8 - File System

Un file system è un componente del sistema operativo che gestisce e organizza i dati su un dispositivo di archiviazione, come un disco rigido, una chiavetta USB o un'unità SSD. Fornisce una struttura per memorizzare, accedere e gestire i file e le directory, consentendo agli utenti e ai programmi di trovare e utilizzare i dati in modo efficiente. I file system più comuni includono NTFS, FAT32, ext4 e APFS.

File

I file sono un meccanismo di astrazione che consente di salvare e leggere informazioni sul disco, nascondendo i dettagli di come e dove sono memorizzate. I file vengono identificati tramite nomi ed estensioni.

Struttura dei File

I file possono essere strutturati in modi diversi, a seconda delle esigenze del sistema operativo e delle applicazioni:

- **Sequenza Non Strutturata di Byte**: I file sono visti come una serie non strutturata di byte. Utilizzato da UNIX, Linux, macOS e Windows.
- **Sequenza di Record di Lunghezza Fissa**: Un file è una sequenza di record con lunghezza fissa. Meno comune nei sistemi moderni.
- **File come Albero di Record**: Il file è organizzato come un albero di record, con lunghezze variabili e un campo chiave fisso. Utilizzato in sistemi come DBMS.

Tipi di File

- **File e Directory Normali:** Utilizzati in sistemi come UNIX e Windows. I file normali contengono informazioni dell'utente e sono i più comuni. Le directory sono file di sistema che mantengono la struttura del file system.
- **File Speciali:**
  - A caratteri: Usati per modellare porte seriali di I/O, come terminali e stampanti.
  - A blocchi: Usati per modellare dischi.
- **File Normali:**

    - File ASCII: Composti da righe di testo, visualizzabili e stampabili, con variazioni nella terminazione delle righe.
    - File Binari: Non leggibili come testo, con una struttura interna conosciuta dai programmi che li utilizzano, come file eseguibili o archivi.
- **File Eseguibile - Componenti:**
    - Intestazione (Header): Contiene un numero che identifica il file come eseguibile, le dimensioni, l'indirizzo di esecuzione iniziale (punto d'ingresso) e vari flag.
    - Testo e Dati: Parti effettive del programma, da caricare e rilocare in memoria.
    - Tabella dei simboli: Utilizzata per il debug.
- **File di Archivio:**
    - Descrizione: Raccolta di procedure di libreria (moduli) compilate ma non collegate.
    - Intestazione dei Moduli: Indicano nome, data di creazione, codice di protezione e dimensione.
    - Carattere Binario: Non leggibile come testo.

Metodi di Accesso ai File

I metodi di accesso ai file includono:

- **Accesso Sequenziale**: Lettura di byte o record in ordine senza poter saltare o leggere in ordine sparso.
- **Accesso Causale**: Lettura di byte o record in qualsiasi ordine. Utilizza l'operazione `seek` per impostare la posizione corrente.

Attributi dei File

Ogni file ha un nome e i propri dati, oltre a ulteriori informazioni chiamate attributi (metadati), come la data e l'ora dell'ultima modifica e la dimensione. Gli attributi sono cruciali per:

- La protezione e il controllo dell'accesso.
- La gestione efficace dei file nei sistemi operativi. L'elenco degli attributi varia considerevolmente tra i diversi sistemi operativi.

Directory

Per tenere traccia dei file, i file system normalmente utilizzano directory o cartelle, che sono anch'esse dei file.

Sistemi di Directory a Livello Singolo

La forma più semplice di sistema di directory è una sola directory contenente tutti i file, chiamata directory principale (root directory). Questo schema è semplice e permette di localizzare i file rapidamente. Viene spesso usato nei dispositivi embedded come fotocamere digitali o MP3.

Sistemi di Directory Gerarchici

Un sistema di directory gerarchico è organizzato come un albero di directory. In un ambiente condiviso, ogni utente può avere una directory principale privata per la propria gerarchia. Questo approccio permette agli utenti di creare un numero arbitrario di sottodirectory per organizzare il proprio lavoro. I file system moderni adottano questo schema.

Percorso Assoluto

Ogni file ha un nome di percorso assoluto che inizia dalla directory principale. Ad esempio, `/usr/ast/mailbox` indica che il file `mailbox` si trova nella directory `ast`, che è una sottodirectory di `usr`, a sua volta sottodirectory della root.

