IMPLEMENTAZIONE DELLE DIRECTORY

Le directory sono tipologie speciali di File che hanno il compito di contenere altri oggetti, il loro contenuto non è altro che un elenco di File. Il contenuto di un generico record di una directory dipende dal File system usato e dalle sue strategie. Tutti i metadati vengono memorizzati all'interno della directory insieme al nome del File e al numero del primo blocco (nel caso di FAT). Questo cambia usando gli i-node, essi infatti gestiscono e contengono tutte le informazioni che riguardano il singolo File, grazie a questo, utilizzando gli i-node la voce della directory viene a semplificarsi poiché contiene solo il nome del File e il numero di i-node (conoscere il numero di i-node mi permette di individuarlo sul disco e dunque di portarlo in memoria). Per rappresentare la generica voce di una directory, abbiamo diverse strategie, distinguiamo però sempre una parte a dimensione fissa, ovvero, i metadati e una parte a dimensione variabile che riguarda il nome del File.
La strategia che spreca meno spazio consiste nel rappresentare l'entry di ogni File. Per ogni entry la prima informazione è la lunghezza in byte di tutta la entry, inclusa la parte variabile. Questa informazione "lenght", permette di scorrere rapidamente la entry senza dover attenzionare il contenuto dei singoli slot.
Detto questo, un modo per rappresentare il nome è quello di rappresentarlo byte per byte e farlo terminare con un carattere di terminazione, inoltre vengono aggiunti dei byte di padding perché il nome del File viene arrotondato alla word, per avere entry allineate.
Una strategia migliore permette di rappresentare gli stessi record in forma alternativa, ovvero, la parte iniziale della directory viene gestita come slot di dimensione fissa e contiene tutti i metadati, più un puntatore a un heap destinato a contenere in modo compatto tutti i nomi dei File in modo sequenziale, in questo modo, la prima parte del File è intercambiabile perché di dimensione fissa e, nell'heap, nonostante si debba compattare, la compattazione riguarda meno dati e dunque è molto più facilmente organizzabile. Per quanto riguarda la ricerca dei File all'interno della directory, i File system moderni

predispongono i dati in tabelle hash che garantiscono ricerche più efficienti. Inoltre, tutti i meccanismi di ricerca sono favoriti da una cache del disco che velocizza l'accesso, questa cache è implementata via software dal s.o. sfruttando la memoria centrale.

CONDIVISIONE DI FILE SU UN FILE SYSTEM

Quando due o più utenti vogliono condividere un file, nel caso degli i-node è possibile usare gli hard-link, ovvero, dei riferimenti agli i-node che voglio gestire all'interno delle directory. Tra i metadati dell'i-node esiste un contatore di hard-link che è utile al file system per questioni di integrità, infatti, in fase di rimozione, non faccio altro che cancellare la voce nella directory e ridurne il contatore, se questo contatore arriva a 0, vengono liberati anche tutti i blocchi dell'i-node, poiché il file è usato. Quando il file viene aperto in una dir B, essa diventa proprietaria, se dovesse essere indicizzata da una directory C, in seguito alla chiusura di B, si verificherebbe un'anomalia di accounting, ovvero, il file rimarrebbe puntato da una directory non proprietaria.

Un altro meccanismo è quello dei soft link, qui l'idea è quella di avere dei veri e propri oggetti che prendono il nome appunto di soft link ed hanno come contenuto il pathname dell'oggetto riferito, dunque aprendo un soft link in realtà andrò ad aprire il file a cui punta. Quando creo un soft-link, creo un nuovo oggetto che avrà il suo i-node e il suo contenuto, questo appesantisce il sistema ma è l'unico modo per fare riferimento a oggetti al di fuori del file system. Inoltre, i soft-link possono essere fatti a qualsiasi tipo di oggetto, questo cosa non è fattibile con gli hard-link poiché nella slot della directory andrò ad inserire direttamente il numero di i-node dell'oggetto che voglio referenziare, questo non sarebbe possibile poiché file system differenti hanno oggetti diversi a i-number uguali.

