CAPITOLO 4

#Sistemi_Operativi

Architettura e struttura del File System

Il File System rappresenta un metodo per organizzare e memorizzare le informazioni, fornendo un 'astrazione sui dispositivi di memorizzazione non volatile (Disco rigido, SSD, rete, RAM). Le informazioni sono organizzate in file e directory. Esempi di file system: FAT12/FAT16 per MS-DOS, NTFS per Windows, Ext4 per Linux, APFS per macOS/iOS.

File

Un File è un'astrazione di un dispositivo di memorizzazione, fornisce un metodo per salvare e leggere informazioni nascondendo all'utente i dettagli implementativi.
L'importanza dei file infatti sta proprio nella capacità di fornire un'interfaccia di memorizzazione che astrae dal complicato hardware e consente quindi un uso più semplice di queste funzioni da parte dell'utente.

I File sono identificati da nome e estensione che indica il tipo di file e quali sono i programmi che ne favoriscono l'esecuzione.

I File possono essere strutturati in vari modi, in case alle esigenze e applicazioni:

- Sequenza non strutturata di byte : il File è visto semplicemente come una sequenza continua di byte senza alcuna struttura imposta dal SO. I dati al suo interno sono interpretati direttamente dai programmi che li utilizzano.
- Sequenza di record di lunghezza fissa : il File è composto da record, ognuno dei quali dalla lunghezza predefinita e dalla struttura chiara. Questo formato è particolarmente efficiente per accessi sequenziali e consente un semplice calcolo degli offset per l'accesso diretto a specifici record.
- File come Albero di Record : i File sono organizzati in una struttura ad albero. Ogni File ha record di lunghezze variabili, ed ogni record ha un campo chiave in una posizione fissa che semplifica la ricerca rapida di specifici record. Questo approccio è comune nei sistemi e database contenenti grandi quantità di dati.

In un SO esistono diversi tipi di File:

1. File normali : sono la forma più comune, contengono informazioni utente e possono essere principalmente di due tipi:

    - ASCII : composti da righe di testo visualizzabili e stampabili.
    - Binari: non leggibili come testo, hanno una struttura interna conosciuta dai programmi che li utilizzano. Distinguiamo tra i File Binari i:
        - File eseguibili : contengono codice eseguibile dalla CPU e sono progettati per essere eseguit dal SO. Hanno Intestazione (header, contiene informazion cruciali per l'esecuzione), Testo del programma e dati (codice effettivo), Tabelle dei simboli (per il debug).
        - File di Archivio : utilizzati per memorizzare insiemi di file in un unico file compresso o raggruppato.
2. File Speciali : utilizzati dai SO per rappresentare dispositivi hardware o fornire meccanismo specifici per gestire le risorse del sistema. Sono di due tipi:
    - A Caratteri : permettono operazioni di lettura o scrittura che coinvolgono singoli caratteri o flussi di caratteri. Sono utilizzati per modellare dispositivi che gestiscono carattere per carattere come terminali e stampanti.
    - A Blocchi : utilizzati per modellare dispositivi che gestiscono dati a blocchi di dimensioni fisse come Dischi Rigidi, USB etc...
3. File e Directory normali : sono utilizzati in sistemi come UNIX e Windows. I file normali contengono informazioni utente e sono la forma più comune. Le directory sono file di sistema per mantenere la struttura del file system.

Accesso ai File

Nei primi SO, l'accesso ai file era principalmente sequenziale (ossia si poteva leggere o scrivere solo procedendo in ordine dall'inizio alla fine, tipico nei nastri magnetici).
Fu con l'introduzione dei dischi che finalmente arrivò la possibilità di accedere in qualsiasi posizione in un file (accesso casuale, "random access").
Per specificare da dove iniziare a leggere si possono usare due metodi:

- com il primo metodo ogni operazione di Read fornisce la posizione del file dalla quale cominciare a leggere.
- con il secondo è fornita una funzione speciale, seek, per impostare la posizione corrente (usato da Windows e Unix).