Percorso Relativo

Il percorso relativo è usato congiuntamente al concetto di directory di lavoro (o directory corrente). Se un utente è nella directory `/usr/hjb`, il comando `cp mailbox mailbox.bak` è equivalente a `cp /usr/hjb/mailbox /usr/hjb/mailbox.bak`.

Nomi Speciali

Nei sistemi che supportano directory gerarchiche, ogni directory contiene due voci speciali:

- `.` (dot): Si riferisce alla directory corrente.
- `..` (dotdot): Si riferisce alla directory genitore, tranne nella directory radice.

Operazioni sulle Directory

Le operazioni principali sulle directory sono:

1. **create**: Creazione di una directory vuota con le voci `.` e `..`.
2. **delete**: Eliminazione di una directory vuota.
3. **opendir**: Apertura di una directory per leggere il suo contenuto.
4. **closedir**: Chiusura di una directory dopo la lettura per liberare risorse.
5. **readdir**: Restituisce la prossima voce in una directory aperta senza esporre la struttura interna.
6. **rename**: Rinomina una cartella, simile al rinomino di un file.
7. **link**: Crea un hard link, collegando un file esistente a un nuovo percorso condividendo l'i-node.
8. **unlink**: Rimuove una voce di una directory, cancellando il file se è l'unico link.

Layout del File System

MBR

Il settore 0 del disco, chiamato MBR, è essenziale per l'avvio del computer. Contiene la tabella delle partizioni e identifica la partizione attiva da cui avviare il sistema. Quando il computer si avvia, il BIOS legge ed esegue l'MBR, che localizza la partizione attiva e ne legge il primo blocco, chiamato blocco di boot.

3. **Struttura del file system:** Dopo il MBR e il settore di boot, la struttura del file system varia a seconda del tipo di file system utilizzato (come FAT, NTFS, ext4, etc.). Tuttavia, alcuni elementi generali includono:
   - **Superblocco:** Contiene parametri chiave del file system come un numero che lo identifica. Viene letto in memoria all'avvio del computer.
   - **Bitmap o Linked List:** Utilizzati per la gestione dello spazio libero nel disco.
   - **I-node:** Strutture di dati che rappresentano i file nel file system. Contine etutte le info di un file.
   - **Directory radice:** È il punto di partenza dell'albero della struttura gerarchica del file system.
     Il MBR e la struttura del file system sono fondamentali per il corretto funzionamento del disco e per consentire al sistema operativo di gestire e organizzare i dati in modo efficiente.

UEFI

Con l'introduzione di UEFI (Unified Extensible Firmware Interface), il processo di avvio del computer è diventato più veloce e supporta una serie di miglioramenti rispetto al vecchio MBR (Master Boot Record):

1. **Velocità:** UEFI è più rapido nel processo di avvio rispetto al MBR.
2. Supporto a dischi più grandi: UEFI supporta dischi di dimensioni fino a 8 ZiB (zettabyte)
3. **Partizioni infinite:** UEFI non ha il limite di 4 partizioni primarie come nel MBR. Utilizza la GPT (GUID Partition Table).
4. **Struttura della GPT:** La GPT è una tabella delle partizioni che si trova nel secondo blocco del disco. Contiene informazioni dettagliate sulla posizione di ogni partizione sul disco utilizzando identificatori univoci (GUID).
5. **Backup della GPT:** UEFI conserva un backup della GPT nell'ultimo blocco del disco, garantendo la ridondanza e la sicurezza dei dati relativi alle partizioni.
6. **Supporto a vari tipi di file system:** Una volta individuata la GPT, il firmware UEFI può leggere e supportare diversi tipi di file system, permettendo una maggiore flessibilità nell'utilizzo di dischi formattati con file system specifici.
   In sintesi, UEFI con GPT rappresenta uno standard moderno per il boot dei computer, offrendo prestazioni migliorate, supporto per dischi di grandi dimensioni e una gestione avanzata delle partizioni rispetto al tradizionale MBR.

Implementazione dei File

È importante avere una buona implementazione dei file per assicurare l'integritá, l'accesso efficiente e la gestione dello spazio sul disco.

Allocazione Continua

Nel sistema di allocazione continua, ogni file è memorizzato come una sequenza contigua di blocchi sul disco. Questo approccio presenta alcuni vantaggi e svantaggi significativi.
In sintesi, l'allocazione continua è un metodo semplice ed efficiente per la memorizzazione dei file, ma può causare problemi di frammentazione del disco nel tempo, rendendo difficile l'aggiunta di nuovi file e richiedendo operazioni aggiuntive per mantenere le prestazioni ottimali del sistema.