Gli hard link, possono essere fatti a directory ma trasformerei la struttura ad albero del file system in un grafo aciclico, questo potrebbe creare problemi con la scansione delle cartelle, rendendola infinita.

GESTIONE BLOCCHI LIBERI

Tra i vari compiti di cui si deve occupare il File system, è presente quello di tenere traccia dello stato di allocazione dei vari blocchi del disco. A tal proposito viene utilizzata una bitmap che rappresenta ogni singolo blocco, la bitmap ha una dimensione abbastanza piccola, essa può essere caricata tutta in memoria o può essere paginata e caricata all'evenienza. Per evitare che la bitmap sottragga spazio ai processi, essa viene rappresentata tramite una lista di blocchi liberi, questa strategia richiede più spazio sul disco ma vengono usati i blocchi liberi stessi, inoltre, vengono inseriti dei contatori per blocchi contigui. Nonostante ciò, le liste sono scarsamente utilizzate e si prediligono le bitmap.

CONTROLLI DI CONSISTENZA

Vengono effettuati dal File system o da apposite utility esterne e vengono innescati o periodicamente o a seguito di un crash del sistema. Quando avvengono questi eventi, la RAM perde il suo contenuto ma il File system, rappresentato dai blocchi del disco non è volatile e le metaoperazioni su di essi devono rimanere. Queste operazioni però non sono atomiche e il S.O. al riavvio non è in grado di capire dove è stato interrotto, per questo motivo, dopo il riavvio, non è detto che il File system sia coerente, infatti potrebbero esserci ad esempio blocchi segnati come occupati ma liberi in realtà poiché l'operazione è stata interrotta a metà. Per questo motivo, vengono attivate queste utility di controlli di consistenza che cercano di capire quali sono e se ci sono delle inconsistenze nel sistema ed eventualmente correggerle. Per effettuare questi controlli vengono mantenuti due vettori, uno che indica il numero di volte che il blocco è stato citato come usato mentre il secondo indica le volte in cui quel blocco è stato citato come libero. Nel caso degli i-node, ogni volta che un blocco viene citato all'interno di un i-node, il suo posto "in uso" va ad 1 nel vettore, ovviamente, per funzionare correttamente esso dovrà essere citato solo una volta. Dunque se un blocco presentasse 2, sarebbe un'anomalia.

Dopo aver scandito entrambi i vettori, per correttezza dovrebbero contenere tutti 0 o 1 e soprattutto l'uno deve essere il complementare dell'altro. Abbiamo vari tipi di anomalia:

- [0,0]: devo portare il blocco "libero" come uno per poterlo usare
- [0,2]: questa correzione serve ad evitare che lo stesso blocco venga dedicato a 2 entità differenti, per risolvere devo portare il 2 a 1.
- [2,0]: in questo caso, esistono 2 i-node che citano lo stesso blocco, questo causa anch'esso un danno ad almeno uno dei due oggetti. L'unico modo per correggere la cosa è quello di ricreare l'indipendenza, ovvero, allocare un secondo blocco esterno ad uno dei 2 i-node. Qui potrebbero verificarsi file corrotti.

JOURNALING

La maggior parte dei File system moderni impiegano questa tecnica che consiste nell'effettuare una meta-operazione (che contiene al suo interno una serie di micro operazioni) e di prevedere una struttura dati persistente chiamata Journal su cui scrive una descrizione dell'operazione che sta per effettuare. Ovviamente non viene scritto tutto ma solo ciò che tocca i metadati per motivi di spazio. Una volta eseguita l'operazione i log presenti sul journal vengono eliminati per fare spazio alla prossima meta operazione.
L'idea è quella di ripetere le operazioni che sono salvate sul log (presente sul disco) in caso di crash del sistema in modo da limitare i danni. Il journaling da maggiore robustezza ai metadati e offre un reboot veloce.
Questo meccanismo funziona solo se le operazioni presenti nel log sono idempotenti, ovvero, sono ripetibili senza provocare danni

