1 Lezione del 03-12-25

1.1 File system

Nella maggior parte dei S/O general purpose odierni è definito un componente destinato alla gestione dei **file system**. Un *file system* è un sistema che governa l'organizzazione e l'accesso ai *file*, spesso allocati su dispositivi a blocchi come i *dischi*.

Con un file system andiamo quindi a realizzare tutta una serie di concetti astratti, fra cui:

- Il **file**, unità logica di memorizzazione dati;
- La **directory** (o *direttorio*), insieme di file o altre directory;
- La **partizione**, un insieme di file associati ad un particolare dispositivo fisico (o una sua porzione).

File e directory rappresentano i nodi di una struttura ad *albero*. Le caratteristiche di file, direttorio e partizione sono del tutto indipendenti dalla natura e dal tipo di dispositivo fisico utilizzato. Sono, appunto, *astrazioni*.

1.1.1 Organizzazione logica del file system

Il file system è, come tutti i moduli del sistema, una struttura gerarchica:

- Il livello più alto è quello *logico*, dove esiste solamente l'astrazione di file e directory. Questo è il livello che viene offerto alle *applicaziioni*;
- Segue il livello di *accesso*, che governa le modalità in cui si accede ai file (sequenziale, diretta, ecc...), e i vari meccanismi di protezione che possono essere implementati;
- Segue il livello di *organizzazione fisica*, che riguarda l'allocazione dei file nei blocchi fisici. Vediamo infatto ogni *disco virtuale* (più informazioni sotto) come un vettore di blocchi fisici, che vengono quindi distribuiti fra file;
- In fondo c'è quindi l'astrazione del *dispositivo virtuale*, costruita al di sopra dell'hardware (la memoria secondaria, cioè i dispositivi a blocchi), e visto come già introdotto come un vettore lineare di blocchi fisici.

1.1.2 File

Un **file** è un insieme di informazioni, rappresentate secondo un insieme di *record logici* (bit, byte, parole, ecc...). In UNIX il record è 1 byte.

Ogni file è ulteriormente caratterizzato da un insieme di *attributi*, cioè metadati che contengono il file:

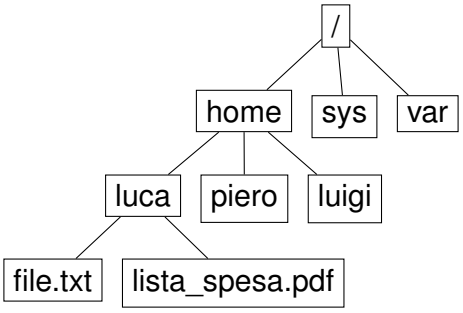
- **Nome** del file;
- **Tipo** del file (si distingue fra file eseguibili, batch, di testo, ecc...);
- **Indirizzo** del file nella memoria secondaria;
- **Dimensione** del file, cioè il numero di record da cui è composto in memoria secondaria;

- **Timestamp** di creazione del file, e di ultima modifica.

Nei sistemi operativi multiutente, inoltre, si vuole includere informazioni riguardo all'utente **proprietario** del file, e i **permessi** degli altri utenti riguardo alla manipolazione del file.

1.1.3 Alberi di file

Abbiamo tutti nota l'organizzazione ad albero dei moderni file system. A scopo di ripasso, notiamo che si definisce un primo direttorio detto *radice*, rappresentato in sintassi come `/`. Per individuare un oggetto (un'altra directory o un file) a partire dalla radice si continuano a frappore nomi di oggetti fra `/`. Ad esempio, `/home/luca/file.txt` cerca il file `file.txt`, nella directory `luca`, a sua volta nella directory `home`, a sua volta nella diectory radice:



Notiamo che in verità la presenza di *link* all'interno del file system, cioè riferimenti allo stesso file fisico da più locazioni, rende l'astrazione migliore non più l'albero ma il **DAG** (*Directed Acyclic Graph*), cioè il grafo aciclico diretto (che rispecchia un albero ma permette la connessione fra nodi con radici mutualmente diverse).

1.1.4 Operazioni su file system

Su un file system dobbiamo permettere, di base, un insieme minimo di operazioni:

- **Creazione/cancellazione directory**: modificano la struttura logica del file system, aggiungendo/eliminando rami al grafo che rappresenta il file system;
- **Aggiunta/cancellazione file**: inseriscono nuovi dati all'interno del file system;
- **Listing**: generano listati dei contenuti delle directory;
- **Attraversamento directory**: permettono il passagigo da una directory all'altra, e quindi la navigazione del file system.

1.1.5 Descrittore di file

Per realizzare l'astrazione del file dobbiamo implementare un qualche tipo di struttura dati per la sua rappresentazione, cioè un **descrittore di file**.

Questo conterrà gli attributi già notati in 20.1.2. I descrittori di file devono essere memorizzati in modo persistente, e quindi vengono allocati in apposite strutture in memoria secondaria. In particolare, ricordiamo la terminologia UNIX di *i-node*, *i-list* e *i-number*.

Per la rappresentazione delle directory, che assumiamo come categorie particolari di file, dobbiamo sicuramente mantenere collegamenti ai descrittori di tutti i file che questa contiene.

1.1.6 Accesso ai file

Il compito del S/O p quello di consentire l'accesso *on-line* ai file. Le operazioni permesse saranno quelle di **accesso** ai file, cioè:

- **Lettura** di record logici dal file;
- **Scrittura** su file, cioè inserimento di nuovi record logici all'interno del file.