Allocazione a liste concatenate

Nel metodo di allocazione a liste concatenate, ciascun file è memorizzato come una serie di blocchi sul disco, dove ogni blocco contiene un puntatore al successivo nella lista concatenata, insieme ai dati del file.
In sintesi, l'allocazione a liste concatenate è efficace nel massimizzare l'utilizzo dello spazio sul disco e nel ridurre la frammentazione esterna. Tuttavia, a causa dell'accesso casuale lento e della gestione delle dimensioni dei blocchi, potrebbe non essere ideale per applicazioni che richiedono un accesso rapido e efficiente ai dati, come sistemi di database o di gestione dei file molto attivi.

Allocazione a liste concatenate con FAT

Nel metodo di allocazione a liste concatenate con FAT (File Allocation Table), ogni blocco del disco è rappresentato da una voce nella FAT, che è una tabella di allocazione dei file memorizzata in memoria RAM.
In sintesi, l'allocazione a liste concatenate con FAT risolve l'accesso casuale lento tipico dell'allocazione a liste concatenate tradizionale spostando i puntatori in una tabella di memoria in RAM. Tuttavia, ciò comporta il vincolo di dover mantenere la FAT sempre in memoria, con il rischio di occupare una quantità considerevole di RAM, soprattutto su dischi di grandi dimensioni.

I-Node

Gli i-node (index-node) sono una struttura dati fondamentale utilizzata dai sistemi operativi UNIX e dai suoi file system derivati, come ext2, ext3, ext4, e altri. Essi elencano gli attributi di un file o directory, escludendo nome e contenuto, come permessi di accesso, proprietario, timestamp e gli indirizzi dei blocchi dei dati del file.
In sintesi, gli i-node sono una componente cruciale per la gestione dei file nei sistemi UNIX e derivati, fornendo un meccanismo efficiente per memorizzare e gestire gli attributi dei file e le informazioni di allocazione dei blocchi di dati.

Implementazione delle Directory

Il ruolo principale delle directory è quello di mappare il nome ASCII di un file alle informazioni necessarie per localizzare i dati associati sul disco. A seconda del sistema di file utilizzato, queste informazioni possono variare.

In sintesi, le directory svolgono il compito fondamentale di fornire un'interfaccia tra il nome di un file, che è generalmente in formato ASCII, e le informazioni specifiche sul suo posizionamento e struttura fisica nel sistema di archiviazione del disco. A seconda del metodo di allocazione utilizzato dal sistema di file (contiguo, liste concatenate o basato su i-node), le directory forniscono le informazioni adeguate per localizzare e accedere ai dati del file in modo efficiente.

File Condivisi e Link nel File System

I file condivisi sono essenziali in ambienti collaborativi per permettere a piú utenti di lavorare sugli stessi file.

Hard Link:

- Gli hard link sono collegamenti diretti all'i-node di un file condiviso nel sistema di file.
- Un file con hard link non viene rimosso fisicamente dal disco finché esistono riferimenti ad esso tramite hard link.
- Sono efficienti in termini di spazio perché utilizzano un solo i-node indipendentemente dal numero di link.
- Possono causare confusione sulla proprietà del file poiché tutti i link condividono lo stesso i-node e quindi le stesse proprietà.

  Soft Link:
- I soft link puntano al nome del file anziché direttamente all'i-node.
- Sono più flessibili rispetto agli hard link in quanto possono riferirsi a nomi di file oltre i confini del file system e su macchine remote.
- Richiedono un i-node separato per ogni link, rendendoli meno efficienti in termini di spazio rispetto agli hard link.
- Diventano invalidi se il file originale viene rimosso, poiché puntano solo al nome del file e non al suo i-node.
- Il sistema operativo impiega più tempo nella risoluzione del percorso rispetto agli hard link a causa della necessità di seguire il percorso del nome del file fino al file effettivo.

  In sintesi, i hard link offrono una gestione efficiente dello spazio e sono robusti rispetto alla rimozione accidentale di file, ma possono confondere la proprietà del file. I soft link, invece, offrono maggiore flessibilità nel riferimento a file in diverse posizioni, ma sono meno efficienti in termini di spazio e possono diventare invalidi se il file originale viene eliminato.