I SO inoltre associano ai file ulteriori informazioni dette attributi o metadati, che sono cruciali per la protezione o la gestione del file stesso:

Attributo	Significato	Attributo	Significato
Protezione	Chi può accedere al file e in che modalità	Flag temporaneo	0 per normale; 1 per cancellare il file al termine del processo
Password	Password necessaria per accedere al file	Flag di file bloccato	0 per non bloccato; non zero per bloccato
Creatore	ID della persona che ha creato il file	Lunghezza del record	Numero di byte nel record
Proprietario	Proprietario attuale	Posizione della chiave	Offset della chiave in ciascun record
Flag di sola lettura	0 per lettura/scrittura; 1 per sola lettura	Lunghezza della chiave	Numero di byte del campo chiave
Flag di file nascosto	0 per normale; 1 per non visualizzare negli elenchi	Data e ora di creazione del file	Data e ora di quando il file è stato creato
Flag di file di sistema	0 per file normali; 1 per file di sistema	Data e ora di ultimo accesso al file	Data e ora di quando è avvenuto l'ultimo accesso al file
Flag di file archivio	0 per già sottoposto a backup; 1 per file di cui fare il backup	Data e ora di ultima modifica al file	Data e ora di quando è avvenuta l'ultima modifica al file
Flag ASCII/binario	0 per file ASCII; 1 per file binari	Dimensione attuale	Numero di byte nel file
Flag di accesso casuale	0 per accesso sequenziale; 1 per accesso casuale	Dimensione massima	Numero di byte di cui può aumentare il file

Figure 3: Attributi di un file.

Operazione sui File:

1. Create : Creazione di un file senza dati.
2. Delete : Eliminazione di un file per liberare spazio sul disco, attraverso una specifica chiamata di sistema.
3. Open : apertura del file per consentire al sistema di caricare in memoria gli attributi e gli indirizzi dal disco.
4. Close : chiusura del file al termine degli accessi per liberare lo spazio all'interno delle tabelle interne.
5. Read : lettura dei dati da un file, generalmente dalla posizione corrente, specificando la quantità di dati richiesti e fornendo un buffer per la loro memorizzazione.
6. Write : scrittura di dati nel file, tipicamente alla posizione corrente, può comportare l'ampliamento del file o la sovrascrittura dei dati esistenti.
7. Append : aggiunta dei dati solo alla fine del file, usata in alcuni sistemi operativi come forma limitata di scrittura.
8. Seek : Riposizionamento del puntatore del file su una posizione specifica per file ad accesso casuale, permettendo la lettura o la scrittura da quella posizione.
9. GetAttributes : Lettura degli attributi del file.
10. SetAttributes : modifica degli attributi di un file da parte dell'utente, come la modalità di protezione o altri flag, dopo la creazione del file.
11. Rename : ridenominazione di un file, utilizzata come alternativa al processo di copia ed eliminazione del file originale, specialmente utile per file di grandi dimensioni.

Directory

Per tener traccia dei file, i file system usano normalmente directory o cartelle.
Anch'esse sono dei file e rappresentano un modo per mantenere la struttura del File System.
Possono essere implementate in vari modi:

- Sistemi di directory a livello singolo: è il sistema più semplice. Consiste in un'unica directory (detta root directory) contenente tutti i file. Il vantaggio è la semplicità e la capacità di localizzare rapidamente i file (usato spesso in sistemi embedded, ossia specializzati nello svolgere un singolo compito).
- Sistemi a directory gerarchici : directory ramificate ad albero che offrono la capacità di creare un numero arbitrario di sottodirectory (tutti i moderni file system sono organizzati in questo modo). Quando si ha questo sistema con albero delle directory, i nomi dei file devono essere specificati in qualche modo. Sono usati comunemente due metodi:
  - Percorso Assoluto : prevede di assegnare a ogni file un nome di percorso assoluto, composto dal percorso che inizia dalla directory principale (root "/") e arriva al file (iniziano dalla directory principale e sono univoci).
  - Percorso Relativo: è un percorso che si riferisce a un file o a una directory partendo dalla directory di lavoro corrente. Se l'utente ha selezionato una directory specifica come directory di lavoro, i percorsi che non iniziano con la directory principale vengono interpretati come relativi a quella directory.
  Voci in ogni directory: "." per indicare quella corrente e ".." per indicare la genitrice.

Operazioni sulle Directory

1. create : creazione di una directory vuota con le voci '.' e '..'.
2. delete : eliminazione di una directory, possibile solo se la directory è vuota.
3. opendir : apertura di una directory per la lettura del 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 (blocco di dati presente sul disco).
8. unlink : rimuove una voce di una directory, cancellando il file se è l'unico link.