CACHE DEL DISCO (BUFFER CACHE)

La cache del disco è completamente implementata via software grazie al supporto della memoria del sistema ed ha lo scopo di evitare alcune operazioni di input/output dal disco quando possibile. La logica è quella di memorizzare il contenuto dei blocchi più recentemente utilizzati nella speranza che gli stessi siano richiesti nell'immediato futuro.
Nei sistemi UNIX, qualsiasi blocco di memoria centrale libero viene usato come cache del disco, infatti, la cache non ha memoria fissa poiché qualsiasi blocco di RAM libero è sprecato. Tutti i frame che contengono informazioni del disco vengono inseriti in una lista doppiamente concatenata in modo da poter gestire lo spazio tramite l'algoritmo LRU, ovvero, ogni volta che uso un nodo, esso sarà portato in coda alla lista (MRU) e se ci sarà necessità sarà rimosso l'LRU, ovviamente il costo di operazioni su lista è trascurabile rispetto a quello della lettura da disco. Per quanto riguarda la lettura della cache invece, sarebbe troppo dispendiosa da effettuare su una lista, per questo viene usata una tabella hash che mappa i nodi della lista. Dunque questi dati sono mantenuti in una struttura ibrida che mi consente l'accesso rapido e l'implementazione dell'LRU.
Ovviamente la cache funziona anche in scrittura, infatti, permette di ritardare la scrittura su disco del blocco modificato contenuto in memoria, inoltre, ritardare le scritture permette di raggrupparle ed eseguirle in ordine fisico sul disco risparmiando tempo (inteso come tempo di posizionamento della testina).
D'altra parte, il blocco sporco verrebbe scritto solo quando scartato dall'LRU e quindi sincronizzato con il disco, questo però causa un problema poiché i blocchi sporchi e molto utilizzati, quindi molto importanti, verrebbero persi in caso di crash, i danni sarebbero molti. Però l'integrità è molto più importante dell'efficienza, per questo viene usato un LRU modificato, poiché la scrittura non viene ritardata il più possibile, ma anzi, vengono subito effettuate se riguardano metadati e ritardate in modo controllato se riguardanti il contenuto del file. Viene posto un limite al ritardo.

Un'ulteriore pratica di ottimizzazione che sfrutta la cache è la tecnica del read-ahead, ovvero, il S.O. fa una scommessa e preventivamente carica in cache anche i blocchi consecutivi a quello richiesto. Questo potrebbe portare grossi vantaggi se il processo usasse il file in modo sequenziale, questo però non è sempre vero e inoltre, essendo i file non obbligatoriamente in blocchi consecutivi, potremmo caricare blocchi che non c'entrano nel contesto. Parallelamente si presenta la tecnica del free-behind: l'idea è che se un processo sta leggendo un determinato blocco, esso difficilmente userà quelli precedenti, essi dunque possono essere rilasciati liberando spazio in memoria centrale.

ALTRE TECNICHE PER MIGLIORARE LE PRESTAZIONI

Il S.O. gestisce anche come scrivere i dati sul disco nel modo più efficiente possibile, ad esempio, gli i-node, essendo preallocati, sono disposti in una posizione ben precisa, ad esempio, possono essere allocati in modo contiguo nella traccia più esterna del disco, a ogni i-node corrisponderanno dei blocchi che contengono il file. Considerando che prima verrà letto l'i-node e poi il file, non conviene posizionarli distanti poiché pagheremmo il prezzo del posizionamento della testina. Per rendere più efficiente la lettura di un file, conviene posizionare gli i-node in tracce diverse e disporre il file in modo sequenziale sulla stessa traccia dell'i-node, in modo da far muovere il meno possibile la testina durante la lettura di un file.