FILE SYSTEM

# FILE SYSTEM E DISCHI

Il file system è un'astrazione offerta dal sistema operativo e rende un supporto fornito dall'hardware qualcosa di più comodo e intuitivo da usare come i File e le directory, inoltre, tramite queste astrazioni si occupa di gestire grandi quantità di informazioni, in modo persistente e condiviso tra più processi. I dettagli di gestione ed implementazione costituiscono il File. Alcuni dettagli sono:

- nomenclatura;
- tipo di file: su Linux vengono usati i magic byte, ovvero, i primi due byte di ogni File identificano il tipo di oggetto. Questo approccio diventa ridondante insieme all'estensione.
- metadati: in questa sezione sono presenti timestamp e maschere di accesso principalmente. Nei sistemi MACOS, viene memorizzato anche il tipo di file e l'app con cui è stato creato.
- operazioni: le operazioni che possiamo compiere sui File sono tipicamente intese come chiamate di sistema e sono le azioni a supporto di questi nuovi oggetti che mette a disposizione il sistema operativo. Nell'ambito delle operazioni che si possono svolgere su tipi di oggetti un cenno particolare va alla condivisione: quando due processi tentano di aprire lo stesso File presente sul file system, si presentano le stesse problematiche presenti in memoria centrale. Per gestire l'accesso anche qui il s.o. mette a disposizione meccanismi di lock che si distinguono in:
  - lock shared: più processi accedono allo stesso File ma in sola lettura.
  - lock exclusive: un solo processo può possederlo per volta e permette di modificare il file.

Un'altra distinsione si può fare tra due meccanismi di lock

- mandatory: in questo caso, i meccanismi di lock sono obbligatori e imposti dal s.o. sui processi.
- advisory: il S.O. non impone il lock ma lo suggerisce, un processo potrebbe ignorare il consiglio e correre il rischio di finire in errore. La responsabilità è del progettista.

3/17 MATERIA

Il s.o. attraverso il PCB di ogni processo tiene traccia anche dei file attualmente aperti attraverso la tabella dei file aperti. In particolare, esiste una tabella globale dei file aperti e poi ogni processo ha la propria, in realtà, queste n+1 tabelle sono collegate poiché le tabelle locali non contengono altro che riferimenti alla tabella globale.