Una variante del concetto di collegare i file è il Link Simbolico (o Collegamento o Alias).
Si tratta di un file speciale che punta a un altro file o directory. In pratica è un riferimento a un percorso file piuttosto che al contenuto effettivo del file. Quando si accede al link simbolico, il SO reindirizza l'operazione al file o directory a cui punta.
I vantaggi del Link Simbolico è che può attraversare i confini del File System: un link simbolico può puntare a un file che si trova su un disco diverso o anche su un file system remoto, se montato sulla macchina locale.

Implementazione del File System

Abbiamo descritto il File System come il metodo di organizzazione e gestione dati all'interno di memoria non volatili, come il disco rigido.
La sua implementazione e layout su disco rigido variano a seconda del sistema utilizzato.
Un disco può infatti essere suddiviso in più partizioni, ognuno dei quali con il proprio File System indipendente.

Mentre quindi il layout specifico del File System può variare notevolmente a seconda del tipo utilizzato (es. FAT, NTFS etc...) esistono delle componenti comuni:

- Superblocco: contiene parametri critici del file system come il "numero magico" per identificarne il tipo, avere informazioni sullo stato e altre configurazioni.
- Bitmap o Linked List : utilizzate per gestire lo spazio libero nel disco. Similmente a quanto visto con l'utilizzo di queste strutture in RAM, le Bitmap tengono traccia di quali blocchi sono liberi o utilizzati mentre le liste concatenate collegano i blocchi liberi.
- I-node : è un array di strutture dati, una per ogni file, che contiene dati importanti come permessi, proprietario, data e ora di creazione, dimensione e indirizzi dei dati fisici del file.
- Directory radice : contiene le informazioni di base e l'elenco delle directory e file di livello superiore.

Gestione dell'avvio

Esistono diversi approcci per la gestione dell'avvio e delle partizioni di un disco rigido:

- MBR : il Master Boot Record (MBR) è il settore 0 del disco rigido ed è essenziale per l'avvio del calcolatore. Questo contiene infatti la Tabella delle Partizioni, che può gestire soltanto fino a 4 partizioni primarie. Ogni voce della tabella contiene indirizzo di inizio e di fine partizione. La Partizione attiva identifica quale di queste 4 partizioni deve essere utilizzata al fine di avviare il SO. Quando si avvia il computer, il BIOS legge ed esegue il codice presente nell'MBR. Questo a sua volta localizza la partizione attiva, ne legge il primo settore (detto blocco di boot/boot sector) e lo esegue per avviare il SO.
- UEFI (Unified Extensible Firmware Interface): si tratta di un'interfaccia firmware (tipo di software strettamente correlato all'hardware) che ha sostituito il BIOS tradizionale in molti sistemi moderni. Offre infatti un avvio più veloce supportando caratteristiche avanzate come il secure boot e l'uso di dischi di grandi dimensioni. UEFI non dipende da MBR all'avvio in quanto sfrutta una struttura detta GUID Partition Table (GPT). Questa struttura è molto più flessibile di MBR in quanto non si limita a 4 partizioni ma ne può supportare un numero sostanzialmente illimitato (come MBR ogni partizione è identificata da indirizzo di inizio e fine). La sua struttura principale non risiede nel primo settore del disco (come il

MBR) ma nel secondo. In questo modo è garantito il supporto per software legacy (meno moderni) supportati da MBR.

Implementazione dei File

Esistono diversi metodi implementativi per i file:

