File system

Il problema di base è la gestione di grandi quantità di informazioni, in modo persistente e condiviso tra i processi.

Il filesystem è una componente software che permette di gestire l'astrazione dei file e delle directory su un dispositivo di archiviazione. Il filesystem definisce come i dati vengono organizzati e strutturati sul dispositivo, come vengono gestiti i nomi dei file, le cartelle e le autorizzazioni di accesso ai file.
Il filesystem permette di accedere ai dati archiviati sul dispositivo e per effettuare operazioni di lettura, scrittura, modifica e cancellazione dei file. Esistono diversi tipi di filesystem con caratteristiche e prestazioni differenti, che possono essere scelti in base alle esigenze specifiche del sistema in cui vengono utilizzati.

Ogni file system presenta delle differenze nei dettagli della gestione dei file, come ad esempio:

- **Nomenclatura** : il file system può essere case sensitive o case insensitive per i nomi dei file, ovvero può distinguere o meno tra maiuscole e minuscole. Inoltre, il file system può imporre limiti sulla lunghezza dei nomi dei file o sui caratteri ammessi.

- **Tipi di file** : il file system può supportare diversi tipi di file, come ad esempio file di testo, immagini, audio, video, file eseguibili, file di sistema, ecc.

- **Tipi di accesso** : il file system può gestire diversi tipi di accesso ai file, come ad esempio lettura, scrittura, esecuzione, condivisione, permessi di accesso, ecc. Inoltre, il file system può definire regole di accesso specifiche per gli utenti o i gruppi di utenti.

- **Metadati (attributi)** : il file system può memorizzare diversi attributi per i file, come ad esempio la data di creazione, la data di modifica, il proprietario del file, i permessi di accesso, la dimensione del file, l'attributo "nascosto" per i file di sistema, ecc. Questi attributi sono utilizzati dal sistema operativo per gestire i file e per fornire informazioni sui file agli utenti.

- **Operazioni supportate sui file** : il file system prevedono la possibilità di effettuare operazioni sui file come apertura, creazione, spostamento, rinominazione, cancellazione ecc . Si sfruttano dei meccanismi di lock e mutex per la gestione degli accessi concorrenti. Riconducibile al problema lettori-scrittori, in particolare i file system distinguono tra **lock condivisi** (shared lock) e **lock esclusivi** (exclusive lock) . Nel caso di lock condivisi si hanno più processi che aprono in lettura e che possono accedere in modo concorrente al file , quindi il lock è nei confronti di altri processi che cercano di scrivere sul file. Il lock esclusivo è dato tra operazioni in scritture con gli strumenti già visti. Si distingue ancora tra lock **mandatory** ed **advisory** a prescindere dalla natura del lock (shared o exclusive). Il lock mandatory blocca processi che vogliono ottenere l'accesso ad un file già aperto in scrittura da un altro file e non c'è modo di bypassare il lock, mentre il lock advisory segnalano che il lock è stato preso da un altro processo ma permettono comunque l'accesso.

Strutture locali e condivise tra processi :

Nel PCB di un processo è presente una struttura dati per la gestione dei file aperti **locali al processo** :

- a tabella dei file aperti locali al processo contiene informazioni sui file che sono stati aperti dal processo stesso, come ad esempio il loro stato (aperto o chiuso), il puntatore al file, il puntatore alla posizione corrente nel file, i permessi di accesso associati al file e altri metadati relativi al file.

Esiste anche una struttura dati simile ma **condivisa tra processi** :

- la tabella dei file globali e condivisi tra i vari processi contiene informazioni sui file che sono stati aperti da più processi contemporaneamente, come ad esempio i file di sistema o le librerie condivise. Questa tabella contiene informazioni simili a quelle presenti nella tabella dei file aperti locali al processo, ma viene condivisa tra tutti i processi che necessitano di accedere ai file e ne gestisce l'accesso in modo che le informazioni rimangano coerenti tra i vari processi. L'esistenza di questa tabella evita la duplicazione dei file all'interno della memoria di ogni processo, risparmiando spazio di memoria e semplificando la gestione dei file condivisi.

Struttura di un file system

Il disco è un insieme di locazioni di memoria contigue. Si necessita di implementare l'astrazione del partizionamento del disco e si deve tenere conto di dove iniziano e finiscono i vari blocchi di partizioni.

Master boot record (MBR)

Il MBR è un layout di partizione obsoleto utilizzato sui dischi rigidi per definire la struttura delle partizioni del disco. Il master-boot-record si trova nella prima sezione del disco rigido ed è una partizione costituita da codice eseguibile che viene caricata in memoria al momento dell'avvio del sistema (fase di boot).

Il MBR contiene una **tabella delle partizioni**, che definisce la posizione e la dimensione delle partizioni presenti sul disco rigido e contiene un programma di avvio detto **boot loader**, che consente di caricare il sistema operativo dalla partizione di avvio. Il programma di avvio viene eseguito al momento dell'avvio del sistema e cerca la partizione contenente il sistema operativo da avviare.

