Implementazione del File System

Da un punto di vista tecnico ci interessa il modo in cui i file sono memorizzati, com'è gestito lo spazio su disco e come far funzionare tutto in modo efficace.

Layout del file system

I file system sono memorizzati su disco che può essere suddiviso in diverse partizioni indipendenti.

> ⓘ **Tipi di layout**
>
> Dipende dall'età del computer:
>
> - BIOS e un master boot record -> vecchi.
> - UEFI -> moderno.

Vecchio stile: Il Master Boot Record

MBR si trova nel settore 0 e contiene alla fine la tabella degli indirizzi di inizio e fine delle partizioni.
Quando accendiamo il computer, il BIOS chiama l'MBR che legge la partizione attiva, quindi il primo blocco (**Blocco di boot**, contiene il bootloader) e lo esegue.
Dal primo blocco viene caricato il sistema operativo da parte di un programma.

> 🖉 **Layout di una partizione**
>
> Spesso ogni partizione contiene dei blocchi particolari:
>
> - **Superblocco:** Contiene parametri chiave e letto in memoria all'accensione del pc o al primo avvio del file system. Informazioni del superblocco sono: numero magico che identifica il file system, numero di blocchi del file system, altre informazioni chiave.
> - Informazioni sui blocchi liberi del file system (Bitmap o Puntatori).
> - I-node, array di strutture dati che descrivono il file.
> - Directory radice contenente la cima dell'albero.
> - Resto delle altre directory e file.

Nuova scuola: Unified Extensible Firmware Interface

L'avvio con il MBR è lento ed è dipendente dall'architettura usata. Quindi Intel ha proposto di sostituirlo con **UEFI**.
Non ci sta un MBR nel settore 0, bensì un marcatore, il quale compito è quello di non fare trovare l'MBR al BIOS.
La **GPT (GUID Partition Table)** mantiene le informazioni sulle posizioni delle partizioni del disco.

> **GUID: Globally Unique Identifier**
>
> Nell'ultimo blocco abbiamo un backup di GPT. Inoltre:
>
> - Supporta dischi fino a 8ZiB
> - Consente numero illimitato di partizioni
> - Include backup della tabella delle partizioni
> - Utilizza un controllo di integrtià per evitare la corruzione di messaggi (CRC)

GPT -> Contiene inizio e fine di ogni partizione.

Trovata la GPT, il **firmware** può leggere i file system di tipi specifici (FAT), questo è posizionata nella partizione **EFI system partition (ESP)** che ci permette di archiviare i file di avvio come bootloader, driver ed è quindi essenziale per l'avvio del sistema operativo.

> **Secure Boot**
>
> Funzionalità UEFI progettata per impedire l'avvio di software non autorizzato.
> Lo fa attraverso il controllo delle firme digitali di bootloader, driver e SO, avvia solo software autorizzato e firmato bloccando malware durante l'avvio.
> Quindi è una protezione in più, ma alcuni SO richiedono la disattivazione per funzionare.

Vantaggi del UEFI sono:

- Avvio più veloce.
- Compatibilità migliorata.
- Interfaccia utente avanzata
- Supporto dei dischi moderni.
- Sicurezza

Implementazione dei file nei file system

Aspetto più importante -> Tenere traccia dei blocchi di memoria assoviati a un file.
Esistono diverse modi per implemetarlo, solitamente dipende dal SO usato:

- **Allocazione contigua**: Schema più semplice, memorizziamo i file in memoria come una sequenza contigua di blocchi nel disco con grandezza fissata.

Vantaggi

1. Semplice da implementare: bastano due numeri per tenere traccia delle posizioni dei blocchi di un file: l'indirizzo del primo bloccco e il numero dei file del blocco.
2. Prestazioni di lettura sono eccellenti su disco magnetico, perchè per leggere l'intero file è richiesta una sola operazione.

⚠ Svantaggi

Con il passare del tempo i dischi si frammentano, quindi con l'eliminazione dei file vengono lasciati degli spazi vuoti.
Inizialmente non sono un problema perché possiamo riempite gli spazi alla fine del disco. Poi diventa necessario compattare il disco che è un'operazione molto complessa o di riutilizzare gli spazi vuoti, il che richiede il mantenimento di una lista di spazi vuoti, e la conoscenza a priori dello spazio dei file che sto creando per trovare abbastanza spazio nella lista.

- **Allocazione a liste concatenate** -> File memorizzati come liste concatenate di blocchi, composti da due parti: la prima per il puntatore successivo, il resto per i dati.

Vantaggi

Possiamo usare ogni blocco del disco, senza perdere spazio a causa della frammentazione, bisogna solo memorizzare l'indirizzo su disco del primo blocco

⚠ Svantaggi

Leggere un file sequenzialmente è facile, ma l'accesso casuale è lento perché devo leggere tutti gli $n-1$ blocchi prima, inoltre la quantità di spazio usata per memorizzare i dati non è standard il che potrebbere richiedere di leggere dati da più blocchi del disco, aumentando l'overhead.