- Allocazione continua : ciascun file viene memorizzato come una sequenza contigua di blocchi sul disco. E' semplice da implementare ed offre eccellenti prestazioni di lettura in quanto i blocchi dei file sono adiacenti. Il principale svantaggio è invece la Frammentazione esterna, che può rendere difficile trovare uno spazio contiguo sufficientemente grande per un nuovo file o per espandere un file esistente. Per ridurre la frammentazione è necessario compattare frequentemente il disco, e ciò è costoso in termini di tempo.
- Allocazione a liste concatenate : utilizza una lista collegata a blocchi, dove ogni blocco contiene un puntatore a quello successivo oltre ai dati del file. Ciò riduce la frammentazione esterna in quanto non richiede necessariamente blocchi contigui, tuttavia l'accesso casuale ai dati è lento in quanto per raggiungere un blocco specifico è necessario attraversare la lista a partire dal primo blocco. Inoltre, essendo una parte del blocco occupato dal puntatore, si riduce lo spazio per i dati effettivi.
- Allocazione a liste concatenate con FAT (File Allocation Table) : è un miglioramento del metodo precedente. La FAT è infatti una tabella di memoria che contiene i puntatori ai blocchi successivi di un file, migliorando l'accesso casuale rispetto a una semplice lista concatenata. Ciò riduce i tempi di accesso perché i puntatori sono memorizzati in una tabella e non nei blocchi stessi. Tuttavia la FAT deve essere mantenuta in RAM, e la situazione diventa problematica se si utilizzano dischi di grandi dimensioni in quanto la tabella potrebbe occupare quantità significative di memoria.
- I-node : si tratta di una struttura dati utilizzata nei sistemi di file come UNIX per memorizzare le informazioni di un file, come attributi (proprietario, permessi, timestamp) e indirizzi dei blocchi di dati. Ogni file e directory è rappresentata da i-node univoco. Tuttavia gli i-node hanno spazio limitato per memorizzare gli indirizzi dei blocchi, per file di grandi dimensioni vengono utilizzati blocchi indiretti (un blocco che contiene indirizzi di altri blocchi) per gestire più blocchi di dati.

Implementazione delle directory

Il principale ruolo delle directory è associare un nome ASCII di un file alle informazioni necessarie per localizzare i dati sul disco. Queste informazioni variano a seconda del sistema utilizzato per implementare i file. Nei moderni sistemi i nomi dei file possono variare con caratteri da 1 a 255. Per gestire questa variabilità, si utilizzano due modi per strutturare le directory:

- Struttura con Header di lunghezza fissa : ogni voce nella directory inizia con un header di lunghezza fissa e termina con il nome del file. Ogni file termia con un carattere speciale, che viene utilizzato anche più volte per "riempire" (padding) il nome affinché questo abbia un numero intero di parole (32 o 63 bit in base al sistema). Questo sistema può tuttavia portare a una Frammentazione Interna, in quanto quando un file viene cancellato il suo spazio potrebbe non poter essere utilizzato in modo efficiente se il nuovo file da inserire non ha la stessa lunghezza (si possono creare piccoli "buchi" inutilizzabili).
- Utilizzo di Heap per i nomi dei file : i nomi dei file sono salvati in un heap, struttura dati che permette di memorizzarli in modo dinamico e non sequenziale. Quando si aggiunge il nome di un file, lo spazio viene allocato dinamicamente dall'heap, e non è necessario riempire i nomi dei file fino a raggiungere lunghezza fissa.

In aggiunta ai due metodi principali, per implementare la ricerca dei file all'interno di una directory, è stato introdotto l'uso delle tabelle hash.
Quando si cerca un file, il suo nome viene trasformato attraverso una funzione di hashing in un indice numerico che punta ad una posizione specifica della tabella. Se più file dovessero generare lo stesso indice (collisione) allora viene utilizzata una lista concatenata per collegare i file che hanno generato lo stesso hash. Un'ulteriore ottimizzazione è l'uso di caching per memorizzare i risultati delle ricerche precedenti.

N.B. : File nel disco rigido (non volatile). Quando lo apro l'I-node a lui associato (se è questa la struttura scelta per gestire i file) deve essere caricato in RAM.

File condivisi e Link nel File System

Quando si parla di File condivisi, si fa riferimento alla possibilità per più utenti di accedere e lavorare contemporaneamente allo stesso file in ambiti collaborativi.
Esistono due principali modalità di collegamento per un file:

- Hard Link : punta direttamente all' I-node del file condiviso. E' vantaggioso in quanto è necessario mantenere in RAM il singolo i-node indipendentemente dal numero di link creati per lo stesso file. Il file condiviso tuttavia rimane nel sistema fin quando non sono stati rimossi tutti i riferimenti ad esso, e ciò può portare confusione sulla proprietà del file in quanto questo viene eliminato dal sistema solo quando viene rimosso l'ultimo link che lo referenzia. Gli hard link non possono essere creati per directory.
- Soft Link : questo tipo di collegamento punta al nome del file e non al suo I-node. E' una modalità più flessibile in quanto consente di riferirsi a nomi del file al di fuori dei confini del file system locale e anche su macchine remote. Tuttavia è meno efficiente in termini di spazio poiché richiede un I-node per ogni link creato. Inoltre i soft link diventano invalidi se il file originale viene rimosso.

