Parte quarta

Gestione della memoria secondaria

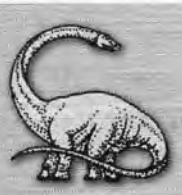
Poiché la memoria centrale è in genere troppo piccola per contenere in modo permanente tutti i dati e tutti i programmi, il calcolatore deve disporre di una memoria secondaria a sostegno della memoria centrale. I calcolatori moderni impiegano i dischi come mezzo principale di registrazione delle informazioni, cioè programmi e dati. Il file system fornisce i meccanismi sia per l'accesso ai dati e ai programmi residenti nei dischi sia per la loro registrazione. Un file è una raccolta d'informazioni tra loro correlate definite dal suo creatore. Il sistema operativo gestisce la corrispondenza tra i file e i dispositivi che li contengono fisicamente; i file sono normalmente organizzati in directory che ne facilitano l'uso.

I dispositivi da collegare a un calcolatore possono variare sotto molti aspetti. Alcuni di loro trasferiscono un carattere o un blocco di caratteri per volta. Alcuni prevedono esclusivamente l'accesso sequenziale, altri solo l'accesso casuale. Talvolta il trasferimento dei dati è sincrono, talaltra è asincrono. Esistono dispositivi dedicati, mentre altri possono essere condivisi. E ancora, a differenza di quelli a sola lettura, alcuni dispositivi ammettono sia la lettura che la scrittura. Pur essendo caratterizzati da notevoli differenze di velocità, i dispositivi rappresentano, nel loro insieme, la componente più lenta e voluminosa dell'elaboratore.

Il sistema operativo, per trattare efficacemente tutte queste varianti, deve offrire alle applicazioni funzionalità che consentano loro un minuzioso controllo dei dispositivi. Uno degli obiettivi cruciali del sottosistema di I/O è fornire la più semplice interfaccia possibile al resto del sistema. Poiché i dispositivi rappresentano un collo di bottiglia per le prestazioni, al fine di sfruttare l'accesso concorrente nel migliore dei modi, l'ottimizzazione dell'I/O è un altro elemento chiave.

Capitolo 10

Interfaccia del file system



OBIETTIVI

- **Qual è la funzione dei file system.**
- **Descrizione delle interfacce dei file system.**
- **Presentazione dei compromessi di progettazione dei file system, compresi metodi d'accesso, condivisione dei file, uso dei lock e strutture della directory.**
- **Analisi della protezione dei file system.**

Per la maggior parte degli utenti il file system è l'aspetto più visibile di un sistema operativo. Esso fornisce il meccanismo per la registrazione e l'accesso in linea a dati e programmi appartenenti al sistema operativo e a tutti gli utenti del sistema di calcolo. Il file system consiste di 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. I file system – brevemente accennati nel presente capitolo, ma oggetto di approfondimento nei capitoli successivi – basano la loro esistenza su dispositivi. Si analizza inoltre la semantica dei file condivisi da più processi, utenti e calcolatori. In questo capitolo infine si considerano i vari aspetti dei file e i principali tipi di strutture della directory. Si esamina inoltre la gestione della *protezione dei file*, necessaria in un ambiente in cui più utenti hanno accesso ai file, e dove si vuole controllare chi e in che modo vi ha accesso.

10.1 Concetto di file

I calcolatori possono memorizzare le informazioni su diversi supporti, come dischi, nastri magnetici e dischi ottici. Per rendere agevole l'uso del calcolatore, il sistema operativo offre una visione logica uniforme della memorizzazione delle informazioni; astrae il *file* dalle caratteristiche fisiche dei propri dispositivi di memoria per definire un'unità di memoria logica. Il sistema operativo associa i file ai dispositivi fisici, di solito non volatili, in modo che il loro contenuto non vada perduto a causa delle interruzioni dell'alimentazione elettrica e dei riavvii del sistema.

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; i dati si possono cioè scrivere in memoria secondaria soltanto all'interno di un file. Di solito i file rappresentano programmi, in forma sorgente e oggetto, e dati. I file di dati possono essere numerici, alfabetici, alfanumerici o binari, e non possedere

un formato specifico, come i file di testo; oppure essere rigidamente formattati. In genere un file è formato da una sequenza di bit, byte, righe o *record* il cui significato è definito dal creatore e dall'utente del file stesso. Il concetto di file è quindi estremamente generale.

Le informazioni contenute in un file sono definite dal suo creatore e possono essere di molti tipi: programmi sorgente, programmi oggetto, dati numerici, testo, dati contabili, immagini, registrazioni sonore, e così via. Un file ha una **struttura** definita secondo il tipo: un file di *testo* è formato da una sequenza di caratteri organizzati in righe, e probabilmente pagine; un file *sorgente* è formato da una sequenza di procedure e funzioni, ciascuna delle quali è a sua volta organizzata in dichiarazioni seguite da istruzioni eseguibili; un file *oggetto* è formato da una sequenza di byte, organizzati in blocchi, comprensibile al modulo di collegamento del sistema; un file *eseguibile* consiste di una serie di sezioni di codice che il caricatore può caricare in memoria ed eseguire.

10.1.1 Attributi dei file

Per comodità degli utenti, ogni file ha un nome che si usa come riferimento. Un nome, di solito, è una sequenza di caratteri come `esempio.c`. Alcuni sistemi, nella composizione dei nomi, distinguono le lettere maiuscole dalle minuscole, altri le considerano equivalenti. Una volta ricevuto il nome, il file diviene indipendente dal processo, dall'utente, e anche dal sistema da cui è stato creato. Ad esempio, un utente potrebbe creare il file `esempio.c` e un altro utente potrebbe modificarlo specificandone il nome. Il proprietario del file potrebbe registrare il file in un dischetto, inviarlo per posta elettronica, o copiarlo attraverso la rete, ed esso potrebbe ancora chiamarsi `esempio.c` nel sistema di destinazione.

Un file ha altri attributi che possono variare secondo il sistema operativo, ma che tipicamente comprendono i seguenti.

- **Nome.** Il nome simbolico del file è l'unica informazione in forma umanamente leggibile.
- **Identificatore.** Si tratta di un'etichetta unica, di solito un numero, che identifica il file all'interno del file system; è il nome impiegato dal sistema per il file.
- **Tipo.** Questa informazione è necessaria ai sistemi che gestiscono tipi di file diversi.
- **Locazione.** Si tratta di un puntatore al dispositivo e alla locazione del file in tale dispositivo.
- **Dimensione.** Si tratta della dimensione corrente del file (in byte, parole o blocchi) ed eventualmente della massima dimensione consentita.
- **Protezione.** Le informazioni di controllo degli accessi controllano chi può leggere, scrivere o far eseguire il file.
- **Ora, data e identificazione dell'utente.** Queste informazioni possono essere relative alla creazione, l'ultima modifica e l'ultimo uso. Questi dati possono essere utili ai fini della protezione e per controllarne l'uso.

Le informazioni sui file sono conservate nella struttura della directory, che risiede a sua volta in memoria secondaria. Di solito un elemento di directory consiste di un nome di file e di un identificatore unico, che a sua volta individua gli altri attributi del file. Un elemento di directory può richiedere più di un kilobyte per contenere queste informazioni per ciascun file. In un sistema con molti file, la dimensione della stessa directory può essere dell'ordine di megabyte. Poiché le directory, come i file, devono essere non volatili, si devono registrare in memoria secondaria e caricare in memoria centrale un po' per volta, secondo le necessità.

10.1.2 Operazioni sui file

Un file è un **tipo di dato astratto.** Per definire adeguatamente un file è necessario considerare le operazioni che si possono eseguire su di esso. Il sistema operativo può offrire chiamate di sistema per creare, scrivere, leggere, spostare, cancellare e troncare un file. Le successive considerazioni su ciò che deve fare un sistema operativo per ciascuna di queste sei operazioni di base dovrebbero rendere più semplice osservare come si possano realizzare altre operazioni simili, ad esempio la ridenominazione di un file.

