialmente, di dire che la prossima volta che si andrà a leggere/scrivere, non sarà da capo, ma nel punto immediatamente successivo.

Il puntatore all'interno del file, viene gestito automaticamente dal SO.

Accesso randomico (riposizionamento - seek)

Questo metodo funziona bene nel momento in cui la lettura/scrittura è **sequenziale**, ma cosa succede quando voglio accedere al file in una posizione random?

Il read e write continua a funzionare allo stesso modo, ma si riposiziona all'interno del file, spostando semplicemente il puntatore.

Quindi, l'operazione di **seek** (ricerca) significa spostare il puntatore in una certa posizione all'interno del file.

Open() e Close()

Open()

E' un'operazione che consente, dato un nome simbolico del file, di individuare il file, e predisporre il tutto per l'accesso.

Questo vuol dire che da quel momento il file viene reso noto all'interno del processo che lo ha richiesto, e ci sarà quindi un puntatore che verrà avanzato ad ogni lettura e scrittura.

Close()

Elimina il procedimento che **elimina il file dalle tabelle del processo**, e se ci sono dei buffer, trasferisce il tutto.

Questa operazione ci assicura che tutto ciò che è presente in un buffer non ancora trasferito verso il disco, venga iniziato ad essere spostato sul disco (l'operazione non è immediata).

Files aperti

Il SO utilizza una **tabella dei file aperti** per tenere traccia dei files aperti (meglio di così non si poteva).

Questa tabella consente di accedere al puntatore all'interno del file che corrisponde alla posizione **corrente** all'interno del file, utilizzando delle operazioni di lettura/scrittura.

Supponiamo di aver aggiunto ad un file word un'immagine. Se proviamo ad eliminare l'immagine dal disco, il SO ci consente l'operazione, ma il file non viene realmente eliminato finchè il processo non sarà terminato.

Se ci sono effettive aperture, il SO rimuove il file dalla tabella dei file aperti, solo nel momento in cui non ci sono più aperture.

Open File Locking

Questa è un'aggiunta relativamente recente, infatti i primi sistemi UNIX non permettevano il Locking dei file.

Non bisogna pensare al fatto che un file venga aperto da più processi come una cosa negativa; che succede quando un processo vuole **scrivere** su un file aperto anched altri processi?

Abbiamo il solito problema in un contesto leggermente diverso: se più processi tentano di scrivere contemporaneamente sullo stesso file, abbiamo un gran problema; se diversi processi, che hanno **diversi puntatori nel file**, scrivono sullo stesso file, scrivono in due punti diversi, cosa che non genererebbe particolari problemi.

Se voglio assicurarmi che un solo processo alla volta scriva sullo stesso file, devo richiedere un **lock** su quel dato file:

Questo significa dire a sistema operativo che quel dato processo sta lavorando su quel dato file, e nessun altro processo deve accedervi mentre quel processo vi scrive.

Lock shared

In alcuni casi è possibile anche avere un lock shared, ovvero bloccare **solo una sezione del file**, e quindi permettere ad altri processi di lavorare sulle restanti porzioni del file.

Tipi di lock

I lock possono essere di due tipi:

- **Mandatorio (obbligatorio)** ovvero l'accesso è **negato** assolutamente.
- **Opzionale** in questo caso viene semplicemente **"consigliato"** al processo di non accedervi, ma la decisione spetta solo al processo.

Struttura dei file

Nella maggioranza dei casi, i files **non hanno una struttura**, nel senso che sono semplicemente una **sequenza di word o byte** che vengono interpretati dal programma che gestisce quel tipo di file (estensione).

In altri casi i files potrebbero avere una struttura complessa, gestibile da caratteri di controllo in modo da costruire un file più complesso.

Accesso sequenziale ai file

L'accesso sequenziale ad un file, significa avere un puntatore che punta alla posizione corrente, le letture/scritture avvengono da quella posizione in poi;

Qualora volessi tornare all'inizio del file, invece di effettuare una **seek() (ricerca)**, è possibile effettuare un **rewind** che mi consente di spostarmi all'inizio del file in modo **sequenziale (senza salti)**.