Il layout del MBR è obsoleto perché supporta solo un numero limitato di partizioni . Questo limite rende difficile la gestione di grandi dischi rigidi e di un numero elevato di partizioni , inoltre non permette di gestire partizioni di dimensioni estese.

GPT (Guide Partition Table) EFI

GPT è un nuovo layout di partizione che non presenta la partizione del Master Boot Record, poiché utilizza un sistema di avvio differente chiamato EFI, il quale offre una serie di funzionalità avanzate, tra cui il supporto per dischi di grandi dimensioni. Per gestire le partizioni, il GPT utilizza una **tabella di partizione**, che è una struttura dati che definisce la posizione e la dimensione delle partizioni presenti sul disco rigido.

GPT supporta un maggior numero di partizioni rispetto al MBR e permette di definire attributi aggiuntivi per le partizioni, come ad esempio il tipo di partizione (ad esempio, una partizione di dati o una partizione di sistema), il flag di avvio (per indicare quale partizione contiene il sistema operativo da avviare) e altri attributi specifici del file system utilizzato. GPT utilizza il sistema di avvio EFI/UEFI al posto del vecchio MBR e offre una maggiore flessibilità e affidabilità nella gestione delle partizioni e dei dischi rigidi.

Implementazione dei file

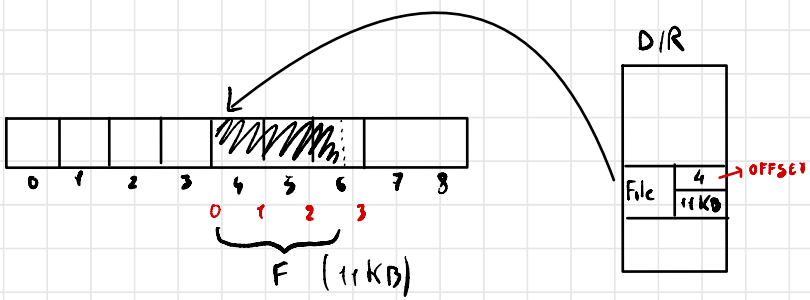
Quando si considera l'allocazione dei file su disco si hanno diverse possibilità di scelta sul tipo di allocazione dei blocchi per i file.

Allocazione contigua :

L'allocazione contigua dei blocchi (o sequenze di blocchi dette **cluster**) prevede la possibilità di frammentazione interna dovuta alla dimensione del blocco stesso. Se la rappresentazione in memoria impone il vincolo di contiguità si ha la possibilità di implementare l'accesso al file in modo efficiente con un overhead minimo implicito nell'operazione stessa.

Nella directory dove è contenuto il file si tiene conto di tre campi :

- nome del file;
- identificativo del primo blocco su disco;
- dimensione effettiva del file;



Vantaggi

L' accesso ad un file, nell'ipotesi di allocazione contigua, si riduce a tener conto di soli due numeri, ovvero l'identificativo del primo blocco su disco e della dimensione del file in blocchi, per tanto è **semplice da implementare**. A partire dal primo blocco si ricavano tutti gli altri blocchi tramite una semplice addizione.
Un secondo vantaggio è dato dal fatto che **l'accesso al file è veloce**, ad esempio per leggere l'intero file è sufficiente la sola ricerca del primo blocco e dopo di che tutti i dati contigui possono essere letti dal disco senza ulteriori ricerche o ritardi di rotazione.

Svantaggi

Tuttavia l'allocazione contigua dei blocchi presenta aspetti negativi, ad esempio negli anni **i dischi si frammentano**, infatti con la rimozione di file si vengono a creare buchi liberi in memoria e la mancata compattazione potrebbe causare un calo nelle prestazioni generali del sistema . Ma un problema più grave è che al **crescere della dimensione** dei file si verificano problemi dovuti al fatto che i blocchi successivi potrebbero non essere liberi e quindi idonei alla crescita della dimensione del file.

Il caso di allocazione contigua è ottimale nel solo caso in cui i file sono di dimensioni fisse a priori come nei CD-ROM, in quanto se la dimensione è nota a priori si può allocare in maniera contigua senza sprechi dovuti alla frammentazione.

Allocazione non contigua

Si pensa di sfruttare l'allocazione non contigua dei blocchi di memoria che contengono il file per permettere l'accrescimento delle dimensioni del file stesso.

- **Approccio con liste collegate (allocazione concatenata)**

Un approccio alternativo per memorizzare i file è quello di utilizzare una struttura di supporto, ovvero una lista concatenata di blocchi di memoria, che supporta l'allocazione non contigua dei file su disco. Ogni blocco della lista contiene un campo NEXT che punta al prossimo blocco di memoria nel quale è memorizzato il file su disco, mentre il resto del blocco contiene dati del file.

