Capitolo 14 : implementazione del File System

Table of contents

Piccolo recap

Un file è un contenitore **logico** contenente delle informazioni. Solo il creatore del file è a conoscenza del suo contenuto ed il formato utilizzato.

Sul disco un file è tipicamente spezzettato in blocchi che corrispondono ai blocchi degli hard disk. In un SSD viene emulata una disposizione dei file simile agli HD, per uniformità. I vari blocchi sono solitamente di 512 bytes.

A bassissimo livello il SO fa utilizzo dei **Device Drivers** che gestiscono l'interfaccia del disco.

File System a strati

1. Programmi applicativi
2. File system logico
3. Modulo di organizzazione dei file
4. File System base
5. Controllo I/O
6. dispositivi

> 6 è il livello più a basso livello

Device Drivers

I drivers a bassissimo livello sono in grado di dare e ricevere comandi del tipo "prendi questo blocco", che storicamente veniva indicato con 3 coordinate: CHS (cylinder head sector);

Attualmente si utilizza una tecnica di **accesso a blocchi** del tipo **LBA**: Questo perchè da alcuni anni, con la crescita del consumo dei dischi, questi vengono visti come un unico array di blocchi con un indirizzo logico; piuttosto che dire "prendi un blocco presente su questo cilindro, utilizzando questa testina, e poi quel particolare settore" (quindi tre coordinate), viene detto semplicemente "prendi il blocco n";

dopo questo comando è direttamente il controllo del disco che converte il comando in **azioni da compiere** (e quindi le coordinate).

Cosa trovo su un disco?

- **Boot control block** : software che viene lanciato per effettuare il boot, qualora il disco in questione è quello scelto per il boot.

Cosa succede quando apro un file

Quando apro un file vengono create delle strutture per processo per sistema, che mi permettono di accedere a quel file; gestiscono sopratutto il puntatore all'interno del file.

Se più processi, che non hanno nulla a che vedere tra loro, tentano di aprire lo stesso file, possono farlo, ma deve esserci un puntatore separato per ognuno dei processi che lo sta aprendo.

Processi che sono in relazione di parentela padre-figlio, devono invece avere **lo stesso puntatore**; in sistemi UNIX, in pratica la cosa funziona nel seguente modo:

C'è una tabella "open-file table" dove vengono specificati i file aperti **per processo**. Questa tabella punta ad una **tabella di sistema** dei file aperti. Il puntatore al file è nella tabella di sistema, se ci sono processi che aprono lo stesso file (per conto proprio) hanno due entry nella tabella di sistema, e quindi hanno puntatori separati.

Se invece c'è una relazione di parentela padre-figlio, il figlio eredita la stessa tabella del padre (tabella per-process) e quindi utilizza lo stesso puntatore del padre, che è lo stesso puntatore al file.

Implementazione di una directory

- **Lista lineare** : Scelta poco intelligente
- **Tabella hash** , ovvero la soluzione adottata ai giorni d'oggi per maggiore efficienza, siccome le liste potrebbero diventare particolarmente enormi.

Il FS potrebbe essere tenuto ordinato mediante delle particolari strutture dati.

Metodi di allocazione - cosa accade sul disco

Nel disco ho milioni di blocchi, alcuni liberi altri occupati. Per creare un file, quali blocchi utilizzo?

Sicuramente dovrò utilizzare dei blocchi **liberi**, altrimenti potrei cancellare files di altri processi o utenti.

La soluzione a cui si potrebbe pensare è la seguente: mi scelgo un bel pezzo continuo di blocchi liberi, e vado a salvare al suo interno il mio file. **SBAGLIATO!** questa tecnica non funziona per il semplice fatto che i files funzionano ad **accrescimento**; il file, al momento della creazione, è vuoto. Quando inizio a scrivere al suo interno le sue dimensioni aumentano sempre di più, quindi se non prendo delle opportune misure, questo tipo di allocazione contigua crea i soliti problemi della gestione dinamica della memoria.

🏁 05-27 1:00

L'allocazione contigua sarebbe la soluzione perfetta, se solo funzionasse; sarebbe infatti un metodo di allocazione molto veloce, proprio per la sua natura, visto che i byte sarebbero disposti in maniera sequenziale.