Metodi di accesso

Accesso sequenziale

```
read next
write next
reset
```

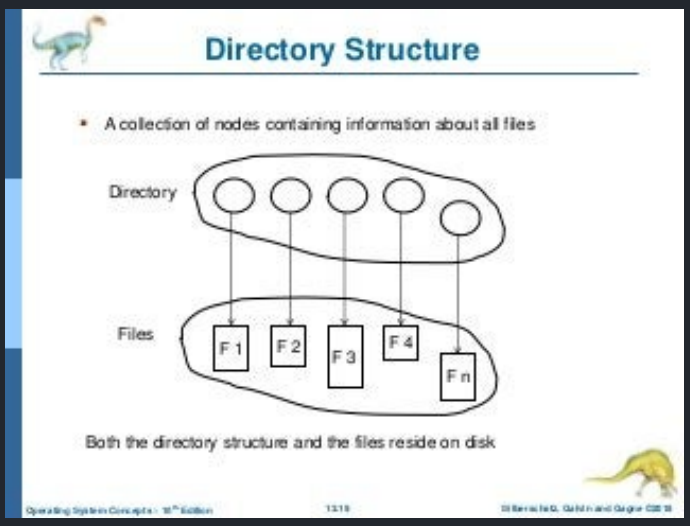
Accesso diretto

```
read n
write n
position to n
    read next
    write next
rewrite n
```

Nell'accesso diretto, bisogna posizionarsi in una posizione precisa con un'operazione di **seek()** e poi (opzionalmente) effettuare una lettura/scrittura sequenziale

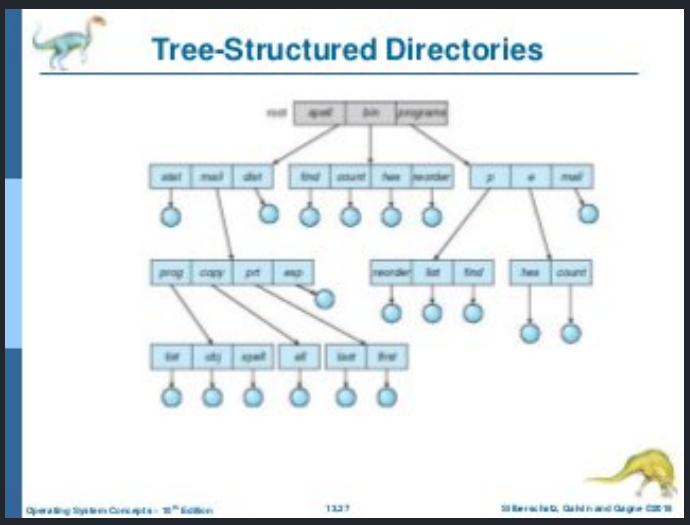
Struttura di una directory

La directory contiene informazioni relative a files, e storicamente la struttura delle directory era fatta in questo modo:



C'era un'unica struttura di directory in cui erano presenti **tutti i files**.

Struttura ad albero per le directory



Nella realtà, già dall'MS-DOS la struttura del file system è ad albero, che permette di avere files aventi lo stesso nome, purchè in directory diverse, oltre a poter separare i documenti dei vari utenti ecc.

Partizione di un disco

Fondamentalmente un disco può essere visto come un unico volume, oppure in maniera partizionata.

Il partizionamento è particolarmente utilizzato dai produttori, anche per motivi di recovery, dove su un'unica unità fisica sono creati più volumi logici. Associata ad un disco, viene associata una **tabella delle partizioni** che individua la posizione di inzio e fine, di unità che **logicamente** sono separate.

Il sistema vede queste **partizioni** come tante unità logiche separate, ma in realtà sono tutte sullo steso disco.

Il vantaggio delle partizioni

Storicamente il vantaggio delle partizioni risiede in una **migliore protezione e migliore backup**: Possiamo ad esempio creare una partizione in cui saranno presenti solo programmi di sistema, renderla solo lettura, in modo che non possa essere alterata o distrutta.

Inoltre, per effettuare un backup, è difficile effettuare il **backup per files utenti** tra un unico enorme disco; conviene maggiormente creare una partizione di files utente, in modo da effettuare il backup dell'intera partizione.

File system

Tipi di file system

Storicamente ogni processore aveva il proprio file system, che non poteva essere cambiato.

Ai giorni d'oggi, l'interfaccia è pressocchè la stessa per ogni sistema, ma ci sono in realtà diversi tipi di file system. Infatti, per ogni unità di archiviazione ha un diverso tipo di file system, in modo che si abbia il FS più ottimizzato per ogni occasione.

Operazioni effettuate su Directory

La differenza fondamentale tra le directory ed un file normale. è che le directory vengono scritte **solo dal sistema operativo.**

- Ricerca di un file
- Creare un file

- Eliminare un file
- "listare" (list) una directory : mostrare tutti i file al suo interno
- Rinominare un file
- "traversare" (traverse) il file system

Tutti i sistemi operativi utilizzano un file system strutturato **ad albero**.

