**Parte II - Capitolo 10 - Interfaccia Del File System**

Per la maggior parte degli utente il ***file system*** è l'aspetto più visibile del sistema operativo. Il file system consiste in due parti distinte: un insieme di *file*, ciascuno dei quali contenente dati correlati, e una *struttura della directory*, che organizza tutti i file nel sistema e fornisce le informazioni relative. Un ***file*** è un insieme di informazioni, correlate e registrate in memoria secondaria, cui è stato assegnato un nome. Dal punto di vista dell'utente un file è la più piccola porzione di memoria secondaria logica. Un file ha attributi che possono variare a seconda del sistema operativo, ma che solitamente sono: nome, identificatore, tipo, locazione, dimensione, protezione, ora, data e identificazione dell'utente. Le informazioni sui file sono conservate nella struttura della ***directory***, che risiede a sua volta in memoria secondaria. Poichè le directory, come i file, devono essere non volatili, si devono registrare in memoria secondaria e caricare in memoria centrale un po' alla volta, secondo le necessità. Un file è un tipo di dato astratto, quindi, è necessario definire le operazioni che possono esser svolte su di esso: creazione, scrittura, lettura, riposizionamento, cancellazione, troncamento. Altre operazioni sono l'aggiunta di informazioni alla fine di un file esistente e la ridenominazione di un file esistente. La maggior parte delle operazioni richiede una ricerca associata al file specificato nella directory. Per evitare questa continua ricerca, il sistema mantiene, dopo che il file è stato aperto, una piccola tabella contenente tutti i file aperti, detta ***tabella dei file aperti***. A un file aperto sono associate altre informazioni quali un puntatore al file, un contatore di file aperti, una posizione nel disco dei file e i diritti d'accesso. Nella progettazione di un file system si deve sempre considerare la possibilità o meno che quest'ultimo riconosca e gestisca i tipi di file. Un sistema operativo che riconosce un tipo di file può trattarlo in modo ragionevole. Un tipo comune di gestione dei tipi di file consiste nell'includere il tipo nel nome del file; infatti il nome del file è suddiviso nel nome vero e proprio e nell'estensione, generalmente divisi da un punto.

I file memorizzano informazioni; al momento dell'uso è necessario accedere a queste informazioni e trasferirle in memoria. Esistono molti metodi per accedere alle informazioni dei file e, la scelta del metodo giusto, è un importante problema di progettazione. Il più semplice metodo d'accesso è l'***accesso sequenziale***: le informazioni del file si elaborano ordinatamente, un record dopo l'altro. Questo metodo è di gran lunga il più comune ed è usato, ad esempio, negli editor e nei compilatori. Un altro metodo di accesso è l'***accesso diretto*** (o ***accesso relativo***): un file è formato da elementi logici di lunghezza fissa detti ***record*** che consentono ai programmi di leggere e scrivere rapidamente tali elementi senza un ordine particolare. Il metodo d'accesso diretto si fonda sul modello di file che si rifà al disco. I file ad accesso diretto sono molto utili quando è necessario accedere immediatamente a grandi quantità di informazioni. Spesso le basi di dati sono di questo tipo: quando si presenta un'interrogazione riguardante un oggetto particolare, occorre stabilire quale blocco contiene la risposta alla richiesta e quindi leggere direttamente quel blocco. Il numero di blocco fornito dall'utente al sistema operativo è normalmente un ***numero di blocco relativo***; l'uso dei numeri di blocco relativo permette al sistema operativo di decidere dove posizionare il file e aiuta a impedire che l'utente acceda a porzioni di file system che possono non far parte del suo file. Sulla base del metodo ad accesso diretto se ne possono costruire altri, che implicano generalmente la costruzione di un indice per i file. L'***indice*** contiene puntatori ai vari blocchi: per trovare un elemento del file occorre prima cercare nell'indice, e quindi usare il puntatore per accedere direttamente al file e trovare l'elemento desiderato. Una struttura fatta in questo modo permette di compiere ricerche in file molto lunghi limitando il numero di operazioni di I/O.

