**Parte II - Capitolo 11 - Realizzazione Del File System**

I dischi costituiscono la maggior parte della memoria secondaria in cui si conserva il file system e hanno due caratteristiche importanti: si possono riscrivere localmente ed è possibile accedere direttamente a qualsiasi blocco di informazioni del disco. Anzichè trasferire un byte alla volta, per migliorare l'efficienza dell'I/O, i trasferimenti tra memoria centra e dischi si eseguono per *blocchi*, ciascuno composto da uno o più settori (di solito 512 byte). Per fornire un efficiente e conveniente accesso al disco, il sistema operativo fa uso di uno o più file system che consentono di memorizzare, individuare e recuperare facilmente i dati. Un file system presenta due problemi di progettazione: il primo riguarda la definizione dell'aspetto del file system agli occhi dell'utente, cioè la definizione del file e dei suoi attributi, delle operazioni permesse su un file e dalla struttura delle directory; il secondo riguarda la creazione di algoritmi e strutture dati che permettano di far corrispondere il file system logico ai dispositivi fisici in memoria secondaria. In generale lo stesso file system è composto da livelli distinti, il più basso dei quali, il ***controllo dell'I/O***, è costituito dai *driver dei dispositivi* e dai gestori dei segnali d'interruzione e si occupa del trasferimento delle informazioni tra memoria centrale e memoria secondaria. Il ***file system di base*** deve soltanto inviare i generici comandi all'appropriato driver di dispositivo per leggere e scrivere blocchi fisici sul disco. Un blocco viene allocato in un buffer prima che possa verificarsi il trasferimento di un blocco del disco. Il ***modulo di organizzazione dei file*** è a conoscenza dei file e dei loro blocchi logici, così come dei blocchi fisici dei dischi; può quindi tradurre gli indirizzi dei blocchi logici negli indirizzi dei blocchi fisici. Il ***file system logico*** gestisce i metadati, cioè tutte le strutture del file system, eccetto gli effettivi dati (il contenuto dei file). Esistono svariati tipi di file system al giorno d'oggi e non è raro che i sistemi operativi ne prevedano più di uno. Ad esempio i CD-ROM sono in formato ISO 9660, UNIX adotta il File System UNIX (***UFS***), Windows NT, 2000 e XP adottano i formati FAT, FAT32 e NTFS, Linux adotta il *file system esteso*, anche se può funzionare con più di quaranta file system diversi.

Per realizzare un file system si usano parecchie strutture dati, sia nei dischi sia nella memoria. Fra le strutture dati presenti nei dischi abbiamo: il ***blocco di controllo dell'avviamento*** (*boot control block*) contenente le informazioni necessarie al sistema per l'avviamento di un sistema operativo da quel volume (blocco d'avviamento o boot block in UFS, settore d'avviamento della partizione o partition boot sector in NTFS); i ***blocchi di controllo dei volumi*** (*volume control block*) contengono i dettagli riguardante i relativi volumi o partizione (superblocco in UFS, tabella principale dei file o master file table in NTFS); le ***strutture delle directory*** sono usate per organizzare i file (comprendono i nomi dei file e i numeri di inode associati in UFS, sono memorizzate nella tabella principale dei file in NTFS); i ***blocchi di controllo di file*** contengono molti dettagli dei file (*inode* in UFS, memorizzate all'interno della tabella principale dei file attraverso una struttura di base di dati relazionale in NTFS). Un disco si può configurare in vari modi, secondo il sistema operativo che lo gestisce. Si può suddividere in più partizioni. Ciascuna partizione è priva di struttura logica (*raw partition*) se non contiene alcun file system; se nessun file system è appropriato si usa un ***disco privo di struttura logica*** (*raw disk*). Nella fase di caricamento del sistema operativo, si esegue il montaggio della ***partizione radice*** (*root partition*), che contiene il kernel del sistema operativo e in alcuni casi altri file del sistema. Un sistema operativo deve gestire contemporaneamente tipi diversi di file system. Un metodo ovvio, ma non ottimale, di realizzare più tipi di file system è scrivere procedure di gestione di file e directory separate per ciascun file system. Al contrario, la maggior parte dei sistemi operativi, UNIX compreso, impiega tecniche orientate agli oggetti per semplificare e organizzare in maniera modulare la soluzione. La realizzazione del file system quindi viene articolata in tre strati principali. Il primo strato è l'interfaccia del file system, basata sulle chiamate di sistema e sui descrittori dei file; il secondo strato è il ***file system virtuale*** (*virtual file system, VFS*) e svolge due funzioni importanti: separa le operazioni generiche del file system dalla loro realizzazione, definendo un'interfaccia uniforme, e permette la rappresentazione univoca di un file su tutta la rete. Il VFS è basato su una struttura di rappresentazione dei file detta ***vnode*** che contiene un indicatore numerico unico per tutta la rete per ciascun file; in questo modo può distinguere tra i file locali e quelli remoti.