Allocazione linkata

Come funzina?

Non chiedo che il file sia **tutto** contiguo; so solo che il file è composto da vari "blocchi" posti in vari posizioni della memoria. In ogni blocco, viene posto un puntatore al blocco successivo, in modo tale che nella directory dove è posizionato il file è presente **il puntatore che punta al primo blocco**, dopodichè gli altri blocchi vengono prelevati seguendo i vari link

Questa soluzione **non funziona tanto bene**, per i vari motivi:

1. Se devo accedere al decimo blocco, sono costretto ad effettuare ad effettuare tutte le nove operazioni di lettura prima di arrivare al blocco che mi interessa.
2. **Motivi di affidabilità:** se per caso uno dei blocchi "viene distrutto", non perdo solo i dati contenuti in quel blocco, ma anche tutti i contenuti successivi.

Variante: File-Allocation table

Questa è una variante del sistema precedente; è adottata dai sistemi Microsoft, non tanto per i sistemi operativi, quanto per le unità rimovibili.

Anche questa è una tecnica di allocazione **linkata**, ma sono stati rimossi i problemi maggiori: i link non sono separati e presenti uno per blocco, ma sono salvati esternamente.

Esiste quindi una tabella chiamata FAT, dove entro (con il blocco di testa), e trovo tutti i puntatori ai blocchi successivi.

Come funziona l'accesso diretto?

Purtroppo non tanto bene: sono costretto a seguire la catena dei puntatori, ma se non altro questa tabella, al momento della prima lettura, viene caricata in memoria, quindi l'accesso (seppur sequenziale) è più veloce.

Per quanto riguarda **l'affidabilità**, questa tabella **non deve essere mai distruttura**, il che significa che vengono effettuate delle copie multiple della tabella.

Questo metodo di allocazione non è particolarmente veloce, e non va assolutamente bene per i dischi grandi (motivazioni ovvie).
Per questo tipo di dischi viene utilizzato NTFS.

Allocazione con indice

Nei sistemi UNIX viene utilizzato questo tipo di allocazione.

Quando viene formattato il disco (gergo windows), o **"make the disk"** (gergo unix, usando il comando `mkfs`), viene creato un File System dove non sono presenti files, dove sono presenti delle **informazioni di riferimento** ed una **directory di testa** (vuota).

Queste informazioni sono dei blocchi detti **blocchi indice**, che conterranno tutte le informazioni relative ai files. Quando eseguiamo il comando mkfs il disco immediatamente diventa più piccolo: ad esempio abbiamo un disco di 1GB e dopo aver usato il comando diventa di 800Mb.

Nei sistemi UNIX nelle directory è presente solo il **nome**(visto dall'esterno) ed un numero che è il numero del **blocco indice corrispondente a quel file.**

Se il file pippo.txt corrisponde all'indice 3, vado nel blocco indice 3 ed in quella locazione sono presenti **tutte le informazioni** del file, come **proprietario, permessi, creazione, ecc**.

Se abbiamo in due directory separate due file con nomi diversi, ma aventi lo stesso numero del blocco indice, abbiamo (in pratica) un **hard link**. Basta mettere lo stesso numero del blocco indice per creare un **alias** per quel file; nel blocco indice sarà riportato che sono presenti due alias per quel file.

Quindi, nel blocco indice sono presenti vari blocchetti, che corrispondono ai file, dove dove sono presenti i puntatori alle locazioni di memoria che contengono i file.

Questo metodo permette **l'accesso diretto**, perchè basta capire qual è il blocco relativo al file, vi accedo (dove sono presenti i puntatori alle locazioni di memoria contenenti il file), capisco quale puntatore mi serve e vi accedo.

Schema combinato: UNIX UFS

Come si fa nei sistemi UNIX ad avere in un blocco molto piccolo, i puntatori a molti blocchi?

Si utilizzano i **livelli**.

Sono presenti tutte le informazioni sul file, solitamente chiamati **metadati**, un piccolo numero di puntatori diretti a blocchi (**direct blocks**), dei blocchi che puntano ad un altro blocco contenente i dati (**single indirect blocks**), blocchi con **doppia indirezione**, blocchi con **tripla indirezione** e così via.