I file vengono salvati in dispositivi di memorizzazione ad accesso casuale che possono essere interamente utilizzati per un file system, oppure essere suddivisi per un controllo più raffinato. Un disco può essere ***partizionato*** e ogni partizione può contenere un file system. La suddivisione in partizioni è utile per limitare la dimensione dei file system individuali, per mettere sullo stesso dispositivo più file system, oppure per liberare ad altri scopi una parte del dispositivo (come per esempio lo spazio di swap o di raw). Le partizioni sono note anche come ***suddivisioni*** (*slice*) o ***minidischi***. Un file system può essere installato su ciascuno di queste parti del disco e ogni entità contenente un file system è detta ***volume***. Ogni volume può essere pensato come un disco virtuale e può contenere diversi sistemi operativi. Ogni volume che contiene un file system deve anche contenere in sè le informazioni sui file presenti nel sistema. Tali informazioni risiedono in una ***directory del dispositivo*** o***indice del volume***. La directory si può considerare come una tabella di simboli che traduce i nomi dei file negli elementi in essa contenuti. Si capisce che una stessa directory può essere organizzata in molti modi diversi, mantenendo le operazioni che si possono fare: ricerca di un file, creazione di un file, cancellazione di un file, elencazione di una directory, ridenominazione di un file, attraversamento del file system. La struttura più semplice per una directory è quella ***a livello singolo***. Tutti i file sono contenuti nella stessa directory. Una directory a livello singolo presenta limiti notevoli che si manifestano con l'aumentare del numero di file in essa contenuti, oppure se il sistema è usato da più utenti. Poichè si trovano nella stessa directory, i file devono avere nomi unici. La confusione dei nomi causa confusione tra i file dei diversi utenti. La soluzione più semplice è quella di creare una directory separata per ogni utente in modo da creare una struttura ***a due livelli***. In questo modo ogni utente dispone della propria ***directory utente*** (*user file directory*, *UFD*). Le directory utente hanno una struttura simile, ma in ciascuna sono elencati solo i file del proprietario. Quando comincia l'elaborazione di un lavoro si fa una ricerca nella ***directory principale*** (*master file directory*, *MFD*): questa viene indirizzata con il nome dell'utente o il numero che lo rappresenta, e ogni suo elemento punta alla relativa directory utente. Quando un utente fa riferimento a un file particolare, il sistema operativo esegue la ricerca solo nella directory di quell'utente. In questo modo utenti diversi possono avere file con lo stesso nome, purchè i file all'interno della stessa directory siano unici. Questa struttura presenta ancora dei problemi: ogni utente risulta isolato dagli altri. L'isolamento può essere un vantaggio se gli utenti sono completamente indipendenti, ma può essere uno svantaggio se gli utenti vogliono cooperare e accedere ai file degli altri. Una generalizzazione della directory a due livelli permette di creare una struttura ***ad albero***. L'albero ha una radice (*root directory*) e ogni file del sistema ha un unico nome di percorso (*path name*). Una directory, o una sottodirectory, contiene un insieme di file o sottodirectory. Le directory sono semplicemente file trattati in modo speciale e contraddistinte da essi da un bit che informa se l'elemento è una directory o un file. Per cancellare o creare directory si adoperano speciali chiamate di sistema. Ogni utente dispone di una ***directory corrente*** che contiene la maggior parte dei file d'interesse corrente per il processo. Se il file non viene trovato nella directory corrente, si deve specificare un nome di percorso, oppure cambiare directory corrente. I nomi di percorso possono essere di due tipi: ***nomi di percorso assoluti***, se segue un percorso che comincia dalla radice e arriva al file desiderato, o ***nomi di percorso relativi***, se definisce un percorso che parte dalla directory corrente. Una decisione importante relativa alla strutturazione ad albero riguarda la cancellazione delle directory. Se una directory è vuota basta cancellare il riferimento nella directory che la contiene (molti sistemi operativi non permettono di cancellare directory se queste non sono vuote); in alternativa si può avere un'opzione che alla richiesta di cancellazione della directory, cancelli anche tutti i file e le sottodirectory contenuti (come il comando rm in UNIX). Un limite della struttura ad albero è quello di non ammettere la condivisione di file o directory. Un ***grafo aciclico*** permette alle directory di avere sottodirectory e file condivisi, cioè lo stesso file o sottodirectory possono essere in due directory diverse. Il fatto che un file sia condiviso non significa che esistano due copie del file: se un programmatore modifica il file, esso appare immediatamente modificato agli altri utenti che condividono il file. I file e le sottodirectory condivise