Vantaggi

Si ha quindi la possibilità di allocare i file in memoria con un maggior grado di libertà e con la possibilità di **accrescere le dimensioni** dei file. Diversamente dall'allocazione contigua **non viene perso spazio** in frammentazione.

Svantaggi

Il principale svantaggio nell'approccio con allocazione concatenata è dato dal **costo di accesso al file** stesso, infatti se si volesse accedere all'n-esimo blocco della lista si necessita di scorrere la lista (che si trova sul disco) ed ogni salto da un nodo all'altro della lista comporta l'accesso al disco.
In oltre, dato che ogni blocco memorizza il puntatore a NEXT, la quantità di spazio per i dati non è più una potenza di due e questo comporta un **disallineamento** in memoria poichè molti programmi lavorano con blocchi la cui dimensione è una potenza di due.

- **FAT (File Allocation Table)**

Entrambi gli svantaggi derivati dall'uso dell'allocazione concatenata vengono risolti da questo approccio.
Si pensa di raggruppare tutti i puntatori NEXT in un'unica struttura dati di appoggio detta FAT che tiene conto degli eventuali puntatori a next ad ogni file, ovvero la generica voce della FAT mi da informazioni sul blocco successivo da leggere. Dalla directory si ottiene l'informazione sul primo blocco della FAT da consultare, ovvero il primo blocco nel quale si memorizza il file e l'informazione sul blocco successivo da consultare è codificata all'interno della FAT stessa. L'ultimo blocco avrà un valore fittizio che indica che non vi è un successore.
La tabella FAT si trova sul disco proprio per la necessità di persistenza delle informazioni e viene caricata in RAM per ragioni di efficienza, con l'accortezza di dover sincronizzare le informazioni tra RAM e disco rapidamente per evitare perdite di dati.

Vantaggi

Questo approccio introduce un primo vantaggio sulla **dimensione del payload** dei blocchi, i quali non dovendo contenere il campo NEXT (che si trova nella FAT) possono essere sfruttati per intero.
Viene risolto il problema che era presente nel modello con la sola lista concatenata in quanto non si pagano N accessi al disco prima della lettura dell'N-esimo blocco in quanto la FAT di trova in RAM e si farà **accesso a disco una sola volta** quando si ha l'informazione sull'esatto blocco da leggere.

Svantaggi

Si necessita di portare l'intera FAT in RAM e non si presta bene a dischi di grandi dimensioni.

- **Allocazione con i-Node**

Un approccio alternativo alla FAT che consente di non portare l'intera tabella in RAM è quello dell' i-Node.

Ogni file presente sul disco rigido è associato ad un i-node, che contiene i metadati relativi al file, come ad esempio il proprietario del file, i permessi di accesso, la data e l'ora di creazione e di modifica del file, la dimensione del file e gli indirizzi dei blocchi di dati che compongono il file.

L'i-node è identificato da un numero univoco chiamato **i-number**, che viene utilizzato per accedere alle informazioni del file associato. Il numero di i-Node viene memorizzato nella directory, insieme al nome del file, per consentire la ricerca del file sulla base del nome.

Vantaggi

Il grande vantaggio rispetto alle liste collegate che usano una FAT in memoria è che l'i-Node ha bisogno di essere spostato in memoria solo quando il file corrispondente è aperto. Lo spazio occupato da un i-Node è molto più piccolo di quello occupato dalla FAT, in quanto la tabella ha una dimensione proporzionale a quella del disco stesso per tenere traccia di ogni blocco del disco. La lista di i-Node che sposto in RAM ha una dimensione al più pari al numero di file che potrebbe essere contemporaneamente aperto.

Svantaggi

Se ogni i-Node ha spazio per un numero fisso di indirizzi del disco, cosa succede se un file cresce sforando questa dimensione?

- **Variante multilivello**

Ogni i-node contiene un numero fisso di indirizzi di blocchi di dati, che varia a seconda del file system utilizzato.
In caso di file di grandi dimensioni, oltre la capacità dei blocchi di dati previsti dall'i-node, vengono utilizzati i cosiddetti **indirizzi indiretti**, che puntano ad altre strutture dati per accedere ai blocchi di dati del file presenti su disco. Ancora meglio sarebbe avere due o più di questi blocchi che puntano altri blocchi su disco pieni di indirizzi.

Si ha un approccio con un i-Node navigabile con accessi diretti fino ad una certa voce, mentre al crescere delle esigenze del file la generica voce diventa un puntatore ad altri blocchi di indirizzi su cui si fanno accessi indiretti. Si ottiene un albero sbilanciato che cresce solo se necessario, in funzione delle esigenze del file.

Vantaggi

Questo accorgimento permette di mantenere un **upper bound** sul numero di blocchi che al più si devono leggere dal disco per accedere al nodo N, garantendo una certa velocità di accesso ai file.