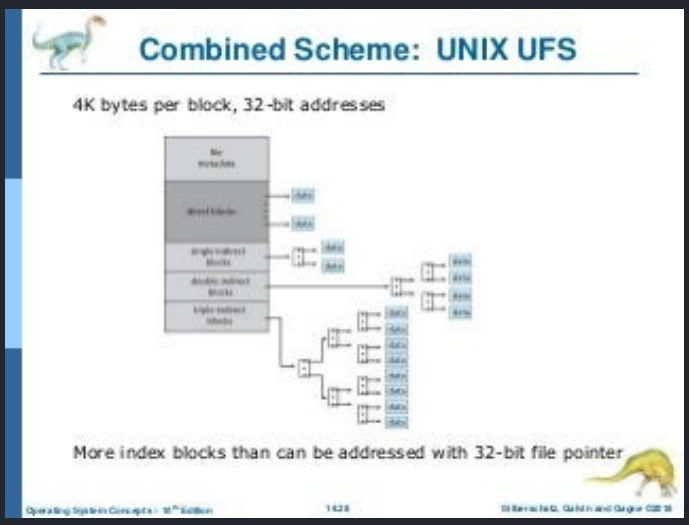
Cosa significa??

Sui file piccoli, questo sistema è particolarmente veloce, perchè sfruttando i **direct blocks** il sistema va direttamente a prelevare i dati.

Via via che il file diventa più grande, sono costretto ad effettuare sempre più accessi sul disco per prelevare i **puntatori** che puntano ai dati.

Quindi

Questo metodo è molto veloce sui file piccoli (maggiormente usati) e diventa via via più lento sui file grandi; d'altra parte permette di avere files di dimensioni elevatissime.



Gestione dello spazio libero

Con questi tre metodi di allocazione (contiguo-ed estensioni, linkato-o con fat, con indice) abbiamo capito che nella maggior parte dei casi i files sono "spezzettati" sul disco.

Lavorare con un disco particolarmente pieno, significa inevitabilmente avere delle prestazioni inferiori.

I programmi che effettuano la **compattazione** (che spostano tutti i dati da una parte lasciando una parte contigua di memoria libera) effettuano uno **spostamento di blocchi** in modo che ogni file sia allocato in maniera **contigua**.

Quindi, dopo aver effettuato questa operazione, per un lasso di tempo molto piccolo, avremo un disco dove i files non sono assolutamente frammentati, e lo spazio libero è tutto da una parte.

Come si capisce qual è lo spazio libero?

Ovviamente i blocchi liberi non sono diversi da quelli pieni, quindi serve una struttura dati che tiene traccia dei blocchi liberi e pieni.

Un modo potrebbe essere quello di tenere una lista (UNIX), o più comunemente, si utilizza un **vettore di bit**.

In questa soluzione, abbiamo un lungo vettore, dove ogni bit rappresenta un blocco e ci dice se quel blocco è libero o occupato; questo è uno dei metodi più comunemente utilizzato.

Ovviamente, anche questo vettore di bit deve essere posizionato sul disco, andando a sottrarre spazio ai files; quando creiamo il FS, viene anche creato spazio per questo vettore di bit.

🏁 05-27 1:26

TRIMing

Questo tipo di utilities sono destinate a dischi allo **stato solido** (SSDs).

Gli SSD sono fatti con sistemi NAND e non è possibile accedere ad una **singola locazione di memoria** ma tipicamente l'accesso è a pagine (tipo la ram); inoltre questo tipo di sistemi, quando dobbiamo modificare i contenuti di un file, bisogna:

1. Prendere l'intera pagina
2. Cancellarla
3. Riscriverla interamente

Quindi, anche se abbiamo modificato un solo byte di un file, siamo costretti a riscrivere interamente la pagina (che è abbastanza grande).

Inoltre, gli accessi in memoria a stato solido, devono essere **distribuiti**, proprio perchè dopo un certo numero di accessi alla stessa locazione, la memoria potrebbe essere danneggiata.