Gerstione dello spazio su Disco

Dimensione dei Blocchi

Generalmente, i file sono memorizzati su disco utilizzando l'allocazione contigua o dividendo il file in blocchi più piccoli per consentire maggiore flessibilità e un migliore utilizzo dello spazio su disco.
Una dimensione comune per i blocchi è di 4KB, che rappresenta un compromesso tra lo spazio su disco e le prestazioni di trasferimento dei dati.
Perché 4KB?

- **Prestazioni di trasferimento dati:** Dischi magnetici con blocchi grandi permettono il trasferimento di piú dati per operazione, ma portano anche a spreco di memoria
- **Efficienza dello spazio:** L'efficienza dello spazio diminuisce con l'aumento della dimensione dei blocchi. Blocchi piccoli minimizzano lo spreco, ma significa anche avere ritardi nella lettura del file sul blocco specifico

Gestione dei blocchi Liberi

Linked List

La gestione dei blocchi liberi mediante una lista concatenata, nota come "free list", utilizza una strategia in cui vengono mantenuti solo i blocchi liberi del disco.
Questo metodo è particolarmente efficace quando il disco è quasi pieno, poiché consente di ottimizzare lo spazio disponibile senza necessariamente allocare grandi aree di memoria.
Questo metodo di gestione dei blocchi liberi è ottimizzato per dischi con capacità quasi esaurita, ma può risultare meno efficiente su dischi molto frammentati dove la lista concatenata dei blocchi liberi potrebbe diventare più complessa da gestire.

BitMap

La bitmap è un metodo efficiente per gestire i blocchi liberi su un disco. Consiste in una struttura dati composta da n bit, dove n rappresenta il numero totale di blocchi sul disco. Ogni bit nella bitmap indica lo stato di un singolo blocco: un valore di 1 indica che il blocco è libero, mentre un valore di 0 indica che il blocco è allocato.
In sintesi, la bitmap è generalmente più efficiente della lista concatenata per

gestire i blocchi liberi su un disco, tranne quando il disco è quasi pieno e la bitmap deve gestire un numero significativo di blocchi allocati.

Performance del File System

Per migliorare le prestazioni dei file system e minimizzare i tempi di accesso al disco o agli SSD vengono utilizzate due tecniche principali.

Cache

Utilizzata per ridurre i tempi di accesso al disco mantenendo i blocchi più usati in memoria. La cache si trova all'interno della memoria RAM.

Caching

- **Buffer Cache:** Memorizza i blocchi del disco in RAM per ridurre gli accessi al disco.
- **Page Cache:** Memorizza le pagine del filesystem virtuale (VFS) in RAM prima di passare al driver del dispositivo.
  In poche parole, i file sono nella cache delle pagine e i blocchi del disco sono nella cache buffer.
  I sistemi operativi possono combinare buffer cache e page cache per un uso piú efficiente della memoria e una riduzione degli accessi al disco.

Implementazione della Cache

Per gestire la cache si puó utilizzare il seguente algoritmo: si verifica tutte le richieste di read per vedere se il blocco necessario si trova nella cache, se c'é, la richiesta é soddisfatta senza accedere al disco. Se non lo é, viene letto dal disco e portato nella cache, quindi copiato ovunque ce ne sia bisogno, cosí che le successive richieste per lo stesso blocco possano essere soddisfatte direttamente nella cache.
Nella cache ci sono molti blocchi e quindi serve un metodo efficiente per determinare se un blocco specifico sia presente o meno, quindi viene utilizzata una hash table per velocizzare la ricerca.
Si manifestano situazioni in cui la cache é piena e altri blocchi non possono essere caricati, quindi bisogna rimuoverne qualcuno. Situazione molto simile alla paginazione, quindi si usano gli stessi algoritmi di sostituzione.

L'unico problema é che LRU puó portare a inconsistenze nel caso di crash, specialmente per blocchi critici come i-node. Soluzione, modifica dell'LRU, si dividono i blocchi per categorie basate sull'importanza. Se un blocco é fondamentale per il file system, verrá scritto immediatamente sul disco indipendentemente dalla posizione nell'LRU.
Oltre a questa tecnica, UNIX mette a disposizione una chiamata `sync` che forza la scrittura di tutti i blocchi modificati. Al momentoi dell'avvio del sistema, viene eseguito un deamon che ogni 30 secondi scrive tutti i blocchi modificati su disco.