Accedere ai files

Solitamente è possibile accedere ai files con un **pathname** assoluto o relativo; un path assoluto è un tipo di path che parte dalla **/root**, ovvero l'inizio del file system, ed arriva al files; termina con il nome del file:
`/root/pianosotto/altropiano/nomefile.txt`

Problematica dei grafi aciclici

Come faccio a cercare, all'interno di un FS ad albero, un file?

Fondamentalmente quello che bisogna fare è una ricerca in profondità o ampiezza (BFS o DFS) finchè non trovo il file.

Da linea di comando un file viene ricercato con il comando find:

```
$ find . -iname '*foo*'
```

> Ricerca di un file non case-sensitive

Che succede quando questo albero "non è più un albero"?

Un albero è un particolare grafo **non avente percorsi chiusi**; cosa succede se invece si verificano dei percorsi chiusi?

Questi particolari percorsi si verificano quando **voglio dare un doppio nome allo stesso file**; pur sembrando una pratica strana, è molto utilizzata:

Quando abbiamo uno **shortcut** di un programma (ad esempio sul desktop), abbiamo un perfetto esempio di questo fenomeno. Questo collegamento non è altro che un link al programma originale.

Nel momento in cui un singolo file è visibile da più directory, potrebbe esserci un problema in cui si verificano **dei percorsi chiusi**.

Il problema è che gli algoritmi di BFS e DFS funzionano su grafi **aciclici**, e di conseguenza in presenza di un loop l'algoritmo rimarrebbe bloccato all'interno del loop. I sistemi operativi possono agire in due modi per difendersi da questa eventualità:

- **Impedire la creazione di link** che creerebbero loop; un chiaro esempio è linux.
- **Uscire dal loop** nonappena ci si accorge di aver ciclato più di una volta.

Creare link

Il metodo storico di UNIX per creare un link, era quello di usare il comando **ln** (link); il comando "ln" senza parametri, permetteva di stabilire un **hard link**: anche se ci troviamo in directory diverse, con files a nomi diversi, in realtà questi puntano allo stesso file (spazio su disco). Questo è possibile se ci troviamo sullo stesso volume del file originale.

Per evitare che il file si trovi sullo stesso volume, sono stati aggiunti i cosiddetti **link simbolici**:

Entriamo in una dir, e al suo interno troviamo semplicemente un collegamento che non è altro che la traduzione di un **pathname** che ci conduce da un'altra parte.

I link simbolici si creano con il comando

```
ln -s {parametri}
```

I link di windows sono proprio dei **link simbolici**; Windows permette gli hard link, ma non fornisce strumenti per crearli (perchè non viene utilizzato dagli utenti babbani).

Eliminazione del file da una dir

Nel momento in cui creiamo un hard link, lo stesso file può essere aperto da due dir diverse. Se il file viene eliminato da una delle due dir, cosa succede?

Evidentemente, non è possibile eliminare il file completamente, se ci sono dei link che puntano ad esso. Bisogna conservare, tra le varie informazioni del file, il numero di puntatori (link) che puntano ad esso. Solo quando il numero di link che puntano al file diventa zero, il file viene fisicamente eliminato.

Tutto questo avviene nel momento in cui abbiamo degli hard link

Nel caso dei link simbolici, invece, non viene tenuto alcun conteggio, e se viene eliminato il file originale, il link dirà qualcosa del tipo *"file non trovato"*

File System Mounting

In windows, ogni volta che viene inserita una nuova unità, ci si trova un disco rimovibile, o qualcosa del genere.

Nel mondo UNIX, tutte le Dir vengono incastrate in un unico albero, mediante delle operazioni di **montaggio** (mount).

A differenza del mondo Windwos, dove abbiamo delle unità logiche (come C:, D:, ecc), in UNIX abbiamo una directory principale chiamata **/root** (solitamente indicata con "/"), dopodichè ci sono delle dir, solitamente vuote, dette **punti di montaggio (mount points)**; tutto ciò che viene "montato" è presente sotto la directory "**mnt**", in Mac chiamato "**/volumes**".

Storicamente, le periferiche andavano **montate e smontate a mano**, usando il comando:

```
mount {}    //montare l'unità
umount {}   // smontare l'unità spostando tutti i dati nelle caches nell'unità
```

Per adeguarsi ad un mondo dove non tutte le persone sono in grado di compiere queste operazioni, la maggior parte degli OS hanno dei **deamons** (programmi di sistema) che appena rilevano la presenza di una nuova unità la montano automaticamente.

Non rimuovere le chiavette senza smontare!