si possono realizzare in molti modi. Un metodo diffuso, usato in molti sistemi UNIX, prevede la creazione di un nuovo elemento di directory chiamato collegamento. Un ***collegamento*** (*link*) è un puntatore a un altro file o un'altra directory. Un altro comune metodo per la realizzazione di file condivisi prevede la duplicazione di tutte le informazioni relative ai file in entrambe le directory di condivisione, quindi i due elementi sono identici. Un collegamento è chiaramente diverso dagli altri elementi della directory. Duplicando gli elementi della directory, la copia e l'originale sono indistinguibili: sorge il problema di mantenere la coerenza se il file viene modificato. Un problema dei grafi aciclici riguarda la cancellazione. In un sistema in cui la condivisione è realizzata da collegamenti simbolici (***linking simbolico***) la gestione di questa situazione è relativamente semplice: la cancellazione di un collegamento non influisce sul file originale, perchè si rimuove solo il collegamento. Se si cancella il file, si libera lo spazio corrispondente lasciando in sospeso il collegamento. Un altro tipo di approccio prevede la conservazione del file finchè non siano stati cancellati tutti i riferimenti ad esso. In questo caso, è possibile tenere una lista di tutti i riferimenti a un file: quando si crea un collegamento si aggiunge un nuovo elemento alla lista dei riferimenti, quando si elimina un collegamento si cancella l'elemento corrispondente dalla lista. Con questo metodo la dimensione della lista di riferimenti può diventare molto grande. Non è necessario mantenere tutta la lista, ma è sufficiente mantenere un contatore del numero di riferimenti. Quando il contatore è uguale a 0 si può cancellare il file. Il sistema operativo UNIX usa questo metodo per i collegamenti non simbolici o ***collegamenti effettivi*** (***hard linking***); il contatore dei riferimenti è tenuto nel blocco di controllo del file. Un altro problema connesso all'uso di una struttura a grafo aciclico consiste nell'assicurare che non vi siano cicli. Il vantaggio principale è dato dalla semplicità degli algoritmi necessari per attraversarlo e per determinare quando non ci sono più riferimenti a un file. Se si formano cicli nella struttura aggiungendo nuovi file o sottodirectory alla struttura, un algoritmo potrebbe causare un ciclo infinito di ricerca. Una soluzione al problema è quella di limitare il numero di directory cui accedere durante la ricerca. Un problema analogo si presenta al momento di stabilire quando sia possibile cancellare un file. È possibile che in un grafo dove sono presenti cicli, un contatore sia non nullo anche se non vi sono più riferimenti al file o alla sottodirectory. Questa anomalia è dovuta alla possibilità di autoriferimento. È possibile risolvere il problema usando il metodo di ripulitura (*garbage collection*) per stabilire quando sia stato cancellato l'ultimo riferimento e quando sia possibile riallocare lo spazio dei dischi. Per evitare i cicli nelle strutture ad albero viste si possono usare algoritmi che però risultano molto onerosi. Una soluzione particolare del caso di directory e collegamenti, un semplice algoritmo prevede di evitare i collegamenti durante l'attraversamento delle directory: in questo modo si evitano cicli senza un ulteriore carico.

Per essere reso accessibile ai processi di un sistema, un file system deve esssere montato. La procedura di ***montaggio*** è molto semplice: si fornisce al sistema operativo il nome del dispositivo e la sua locazione (detta ***punto di montaggio***) nella struttura di file e directory alla quale agganciare il file system. Di solito un punto di montaggio è una directory vuota cui sarà agganciato il file system una volta montato. Il passo successivo consiste nella verifica da parte del sistema operativo della validità del file system contenuto nel dispositivo. La verifica si compie richiedendo al dirver del dispositivo di leggere la directory di dispositivo controllando che tale directory abbia il formato previsto. Infine, il sistema operativo annota nella sua struttura della directory che un certo file system è montato al punto di montaggio specificato. Un sistema potrebbe vietare il montaggio in una directory contenente file, o rendere disponibile il file system montato in tale directory e nascondere i file preesistenti nella directory finchè non si smonti il file system, concludendone l'uso e permettendo l'accesso ai file originariamente presenti in tali directory. L'OS X di Mac si comporta in modo simile a UNIX BSD: tutti i file system vengono montati sotto la directory /Volumes, ma mostrandoli come se fossero montati a livello radice. La famiglia di sistemi operativi Microsoft Windows mantiene una struttura della directory a due livelli estesa, con una lettera di unità associata a dispositivi e volumi. I sistemi Windows rilevano automaticamente tutti i dispositivi e montano tutti i file system rilevati all'avvio dalla macchina; in altri sistemi, per esempio UNIX, occorre montare esplicitamente i file system.