La selezione degli algoritmi di allocazione e di gestione delle directory ha un grande effetto sull'efficienza, sulle prestazioni e l'affidabilità del file system. Il più semplice metodo di realizzazione di una directory è basato sull'uso di una ***lista lineare*** contenente i nomi dei file con puntatori ai blocchi di dati. Questo metodo è più facile da programmare, ma la sua esecuzione è onerosa in termini di tempo. Per creare un nuovo file occorre prima esaminare tutta la directory per essere sicuri che non esista già un file con lo stesso nome, quindi aggiungere un nuovo elemento alla fine della directory. Per cancellare un file bisogna cercarlo nella directory e, una volta trovato, rilasciare lo spazio che gli era stato assegnato. Lo svantaggio dato da una lista lineare di elementi di directory è dato dalla ricerca lineare di un file. Una soluzione potrebbe essere quella di produrre una lista ordinata che permette una ricerca binaria e quindi una riduzione del tempo medio d'attesa. Un'altra struttura dati che si usa per realizzare le directory è la ***tabella hash***. In questo metodo una lista lineare contiene gli elementi di una directory, ma si usa anche una struttura dati hash. La tabella hash riceve un valore calcolato usando come operando il nome del file e riporta un puntatore al nome del file nella lista lineare. Questa struttura può diminuire notevolmente il tempo di ricerca nella directory. L'inserimento e la cancellazione sono abbastanza semplici, anche se occorre prendere provvedimenti per evitare le ***collisioni***, cioè situazioni in cui da due nomi di file si ottiene un riferimento alla stessa locazione. Le maggiori difficoltà legate a una tabella hash sono la sua dimensione, che in genere è fissa, e la dipendenza dalla funzione hash da tale dimensione.

La natura ad accesso diretto dei dischi permette una certa flessibilità nella realizzazione dei file. Il problema consiste nell'allocare lo spazio per questi file in modo che lo spazio nel disco sia usato efficientemente e l'accesso ai file sia rapido. Esistono tre metodi principali per l'allocazione dello spazio di un disco. Per usare il metodo di ***allocazione contigua*** ogni file deve occupare un insieme di blocchi contigui del disco. Il numero dei posizionamenti (*seek*) richiesti per accedere a file il cui spazio è allocato in modo contiguo è trascurabile, così come è trascurabile il tempo di ricerca (*seek time*). L'allocazione contigua dello spazio per un file è definita dall'indirizzo del primo blocco e dalla lunghezza espressa in blocchi. Accedere ai file il cui spazio è assegnato in modo contiguo risulta quindi facile. L'allocazione contigua presenta però alcuni problemi. Una difficoltà riguarda l'individuazione dello spazio per un nuovo file che verrà affrontata in seguito. Il problema dell'allocazione contigua dello spazio dei dischi si può considerare un'applicazione particolare el problema generale dell'***allocazione dinamica della memoria***: il problema generale è infatti quello di soddisfare una richiesta di dimensione n data una lista di buchi liberi. Di solito viene usato l'algoritmo per la scelta del primo buco abbastanza grande (*first fit*) che risulta più rapida. Questi algoritmi soffrono di ***frammentazione esterna***: assegnando e liberando lo spazio per i file, lo spazio dei dischi viene frammentato in tanti piccoli pezzi (una soluzione è quella di copiare l'intero file system in un disco o un nastro, liberarlo e poi copiare nuovamente i dati nel file system). Un altro problema riguarda la determinazione della quantità di spazio necessaria per un file. I file sono ad accrescimento, quindi se un file riceve poco spazio, può diventare impossibile estenderlo. Per ridurre al minimo gli inconvenienti dovuti a questo problema, alcuni sistemi operativi fanno uso di uno schema di allocazione contigua modificato: inizialmente si assegna una porzione di spazio contiguo, e se questa non è abbastanza grande si aggiunge un'altra porzione di spazio, un'*estensione*. La locazione dei blocchi dei file si registra come una locazione e un numero dei blocchi, insieme con l'indirizzo del primo blocco dell'estensione seguente. Un metodo che risolve tutti i problemi dell'allocazione contigua è l'***allocazione concatenata***. Con questo tipo di allocazione ogni file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso. La directory contiene un puntatore al primo e all'ultimo blocco del file. Ogni blocco contiene un puntatore al blocco successivo. Per leggere