Supporto TRIM

Alcune macchine hanno il supporto TRIM abilitato per ottenere le migliori prestazioni per i dischi SSD.

Efficenza e performance - Viene usato del caching?

Ovvio.

Se accediamo ad una directory per la prima volta dopo aver acceso il computer (desktop) noteremo che (solitamente) il led HDD si accende; questo indica che stiamo operando sul disco.

Se vi accediamo poco tempo dopo, la luce HDD non si accende, il che significa che i dati a cui abbiamo fatto accesso, sono stati salvati nelle cache.

I dispositivi, quindi, utilizzano tutta la memoria RAM disponibile per effettuare caching, e quindi velocizzare l'accesso al disco.

Dinamicamente viene utilizzato tutto lo spazio disponibile per velocizzare l'accesso al disco; via via che apriamo nuovi programmi questo spazio si riduce per fare spazio ai processi.

Questo tipo di caching viene chiamato **buffer caching**.

All'atto pratico...

Nei sistemi moderni la buffer cache è **unica** e viene usata sia per la memoria virtuale che per gli accessi al disco, in modo da ottimizzare il tutto.

Il disco d'appoggio (virtual mem) come è fatto?

Si potrebbe pensare di usare un unico grande file usato per la gestione della memoria virtuale.

Questo sistema non è particolarmente efficiente, perchè per accedere alla memoria virtuale devo entrare nella directory del file unico, ed usare i metodi di accesso del FS.

Soluzione

Se sono a conoscenza che le mie pagine di memoria sono di 4Kb, pechè non formattare a blocchi direttamente la memoria, visto che a differenza dei files le pagine non variano?

Quando installiamo linux, ci viene chiesto di creare una partizione per lo swap; oltre al disco normale usato per salvare i vari files, viene creata una piccola partizione sul disco, che serve unicamente per la gestione della memoria virtuale, usando una tecnica di formattazione diversa, opportunamente ideata per questa operazione (molto efficiente).

Solitamente la grandezza di questa memoria di swap andrebbe scelta come **il doppio** della memoria fisica RAM disponibile. LINUX consente, quando la partizione swap non basta, di aggiungere dei file di swap (più lenti) che però aiutano il processo della memoria virtuale.

Buffer cache - Quali sono le sue caratteristiche?

Questo tipo di cache è diversa dalle altre cache nel momento in cui l'accesso **è sequenziale**: nella maggior parte dei casi, l'accesso ai file è proprio sequenziale (come ad esempio la visione di un file). In questi casi la cache non tiene conto dei dati già visualizzati (ad esempio siamo al minuto 25 di un film, la cache non tiene in memoria il minuto 5 del film), proprio perchè leggiamo in sequenza.

Il comportamento della cache, in questo caso, è del tipo **free behind** e **read ahead**, ovvero svuota i dati già letti, e possibilmente inzia a leggere in anticipo il pezzo successivo (ad esempio viene caricato in cache il minuto 26 nel momento in cui vediamo il minuto 25 di un film).

Recovery

Ogni tanto, i dischi vengono corrotti, come i politici; e non ci possiamo fare nulla.

Il caso più semplice è quello della chiavetta che viene staccata al volo senza essere espulsa.

In questi casi il FS si porta in uno stato inconsistente, dove, ad esempio, dei blocchi, che erano stati marcati come occupati, sono in realtà vuoti perchè non abbiamo avuto il tempo materiale per scriverci.

Servono quindi delle **utility che ripristinino il file system**. Queste utility scansionano **tutte le directory** e le esaminano, in modo da verificare la loro consistenza. Come conseguenza abbiamo che queste utility funzionano molto male su dischi di grandi dimensioni.

La soluzione

La maggior parte dei FS sono basati sul **log**. Questo vuol dire che prima di eseguire una qualsiasi operazione, viene scritto in un log che il FS sta tentando di eseguire quella specifica operazione, e poi la esegue.

Se qualcosa "va storto", resta il log per verificare delle operazioni non compiute o compiute male; questo permette di **rieseguire** tutte le operazioni che non sono state completate, in modo da non corrompere nulla.

FINE CAPITOLO 14