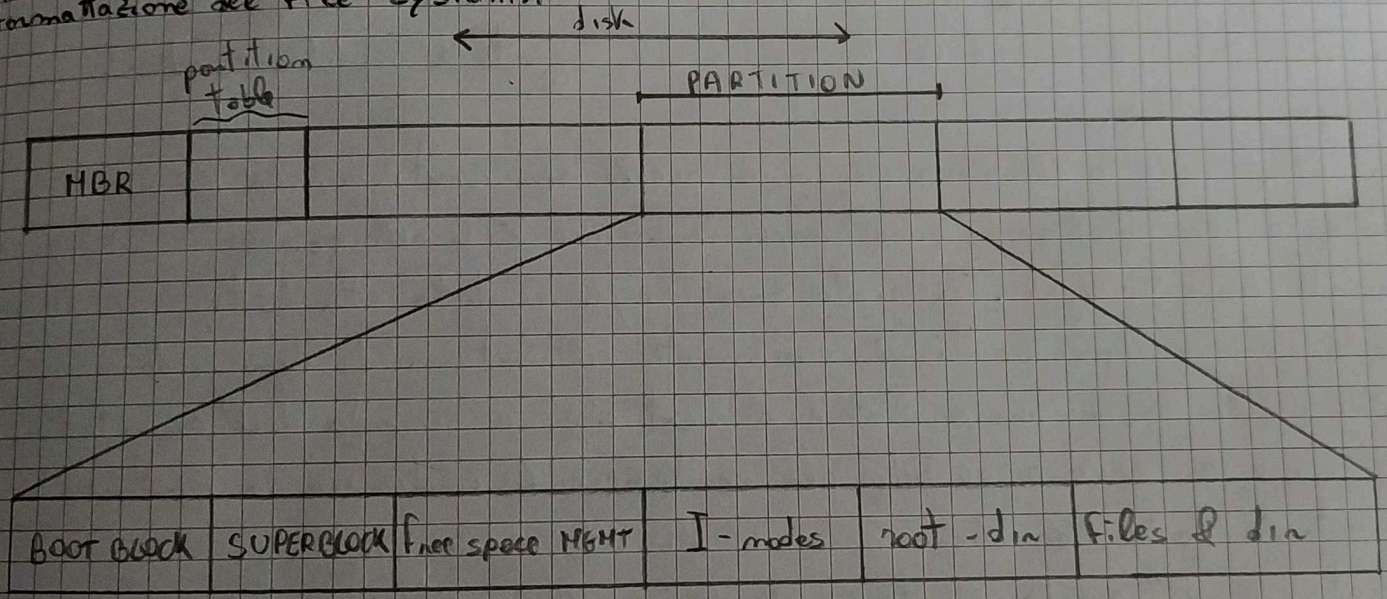
## STRUTTURA DI UN FILE SYSTEM

Parliamo di come organizzare lo spazio di archiviazione di un dispositivo a basso livello, ovvero, del partizionamento di un disco.
Il partizionamento non è altro che un'astrazione che ci permette di creare comparti. Possiamo distinguere due standard per effettuare questa operazione:

MBR: il Master Boot Record si basa sulla tabella delle partizioni ed è il vecchio standard ormai sostituito.
In questo sistema si prevede una partition table che indica le partizioni. Storicamente la tabella contiene 4 voci e permette dunque di creare al più 4 partizioni. Ogni voce indica l'inizio di una partizione e la sua dimensione, in modo da poterne indicare la fine, inoltre, contiene anche alcuni metadati sulla tipologia di file system.
Il master boot record è appunto il primo blocco e contiene un segmento di dati eseguibili che il BIOS legge per avviare il sistema operativo.
La partizione a sua volta contiene una serie di metadati e il primo blocco di ogni partizione che prende il nome di boot block contiene un loader che una volta caricato in memoria e eseguito ha lo scopo di identificare l'immagine del Kernel e portarla in memoria centrale, dopodiché il s.o. prende il controllo totale della macchina.
Un layout moderno alternativo che ricade nello standard EFI (che sostituisce il BIOS) è il GPT.

GPT: il GUID Partition table permette di partizionare dischi di qualsiasi dimensioni e in termini di avvio contiene delle partizioni che contengono vari loader eseguibili per l'esecuzione del s.o.

Scendendo nel dettaglio guardando alla partizione, lo standard è abbastanza libero ma sono presenti alcuni vincoli. In particolare, i primi due blocchi che prendono il nome di boot block e superblock, sono universali e rispettati da tutti gli standard. In particolare, il boot block rimane solo per ragioni storiche di retrocompatibilità, mentre il superblock, è un blocco di metainformazioni e in particolare si trova il tipo di file system usato da quella partizione, in questo modo il s.o. può subito leggere la partizione in modo corretto.
Oltre a questi blocchi troviamo sempre la root directory, ovvero la cartella principale da cui si diramano le altre cartelle e i file del sistema.
Inoltre, nei sistemi UNix, per rappresentare i file si usano gli i-node, ovvero degli oggetti memorizzati sul disco (preallocati in fase di formattazione del file system).



3017
MSTEPA

## IMPLEMENTAZIONE DEI FILE

Il S.O. può decidere vari modi per rappresentare il generico file. La strategia più semplice e universalmente applicabile è quella dell'**allocazione contigua**. Il file viene sempre rappresentato in **blocchi** di dimensione fissa (che potrebbero introdurre frammentazione interna). In questa strategia, il file viene rappresentato dunque da una serie di blocchi contigui. Tra i vantaggi dell'allocazione contigua troviamo la semplicità in termini di gestione, poiché, per tenere riferimento di tutto il file, basta memorizzare il primo blocco di ogni file e la sua dimensione (da cui posso ricavare il numero di blocchi occupati). Un altro vantaggio dato dalla contiguità è anche la contiguità **fisica** sul dispositivo di supporto, questo rende più veloce l'accesso alla memoria poiché abbatte i costi di posizionamento della testina. Questa strategia trova impiego nei dispositivi il cui File system non è soggetto a modifiche, ad esempio i CD.

Un altro modo di allocare memoria è quello non contiguo. La prima strategia per effettuare quest'implementazione è l'allocazione tramite **liste collegate**. La lista sarà implementata tramite blocchi anche non contigui, ovviamente, all'interno di ogni nodo sarà presente anche un **puntatore a next** che, a seconda dell'architettura può essere a 32 o a 64 bit, per contenere l'indirizzo da puntare. La restante parte del blocco sarà spazio allocabile dal file.