un file occorre semplicemente leggere i blocchi seguendo i puntatori da un blocco all'altro. Con questo tipo di allocazione non esiste frammentazione esterna e per soddisfare una richiesta si può usare qualsiasi blocco della lista. Un file può continuare a crescere finchè sono disponibili blocchi liberi. L'allocazione concatenata presenta comunque alcuni svantaggi. Il problema principale riguarda il fatto che può essere usata in modo efficiente solo per i file ad accesso sequenziale: la funzione ad accesso diretto è inefficiente. Un altro svantaggio riguarda lo spazio richiesto per i puntatori. La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi contigui in ***cluster*** (*gruppi di blocchi*), e nell'allocare i cluster anzichè i blocchi. In questo modo i puntatori usano una quantità di spazio di disco che si riduce in modo proporzionale al numero di cluster e viene migliorata la produttività: si hanno meno posizionamenti della testina del disco e diminuisce lo spazio necessario per l'allocazione dei blocchi e la gestione della lista dei blocchi liberi. Il costo di questo metodo è dato da un incremento di frammentazione interna. Un altro problema riguarda l'affidabilità. Poichè i file sono tenuti insieme da puntatori sparsi per il disco, s'immagina che cosa accadrebbe se un puntatore andasse perduto o danneggiato. Una soluzione potrebbe essere quella di adottare liste doppiamente concatenate, ma lo schema sarebbe ancora più oneroso per ogni file. Una variante importante del metodo di allocazione concatenata consiste nell'uso della ***tabella di allocazione dei file*** (*file allocation table, FAT*) usato nei sistemi operativi MS-DOS e OS/2. Per contenere tale tabella si riserva una sezione del disco all'inizio di ciascun volume. La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco: si usa essenzialmente come una lista concatenata. L'elemento di directory contiene il numero del primo blocco del file; l'elemento della tabella indicizzata da quel numero di blocco contiene a sua volta il numer o del blocco successivo del file, fino ad arrivare all'ultimo blocco che ha come elemento della tabella il valore speciale di fine del file. I blocchi inutilizzati sono indicati nella tabella da valore 0. Lo schema di allocazione basato su FAT, se non si usa una cache, può causare un significativo numero di posizionamenti della testina. Tuttavia l'allocazione concatenata non è in grado di sostenere un efficiente accesso diretto perchè i puntatori sono sparsi per tutto il disco e si devono recuperare. L'***allocazione indicizzata*** risolve questo problema, raggruppando tutti i puntatori in una sola locazione: il ***blocco indice***. Ogni file ha il proprio blocco indice: si tratta di un array d'indirizzi di blocchi del disco. Lo spazio aggiuntivo richiesto dai puntatori del blocco indice è generalmente maggiore dello spazio aggiuntivo necessario per l'allocazione concatenata. È auspicabile quindi che il blocco indice sia il più piccolo possibile; se il blocco indice è troppo piccolo non può contenere un numero di puntatori sufficienti per un file di grandi dimensioni, quindi è necessario disporre di un meccanismo per gestire questa situazione. I meccanismi possono essere: *schema concatenato*, cioè un blocco indice è formato normalmente di un solo blocco di disco e ciascun blocco indice può essere letto e scritto esattamente con un'operazione; *indice a più livelli*, cioè viene impiegato un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello e così via. L'UFS utilizza invece uno *schema combinato* che consiste nel tenere i primi 15 puntatori del blocco indice nell'inode del file. I primi 12 puntano a blocchi diretti, cioè contengono direttamente gli indirizzi di blocchi contenenti dati del file. Gli altri 3 puntano a blocchi indiretti. Il primo puntatore di blocco indiretto è l'indirizzo di un blocco indiretto singolo (un blocco indice che non contiene dati, ma indirizzi di blocchi che contengono dati). Quindi c'è un puntatore di blocco indiretto doppio contenente l'indirizzo di un blocco che a sua volta contiene gli indirizzi di blocchi contenenti puntatori agli effettivi blocchi di dati. L'ultimo puntatore contiene l'indirizzo di un blocco indiretto triplo. Consideriamo ora le performance legate agli accessi ai dati di ciascun metodo di allocazione presentato. L'allocazione contigua risulta la migliore per qualsiasi tipo di accesso, l'allocazione concatenata risulta buona per l'accesso sequenziale, ma non usabile per l'accesso diretto.