- **Creazione di un file.** Per creare un file è necessario compiere due passaggi. In primo luogo si deve trovare lo spazio per il file nel file system; la discussione sui criteri di allocazione dei file è rimandata al Capitolo 11. Secondariamente, per il file si deve creare un nuovo elemento nella directory in cui registrare il nome del file, la sua posizione nel file system ed eventualmente altre informazioni.
- **Scrittura di un file.** Per scrivere in un file è indispensabile una chiamata di sistema che specifichi il nome del file e le informazioni che si vogliono scrivere. Dato il nome del file, il sistema cerca la sua posizione nella directory. Il file system deve mantenere un puntatore di *scrittura* alla locazione nel file in cui deve avvenire l'operazione di scrittura successiva. Il puntatore si deve aggiornare ogniqualvolta si esegue una scrittura.
- **Lettura di un file.** Per leggere da un file è necessaria una chiamata di sistema che specifichi il nome del file e la posizione in memoria dove collocare il successivo blocco del file. Anche in questo caso si cerca l'elemento corrispondente nella directory e il sistema deve mantenere un puntatore di *lettura* alla locazione nel file in cui deve avvenire la successiva operazione di lettura. Una volta completata la lettura, si aggiorna il puntatore. Di solito un processo legge o scrive in un file, e la posizione corrente è mantenuta come un **puntatore alla posizione corrente del file.** Sia le operazioni di lettura sia quelle di scrittura adoperano lo stesso puntatore, risparmiando spazio e riducendo la complessità del sistema.
- **Riposizionamento in un file.** Si ricerca l'elemento appropriato nella directory e si assegna un nuovo valore al puntatore alla posizione corrente nel file. Il riposizionamento non richiede alcuna operazione di I/O. Questa operazione è anche nota come *posizionamento* o *ricerca* (*seek*) nel file.
- **Cancellazione di un file.** Per cancellare un file si cerca l'elemento della directory associato al file designato, si rilascia lo spazio associato al file (in modo che possa essere adoperato per altri) e si elimina l'elemento della directory.
- **Troncamento di un file.** Si potrebbe voler cancellare il contenuto di un file, ma mantenere i suoi attributi. Invece di forzare gli utenti a cancellare il file e quindi a ricrearlo, questa funzione consente di mantenere immutati gli attributi (a esclusione della lunghezza del file) pur azzerando la lunghezza del file e rilasciando lo spazio occupato.

Queste sei operazioni di base comprendono sicuramente l'insieme minimo delle operazioni richieste per i file. Altre operazioni comuni comprendono l'*aggiunta* (*appending*) di nuove informazioni alla fine di un file esistente e la *ridenominazione* di un file esistente. Queste operazioni primitive si possono combinare per compiere altre operazioni. Ad esempio, per creare una *copia* di un file o per copiare il file in un altro dispositivo di I/O, come una stampante o un video, è sufficiente creare un nuovo file, leggere i dati dal file vecchio e scriverli nel nuovo. Sono inoltre necessarie operazioni che consentano a un utente di leggere e impostare i vari attributi di un file.

APPLICAZIONE DI LOCK NEL LINGUAGGIO JAVA

L'acquisizione del lock di un file tramite la API Java prevede in primo luogo l'acquisizione del `FileChannel` relativo al file; il metodo `lock()` del `FileChannel` permette poi di ottenere il lock. Il prototipo del metodo `lock()` è:

```
FileLock lock(long begin, long end, boolean shared)
```

dove `begin` ed `end` sono il punto iniziale e finale della parte da sottoporre a lock. Se `shared` vale `true`, si ottiene un lock condiviso; altrimenti, un lock esclusivo. Il lock si rilascia tramite il metodo `release()` invocato sull'oggetto restituito da `lock()`.

Il programma della Figura 10.1 illustra questa tecnica. Esso acquisisce due lock del file *file.txt*. La prima metà del file è soggetta a lock esclusivo, la seconda a lock condiviso.

```
import java.io.*;
import java.nio.channels.*;

public class LockingExample {
  public static final boolean EXCLUSIVE = false;
  public static final boolean SHARED = true;

  public static void main(String args[]) throws IOException {
    FileLock sharedLock = null;
    FileLock exclusiveLock = null;

    try {
      RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");

      // acquisisce il canale per il file
      FileChannel ch = raf.getChannel();

      // acquisisce lock esclusivo per la prima metà del file
      exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);

      /* Modifica i dati . . . */

      // rilascia il lock
      exclusiveLock.release();

      // acquisisce lock condiviso per la seconda metà del file
      sharedLock = ch.lock(raf.length()/2+1, raf.length(), SHARED);

      /* Legge i dati . . . */

      // rilascia il lock
      sharedLock.release();
    } catch (java.io.IOException ioe) {
     System.err.println(ioe);
    }
    finally {
      if (exclusiveLock != null)
            exclusiveLock.release();
      if (sharedLock != null)
            sharedLock.release();
    }
  }
}
```

Figura 10.1 Esempio di applicazione di lock a un file in Java.

Ad esempio, si potrebbe dover ricorrere a un'operazione che consenta all'utente di determinare lo stato di un file, come la lunghezza, e che consenta di definire gli attributi di un file, come il proprietario.

La maggior parte delle operazioni sopra citate richiede una ricerca dell'elemento associato al file specificato nella directory. Per evitare questa continua ricerca, molti sistemi richiedono l'impiego di una chiamata di sistema `open()` la prima volta che si adopera un file in maniera attiva. Il sistema operativo mantiene una piccola tabella contenente informazioni riguardanti tutti i file aperti (detta, per l'appunto, **tabella dei file aperti**). Quando si richiede un'operazione su un file, questo viene individuato tramite un indice in tale tabella, in questo modo si evita qualsiasi ricerca. Quando il file non è più attivamente usato viene *chiuso* dal processo, e il sistema operativo rimuove l'elemento a esso associato dalla tabella dei file aperti.

Alcuni sistemi aprono implicitamente un file al primo riferimento e lo chiudono automaticamente quando il processo che lo ha aperto termina. A ogni modo, la maggior parte dei sistemi esige che il programmatore richieda l'apertura del file in modo esplicito per mezzo di una chiamata di sistema `open()` prima che sia possibile adoperarlo. L'operazione `open()` riceve il nome del file, lo cerca nella directory e copia l'elemento a esso associato nella tabella dei file aperti. La chiamata di sistema `open()` può accettare anche informazioni sui modi d'accesso: creazione, sola lettura, lettura e scrittura, sola aggiunta, ecc. Si controllano i permessi relativi al file, e se la modalità d'accesso richiesta è consentita, si apre il file. La chiamata di sistema `open()` riporta di solito un puntatore all'elemento nella tabella dei file aperti; questo puntatore si adopera al posto dell'effettivo nome del file in tutte le operazioni di I/O, evitando così successive operazioni di ricerca e semplificando l'interfaccia delle chiamate di sistema.

La realizzazione delle operazioni `open()` e `close()` è più complicata in un ambiente multiutente dove più utenti possono aprire un file contemporaneamente. Di solito il sistema operativo introduce due livelli di tabelle interne: una tabella per ciascun processo e una tabella di sistema. La tabella di un processo contiene i riferimenti a tutti i file aperti da quel processo; il puntatore alla posizione di ciascun file si trova in questa tabella e specifica la posizione nel file. Si possono includere anche i diritti d'accesso al file e informazioni di contabilizzazione.

Ciascun elemento della tabella associata a ciascun processo punta a sua volta a una tabella di sistema dei file aperti, contenente le informazioni indipendenti dai processi come la posizione dei file nei dischi, le date degli accessi e le dimensioni dei file. Quando un file è già stato aperto da un processo, l'apertura di un file da parte di un processo comporta l'aggiunta di un elemento relativo al file nella tabella dei file aperti del sistema; una `open()` eseguita da un altro processo comporta solamente l'aggiunta di un nuovo elemento nella tabella dei file aperti associata a quel processo, che punta al corrispondente elemento della tabella di sistema. In genere, la tabella dei file aperti ha anche un *contatore delle aperture* associato a ciascun file, indicante il numero di processi che hanno aperto quel file. Ogni `close()` decrementa questo *contatore*; quando raggiunge il valore zero il file non è più in uso e si elimina l'elemento corrispondente dalla tabella dei file aperti. Riassumendo, a ciascun file aperto sono associate le diverse seguenti informazioni.

- **Puntatore al file.** Nei sistemi che non prevedono lo scostamento come parte delle chiamate di sistema `read()` e `write()`, il sistema deve tener traccia dell'ultima posizione di lettura e scrittura sotto forma di un puntatore alla posizione corrente nel file. Questo puntatore è unico per ogni processo che opera sul file e quindi deve essere tenuto separato dagli attributi del file residenti nel disco.

- **Contatore dei file aperti.** Man mano che si chiudono i file, per evitare di esaurire lo spazio associato alla propria tabella dei file aperti, il sistema operativo deve riutilizzarne gli elementi. Poiché più processi possono aprire uno stesso file, prima di rimuovere l'elemento corrispondente, il sistema deve attendere l'ultima chiusura del file. Questo contatore tiene traccia del numero di `open()` e `close()`, e raggiunge il valore zero dopo l'ultima chiusura, momento in cui il sistema può rimuovere l'elemento della tabella.
- **Posizione nel disco del file.** La maggior parte delle operazioni richiede al sistema di modificare i dati contenuti nel file. L'informazione necessaria per localizzare il file nel disco è mantenuta in memoria, per evitare di doverla prelevare dal disco a ogni operazione.
- **Diritti d'accesso.** Ciascun processo apre un file in una delle modalità d'accesso. Questa informazione è contenuta nella tabella del processo in modo che il sistema operativo possa permettere o negare le successive richieste di I/O.

