Tipi di file system

File system strutturati in log

Con il passare del tempo i file system si sono evoluti, ma il problema maggiore rimane sempre il tempo di ricerca dei dischi.
Questo crea un **collo di bottiglia** nei file system, quindi sono stati progettati i **LFS** (Log-structured File Ststem).

Idea principale -> Al crescere della velocità delle CPU e delle memorie, crescono le cache su disco, in modo tale da soddisfare le richieste senza accedeer al disco.

> ⚠ **Altro problema**
>
> Le operazioni di scrittura per brani piccolo sono estremamente inefficienti

> 🖉 **Dove arrivano tutte queste piccole scritture**
>
> Nuovo file sistema UNIX, per scrivere il file devono essere scritti:
>
> - Gli I-node della directory
> - Il blocco della directory
> - L'I-node del file
> - Il file stesso
>
> Si potrebbero ritardare queste operazioni, per fare un'unica grande scrittura, ma ciò causerebbe grossi problemi di incoerenza in caso di crash.

I progettisti hanno tenuto conto di questa cosa, con l'idea quindi di strutturrare l'intero disco come un file di log.

Semplicemente tutte le operazioni di scrittura vengono inserite in un registro (log), poi sono periodicamente scritte su disco in un segmento finale del log.
Quando apro un file uso la mappa degli I-node per rintracciare il suo i-node, da questo trovo i blocchi degli indirizzi che mi servono che si trovano nel log.

> ⚠ **Il disco non è infinito**

Quindi il log potrebbe occupare tutto lo spazio su disco.

Per risolvere il problema, l'LFS ha un **thread cleaner**, cioè un pulitore il cui compito è vedere se gli I-node dei file sono ancora validi.
Se non lo sono vengono puliti e questa operazione è fatta periodicamente.

File system con journaling

Idea interessante dei file system con log --> Tenere un log di ciò che il file system sta per fare prima che lo faccia così, in caso di crash, si può verificare cosa stesse facendo prima di questo.

Questo tipo di file system è detto anche: **file system journaled** ed è usato da file system come: NTFS, ext4, ReiserFS.

Valutiamolo tramite un esempio: l'eliminazione di un file in UNIX. Questa operazione richiede tre passaggi:

1. Eliminare i file dalla sua directory.
2. Rilasciare L'I-node nel pool degli I-node disponibili.
3. Restituire tutti i blocchi del disco al pool dei blocchi del disco liberi.

L'ordine in cui i passaggi sono eseguiti, in assenza di crash, non ha importanza, mentre invece ha importanza in caso di crash.

Il primo passaggio è stato completato, poi crash

I-Node e i blocchi del file sono in un limbo, non sarranno accessibili a nessun file, ma non potranno nemmeno essere riassegnati.

Crash dopo il secondo passaggio

Vengono persi i blocchi del file.

Cambia l'ordine delle operazioni

- I-node rilasciato per primo --> La vecchia voce della directory continuerà a puntare ad esso.

- Blocchi rilasciati --> Voce di directory valida punta a I-node che elenca blocchi nel pool, il che porta a file che condividono più blocchi.

Il file system con journaling scrive una voce di log che elenca le azioni da fare, la voce di log è scritta sul disco, poi cominciano le operazioni. Una volta finite le operazioni viene eliminato il log dal disco.
Se il sistema crasha, il file system può verificare il log per vedere se vi sono operazioni in attesa, se ci sono, vengno tutte rieseguite.

⚠ Il journaling funziona se le operazioni del log sono idempotenti

Quindi possono essere ripetute tutte le volte che servono senza creare danni.

Ricapitolando il funzionamento del journal

1. **Fase di registrazione** -> Prima di eseguire una modifica, il file system scrive un record nel journal.
2. **Fase di esecuzione** -> Completeta l'operazione, sil file system procede con la modifica dei dati sul disco.
3. **Fase di conferma** -> Completata l'operazione, il file system aggiorna il journal per indicare che l'azione è stata completata con successo.

Se si verifica un crash, al riavvio si consulta il journal, se ci sono operazioni registrate, ma non confermare, allora le completa.

Vantaggi del journaling -> Riduce le possibilità di corruzione del file system in caso di crash inaspettato, inoltre riduce il tempo di recupero dopo un crash, perché il dile system sa quali operazioni sono gia state fatte.

File system virtuali

In un SO possono essere usati molti file system diversi.

Windows

Ha come principale NTFS, ma ha anche partizioni legacy FAT-32 o FAT-16.

Windows gestisce file system diversi identificandoli con una lettera, poiché non ci sta motivo di fare interagire file system eterogenei.

UNIX invece cerca di fare interagire più file system contemporaneamente in una struttura singola.

Per fare interagire più file system in un unica struttura, UNIX usa un implementazione usata inizialmente da Sun Microsytems.

Idea --> Estrapolare la parte di file system comune a tutti i file system, facendo un codice separato che chiama i file system reali sottostanti per gestire i dati.

Chiamate di sistema relative ai file, fatte da parte di processi utente, sono tutte dirette al **file system virtuale.**

File System Virtuale --> Ha un'interfaccia superiore verso i processi utente e ha un'interfaccia inferiore verso i file system reali, questa interfaccia si chiama **interfaccia VFS**.

Interfaccia

- Superiore --> Interagisce con chiamate di sistema POSIX da parte di processi utente.
- Inferiore --> Composta da funzioni che il VFS può inviare ai file system sottostanti.

La maggior parte dei file system sotto il VFS rappresentano partizioni di un disco locale, ma in realtà il VFS nasce per supportare file system remoti tramite il protocollo **NFS (Network File System).

Funzionamento

All' avvio del sistema, il file system radice viene registrato con il VFS, questa registrazione avviene anche per tutti gli altri file system montati.
Nel momento in cui un file system si registra fornisce una tabella delle chiamate (Richieste dal VFS) al VFS.
Quindi dopo la registrazione il VFS sa che per svolgere una determinata funzione sul file system dovrà solo chiamare funzione x del vettore fornito.

Concetti chiave:

- **Superblock nel VFS** --> Rappresenta il descrittore ad alto livello di un file system nel VFS. Mantitene informazioni cruciali (Tipo, Dimensione) usato per identificare e interagire con il file system sottostante.

- **V-node nel VFS** --> Astrazione di un file individuale nel VFS, rappresenta un nodo nel VFS. Contiene **metadati** (Permesso, dimensione, proprietà, riferimenti a dati). Il VFS sfrutta i V-node per fornire un accesso indipendente dal file system ai file, permettendo operazioni di lettura scrittura attraverso vari file system.
- **Directory bel VFS** --> Struttura che gestisce l'organizzazione e il mapping dei file e delle sottodirectory all'interno del VFS. Permette al VFS di **mappare i nomi dei file ai loro V-node** indipendentemente dal file system in cui si trovano. Permette ai processi di interagire con un'interfaccia unificata.

Al montaggio, il file system, fornisce delle informazioni al VFS tramite le quali, il VFS, crea un v-node e mappa le operazioni specifiche del file system reale.

La gestione delle richieste di I/O avviene tramite il tracciamento dei fie aperti nei processi utente tramite v-node e tabelle dei descrittori dei file.

Per aggiungere nuovi tipi di file system basta che i progettisti forniscano delle funzioni che rispettano l'interfaccia VFS.