Ognuna di questa operazioni richiederebbe la localizzazione di informazioni sul disco, fra cui ad esempio:

- Gli indirizzi dei record logici a cui accedere;
- Gli altri attributi del file;
- I record logici.

Per migliorare l'efficienza, il S/O mantiene in memoria centrale una struttura dati che registra i file attualmente in uso. Per ogni file aperto vogliamo mantenere il puntatore al file in memoria centrale (più informazioni sotto), il descrittore del file, e la sua posizione nel disco. I file aperti verranno quindi *mappati* in memoria centrale, cioè temporaneamente copiati, durante l'accesso, per aumentare la velocità.

Le operazioni necessarie saranno:

- In fase di **apertura** del file, introduzione di un nuovo elemento nella tabella dei file aperti e eventuale mapping in memoria (se non era già stato fatto) del file;
- In fase di **chiusura** del file, salvtaggio del file in memoria secondaria e eliminazione dell'elemento corrispondente della tabella dei file aperti.

1.1.7 Metodi di accesso

L'accesso ai file può avvenire secondo varie modalità:

- Accesso **sequenziale**
  In questo caso il file è inteso come una sequenza $[R_1, R_2, ..., R_N]$ di record logici. Per accedere al record R_i, bisogna necessariamente accedere prima ai precedenti $R_1, ..., R_{i-1}$ record.

  In questo caso possiamo prevedere operazioni come:

  - `readn(f, &V)`, che permette la lettura del prossimo record logico (col riferimento `$V`) del file `f`;
  - `writen(f, V)`, che permette la scrittura del prossimo record logico (ottenuto col riferimento `V`) nel file `f`.

  Ad ogni modo, il nodo centrale dell'accesso sequenziale è che ognuna di queste operazioni posiziona il puntatore del file al record successivo a quello letto;

- Accesso **diretto**
  In questo caso il file è inteso come un insieme $\{R_1, R_2, ..., R_N\}$ di record logici. Noto l'indice i, si può accedere direttamente all'i-esimo record.

  In questo caso possiamo prevedere operazioni come:

  - `readd(f, i, &V)`, che permette la lettura del i-esimo record logico (col riferimento `$V`) del file `f`;
  - `writed(f, i, V)`, che permette la scrittura dell'i-esimo record logico (ottenuto col riferimento `V`) nel file `f`. Vediamo quindi come il punto centrale dell'accesso diretto è la possibilità per il programmatore di poter specificare un indice specifico a cui scrivere nel file, senza aver bisogno di scannerizzarlo in una direzione o l'altra.

- Accesso a **indice**
  Con l'accesso a indice andiamo ad interporre fra l'accesso al file e il file stesso una struttura a *indice*, che permette l'accesso alle informazioni nel file sfruttando *chiavi*.

  - `readk(f, key, &V)`, che permette la lettura del record logico indicizzato dalla chiave `key` (col riferimento `$V`) del file `f`;
  - `writek(f, i, V)`, che permette la scrittura del record logico indicizzato dalla chiave `key` (ottenuto col riferimento `V`) nel file `f`.

  In questo caso è chiaro che l'accesso al file avviene solo dopo un'operazione ricerca sull'indice, che dovrà essere memorizzato in un altro file, o comuunque in una locazione accessibile al filesystem.

La modalità di accesso è indipendente dal tipo di dispositivo utilizzato, o dalle tecnica di allocazione dei blocchi in memoria secondaria. Chiaramente, tali soluzioni determineranno le loro modalità di accesso, ma vorremo che il S/O faccia da livello di compatibilità per supportare qualsiasi metodo di accesso.

1.1.8 Organizzazione fisica del file system

Abbiamo giò detto che ogni dispositivo di memorizzazione secondaria verrà partizionato in *blocchi*, che possiamo anche dire *record fisici*. In particolare:

- Un **blocco** è l'unità minima di trasferimento nelle operazioni di I/O da e verso il dispositivo *fisico*. La sua dimensione è per questo costante;
- Un **record fisico** è invece l'unità di trasferimenti minima nelle operazioni di accesso file, viste dai processi (e quindi dalle *applicazioni*).

La corrispondenza fra blocchi è record logici è che un singolo blocco può contenere più record logici. Per questo motivo si ha che la dimensione di un blocco è maggiore di quella di un record logico.

1.1.9 Allocazione contigua

Iniziamo quindi a vedere come possiamo mappare i record logici dei file nei vari record fisici. Il caso pià semplice è quello dell'**allocazione contigua**, dove ogni file è mappato su insieme di blocchi fisicamente contigui.

I vantaggi di questo approccio sono:

- Velocità nella ricerca di un blocco: per trovare il blocco contenente l'i-esimo byte di un file allocato a partire dal blocco B basta prendere:

$$i_B = B + \frac{i}{N_{\text{byte}}}$$

  dove N_{byte} è il numero di byte per blocco.

- La possibilità di fornire accesso sequenziale e diretto in maniera molto semplice, che va in qualche modo di pari passo con la caratteristica di accesso rapido appena nominata.

Gli svantaggi sono invece:

- La **frammentazione esterna**: man mano che il disco si riempie, rimangono zone contigue sempre più piccole e quindi inutilizzabili. A questo punto si rende necessario operare il *compattamento* del file system;
- Cercare spazio libero per un nuovo file ha un certo costo. In oltre bisogna fare le dovute considerazioni riguardo agli approcci *first-fit* o *best-fit*;
- Se la dimensione del file cambia, si hanno dei seri problemi generati dalla riallocazione del file, dovesse questo andare ad impattare per allocazione continua regioni già occupate da altri file.