I sistemi operativi orientati agli utenti devono soddisfare l'esigenza di ***condivisione***. Se un sistema operativo permette l'uso del sistema da parte di più utenti, diventano particolarmente rilevanti i problemi relativi alla condivisione dei file. Il sistema può permettere a ogni utente di accedere ai file degli altri utenti, oppure può richiedere che un utente debba esplicitamente concedere i permessi di accesso ai file. Per realizzare i meccanismi di condivisione e protezione, il sistema deve memorizzare e gestire più attributi di directory e file rispetto a un sistema che consente un singolo utente. La maggior parte dei sistemi ha adottato i concetti di *proprietario*, cioè l'utente che ha il maggiore controllo sul file e directory, e *gruppo*, cioè gli utenti autorizzati a condividere l'accesso al file. Gli identificatori del gruppo e del proprietario di un certo file o directory sono memorizzati con gli altri attributi dei file. L'avvento delle reti ha permesso la comunicazione tra calcolatori separati da grandi distanze. I metodi con cui i file si condividono in una rete sono vari. Un primo metodo consiste nel trasferimento del file richiesto in modo esplicito dagli utenti, attraverso programmi appositi come l'fpt. Un secondo metodo, molto diffuso, è quello del ***file system distribuito*** (*distributed file system*, *DFS*) che permette la visibilità nel calcolatore locale delle directory remote. Il terzo metodo è il ***World Wide Web***: per accedere ai file remoti si usa un programma di consultazione (browser), e operazioni distinte per trasferirli. L'***accesso anonimo*** permette di trasferire i file senza essere utenti accreditati nel sistema remoto. I file system remoti permettono il montaggio di uno o più file system di uno o più calcolatori remoti in un calcolatore locale. In questo caso, il calcolatore contenente il file si chiama *server*, mentre quello che richiede l'accesso al file si chiama *client*. Un server può gestire richieste di più client, un client può accedere a più server. Il server in genere specifica i file disponibili su di un volume o livello di directory. L'identificazione certa dei client è più difficile proprio perchè può avvenire facilmente tramite i relativi nomi simbolici di rete o tramite identificatori, si uò altrettanto facilmente ingannare un server imitando l'identificatore di un client accreditato (*spoofing*). Tra le soluzioni più sicure ci sono quelle che prevedono l'autenticazione reciproca dei client e dei server, che, però, provoca la necessità di compatibilità tra client e server (si devono impiegare gli stessi algoritmi di cifratura). Nel caso di UNIX e del suo file system di rete (*network file system*, *NFS*) l'autenticazione avviene tramite le informazioni di connessione relative al client. In questo schema gli identificatori che identificano l'utente devono coincidere nel client e nel server. I protocolli NFS permettono relazioni da molti a molti, cioè più server possono fornire dati a più client. Una volta montato il file system remoto, le richieste delle operazioni su file sono inviate al server, attraverso la rete, per conto dell'utente, secondo il protocollo DFS. Il ***sistema dei nomi di dominio*** (*domain name system, DNS*) fornisce le traduzioni dai nomi dei calcolatori agli indirizzi di rete per l'intera Internet. Nel caso delle reti di Microsoft (*common internet file system, CIFS*), le informazioni di rete si usano insieme con gli elementi di autenticazione dell'utente per creare un ***nome utente di rete*** (*network login*) che il server usa per decidere se permettere o negare l'accesso a un file system richiesto. Affinchè quest'autenticazione sia valida, i nomi utente devono essere uguali nel calcolatore client e server. Microsoft usa due strutture di nominazione distribuita: i ***domini*** che costituiscono la vecchia tecnologia di nominazione e l'***active directory***: una volta impostata la funzione di nominazione, questa viene usata per autenticare gli utenti da tutti i client a tutti i server. I file system locali possono presentare malfunzionamenti