Poichè la quantità di spazio dei dischi è limitata, è necessario riutilizzare lo spazio lasciato dai file cancellati per scrivere, se possibile, nuovi file. Per tenere traccia dello spazio libero in un disco, il sistema conserva una ***lista dello spazio libero*** in cui sono registrati tutti gli spazi liberi, cioè non allocati ad alcun file o directory. Spesso la lista dello spazio libero si realizza come una ***mappa di bit***, o ***vettore di bit***. Ogni blocco è rappresentato da un bit: se il blocco è libero 1, se il blocco è assegnato 0. I vantaggi legati a questo metodo sono la relativa semplicità ed efficienza nel trovare il primo blocco libero o n blocchi liberi consecutivi nel disco. Sfortunatamente i vettori di bit sono efficienti solo se tutto il vettore è mantenuto in memoria centrale, e viene di tanto in tanto scritto in memoria secondaria allo scopo di consentire eventuali operazioni di ripristino. È possibile tenere il vettore in memoria centrale se i dischi sono piccoli (tale soluzione non è applicabile a dischi più grandi). Un altro metodo di gestione degli spazi liberi consiste nel collegarli tutti, tenere un puntatore al primo di questi in una speciale locazione del disco e caricarlo in memoria. Questo schema non è efficiente in quanto per attraversare la lista è necessario leggere ogni blocco, con un notevole tempo di I/O. Una possibile modifica di questo metodo prevede la memorizzazione degli indirizzi di n blocchi liberi nel primo di questi. Un altro orientamento sfrutta il fatto che, generalmente, più blocchi contigui si possono allocare o liberare contemporaneamente, quindi anzichè tenere una lista di n indirizzi liberi, è sufficiente tenere l'indirizzo del primo blocco libero e il numero di blocchi liberi contigui che lo seguono. L'algoritmo di gestione dello spazio libero utilizzato da ZFS di Sun utilizza una combinazione di tecniche per controllare la dimensione delle strutture di dati e minimizzare l'I/O necessario a gestire tali strutture. Per prima cosa vengono create delle ***metalastre*** (*metaslab*) per dividere lo spazio sul dispositivo in parti che abbiano una dimensione gestibile. Ognuna di queste metalastre è associata a una *mappa di spazio*, cioè un registro di tutte le attività del blocco, in ordine cronologico, in formato di conteggio. Quando ZFS decide di allocare o liberare spazio su una metalastra, carica una mappa di spazio associata alla nella memoria in una struttura ad albero bilanciato, indicizzata agli scostamenti, e replica il log in tale struttura.