- **Allocazione a liste concatenate con una tabella di memoria** -> Eliminiamo i problemi, prendendo la parola puntatore di ogni blocco è ponendola in una tabella in memoria. Oni puntatore punta al blocco successivo del file, finchè non si raggiunge la fine contraddistinta da un carattere speciale. Questo tipo di tabella nella memoria principale si chiama **FAT (File Allocation Table).**

Vantaggi

Intero blocco disponibile per i dati e l'accessi casuale è semplice, tranne per determinati offset. è sufficiente che la voce della directory contenga un singolo intero per localizzare tutti i blocchi.

⚠ Svantaggi

La tabella deve restare sempre in memoria principale (Ex: con un disco da 1TB e blocchi da 1KB dovremmo avere un miliardo di voci, una per ciascun blocco, ogni voce ha 4 byte, la tabella occcupa dai 2,4GB ai 3GB di memoria principale). Non si presenta bene per dischi di grosse dimensioni, infatti era il file system origniale di MD-DOS e viene usato per dischi piccoli.

- **I-node** -> Index-node, elenca gli attributi e gli indirzzi dei blocchi dei file, tramite i-node si possono trovare tutti i blocchi di un file, inoltre è in memoria solo quando il file corrispondente è aperto.

Vantaggi

Se ogni i-node occupa n byte, e possiamo avere k file contemporaneamente aperti, l'array che contiene gli i-node sarà nk byte grande, basta mantenere uno spazio della memoria riservata in anticipo per questa quantità di spazio.
L'array è di dimensioni molto minori rispetto alla tabella.

Motivo per cui l'array è più piccolo della tabella

Tabella -> Cresce in maniera lineare rispetto al disco.
I-node -> Array proporzionale al numero massimo di file in memoria e le dimensioni del disco sono irrilevanti.

> ⚠ **Svantaggi e soluzioni**
>
> Ogni I-node ha spazio per un numero finito di indirizzi del disco, se finiscono? Semplicemente manteniamo l'ultimo indirizzo del disco non per un blocchо di dati, ma per l'indirizzo di un blocco contenente ulteriori indirizzi di blocchi del disco.

Implementazione delle directory

Prima di essere letto, un file deve essere aperto. Per localizzare la vode della direcotory su disco viene usato il nome di percorso, che da l'informazioni per trovare i blocchi sul disco.

Il sistema delle directory ha il compito di mappare il nome ASCII del file sulle informazioni necessarie per localizzare i dati.

Una questione correlata è dove dobbiamo memorizzare gli attributi dei file, abbiamo due scelte:

- **Memorizzarli nella voce della directory** -> Directory composta da lista di voci di dimensione fissa, insieme ad una struttura degli attributi dei file che indicano la posizione dei blocchi.
- **Memorizzare gli attributi negli I-node** -> Memorizziamo nelle voci delle directory solo il nome del file e il numero di I-node.

Fino ad adesso abbiamo pressupposto solo file con nomi di dimensione fissa, ma in tutti i SO abbiamo invece nomi di lungehzze variabile. Abbiamo diversi approcci per permettere ciò:

1. Impostare limite sul nome del file (MAX 255 caratteri) riservando lo spazio per ciascun nome di file, ma spreca una buona parte dello spazio delle directory.
2. Rinunciare a voci di stessa lunghezza, ogni directory contiene una parte fissa, seguita dai dati in un formato fisso.

Abbiamo due approcci proncipali per implementare le directory:

- **Ricerca Lineare:** Manteniamo una lista di nomi che hanndo dei puntatori ai blocchi di dati corrispondenti.

> 🖉 **Vantaggi**
>
> E' molto semplice da implementare.

> ⚠ **Svantaggi**

La ricerca avviene in tempo lineare.

- **Hash Table:** Usiamo delle tabelle di hash in ogni directory per accellerare il processo di ricerca.

Vantaggi

La ricerca avviene in tempo costante.

⚠ Svantaggi

Dobbiamo tenere conto di possibili collisioni.

Una maniera per aumentare l'efficienza della ricerca è quella do usare delle chace che mantegono risultati di ricerce frequenti.
Ciò potrebbe comportare ad una comeplessità maggire nella gestione delle directory.

File condivisi

Ha volte è necessario dovere condividere dei file, questi devono quindi comparire in più directory di utenti diversi contemporaneamente.

La connessione fra un directory di un utente e il file condiviso si chiama **link**.
Ma a causa dei link, il file system non è più un albero, ma un **DAG**, il che complica la gestione.

L'introduzione della condivisione includei dei problemi:

- Se le directory contengono realmente indizzi su disco, quando il file collegato copia gli indirizzi in un directory, i nuovi blocchi saranno elencati solo nella directory dell'utente che ha fatto l'accodamento.

Possiamo risolvere il problema in due modi:

1. I blocchi del disco sono in una piccola struttura dati associata al file stesso , le directory hanno dei puntatori a questa struttura (**HARD LINK)**.

⚠ Svantaggi

B si collega al file condiviso, I-node registra C come proprietario, la creazione del link non cambia la proprietà, ma incrementa il conteggio dei link dell'I-node.

Se C elimina il file il sistema si trova in un problema: Se rimuove il file e pulisce l'I-node, B avrà una voce di directory non valida.
Se poi l'I-node è riassegnato, B punterà al file sbagliato.

3. Introduzione di un file di tipo LINK, contiene solo il nome di percorso del file a cui si collega. Questo approccio si chiama **link simbolico**.