per varie cause: problemi dei dischi che li contengono, alterazione dei dati relativi alle strutture delle directory o a informazioni necessarie alla gestione dei dischi (***metadati***), malfunzionamenti dei controllori dei dischi, problemi ai cavi di connessione o agli adattatori. Anche il comportamento involontario dell'utente o dell'amministratore può causare un malfunzionamento. Di solito questi malfunzionamenti portano al crollo del sistema (*crash*), all'emissione di una condizione di errore e alla necessità di un intervento umano per risolvere il problema. I malfunzionamenti su file system remoti sono più possibili a causa della complessità del sistema di rete. Per realizzare il recupero dei dati dovuti a un malfunzionamento di file system remoto è necessario mantenere alcune ***informazioni di stato*** sia sui client che sui server. Se entrambi tengono traccia delle loro attività correnti e dei loro file aperti, entrambi possono riprendersi senza traumi da un malfunzionamento. Nel caso in cui il server crolli ma debba rilevare la presenza di file system remoti e file aperti, l'NFS segue un criterio semplice realizzando un DFS ***senza stato***, cioè assume che una richiesta di un client per la lettura o la scrittura di un file non sia avvenuta, facendo in modo che il protocollo NFS trasferisca tutte le informazioni necessarie per la localizzazione del file appropriato per svolgere l'operazione richiesta sul file. La ***semantica della coerenza*** è un importante criterio per la valutazione di qualsiasi file system che consenta la condivisione dei file. Essa deve specificare quando le modifiche apportate da un utente possano essere osservate da altri utente. Il file system in UNIX usa la seguente semantica: le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno aperto lo stesso file; gli utenti possono condividere il puntatore alla locazione corrente nel file, quindi l'avanzamento del puntatore da parte di un utente influisce su tutti gli altri. Il file system Andrew (*Andrew file system, AFS*) usa la seguente semantica: le scritture in un file aperto da parte di un utente non sono immediatamente visibili ad altri utenti che hanno aperto lo stesso file; una volta chiuso il file, le modifiche apportate sono visibili nelle sezioni successive. Un altro metodo è quello dei ***file condivisibili immutabili***: una volta dichiarato condiviso dal suo creatore, un file non può essere modificato.

La salvaguardia delle informazioni contenute in un sistema di calcolo dai danni fisici (*affidabilità*) e da accessi impropri (*protezione*) è fondamentale. In genere l'affidabilità è assicurata da più copie dei file. La protezione può essere ottenuta in vari modi. Un primo tipo di controllo offerto dai meccanismi di protezione si ottiene limitando i possibili tipi di accesso. Si possono controllare distinte operazioni: lettura, scrittura, esecuzione, aggiunta, cancellazione, elencazione. Il problema della protezione comunemente si affronta rendendo l'accesso dipendente dall'identità dell'utente. Lo schema più generale per realizzare gli accessi dipendenti dall'identità consiste nell'associare una ***lista di controllo degli accessi*** (*access-control list, ACL*) a ogni file o directory. Quando un utente richiede l'accesso a un file specifico il sistema operativo esamina la lista di controllo degli accessi associata a questo file e si autorizza o si nega l'accesso a seconda della presenza dell'utente sulla lista. Il problema principale della lista di controllo degli accessi è la lunghezza. Questo problema può essere risolto raggruppando gli utenti in tre classi distinte: proprietario, cioè colui che ha creato il file, gruppo, cioè gli utenti che richiedono tipi di accessi simili, e universo, cioè il resto degli utenti del sistema. Nel sistema UNIX sono definiti tre campi di tre bit ciascuno: rwx, dove r controlla l'accesso in lettura, w controlla l'accesso in scrittura, x controlla l'accesso per l'esecuzione. Un altro metodo per la protezione consiste nell'associazione di una parola d'ordine (*password*) a ciascun file per accedervi.