Entrambe le modalità presentano dei problemi comuni. Anzitutto, dal momento che vi sono più collegamenti allo stesso file, i programmi di backup o di ricerca possono dover trattare il file più volte aumentando il rischio di duplicazione dei dati. L'altro problema risiede nella sicurezza in termini di proprietà e permessi di accesso.

Gestione dello spazio su disco

La gestione dello spazio su disco è strettamente correlata a diversi aspetti chiave che influenzano l'efficienza e le prestazioni complessive dei SO.

Dimensione a blocchi : i file su disco possono essere memorizzati in blocchi di diverse dimensioni. La scelta della dimensione di un blocco è compromessa tra efficienza nello spazio e prestazioni nel trasferimento dei dati, infatti:

- Blocchi più grandi consentono di trasferire più dati in una singola operazione di lettura o scrittura, tuttavia portano ad uno spreco di spazio se i file sono piccoli.
- Blocchi piccoli invece riducono lo spreco di spazio poiché è meno probabile che rimangono aree non utilizzate all'interno di un blocco. Tuttavia gestire file su più blocchi può aumentare il tempo necessario per accedere ai dati.

Gestione dei blocchi liberi : esistono due principali metodi:

- Linked List : Utilizza una lista concatenata di blocchi liberi, ciascun blocco quasi pieno (richiede meno memoria per mantenere la lista) ma meno efficiente per dischi molto frammentati o grandi in quanto richiede molte operazioni di ricerca sequenziale per trovare un blocco libero.
- BitMap : usa una mappa di bit, 0 blocco libero e 1 blocco occupato. E' più efficiente delle linked list nella ricerca (è possibile scorrere rapidamente la bitmap), tranne se il disco è quasi pieno.

Quote del disco : si tratta di limitazioni gestite dal SO sull'utilizzo dello spazio del disco, imposte sui singoli utenti per evitare l'utilizzo eccessivo di spazio. Due tipi:

- Limite Soft : il limite può essere momentaneamente superato, ma non permanentemente. Se infatti lo spazio utilizzato rimane sopra il limite soft per un certo periodo di tempo (determinato dal sistema)m l'utente non sarà in grado di salvare nuovi dati finché non libera spazio o ottiene una nuova quota.
- Limite Hard : il limite non può essere superato in nessuna circostanza.

Performance del File System

Il gap tra i tempi di accesso della RAM e della memoria di masse, seppur diminuito con l'avvento delle SSD, rimane significativo. Questa differenza evidenzia la necessità di progettare

file system con ottimizzazioni specifiche per ridurre al minimo gli accessi al disco/SSD. Ci sono due tecniche principali per ottimizzare il file system: l'uso della cache e l'allocazione dei blocchi.

1. Uso della cache : la cache (situata in RAM) riduce i tempi di accesso al disco mantenendo i blocchi più usato in memoria. Esistono due concetti principali di caching:
    - Buffer Cache : memorizza blocchi di dati del disco in RAM per ridurre gli accessi diretti al disco.
    - Page Cache : memorizza le pagine del File System Virtuale (VFS) in RAM. Il VFS è una componente del SO che fornisce un'interfaccia standard ai vari tipi di file system, indipendentemente dal loro formato o posizione fisica. Inoltre il VFS agisce come interfaccia tra file system e kernel, facilitando un accesso più rapido ai file senza dover passare attraverso tutto il codice del file system ogni volta.

Le cache sono inoltre ottimizzate per evitare la duplicazione di dati tra Buffer Cache e Page Cache.

Per implementare la cache si utilizzano algoritmi che controllano se un blocco necessario è già presente in cache prima di effettuare un accesso al disco.
L'algoritmo LRU (Least Recently Used) è il più comune, ma può causare problemi di coerenza se non gestito correttamente, specialmente con blocchi critici come gli I-node.
Alcuni SO dividono i blocchi in categorie basate sull'importanza per evitare problemi d'incosistenza.