Alcuni sistemi operativi offrono la possibilità di applicare lock a un file aperto (o a parti di esso). Quando un processo intende proteggere un file dall'accesso concorrente di altri processi, si serve dei lock. L'utilità dei lock dei file emerge nel caso di file condivisi da diversi processi: un file di log, per esempio, può subire modifiche da parte di molti processi.

I lock dei file sono basati su una funzionalità simile ai lock di lettura-scrittura (Paragrafo 6.6.2). Un **lock condiviso** è assimilabile, per funzionamento, ai lock di lettura: entrambi consentono a più processi concorrenti di appropriarsene. Un **lock esclusivo** mostra invece analogie con i lock di scrittura, perché un solo processo per volta può acquisire questo tipo di lock. Si noti bene che non in tutti i sistemi operativi è possibile scegliere tra i due tipi di lock; alcuni sistemi forniscono solamente lock esclusivi dei file.

Inoltre, il sistema operativo può fornire meccanismi di protezione dei file **obbligatori**, oppure **consigliati**. Se un lock è obbligatorio, il sistema operativo impedirà a qualunque altro processo di accedere al file interessato una volta che il suo lock sia stato acquisito. Poniamo, per esempio, che un processo si appropri del lock esclusivo del file `system.log`. Se un altro processo – per esempio, un editor – tentasse di aprire `system.log`, il sistema operativo negherebbe l'accesso finché il lock esclusivo ritorni disponibile. Ciò accade anche se l'editor non è esplicitamente programmato per acquisire il lock. Qualora invece il lock sia solo consigliato, il sistema operativo non impedirà l'accesso dell'editor a `system.log`. Tuttavia, per poter accedere al file, l'editor deve essere scritto in modo tale da acquisire esplicitamente il lock. In altri termini, se il lock è obbligatorio, il sistema operativo assicura l'integrità dei dati soggetti a lock; se il lock è solo consigliato, è compito dei programmatori garantire la corretta acquisizione e cessione dei lock. In linea generale, i sistemi operativi Windows adottano i lock obbligatori, mentre i sistemi UNIX impiegano i lock consigliati.

L'uso dei lock dei file richiede l'osservazione delle stesse accortezze per la sincronizzazione dei processi. Per esempio, i programmatori impegnati a sviluppare su sistemi con lock obbligatori devono prestare attenzione a detenere i lock esclusivi solo per l'effettiva durata degli accessi ai file; in caso contrario, bloccheranno anche gli accessi da parte di altri processi. Occorre, inoltre, attuare misure appropriate al fine di evitare che due o più processi entrino in stallo nel tentativo di acquisire i lock per i file.

10.1.3 Tipi di file

Nella progettazione di un file system, ma anche dell'intero sistema operativo, si deve sempre considerare la possibilità o meno che quest'ultimo riconosca e gestisca i tipi di file. Un sistema operativo che riconosce il tipo di un file ha la possibilità di trattare il file in modo ragio-

Tipo di file	Estensione usuale	Funzione
Eseguibile	exe, com, bin, o nessuna	Programma, in linguaggio di macchina, eseguibile
Oggetto	obj, o	Compilato, in linguaggio di macchina, non collegato
Codice sorgente	c, cc, java, pas, asm, a	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi all'interprete dei comandi
Testo	txt, doc	Testi, documenti
Elaboratore di testi	wp, tex, rtf, doc	Vari formati per elaboratori di testi
Libreria	lib, a, so, dll	Librerie di procedure per programmatori
Stampa o visualizzazione	ps, pdf, jpeg	File ASCII o binari in formato per stampa o visione
Archivio	arc, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, rm, mp3, avi	File binari contenenti informazioni audio o A/V

Figura 10.2 Comuni tipi di file.

nevole. Ad esempio, un errore abbastanza comune consiste nel tentativo, da parte degli utenti, di stampare un programma oggetto in forma binaria; di solito questo tentativo porta semplicemente a uno spreco di carta, ma si potrebbe impedire se il sistema operativo fosse informato del fatto che il file è un programma oggetto in forma binaria.

Una tecnica comune per realizzare la gestione dei tipi di file consiste nell'includere il tipo nel nome del file. Il nome è suddiviso in due parti, un nome e un'*estensione*, di solito separate da un punto (Figura 10.2); in questo modo l'utente e il sistema operativo possono risalire al tipo del file semplicemente esaminandone il nome. La maggior parte dei sistemi operativi, per esempio, permette agli utenti di specificare i nomi dei file come sequenze di caratteri seguite da un punto e concluse da un'estensione di caratteri aggiuntivi. Esempi di nomi di file sono *resume.doc*, *Server.java* e *ReaderThread.c*. Il sistema usa l'estensione per stabilire il *tipo* del file e le operazioni che si possono eseguire su tale file. Ad esempio, solamente i file con estensione `.com`, `.exe` o `.bat` sono eseguibili; i file con estensione `.com` e `.exe` sono due formati di file eseguibili, mentre i file con estensione `.bat` sono file (**batch**) contenenti una sequenza di comandi, scritti in formato ASCII, diretti al sistema operativo. L'MS-DOS riconosce un numero limitato di estensioni, che però anche i programmi applicativi possono usare per individuare i tipi di file cui sono interessati: gli assemblatori si aspettano che i file sorgente siano caratterizzati dall'estensione `.asm`; l'elaboratore di testi Microsoft Word presuppone che i propri file terminino con l'estensione `.doc`. Queste estensioni non sono necessarie, ma la loro presenza consente a un utente di ridurre il numero delle battute specificando il nome del file senza estensione e lasciando all'applicazione il compito di cercare il file con il nome impostato e l'estensione attesa. Poiché queste estensioni non sono gestite dal sistema operativo, si possono considerare un suggerimento rivolto alle applicazioni che operano su di loro.

Consideriamo, inoltre, il sistema operativo Mac OS X. In questo sistema ciascun file ha un tipo, come *TEXT* (text file) oppure *APPL* (applicazione). Ciascun file possiede anche un at-

tributo di creazione contenente il nome del programma che lo ha creato. Questo attributo è impostato dal sistema operativo durante la chiamata di sistema `create()`, quindi la sua presenza è forzata e gestita dal sistema operativo. Per esempio, un file prodotto da un elaboratore di testi avrà il nome dell'elaboratore di testi come attributo di creazione. Quando un utente apre il file, con un doppio clic del mouse sull'icona che lo rappresenta, si attiva automaticamente l'elaboratore di testi che apre il file, pronto per essere letto e modificato.

Il sistema operativo UNIX non fornisce una funzione di questo tipo, ma si limita a memorizzare un codice (noto come **magic number**) all'inizio di alcuni tipi di file allo scopo di indicarne in modo generico il tipo: eseguibili, sequenze di comandi (batch file, noti come *shell script*), PostScript e così via. Non tutti i file possiedono tale codice, quindi il sistema non può affidarsi unicamente a questo tipo d'informazione; inoltre, non memorizza il nome del programma che ha creato il file. UNIX consente di sfruttare le estensioni come suggerimento del tipo di file; queste non vengono però imposte né dipendono dal sistema operativo; il loro compito consiste principalmente nell'aiutare gli utenti a riconoscere il tipo di contenuto del file. Un'applicazione può usare o ignorare le estensioni; dipende dalle scelte dei programmatori.

10.1.4 Struttura dei file

I tipi di file si possono anche adoperare per indicare la struttura interna dei file. Come si è accennato nel Paragrafo 10.1.3, i file sorgente e i file oggetto hanno una struttura corrispondente a ciò che il programma che dovrà leggerli si attende. Inoltre alcuni file devono rispettare una determinata struttura comprensibile al sistema operativo. Ad esempio, il sistema operativo può richiedere che un file eseguibile abbia una struttura specifica che consenta di determinare dove caricare il file in memoria e quale sia la locazione della prima istruzione. Alcuni sistemi operativi estendono questa idea a un insieme di strutture di file gestite dal sistema, con un insieme di operazioni specifiche per la manipolazione dei file con queste strutture. Ad esempio, il sistema operativo DEC VMS è dotato di un file system che gestisce tre strutture di file.