Dopo aver descritto le operazioni di allocazione dei blocchi e di gestione della directory, è possibile considerare i loro effetti sulle prestazioni e l'efficienza d'uso dei dischi. L'***efficienza*** di un disco dipende fortemente dagli algoritmi usati per l'allocazione del disco e la gestione della directory. In questo campo la scelta di UNIX di utilizzare gli inode risulta migliore rispetto agli altri sistemi operativi. Si devono tenere in considerazione anche il tipo di dati normalmente contenuti in un elemento della directory. Di solito si memorizza la data di ultima scrittura o la data dell'ultimo accesso per fornire informazioni all'utente e per determinare se per il file occorre la creazione o l'aggiornamento di una copia di riserva. Dopo aver scelto gli algoritmi fondamentali del file system le ***prestazioni*** possono essere migliorate in diversi modi. Alcuni controllori di unità a disco contengono una quantità di memoria locale sufficiente per la creazione di una *cache* interna al controllore sufficientemente grande da memorizzare un'intera traccia del disco alla volta. Eseguito il posizionamento della testina, si legge la traccia nella cache del controllore del disco a partire dal settore sotto cui si viene a trovare la testina in modo da ridurre il tempo di latenza. Il controllore trasferisce quindi tutte le richieste di settori al sistema operativo. Alcuni sistemi riservano una sezione separata della memoria centrale come ***cache del disco***, dove tenere i blocchi in previsione di un loro utilizzo entro breve tempo. Altri sistemi impiegano una ***cache delle pagine*** per i file, soluzione che utilizza tecniche di memoria virtuale per la gestione dei dati dei file come pagine anzichè come blocchi di file system. L'uso di indirizzi virtuali è molto più efficiente dell'uso dei blocchi fisici di un disco. Questo metodo utilizzato da diversi sistemi quali Solaris, Linux, Windows NT, 2000, XP è noto come ***memoria virtuale unificata***. Alcune versioni di UNIX e Linux prevedono la cosiddetta ***buffer cache unificata***. Ci sono altri aspetti che possono influenzare le prestazioni di I/O, come quelli che riguardano la necessità di scritture sincrone o asincrone. Le ***scritture sincrone*** avvengono nell'ordine in cui le riceve il sottosistema per la gesione del disco e non subiscono la memorizzazione transitoria. Nelle ***scritture asincrone*** si memorizzano i dati nella cache e si restituisce immediatamente il controllo alla procedura chiamante. Alcuni sistemi ottimizzano la cache delle pagine adottando, secondo il tipo d'accesso ai file, differenti algoritmi di sostituzione. Gli accessi sequenziali si potrebbero ottimizzare con le tecniche di rilascio indietro o lettura anticipata. Il ***rilascio indietro*** (***free-behind***) rimuove una pagina dalla memoria di transito non appena si verifica una richiesta della pagina successiva. Con la ***lettura anticipata*** (***read-ahead***) si leggono e si mettono nella cache della pagina richiesta parecchie pagine successive poichè è molto probabile che queste pagine siano richieste una volta terminata l'elaborazione della pagina corrente.

Poichè i file e le directory sono mantenuti sia in memoria centrale che nei dischi, è necessario aver cura di assicurare che il verificarsi di un malfunzionamento nel sistema non comporti la perdita dei dati o la loro incoerenza. Vediamo ora come può avvenire il ***ripristino*** di un sistema. Qualunque sia la causa degli errori, un file system deve prima scoprire i problemi e poi correggerli. Per scoprire gli errori vengono esaminati tutti i metadati su ongi file system per verificare la coerenza del sistema. Il ***verificatore della coerenza*** confronta i dati delle directory con quelli contenuti nei blocchi dei dischi, tentendo di correggere ogni incoerenza. Gli algoritmi per il ripristino sviluppati nell'area della delle basi di dati, basati sulla registrazione delle modifiche, vengono applicati con successo al problema della verifica della coerenza realizzando i ***file system orientati alle transazioni e basati sulla registrazione delle modifiche*** (*log based transiction-oriented file system*), anche noti come ***file system annotati*** (*journaling file system*). Tutte le modifiche dei metadati si annotano in modo sequenziale in un file di registrazione, detto *log*. Una volta che le modifiche sono state riportate nel log, le operazioni si considerano portate al termine con successo (*committed*) e la chiamata di sistema può restituire il controllo al processo utente, permettendogli di proseguire la sua esecuzione. Quando l'intera transazione è stata completata, se ne rimuovono le annotazioni dal log, che è in realtà un buffer circolare. I ***buffer circolari*** scrivono fino alla fine dello spazio disponibile, e poi ricominciano dall'inizio sovrascrivendo i vecchi contenuti. Se si verifica un crollo del sistema e nel log sono presenti transazioni, queste risultano mai state ultimate nel file system, anche se il sistema operativo le definisce portate a termine con successo, e quindi devono essere completate. Le transazioni si possono eseguire a partire dalla posizione corrente del puntatore fino al completamento, e le strutture del file system rimangono coerenti. Per evitare la perdita dei dati dovuta a malfunzionamenti o guasti nei dischi magnetici si possono usare programmi di sistema che consentono di fare delle ***copie di riserva*** (*backup*) dei dati residenti nei dischi o in altri dispositivi di registrazione dati. Il ripristino della situazione antecedente la perdita di un singolo file richiederà il ***recupero*** (*restore*) dei dati dalle copie di riserva.