Il vantaggio di questa strategia è oltre alla possibilità di allocare memoria in modo non contiguo, l'eliminazione del fenomeno della frammentazione esterna poiché qualunque blocco sarà utilizzabile. D'altra parte però, il singolo blocco viene solo **parzialmente utilizzato** poiché una porzione è riservata al puntatore. Questo disallineamento rende la dimensione non una potenza del 2, questo potrebbe creare delle incongruenze.

Un altro problema è dovuto all'**accesso casuale**, infatti, essendo esso sempre implementato tramite ingresso sequenziale, diventa particolarmente inefficiente su file di grosse dimensioni poiché per leggere l'i-esimo blocco, devo leggere i precedenti i-1 blocchi. A causa di questi problemi, questa strategia non è utilizzata.

Per sfruttare i vantaggi introdotti dall'allocazione concatenata ma evitare i suoi svantaggi, è stata introdotta l'allocazione tabellare che sfrutta appunto la File allocation table (FAT), ovvero, una particolare tabella che gestisce il problema dell'allocazione dei File mantenendo al suo interno i puntatori, liberando dunque i singoli blocchi. La FAT è una tabella monodimensionale dove l'i-esima generica voce indica il blocco successivo a quello che stiamo osservando (se allocato), in altri termini, riporta il puntatore a next dell'i-esimo blocco.
La FAT è una struttura dati globale e non legata al singolo File, dunque è possibile rappresentare più File sulla stessa tabella.
Nella pratica vengono mantenute due FAT speculari per questioni di sicurezza e tolleranza ai guasti. Questo approccio permette di usare la totalità del blocco per il payload del File, inoltre, la FAT viene totalmente caricata in memoria, questo permette al S.O. di calcolare direttamente la posizione dell'i-esimo blocco evitando ritardi e inefficienze. I problemi insorgono nei moderni dischi nel momento in cui carichiamo la FAT in memoria, essa infatti è troppo grande per dischi molto capienti.
Un approccio precedente a FAT ma che risponde alle sue criticità è quello basato su i-node (index-node).
Un i-node è una struttura dati (abbastanza piccola) che rappresenta le informazioni che abbiamo su un dato oggetto sul disco.
L'idea è che un determinato i-node viene caricato in memoria centrale solo nel momento in cui uno specifico File viene aperto da un processo del sistema, questo riduce di molto l'occupazione in memoria centrale a differenza della FAT. Gli i-node sono strutture preallocate e identificate da un i-number, essendo le strutture preallocate, l'i-number permette di individuare la posizione in memoria dell'i-node.
L'i-node contiene tutti i metadati che riguardano l'oggetto, tutti ad eccezione del nome, poiché esso è nella voce della directory che contiene il File. Dentro l'inode sono presenti le indicazioni su dove si trova il File (la lista di blocchi). Per garantire questo servizio, l'i-node è composto da due parti, una che contiene i metadati e un certo numero di voci che sono deputate a contenere

3317 MOTETA

i blocchi che contengono le parti del file. Ovviamente queste voci sono in numero limitato e vanno adattate per file di grosse dimensioni. Una prima idea potrebbe essere quella di far puntare l'ultima voce di quelle predisposte ad un blocco del disco che contiene altre voci che estendono la tabella, qui però si presenta il problema di leggere in modo sequenziale una lista anche se in piccola grandezza. Un altro approccio è quello di creare una struttura ad albero in cui ogni voce dell' i-node punta ad un blocco che contiene altre voci. Questo approccio migliora l'accesso diretto perché mi permette di identificare l'i-node su cui effettuare la ricerca, ovviamente questo approccio gerarchico mi dà l'aggravio di una lettura in più in memoria. Nonostante questo, per file molto grandi dovrei aggiungere gerarchie e aumentare il costo di accesso, questo d'altra parte peggiora l'accesso ai file piccoli. Nella pratica si utilizza un approccio ibrido, ovvero, un i-node reale, contiene una parte di metadati e 13 voci per puntare a blocchi le cui prime 10 voci puntano a blocchi reali mentre le ultime 3 vengono usate per espandere le voci, in particolare l'estensione avviene in modo ibrido (ne gerarchico ne lineare), la terzultima voce punta ad un blocco che contiene un'estensione della lista. Se anche queste voci non bastassero, viene utilizzata la penultima voce che punta a un blocco indiretto doppio, ovvero, voci con ordine gerarchico superiore. che permettono maggiore espansione con overhead ridotto (+2). L'ultimo blocco sarà un blocco indiretto triplo, sarà dunque la radice di un albero a 3 livelli.