L'analisi precedente porta a considerare uno degli svantaggi dei sistemi operativi che gestiscono più strutture di file: la dimensione risultante del sistema operativo è ingombrante. Se definisce cinque strutture di file differenti, il sistema operativo deve contenere il codice per gestirle tutte; inoltre qualsiasi file potrebbe dover essere definito come uno dei tipi gestiti dal sistema operativo, con la conseguenza di introdurre notevoli problemi per le applicazioni che richiedono una strutturazione dei propri dati in modi non previsti dal sistema operativo.

Ad esempio, si supponga che un sistema operativo preveda due tipi di file: file di testo (composti da caratteri ASCII separati da caratteri di ritorno del carrello e avanzamento di riga) e file binari eseguibili. Un utente che volesse definire un file cifrato per proteggere i propri dati da letture non autorizzate potrebbe scoprire che nessuna delle due strutture si adatta al problema: non è un file di righe di testo ASCII, ma un insieme di bit (apparentemente casuali), e sebbene possa sembrare un file binario, non è eseguibile. Queste limitazioni impongono all'utente di raggirare o usare in modo scorretto il meccanismo dei tipi dei file definito dal sistema operativo, oppure di abbandonare lo schema di codifica.

Alcuni sistemi operativi impongono (e gestiscono) un numero minimo di strutture di file. Questo orientamento è stato seguito da UNIX, dall'MS-DOS e altri. UNIX considera ciascun file come una sequenza di byte, senza alcuna interpretazione. Questo schema garantisce la massima flessibilità, ma il minimo sostegno. Qualsiasi programma applicativo deve contenere il proprio codice per interpretare in modo appropriato la struttura di un file. A ogni modo, per poter caricare ed eseguire i programmi, tutti i sistemi operativi devono prevedere almeno un tipo di struttura, quella dei file eseguibili.

Anche il sistema operativo Macintosh gestisce un numero ridotto di strutture di file. Prevede che i file eseguibili consistano di due parti, **resource fork** e **data fork**. La prima contiene le informazioni che interessano l'utente, ad esempio le etichette dei pulsanti dell'interfaccia del programma. (Per tradurre in un'altra lingua le etichette dei pulsanti, si possono adoperare gli strumenti messi a disposizione dal sistema operativo per la modifica delle informazioni contenute nella prima parte del file.) La seconda contiene il codice del programma o dati: l'usuale contenuto dei file. Per ottenere i medesimi risultati con UNIX o MS-DOS, il programmatore dovrebbe modificare e ricompilare il codice sorgente, a meno che non abbia creato il proprio tipo di file di dati modificabile dall'utente. Evidentemente è utile che un sistema operativo gestisca le strutture che si adoperano spesso, ciò risparmia molto lavoro ai programmatori. Un numero eccessivamente limitato di strutture rende scomoda la programmazione, mentre troppe strutture appesantiscono il sistema operativo e confondono i programmatori.

10.1.5 Struttura interna dei file

Per il sistema operativo la localizzazione di uno scostamento all'interno di un file può essere complicata. I dischi hanno una dimensione dei blocchi ben definita, stabilita secondo la dimensione di un settore. Tutti gli I/O su disco si eseguono in unità di un blocco (record fisico), e tutti i blocchi hanno la stessa dimensione. È improbabile che la dimensione del record fisico corrisponda esattamente alla lunghezza del record logico desiderato, che può anche essere variabile. Una soluzione diffusa per questo tipo di problema consiste nell'**impaccamento** di un certo numero di record logici in blocchi fisici.

Il sistema operativo UNIX, ad esempio, definisce tutti i file semplicemente come un flusso di byte. A ciascun byte si può accedere in modo individuale tramite il suo scostamento a partire dall'inizio, o dalla fine, del file. In questo caso il record logico è un byte. Il file system impacca e deimpacca automaticamente i byte in blocchi fisici (ad esempio 512 byte per blocco) come è necessario.

La dimensione dei record logici, quella dei blocchi fisici e la tecnica d'impaccamento determinano il numero dei record logici all'interno di ogni blocco fisico. L'impaccamento può essere fatto dal programma applicativo dell'utente oppure dal sistema operativo.

In entrambi i casi il file si può considerare come una sequenza di blocchi. Tutte le funzioni di I/O di base operano in termini di blocchi. La conversione da record logici a blocchi fisici è un problema di programmazione relativamente semplice.

Poiché lo spazio del disco è sempre assegnato in blocchi, una parte dell'ultimo blocco di ogni file in genere è sprecata. Se ogni blocco è composto di 512 byte, a un file di 1949 byte si assegnano quattro blocchi (2048 byte); gli ultimi 99 byte sono sprecati. I byte sprecati, assegnati per la gestione in multipli di blocchi invece che di byte, costituiscono la **frammentazione interna**. Tutti i file system ne soffrono; maggiore è la dimensione dei blocchi, maggiore sarà la frammentazione interna.

10.2 Metodi d'accesso

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; alcuni sistemi consentono un solo metodo d'accesso ai file; altri, come quelli IBM, offrono diversi metodi d'accesso; la scelta del metodo giusto per una particolare applicazione è un importante problema di progettazione.

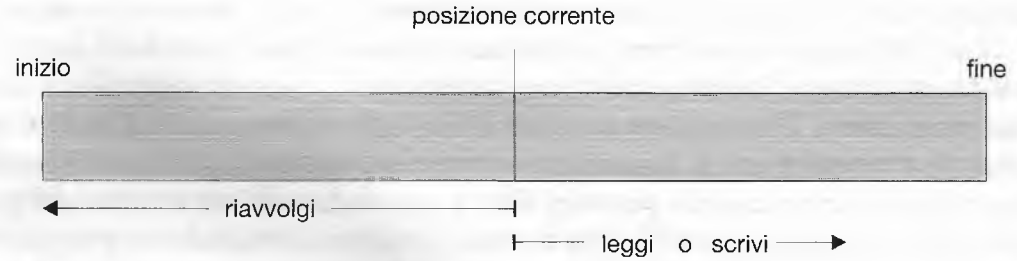


Figura 10.3 File ad accesso sequenziale.

10.2.1 Accesso sequenziale

Il più semplice metodo d'accesso è l'**accesso sequenziale**: le informazioni del file si elaborano ordinatamente, un record dopo l'altro; questo metodo d'accesso è di gran lunga il più comune, ed è usato, ad esempio, dagli editor e dai compilatori.

Le più comuni operazioni che si compiono sui file sono le letture e le scritture: un'operazione di lettura legge la prima porzione e fa avanzare automaticamente il puntatore del file che tiene traccia della locazione di I/O; analogamente, un'operazione di scrittura fa un'aggiunta in coda al file e avanza fino alla fine delle informazioni appena scritte, che costituisce la nuova fine del file. Un file siffatto si può reimpostare sull'inizio e, in alcuni sistemi, un programma può riuscire ad andare avanti o indietro di n record, con n intero e alcune volte solo per $n = 1$. L'accesso sequenziale è illustrato nella Figura 10.3. L'accesso sequenziale è basato su un modello di file che si rifà al nastro, e funziona nei dispositivi ad accesso sequenziale così come nei dispositivi ad accesso diretto.

10.2.2 Accesso diretto

Un altro metodo è l'**accesso diretto** (o **accesso relativo**). Un file è formato da elementi logici (**record**) di lunghezza fissa; ciò consente ai programmi di leggere e scrivere rapidamente tali elementi senza un ordine particolare. Il metodo ad accesso diretto si fonda su un modello di file che si rifà al disco: i dischi permettono, infatti, l'accesso diretto a ogni blocco di file. Il file si considera come una sequenza numerata di blocchi o record che si possono leggere o scrivere in modo arbitrario: si può ad esempio leggere il blocco 14, quindi il blocco 53 e poi scrivere il blocco 7. Non esistono limiti all'ordine di lettura o scrittura di un file ad accesso diretto.

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, ottenendo così le informazioni richieste.

In un sistema di prenotazione di volo, ad esempio, si possono registrare tutte le informazioni su un particolare volo, ad esempio il volo 713, nel blocco identificato da tale numero di volo. Quindi, il numero di posti disponibili per il volo 713 si memorizza nel blocco 713 del file di prenotazione. Per registrare informazioni riguardanti un gruppo più grande, ad esempio una popolazione, si può eseguire la ricerca calcolando una funzione hash sui nomi delle persone, oppure usando un piccolo indice per determinare il blocco da leggere.

Per il metodo ad accesso diretto, si devono modificare le operazioni sui file per inserire il numero del blocco in forma di parametro. Quindi, si hanno `read` n, dove n è il numero del blocco, al posto di `read next`, e `write` n, invece che `write next`. Un metodo alternativo prevede di mantenere `read next` e `write next`, come nell'accesso sequen-

ziale, e di aggiungere un'operazione `position to` *n*, dove *n* è il numero del blocco. Quindi a un'operazione `read` *n* corrispondono una `position to` *n* e una `read next`.