L'***NFS*** è un buon esempio di file system di rete client-server ampiamente usato e ben realizzato. È una definizione e una realizzazione di un sistema per l'accesso a file remoti attraverso LAN o WAN. Descriviamo la versione utilizzata dal sistema operativo Solaris. Nel contesto dell'NFS si considera un insieme di stazioni di lavoro interconnesse come un insieme di calcolatori indipendenti con file system indipendenti. La condivisione è basata su una relazione client-server e un calcolatore può essere sia client che server. Affinchè una directory remota sia accessibile in modo trasparente a un calcolatore particolare, un client di quel calcolatore deve prima eseguire un'operazione di montaggio. La semantica dell'operazione consiste nel fatto che una directory remota si monta in corrispondenza di una directory di un file system locale. Una volta completata l'operazione di montaggio, la directory montata assume l'aspetto di un sottoalbero integrante del file system locale, e sostituisce il sottoalbero che discende dalla directory locale, che a sua volta rappresenta la radice della directory appena montata. La directory remota si specifica come argomento dell'operazione di montaggio in modo esplicito: si deve fornire la locazione o il nome del calcolatore della directory remota. Da questo momento gli utenti del calcolatore possono accedere ai file della directory remota in modo trasparente. Montando un file system remoto, il client non acquisisce l'accesso ai file system che erano montati sopra il primo. Uno degli scopi nella progettazione dell'NFS era quello di operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete. La definizione dell'NFS è indipendente da questi mezzi e quindi incoraggia altre realizzazioni. Questa indipendenza si ottiene usando primitive RPC costruite su un protocollo di rappresentazione esterna dei dati (*external data representation, XDR*) usato tra due interfacce indipendenti dalla realizzazione. La definizione dell'NFS distingue tra i servizi offerti da un meccanismo di montaggio e i servizi effettivi d'accesso ai file remoti. Per questi servizi si definiscono due protocolli distinti: il protocollo di montaggio e il protocollo per gli accessi ai file remoti o protocollo NFS. Il ***protocollo di montaggio*** stabilisce la connessione logica iniziale tra un server e un client. Un'operazione di montaggio comprende il nome della directory remota da montare e il nome del calcolatore server in cui tale directory è memorizzata. La richiesta di montaggio si associa alla RPC corrispondente e s'invia al server di montaggio in esecuzione nello specifico calcolatore server. Il server conserva una ***lista di esportazione*** che specifica i file system locali esportati per il montaggio e i nomi dei calcolatori a cui tale operazione è stata permessa. Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client un ***handle del file*** da usare come chiave per ulteriori accessi ai file che si trovano all'interno del file system montato. L'handle del file contiene tutte le informazioni di cui ha bisogno il server per gestire un proprio file. Il server contiene anche una lista dei calcolatori client e delle corrispondenti directory correntemente montate (questa lista viene usata soprattutto per scopi amministrativi). Il ***protocollo NFS*** offre un insieme di RPC per operazioni su file remoti che svolgono le seguenti operazioni: ricerca di un file in una diretory, lettura di un insieme di elementi di una directory, manipolazione di collegamenti e di directory, accesso ad attributi di file, lettura e scrittura di file. (Vedere sul libro)

- Esempio: il file system WAFL