2. Allocazione dei blocchi : un'altra tecnica, usata per i dischi magnetici, è quella di ridurre la frequenza di movimenti del braccio mettendo vicini (blocchi contigui) i blocchi ai quali è probabile che si acceda in sequenza. Quando viene scritto un file di output, il file system deve quindi allocare i blocchi uno alla volta.
    - Se i blocchi liberi sono registrati in una bitmap e questa bitmap è memorizzata interamente nella memoria principale (RAM), è abbastanza semplice scegliere un blocco libero che sia il più vicino possibile al blocco precedente.
    - Se invece si usa la lista dei blocchi liberi, parte della quale sta su disco, è molto più difficile allocare blocchi vicini l'uno all'altro (la ricerca di blocchi liberi richiede più tempo e può risultare in una frammentazione maggiore).

Altre tecniche per migliorare la performance del file system sono:

- Read Ahead : tecnica usata per ottimizzare le letture dai dischi. Prevede le richieste future di dati e li carica in anticipo nella memoria cache. E' particolarmente efficace per accessi sequenziali, dove anticipa le letture successive, ma meno per accessi casuali.
- Deframmentazione dei dischi : Una pratica utile (sebbene sia meno necessaria negli SSD) per ridurre la frammentazione dei file sul disco, che può verificarsi naturalmente con l'uso

continuo. Sistemi operativi moderni gestiscono la deframmentazione in maniera automatica senza richiedere l'intervento dell'utente.

- Compressione : il processo di ridurre la dimensione di un file eliminando ridondanze o utilizzando metodi di codifica efficienti. Questo permette di risparmiare spazio di archiviazione e di velocizzare il trasferimento dei dati.
- Deduplicazione : utile per eliminare i duplicati di file, può essere eseguita in tempo reale o come processo post-elaborazione, a seconda delle esigenze del sistema e dell'hardware disponibile.
- Posizionamento degli I-node : la posizione degli I-node influisce significativamente sulle prestazioni del file system:
  - Posizionamento tradizionale : posizionati vicino all'inizio del disco, porta a tempi di ricerca più lunghi, poiché il braccio del disco deve spostarsi dall'inizio del disco ai blocchi dati sparsi.
  - Centrare gli I-node : Un'alternativa più efficiente è quella di posizionare gli I-node al centro del disco per ridurre il tempo di ricerca medio. Inoltre si può dividere il disco in gruppi di cilindri, ciascuno con i proprio I-node, blocchi e lista dei blocchi liberi.

Affidabilità del File System

L'affidabilità di un file system è fondamentale per la protezione dei dati.
Le principali minacce includono : Guasti del disco, Interruzioni di energia, Bug software, errori umani e malware che possono compromettere la sicurezza dei dati.

Per mitigare questi rischi, i Backup sono essenziali.
I principali scopi dei backup sono recupero da disastri (come crash del disco o disastri naturali) e recupero da errori umani (come l'eliminazione accidentale di un file).
Esistono due strategie di Backup:

- Backup Fisico : consiste nel copiare sequenzialmente tutti i blocchi del disco, indipendentemente dal loro contenuto. Questo metodo è semplice e veloce, ma comporta alcune limitazioni. In particolare, non è flessibile nel saltare specifiche directory o escludere file non necessari, rendendo quindi difficile la realizzazione di Backup incrementali (backup che interessano solo file modificati dall'ultimo backup). Inoltre non è possibile ripristinare singoli file senza ripristinare l'intero sistema.
- Backup Logico : il backup logico si occupa di copiare file e directory specifici, permettendo una maggiore flessibilità rispetto al backup fisico. Questo lo rende ideale per backup incrementali o differenziali, in cui solo i file modificati vengono salvati.

L'algoritmo del backup logico consiste in quattro fasi principali:

1. Rilevamento delle modifiche : fase che implica il controllo dei file e directory modificate rispetto all'ultimo backup.
2. Pulizia della bitmap : vengono esclusi file e directory non modificate dall'ultimo backup.
3. Backup delle directory : si effettua il backup delle directory contrassegnate insieme ai loro attributi.
4. Backup dei file : si effettua il backup dei file contrassegnati insiemi ai loro attributi.