Il numero del blocco fornito dall'utente al sistema operativo è normalmente un **numero di blocco relativo**. Si tratta di un indice relativo all'inizio del file, quindi il primo blocco relativo del file è 0, il successivo è 1 e così via, anche se l'indirizzo assoluto nel disco del blocco può essere 14703 per il primo blocco e 3192 per il secondo. L'uso dei numeri di blocco relativi permette al sistema operativo di decidere dove posizionare il file (si tratta del *problema dell'allocazione* trattato nel Capitolo 11) e aiuta a impedire che l'utente acceda a porzioni del file system che possono non far parte del suo file. Alcuni sistemi iniziano la numerazione dei blocchi relativi da 0, altri da 1.

Come soddisfa il sistema operativo una richiesta di un record *n* in un file? Dato un record logico di lunghezza *l*, una richiesta per il record *n* determina una richiesta di I/O per *l* byte alla locazione $l \times n$ all'interno del file (assumendo che il primo record sia $n = 0$). Lettura, scrittura e cancellazione di un record sono rese più semplici dalla sua dimensione fissa.

Non tutti i sistemi operativi gestiscono ambedue i tipi di accesso; alcuni permettono il solo accesso sequenziale, altri solo quello diretto. Alcuni sistemi richiedono che si definisca il tipo d'accesso al file al momento della sua creazione; a tale file si può accedere soltanto nel modo definito. Tuttavia si può facilmente simulare l'accesso sequenziale a un file ad accesso diretto mantenendo una variabile `cp` che, come illustra la Figura 10.4, definisce la posizione corrente. D'altra parte è estremamente goffo e inefficiente simulare l'accesso diretto a un file che di per sé è ad accesso sequenziale.

10.2.3 Altri metodi d'accesso

Sulla base di un metodo d'accesso diretto se ne possono costruire altri, che implicano generalmente la costruzione di un indice per il 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.

Si consideri, ad esempio, un file contenente prezzi al dettaglio, contenente una lista dei codici universali dei prodotti (*universal product codes*, UPC), a ciascuno dei quali è associato un prezzo. Dato un elemento di 16 byte, questo è composto da un codice UPC a 10 cifre e un prezzo a 6 cifre. Se il disco usato ha 1024 byte per blocco, in ogni blocco si possono memorizzare 64 elementi. Un file di 120.000 elementi occupa circa 2000 blocchi (2 milioni di byte). Ordinando il file secondo il codice UPC si può definire un indice composto dal primo codice UPC di ogni blocco. Tale indice è costituito di 2000 elementi di 10 cifre ciascuno (20.000 byte) e quindi può essere tenuto in memoria. Per trovare il prezzo di un oggetto specifico si può fare una ricerca (binaria) nell'indice, che permette di sapere esattamente quale

Accesso sequenziale	Realizzazione nel caso di accesso diretto
`reset`	`cp = 0;`
`read next`	`read cp;` `cp = cp + 1;`
`write next`	`write cp;` `cp = cp + 1;`

Figura 10.4 Simulazione dell'accesso sequenziale a un file ad accesso diretto.

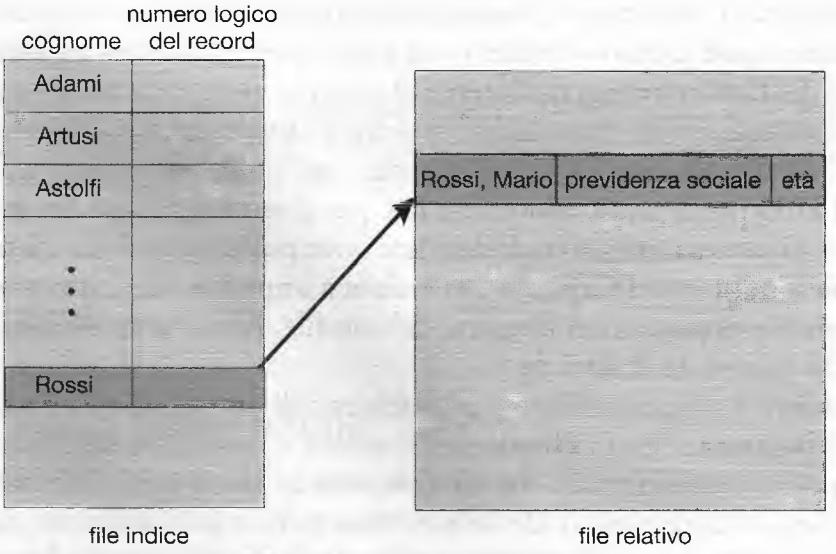


Figura 10.5 Esempio di indice e relativo file.

blocco contiene l'elemento desiderato e quindi accedere a quel blocco. Questa struttura permette di compiere ricerche in file molto lunghi limitando il numero di operazioni di I/O.

Nel caso di file molto lunghi, lo stesso file indice può diventare troppo lungo perché sia tenuto in memoria. Una soluzione a questo problema è data dalla creazione di un indice per il file indice. Il file indice principale contiene puntatori ai file indice secondari, che puntano agli effettivi elementi di dati.

Il metodo ad accesso sequenziale indicizzato di IBM (*indexed sequential access method*, ISAM), ad esempio, usa un piccolo indice principale che punta ai blocchi del disco di un indice secondario, e i blocchi dell'indice secondario puntano ai blocchi del file effettivo. Il file è ordinato rispetto a una chiave definita. Per trovare un particolare elemento, si fa inizialmente una ricerca binaria nell'indice principale, che fornisce il numero del blocco dell'indice secondario. Questo blocco viene letto e sottoposto a una seconda ricerca binaria che individui il blocco contenente l'elemento richiesto. Infine, si fa una ricerca sequenziale sul blocco. In questo modo si può localizzare ogni elemento tramite il suo codice con al massimo due letture ad accesso diretto. La Figura 10.5 mostra uno schema simile, com'è realizzato nel VMS con indici e relativi file.

10.3 Struttura della directory e del disco

Si prenderanno ora in considerazione le modalità di memorizzazione dei file. Ovviamente, nessun computer a uso generale contiene un unico file. Vi sono infatti memorizzati spesso migliaia, milioni o persino bilioni di file. I file vengono salvati in dispositivi di memorizzazione ad accesso casuale, come dischi fissi, dischi ottici e dischi a stato solido (basati sulla memoria).

Un dispositivo di memorizzazione può essere interamente utilizzato per un file system, ma può anche essere suddiviso per un controllo più raffinato. Un disco può ad esempio essere **partizionato** e ogni partizione può contenere un file system. I dispositivi di memorizzazione possono essere raccolti in insiemi RAID che proteggono dal fallimento di un singolo disco (come descritto al Paragrafo 12.7). Alcune volte, i dischi sono suddivisi e al contempo raccolti in insiemi RAID.

La suddivisione in partizioni è utile anche per limitare la dimensione dei file system individuali, per mettere sullo stesso dispositivo tipi diversi di file system, oppure per liberare ad altri scopi una parte del dispositivo, come nel caso dello spazio di swap (*avvicendamento*) o dello spazio su disco non formattato (**raw**). Le partizioni sono note anche come **suddivisioni** (*slice*) o **minidischi** (nel mondo IBM). Un file system può essere installato su ciascuna di queste parti del disco. Ogni entità contenente un file system è generalmente nota come **volume**. Il volume può essere un sottoinsieme di dispositivi, un dispositivo intero o dispositivi multipli collegati in RAID. Ogni volume può essere pensato come un disco virtuale. I volumi possono inoltre contenere diversi sistemi operativi, permettendo al sistema di avviare ed eseguire più di un sistema operativo.

Ogni volume contenente un file system deve anche avere in sé le informazioni sui file presenti nel sistema. Tali informazioni risiedono in una **directory del dispositivo** o **indice del volume**. La directory del dispositivo (in breve **directory**) registra informazioni, quali nome, posizione e tipo, di tutti i file del volume. La Figura 10.6 mostra la tipica organizzazione dei file system.

10.3.1 Struttura della memorizzazione di massa

Come discusso, un sistema informatico a uso generale dispone di dispositivi di memorizzazione multipli, che possono essere ripartiti nei volumi che contengono i file system. Il numero di file system in un sistema informatico può variare da zero a molti, e ne esistono di diverso tipo. Per esempio, un tipico sistema Solaris può avere molti file system diversi, come evidenziato nella Figura 10.7.