Nei sistemi come windows è possibile rimuovere le chiavette (esempio classico) senza espellerle; questo perchè windows, abituato ad essere utilizzato da utenti non "studiati", ha deciso di non usare alcuna tecnica di **caching**;

Nei sistemi più seri, come linux, mac os, ecc, viene utilzzata la tecnica del caching, quindi se rimuoviamo delle periferiche senza espellerle, dei dati potrebbero non essere stati ancora spostati sull'unità, avendo quindi una perdita di dati, o peggio una corruzione della chiavetta.

File systems remoti

Possiamo montare delle directory appartenenti ad unità remote ed utilizzarle come se fossero delle dir locali.

Ci sono dei protocolli ben precisi per questi tipi di scambi:

- **NFS** protocollo di condivisione UNIX
- **CIFS** protocollo standard windows (anche se probabilmente SAMBA è più usato)

Resta il problema della consistenza, perchè ovviamente se abbiamo più utenti che si connettono ad un unità remota, le modifiche di un utente potrebbero essere viste in ritardo da un altro, ed altre problematiche del genere.

Semantica della consistenza

Che l'accesso sia locale o remoto, se ci sono più accessi contemporanei, e più modifiche, evidentemente bisogna gestire opportunamente queste modifiche.

La semantica classica di UNIX è: Nonappena le modifiche vengono applicate, tutti gli utenti le visualizzano; questa politica funziona molto bene in locale, ma qualora si avesse un file in remoto (rete) non è la scelta migliore. Questo perchè tramite la rete, il sistema operativo di ogni utente crea, senza che l'utente se ne accorga, una copia del file, che poi modifica in locale.

Protezione

Come si proteggono i files?

Il creatore del file o il suo proprietario, vorrebbero dichiarare quali operazioni possono essere effettuate sul file. Il metodo classico di UNIX (**DA CONOSCERE!**) fa utilizzo di 9 bit.

Distingue gli accessi di:

- **accesso del Proprietario**
- **Accesso del gruppo del proprietario**
- **tutto il resto**

Per visualizzare il **proprietario di un file si digita in shell:**

```
ls -l (filename)
```

E' possibile stabilire dei **gruppi** in cui si garantisce l'accesso ad un intero gruppo, e non un singolo utente.

Si utilizzano i bit per garantire l'accesso nel seguente modo:

	R	W	X
accesso del proprietario	1	1	1
accesso del gruppo	1	1	0
Accesso pubblico	0	0	1

> R: Read
>
> W: Write
>
> X: Execution

La "X" in unix è comunemente usata per dire che un qualche file è eseguibile. La X può comparire anche sotto delle directory (non indica che essa è eseguibile).

I permessi possono essere cambiati dal **proprietario** tramite un comando chiamato **chmod**

Valore binario (`rwx`)	Valore decimale	Permessi
111	7	lettura, scrittura ed esecuzione
110	6	lettura e scrittura
101	5	lettura ed esecuzione
100	4	solo lettura
011	3	scrittura ed esecuzione
010	2	solo scrittura
001	1	solo esecuzione
000	0	nessuno

Esempi:

```
chmod 734 nomefile
```

assegna tutti i permessi all'utente, scrittura ed esecuzione per il gruppo e solo lettura per tutti gli altri.

```
chmod 777 nomefile
```

assegna tutti i permessi all'utente corrente, al suo gruppo ed anche a tutti gli altri.

```
chmod -R 777 nomedirectory
```

come il precedente ma riguarda una directory e tutti i file esistenti all'interno della stessa.

Se ad esempio creiamo un file con

```
touch prova.txt
```

Il file viene creato con le impostazioni di default, ovvero scrittura/lettura-lettura-lettura.

Per cambiare uno dei parametri, ad esempio la sua eseguibilità usiamo chmod:

```
chmod +x prova.txt
```

Il file è ora eseguibile.

Per eseguirlo si usa il comando:

```
./prova.txt
```

Ovviamente un .txt non può esssere eseguito.

File che iniziano con "."

I file che iniziano con il carattere "." sono comunemente dei file nascosti, ad esempio i file .lock sono dei file che servono a mettere un lock su un qualcosa per evitare accessi concorrenti.

Essi possono essere visualizzati con il comando:

```
ls -la
```

I file che iniziano con ". ." sono degli **hard link** assegnati alla directory presente e a quella precedente.

Access control list

Questo metodo non è molto selettivo, ed il metodo più serio, implementato anche nei sistemi più importanti, è il sistema access-control list.

Con questo metodo, posso dire per ogni file, quali sono gli utenti che possono accedervi e quali operazioni possono compiere.