In generale, la procedura di ripristino consiste nella creazione di un nuovo file system seguito dal ripristino del backup completo per poi procedere con quelli incrementali.

NB : Le pipe sono un meccanismo di comunicazione interprocesso (IPC, Inter-Process Communication) utilizzato nei sistemi operativi Unix e Unix-like, come Linux. Permettono a due processi di comunicare tra loro scambiando dati in modo sequenziale, simile a un flusso, attraverso una connessione unidirezionale.

La Coerenza dei Dati nel file system è essenziale per evitare la corruzione dei dati, che può avvenire in caso di un crash durante la scrittura dei blocchi. Comandi come *fsck* in UNIX o *sfc* in Windows sono utilizzati per verificare e riparare la coerenza all'avvio.

Controllo dei blocchi : si tratta di una procedura utilizzata per verificare l'allocazione dei blocchi sul disco, assicurandosi che non vi siano blocchi persi o duplicati (sia tra file che tra blocchi liberi). Questa procedura prevede la costruzione di due tabelle : la Tabella dei contatori per Blocchi di File che contiene il conteggio di quante volte un blocco è referenziato da un file e la Tabella dei contatori per i Blocchi Liberi che tiene traccia di blocchi che dovrebbero essere liberi e quindi non assegnati ai file.

Analizzando gli I-node (che contengono informazioni sui blocchi di dati di ogni file) si possono identificare delle anomalie come:

- Blocchi mancanti : blocchi non referenziati né da file né dalla lista dei blocchi liberi e quindi "persi" nel sistema.
- Blocchi duplicati : blocchi referenziati sia dalla lista dei blocchi liberi che da uno o più file.

Per risolvere questi problemi i blocchi mancanti possono essere aggiunti alla lista dei blocchi liberi, rendendoli nuovamente disponibili per l'allocazione mentre i blocchi duplicati devono essere correttamente riassegnati a un solo file (in modo che ogni blocco sia utilizzato da un solo file o sia libero, ma non entrambi).

Journaling : utilizzato in file system come NTFS e ext4, è una tecnica che protegge l'integrità del file system in caso di crash registrando le operazioni prima ancora di eseguirle in un registro (journal). Vediamo il suo funzionamento:

- Fase di Registrazione : prima di eseguire qualsiasi modifica di dati sul disco, viene scritto un record nel journal che descrive dettagliatamente l'operazione pianificata (include anche i blocchi da modificare e come li si vuole modificare). Questo passaggio è cruciale poiché assicura che il sistema possa sapere quali operazioni erano in corso in caso di crash.
- Fase di Esecuzione : dopo aver registrato l'operazione nel journal, questa viene eseguita effettivamente.
- Fase di Conferma : una volta che l'operazione è stata svolta con successo, il journal viene aggiornato per indicare che l'operazione è stata portata a termine (commit).

Se il sistema subisce crash prima che una modifica sia completata, al riavvio, il file system consulta il journal per verificare lo stato delle operazioni ancora in sospeso.
Se vengono trovate operazioni registrate nel journal che non sono state ancora completate, il file system può completarle o annullarle, garantendo che il file system ritorni a uno stato coerente e evitando la corruzione dei dati.

I vantaggi del Journaling risiedono nel mantenere coerenza e integrità dei dati nonché un recupero rapido in caso di crash.

Eliminazione sicura : La cancellazione standard dei file non rimuove completamente i dati, il sistema operativo segna lo spazio occupato dal file come disponibile per essere sovrascritto. Tuttavia, i dati originali rimangono fisicamente presenti sul disco fino alla successiva sovrascrittura.
Per la sicurezza dei dati, è stata implementata l'eliminazione sicura, un processo che sovrascrive i dati originali con dati casuali o con sequenze di zeri multiple, ripetendo l'operazione più volte e garantendo che i dati eliminati non siano più recuperabili.

Cifratura del disco : tecnica di sicurezza avanzata che consiste nel crittografare l'intero contenuto di un disco, rendendo i dati inaccessibili e illeggibili a chiunque non disponga della chiave di decrittografia corretta.