Nel testo prenderemo in considerazione solo file system a scopo generico. Vale la pena però di notare che esistono molti file system a scopo specifico. Consideriamo i tipi di file system dell'esempio menzionato precedentemente riguardante Solaris:

- **tmpfs** - un file system "temporaneo" creato nella memoria centrale volatile e i cui contenuti vengono cancellati se il sistema si riavvia o si blocca;
- **objfs** - un file system "virtuale" (essenzialmente un'interfaccia con il kernel che è simile a un file system) che permette agli strumenti che eseguono il debug di accedere ai simboli del kernel;

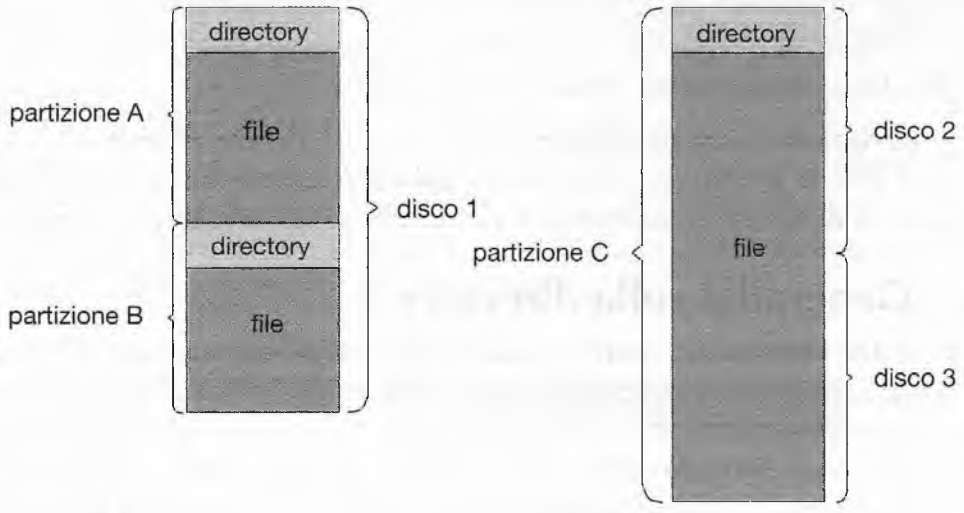


Figura 10.6 Tipica organizzazione di un file system.

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	procfs
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fdfs
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

Figura 10.7 File system in Solaris.

- **ctfs** - un file system virtuale che mantiene le informazioni "contrattuali" per gestire quali sono i processi che partono quando il sistema si avvia e quali devono continuare a girare durante il funzionamento del sistema;
- **lofs** - un file system di tipo "loop back" che permette di accedere a un file system piuttosto che a un altro;
- **procfs** - un file system virtuale che presenta le informazioni su tutti i processi presenti come se esse risiedessero su un file system;
- **ufs, zfs** - file system a scopo generico.

I file system dei computer possono quindi essere numerosi. Persino all'interno di un file system è utile dividere i file in gruppi da gestire e sui quali agire. Un'organizzazione di questo tipo richiede l'utilizzo di directory, argomento che esamineremo nel resto di questo paragrafo.

10.3.2 Generalità sulla directory

La directory si può considerare come una tabella di simboli che traduce i nomi dei file negli elementi in essa contenuti. Da questo punto di vista, si capisce che la stessa directory si può organizzare in molti modi diversi; deve essere possibile inserire nuovi elementi, cancellarne di esistenti, cercare un elemento ed elencare tutti gli elementi della directory. In questo paragrafo si affronta l'analisi delle appropriate strutture dati utilizzabili per la realizzazione delle directory. Nel considerare una particolare struttura della directory si deve tenere presente l'insieme delle seguenti operazioni che si possono eseguire su una directory.

- **Ricerca di un file.** Deve esserci la possibilità di scorrere una directory per individuare l'elemento associato a un particolare file. Poiché i file possono avere nomi simbolici, e poiché nomi simili possono indicare relazioni tra file, deve esistere la possibilità di trovare tutti i file il cui nome soddisfi una particolare espressione.
- **Creazione di un file.** Deve essere possibile creare nuovi file e aggiungerli alla directory.
- **Cancellazione di un file.** Quando non serve più, si deve poter rimuovere un file dalla directory.
- **Elencazione di una directory.** Deve esistere la possibilità di elencare tutti i file di una directory, e il contenuto degli elementi della directory associati ai rispettivi file nell'elenco.
- **Ridenominazione di un file.** Poiché il nome di un file rappresenta per i suoi utenti il contenuto del file, questo nome deve poter essere modificato quando il contenuto o l'uso del file subiscono cambiamenti. La ridenominazione di un file potrebbe comportare la variazione della posizione del file nella directory.
- **Attraversamento del file system.** Si potrebbe voler accedere a ogni directory e a ciascun file contenuto in una directory. Per motivi di affidabilità, è opportuno salvare il contenuto e la struttura dell'intero file system a intervalli regolari. Questo salvataggio consiste nella copiatura di tutti i file in un nastro magnetico; tale tecnica consente di avere una copia di riserva (*backup*) che sarebbe utile nel caso in cui si dovesse verificare un guasto nel sistema o più semplicemente se un file non è più in uso. In quest'ultimo caso si può liberare lo spazio da esso occupato nel disco, riutilizzabile quindi per altri file.

Nei paragrafi seguenti sono descritti gli schemi più comuni per la definizione della struttura logica di una directory.

10.3.3 Directory a livello singolo

La struttura più semplice per una directory è quella a livello singolo. Tutti i file sono contenuti nella stessa directory, facilmente gestibile e comprensibile (Figura 10.8).

Una directory a livello singolo presenta però limiti notevoli che si manifestano all'aumentare del numero dei file in essa contenuti, oppure se il sistema è usato da più utenti. Poiché si trovano tutti nella stessa directory, i file devono avere nomi unici; se due utenti attribuiscono lo stesso nome al loro file di dati, ad esempio `prova`, si vìola la regola del nome unico. Anche se i nomi dei file generalmente si scelgono in modo da riflettere il contenuto del file stesso, spesso hanno una lunghezza limitata (il sistema operativo MS-DOS permette nomi di non più di 11 caratteri, lo UNIX permette lunghezze di 255 caratteri).

Anche per un solo utente, con una directory a livello singolo, con l'aumentare del loro numero diventa difficile ricordare i nomi dei file. Non è affatto raro che un utente abbia centinaia di file in un calcolatore e altrettanti file in un altro sistema. In un tale ambiente, sarebbe un compito arduo dover ricordare tanti nomi di file.

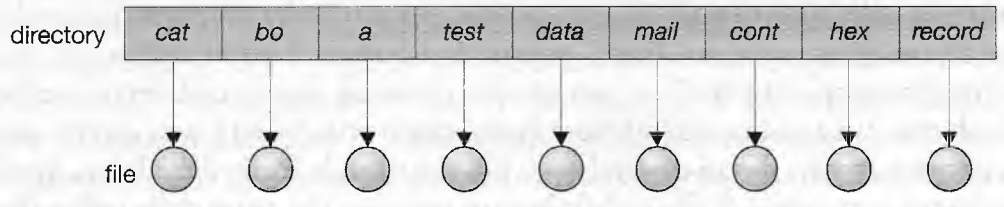


Figura 10.8 Directory a livello singolo.

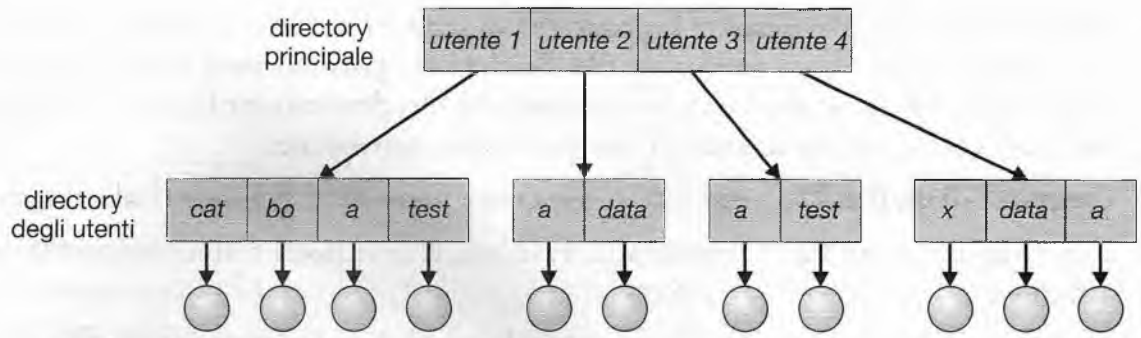


Figura 10.9 Struttura della directory a due livelli.

10.3.4 Directory a due livelli

Come abbiamo visto, una directory a livello singolo spesso causa confusione dei nomi dei file tra diversi utenti. La soluzione più ovvia prevede la creazione di una directory *separata* per ogni utente.