Deframmentazione del disco

Con il tempo, i file vengono creati, modificati ed eliminati, causando frammentazione dei dati sul disco. Questo significa che i blocchi di dati di un singolo file possono essere dispersi in diverse parti del disco rigido, rallentando le prestazioni di lettura e scrittura. Per migliorare le prestazioni, è necessario consolidare i blocchi di dati in modo che siano contigui, riducendo così il tempo necessario per accedere ai dati dispersi. I file system di Linux gestiscono meglio la frammentazione rispetto a Windows

Compressione e Deduplicazione

Alcuni file system come NTFS e Btrfs supportano la compressione automatica dei dati per risparmiare spazio su disco. Inoltre, implementano la deduplicazione per eliminare i dati duplicati tra i file, riducendo lo spazio di archiviazione utilizzato. La deduplicazione può essere eseguita inline durante la scrittura dei dati o in modalità post-process, dove l'hashing e i confronti vengono eseguiti successivamente in background senza influenzare le operazioni di scrittura dei file.

Affidabilitá del File System

Minacce all'affidabilitá del File System

I rischi per i dati includono:

- **Guasti del disco
- **Interruzioni di energia**
- **Bug del software**
- **Errori umani**

- **Perdita o furto del computer**
- **Malware/ransomware**

Backup

I backup su disco sono necessari per affrontare problemi di recupero da disastri come crash del disco o catastrofi naturali, e per recuperare da errori umani come l'eliminazione accidentale dei file. È essenziale considerare:

1. **Selezione dei file:** È importante decidere quali file includere nel backup, se tutto il file system o solo specifiche directory.
2. **Backup incrementale:** Per ottimizzare tempo e spazio, si utilizzano backup incrementali che aggiornano solo i file modificati dall'ultimo backup completo.
3. **Compressione dei dati:** Comprimere i dati prima di salvarli nel backup può ridurre lo spazio necessario per il salvataggio.
4. **Snapshot del file system:** Algoritmi permettono di creare istantanee del file system per facilitare il backup di file system attivi.
5. **Sicurezza del backup:** È fondamentale mantenere i backup in un luogo sicuro, separato dai computer principali, per proteggerli da rischi aggiuntivi.

Strategie di Backup

- **Backup Fisico:** Copia sequenziale di tutti i blocchi del disco, semplice e veloce, ma ha diffcoltá nel saltare le directory specifiche o nel fare backup incrementali.
- **Backup Logico:** Seleziona e copia solo i file e le directory specifici, ideale per backup incrementali o completi.

Coerenza

La coerenza del file system é cruciale per mantenere l'integritá dei dati. Problemi di incoerenza
possono sorgere a seguito di crash durante la scrittura dei blocchi. In UNIX si utilizza fsck, e in
Windows sfc per verificare la coerenza all'avvio dopo un crash.

File System con Journaling

Registra anticipatamente le operazioni da eseguire in un log, per garantire la coerenza in caso di
crash. Ampiamente usato in NTFS, ext4, macOS. Un journal in un file system é come un registro
che tiene traccia delle modifiche che verranno apportate al file system prima che esse avvengano
effettivamente.

Funzionamento

- **Fase di Registrazione:** Prima di qualsiasi modifica il file system scrive un record nel journal, ovvero l'operazione che verrá eseguita
- **Fase di Esecuzione:** Dopo aver registrato l'operazione, il file system modifica i dati sul disco
- **Fase di Conferma:** A operazione completa, il file system aggiorna il journal per confermare il successo dell'azione

File System virtuali

Il VFS (Virtual File System) è una struttura che integra vari file system in una struttura unificata. Utilizza un livello di codice comune che interagisce con i diversi file system reali sottostanti. L'interfaccia superiore gestisce le chiamate di sistema POSIX dai processi utente, mentre l'interfaccia inferiore consiste in funzioni inviate dal VFS ai file system sottostanti.

- **Superblock nel VFS:** È il descrittore di alto livello di un file system specifico nel VFS. Contiene informazioni cruciali sul file system e permette di identificarlo e interagire con esso.
- **V-node nel VFS:** Rappresenta un file individuale all'interno del VFS e funge da nodo nel file system virtuale. Contiene metadati come permessi, proprietà e dimensione del file. Il VFS utilizza i v-node per gestire e fornire accesso ai file.
- **Directory nel VFS:** È una struttura che organizza e mappa i file e le sottodirectory all'interno del VFS. Consente al VFS di mappare i nomi dei file ai rispettivi v-node, indipendentemente dal file system sottostante.
  Questi componenti consentono al VFS di fornire un'interfaccia uniforme agli utenti e ai processi del sistema operativo, indipendentemente dal file system reale utilizzato.