File System Virtuali VFS : si tratta di una struttura astratta che consente al SO di supportare diversi tipi di file system in modo unificato. Il VFS funge da intermediario tra le chiamate di sistema (come le operazioni sui file, in generale le applicazioni utente) e i file system reali., permettendo così al kernel del SO di interagire con vari file system senza doversi preoccupare delle loro specifiche implementazioni.
Si basa su un livello di codice comune che interagisce con i file system reali sottostanti.
In particolare l'interfaccia superiore interagisce con le chiamate di sistema POSIX di processi utente, mentre quella inferiore è composta da decine di funzioni che il VFS può inviare ai file system reali sottostanti.
Nel contesto VFS alcune componenti chiave sono:

- SuperBlock : è un descrittore ad alto livello che rappresenta un file system specifico. Esso contiene informazioni fondamentali riguardo quel file system, come la dimensione, il tipo, lo spazio libero e altro ancora.
- V-Node : Il Virtual Node è un'astrazione che rappresenta un singolo file o directory all'interno del VFS. Contiene i metadati essenziali per quello specifico file come permessi di accesso, proprietario, timestamp. La particolarità dei v-node è che offrono un'interfaccia comune per i file, indipendentemente dal file system sottostante in cui si trovano.
- Directory : gestiscono nel VFS la mappatura dei nomi dei file e delle sottodirectory ai rispettivi V-Nodes. Ciò permette di organizzare i file e navigare attraverso di essi in modo coerente, indipendentemente dal file system in cui risiedono.

Quando un nuovo file system viene registrato nel VFS, esso deve fornire un insieme di funzioni specifiche che il VFS utilizzerà per interagire con esso.
Queste funzioni sono necessarie per eseguire operazioni di base come lettura, scrittura, apertura, chiusura e altro. Questo meccanismo consente al VFS di essere estensibile e di supportare nuovi file system senza dover modificare il resto del kernel.

RAID

Il RAID è una tecnologia che migliora le prestazioni e l'affidabilità della memoria non volatile attraverso la gestione di dischi multipli. Ne esistono di diversi tipi:

- Raid di Livello 0 : utilizza lo striping, ossia i dati sono divisi in segmenti più piccoli (stripes) e distribuite simultaneamente su più dischi del RAID. Ciò migliora le prestazioni in quanto i dati possono essere letti e scritti in parallelo. Tuttavia Raid 0 non fornisce alcuna tolleranza agli errori o ridondanza, per cui se un singolo disco fallisce tutti i dati vengono persi.
- Raid di livello 1 : questo RAID utilizza il mirroring, per cui i dati vengono duplicati su due dischi separati. Ciò offre ridondanza, che migliora la tolleranza agli errori poiché i dati sono mantenuti su copie identiche in entrambi i dischi. Se un disco fallisce, l'altro può continuare senza perdita di dati.
- Raid di livello 2 : RAID 2 utilizza la codifica di Hamming per la correzione degli errori e lo striping a livello di bit. Questo tipo di RAID è molto complesso e richiede un gran numero di dischi, motivo per cui è raramente implementato in pratica.
- RAID di livello 3 : RAID 3 utilizza lo striping a livello di byte con un singolo disco dedicato alla parità, che consente di ricostruire i dati in caso di guasto di un disco. Le unità devono essere sincronizzate poiché i dati sono distribuiti a livello di byte.
- RAID di livello 4 : RAID 4 è simile al RAID 3 ma utilizza lo striping a livello di blocco anziché a livello di byte. Un singolo disco è dedicato alla parità, il che può creare un collo di bottiglia durante le operazioni di scrittura poiché tutte le operazioni di parità devono essere scritte su un unico disco.

- RAID di livello 5 : RAID 5 distribuisce i dati e la parità tra tutti i dischi, eliminando il collo di bottiglia presente in RAID 4. E' una dei livelli RAID più utilizzati poiché offre un buon equilibrio tra prestazioni, capacità e tolleranza agli errori.
- RAID di livello 6 : RAID 6 è simile a RAID 5 ma utilizza due blocchi di parità distribuiti, il che consente di tollerare la perdita di due dischi anziché uno solo. Questo migliora ulteriormente l'affidabilità, ma a costo di una maggiore complessità e overhead.
- RAID 0+1 : RAID 0+1 combina lo striping di RAID 0 con il mirroring di RAID 1. Questo offre sia le prestazioni di RAID 0 che la ridondanza di RAID 1, ma richiede un minimo di quattro dischi.