Nella struttura a due livelli, ogni utente dispone della propria **directory utente** (*user file directory*, UFD). Tutte le directory utente hanno una struttura simile, ma in ciascuna sono elencati solo i file del proprietario. Quando comincia l'elaborazione di un lavoro dell'utente, oppure un utente inizia una sessione di lavoro, si fa una ricerca nella **directory principale** (*master file directory*, MFD) del sistema. La directory principale viene indirizzata con il nome dell'utente o il numero che lo rappresenta, e ogni suo elemento punta alla relativa directory utente (Figura 10.9).

Quando un utente fa un 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, purché tutti i nomi di file all'interno di ciascuna directory utente siano unici. Per creare un file per un utente, il sistema operativo controlla che non ci sia un altro file con lo stesso nome soltanto nella directory di tale utente. Per cancellare un file il sistema operativo limita la propria ricerca alla directory utente locale, quindi non può cancellare per errore un file con lo stesso nome che appartenga a un altro utente.

Le stesse directory utente devono essere create e cancellate come è necessario; a tale scopo si esegue uno speciale programma di sistema con nome dell'utente e dati contabili adeguati. Il programma crea una nuova directory utente e aggiunge l'elemento a essa corrispondente nella directory principale. L'esecuzione di questo programma può essere limitata all'amministratore del sistema. L'allocazione dello spazio nei dischi per le directory utente può essere gestita con le tecniche descritte per i file nel Capitolo 11.

Sebbene risolva la questione delle collisioni dei nomi, la struttura della directory a due livelli presenta ancora dei problemi. In effetti, questa struttura isola un utente dagli altri. Questo isolamento può essere un vantaggio quando gli utenti sono completamente indipendenti, ma è uno svantaggio quando gli utenti *vogliono* cooperare e accedere a file di altri utenti. Alcuni sistemi non permettono l'accesso a file di utenti locali da parte di altri utenti.

Se l'accesso è autorizzato, un utente deve avere la possibilità di riferirsi al nome di un file che si trova nella directory di un altro utente. Per attribuire un nome unico a un particolare file di una directory a due livelli, occorre indicare sia il nome dell'utente sia il nome del file. Una directory a due livelli si può pensare come un albero, o almeno un albero rovesciato, di altezza 2. La radice dell'albero è la directory principale, i suoi diretti discendenti sono le directory utente, da cui discendono i file che sono le foglie dell'albero. Specificando un nome utente e un nome di file si definisce un percorso che parte dalla radice (la directo-

ry principale) e arriva a una specifica foglia (il file specificato). Quindi, un nome utente e un nome di file definiscono un *nome di percorso* (*path name*). Ogni file del sistema ha un nome di percorso. Per attribuire un nome unico a un file, un utente deve conoscere il nome di percorso del file desiderato.

Se, ad esempio, l'utente *A* desidera accedere al proprio file chiamato `prova`, è sufficiente che faccia riferimento a `prova`. Invece, per accedere al file denominato `prova` dell'utente *B*, con nome di elemento della directory `utenteB`, l'utente *A* deve fare riferimento a `/utenteB/prova`. Ogni sistema ha la propria sintassi per riferirsi ai file delle directory diverse da quella dell'utente.

Per specificare il volume cui appartiene un file occorrono ulteriori regole sintattiche. Nell'MS-DOS, ad esempio, il volume è indicato da una lettera seguita dai due punti. Quindi l'indicazione di un file potrebbe essere del tipo `C:\utenteB\prova`. Alcuni sistemi vanno oltre e separano: volume, nome della directory, e nome del file. Nel VMS, ad esempio, il file `login.com` potrebbe essere indicato come `u:[sst.jdeck]login.com;1`, dove `u` è il nome del volume, `sst` è il nome della directory, `jdeck` è il nome della sottodirectory e `1` è il numero della versione. Altri sistemi trattano il nome del volume semplicemente come parte del nome della directory. Il primo elemento è quello del volume, il resto è composto dalla directory e dal file. Ad esempio, `/u/pbg/prova` potrebbe indicare il volume `u`, la directory `pbg` e il file `prova`.

Un caso particolare di questa situazione riguarda i file di sistema. I programmi forniti come elementi integranti del sistema, come caricatori, assemblatori, compilatori, strumenti, librerie e così via, sono infatti definiti come file. Quando si impartiscono al sistema operativo i comandi appropriati, il caricatore legge questi file che poi vengono eseguiti. Molti interpreti di comandi operano semplicemente trattando il comando come il nome di un file da caricare ed eseguire. Poiché il sistema delle directory è già stato definito, questo nome di file viene cercato nella directory utente locale. Una soluzione prevede la copiatura dei file di sistema in ciascuna directory utente. Tuttavia, con la copiatura di tutti i file di sistema si spreca un'enorme quantità di spazio. Se i file di sistema occupano 5 MB, con 12 utenti si avrebbe un'occupazione di spazio pari a $5 \times 12 = 60$ MB, solo per le copie dei file di sistema.

La soluzione standard prevede una leggera complicazione della procedura di ricerca; si definisce una speciale directory utente contenente i file di sistema, ad esempio la directory utente 0. Ogni volta che si indica un file da caricare, il sistema operativo lo cerca innanzitutto nella directory utente locale, e, se lo trova, lo usa; se non lo trova, il sistema cerca automaticamente nella speciale directory utente contenente i file di sistema. La sequenza delle directory in cui è stata fatta la ricerca avviata dal riferimento a un file è detta **percorso di ricerca** (*search path*). Tale percorso si può estendere in modo da contenere una lista illimitata di directory in cui fare le ricerche quando si dà il nome di un comando. Questo metodo è il più usato in UNIX e MS-DOS. Alcuni sistemi prevedono anche che ogni utente disponga del proprio percorso di ricerca personale.

10.3.5 Directory con struttura ad albero

La corrispondenza strutturale tra directory a due livelli e albero a due livelli permette di generalizzare facilmente il concetto, estendendo la struttura della directory a un albero di altezza arbitraria (Figura 10.10). Questa generalizzazione permette agli utenti di creare proprie sottodirectory e di organizzare i file di conseguenza. Il file system dell'MS-DOS, ad esempio, è strutturato ad albero; si tratta infatti del più comune tipo di struttura delle directory. L'albero ha una directory radice (*root directory*), e ogni file del sistema ha un unico nome di percorso.

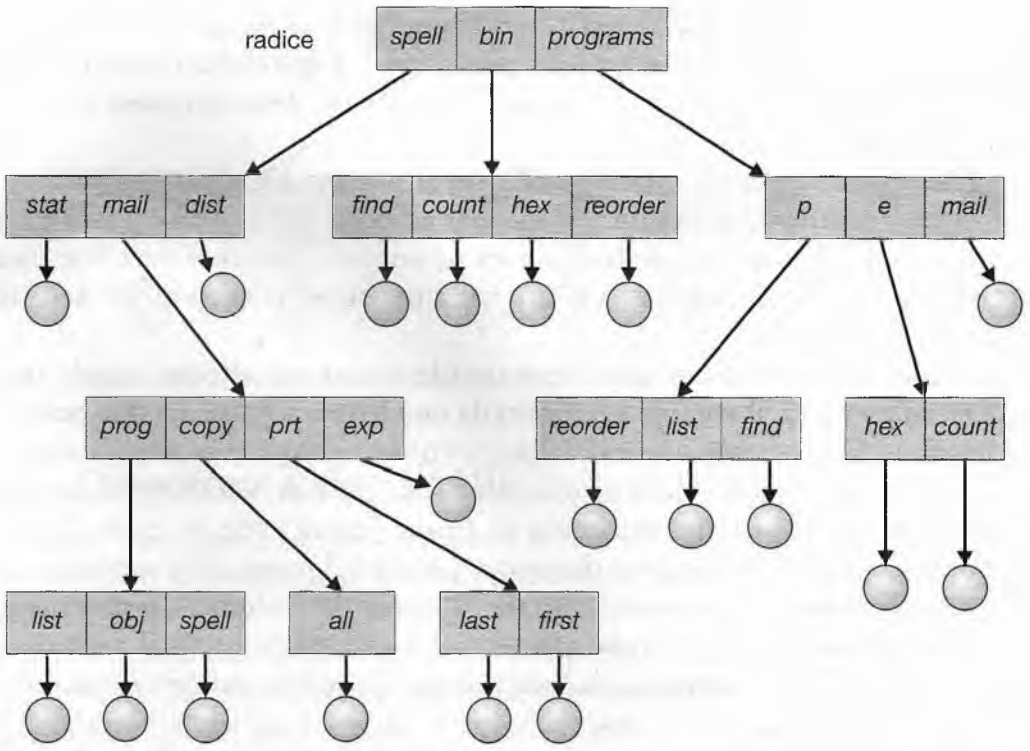


Figura 10.10 Struttura della directory ad albero.

Una directory, o una sottodirectory, contiene un insieme di file o sottodirectory. Le directory sono semplicemente file, trattati però in modo speciale. Tutte le directory hanno lo stesso formato interno. La distinzione tra file e directory è data dal bit, rispettivamente 0 e 1, di ogni elemento della directory. Per creare e cancellare le directory si adoperano speciali chiamate di sistema.

Normalmente, ogni utente dispone di una directory corrente. La **directory corrente** deve contenere la maggior parte dei file di interesse corrente per il processo. Quando si fa un riferimento a un file, si esegue una ricerca nella directory corrente; se il file non si trova in tale directory, l'utente deve specificare un nome di percorso oppure cambiare la directory corrente facendo diventare tale la directory contenente il file desiderato. Per cambiare directory corrente si fa uso di una chiamata di sistema che preleva un nome di directory come parametro e lo usa per ridefinire la directory corrente. Quindi, l'utente può cambiare la propria directory corrente ogni volta che lo desidera. Da una chiamata di sistema `change directory` alla successiva, tutte le chiamate di sistema `open()` cercano i file specificati nella directory corrente. Si noti che il percorso di ricerca può contenere o meno uno speciale elemento che rappresenta "la directory corrente".

La directory corrente iniziale di un utente è stabilita all'avvio del lavoro d'elaborazione dell'utente, oppure quando quest'ultimo inizia una sessione di lavoro; il sistema operativo cerca nel file di contabilizzazione, o in qualche altra locazione predefinita, l'elemento relativo a questo utente. Nel file di contabilizzazione è memorizzato un puntatore alla (oppure il nome della) directory iniziale dell'utente. Tale puntatore viene copiato in una variabile locale per l'utente che specifica la sua directory corrente iniziale. Dalla shell dell'utente si possono poi avviare altri processi: la loro directory corrente è solitamente la directory corrente del processo genitore al momento della creazione del figlio.

I nomi di percorso possono essere di due tipi: **nomi di percorso assoluti** e **nomi di percorso relativi**. Un *nome di percorso assoluto* comincia dalla radice dell'albero di directory

e segue un percorso che lo porta fino al file specificato indicando i nomi delle directory che costruiscono le tappe del suo cammino. Un *nome di percorso relativo* definisce un percorso che parte dalla directory corrente. Ad esempio, nel file system con struttura ad albero della Figura 10.10, se la directory corrente è `root/spell/mail`, il nome di percorso relativo `prt/first` si riferisce allo stesso file indicato dal percorso assoluto `root/spell/mail/prt/first`.

Se si permette all'utente di definire le proprie sottodirectory, gli si consente anche di dare una struttura ai suoi file. Questa struttura può presentare directory distinte per file associati a soggetti diversi; ad esempio, si può creare una sottodirectory contenente il testo di questo libro, oppure diversi tipi di informazioni, ad esempio la directory `programmi` può contenere programmi sorgente; la directory `bin` può contenere tutti i programmi eseguibili.

Una decisione importante relativa alla strutturazione ad albero delle directory riguarda il modo di gestire la cancellazione di una directory. Se una directory è vuota, è sufficiente cancellare l'elemento che la designa nella directory che la contiene. Tuttavia se la directory da cancellare non è vuota, ma contiene file oppure sottodirectory, è possibile procedere in due modi. Alcuni sistemi, come l'MS-DOS, non cancellano una directory a meno che non sia vuota; per cancellarla l'utente deve prima cancellare i file in essa contenuti. Se esiste qualche sottodirectory, questa procedura si deve applicare anche alle sottodirectory. Questo metodo può richiedere una discreta quantità di lavoro. In alternativa, come nel comando `rm` di UNIX, si può avere un'opzione che, alla richiesta di cancellazione di una directory, cancelli anche tutti i file e tutte le sottodirectory in essa contenuti. Entrambi i criteri sono abbastanza facili da realizzare; si tratta soltanto di stabilire quale seguire. Il secondo criterio è più comodo, anche se più pericoloso, poiché si può rimuovere un'intera struttura della directory con un solo comando. Se si eseguisse tale comando per sbaglio sarebbe necessario ripristinare un gran numero di file e directory dalle copie di riserva (ipotizzandone l'esistenza).

Con un sistema di directory strutturato ad albero anche l'accesso ai file di altri utenti è di facile realizzazione. Ad esempio, l'utente *B* può accedere ai file dell'utente *A* specificando i nomi di percorso assoluti oppure relativi. In alternativa, l'utente B può far sì che la propria directory corrente sia quella dell'utente *A* e accedere ai file usando direttamente i loro nomi.

Un percorso per un file in una directory strutturata ad albero può ovviamente essere più lungo di quello in una directory a due livelli. Per consentire agli utenti di accedere ai programmi senza dover ricordare tali lunghi percorsi, il sistema operativo Macintosh cerca automaticamente i programmi eseguibili attraverso un file detto `Desktop file` contenente il nome e la locazione di tutti i programmi eseguibili a esso noti. Quando si aggiunge al sistema un nuovo disco o si accede a una rete di comunicazione, il sistema operativo attraversa la struttura delle directory del dispositivo alla ricerca dei programmi eseguibili e registra le informazioni relative. Questo meccanismo consente la funzione del doppio clic descritta precedentemente. Un doppio clic sul nome di un file, o sull'icona che lo rappresenta, determina la lettura del suo attributo di creazione e la ricerca dell'elemento corrispondente nel `Desktop file`. Una volta trovato tale elemento, si avvia il programma eseguibile associato che apre il file su cui si è fatto il doppio clic.

10.3.6 Directory con struttura a grafo aciclico

Si considerino due programmatori che lavorano a un progetto comune. I file associati a quel progetto si possono memorizzare in una sottodirectory, separandoli da altri progetti e file dei due programmatori, ma poiché entrambi i programmatori hanno le stesse responsabilità sul progetto, ciascuno richiede che la sottodirectory si trovi nelle proprie directory. La sot-

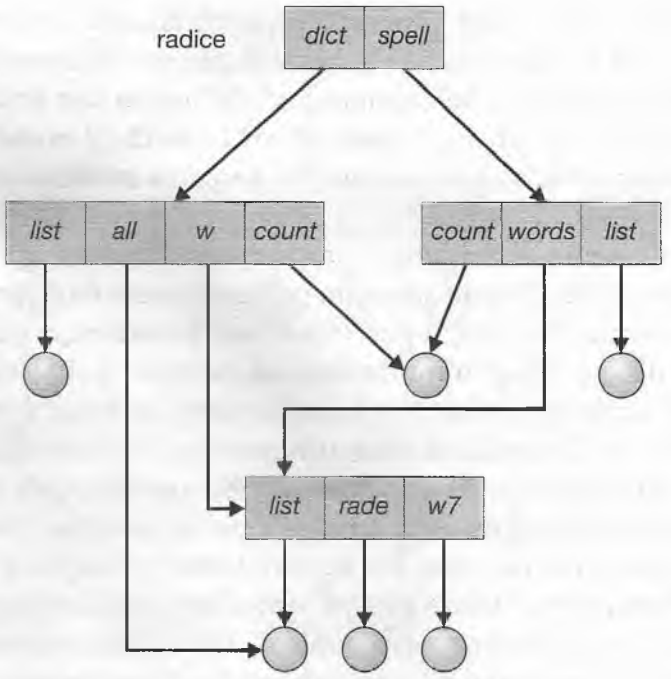


Figura 10.11 Struttura della directory a grafo aciclico.

todirectory comune deve essere *condivisa*. Nel sistema esiste quindi una directory condivisa o un file condiviso in due, o più, posizioni alla volta.

La struttura ad albero non ammette la condivisione di file o directory. Un **grafo aciclico** permette alle directory di avere sottodirectory e file condivisi (Figura 10.11). Lo *stesso* file o la *stessa* sottodirectory possono essere in due directory diverse. Un grafo aciclico, cioè senza cicli, rappresenta la generalizzazione naturale dello schema delle directory con struttura ad albero.

Il fatto che un file sia condiviso, o che lo sia una directory, non significa che ci siano due copie del file: con due copie ciascun programmatore potrebbe vedere la copia presente nella propria directory e non l'originale; se un programmatore modifica il file, le modifiche non appaiono nell'altra copia. Se invece il file è condiviso esiste *un* solo file effettivo, perciò tutte le modifiche sono immediatamente visibili. La condivisione è di particolare importanza se applicata alle sottodirectory; un nuovo file appare automaticamente in tutte le sottodirectory condivise.

Quando più persone lavorano insieme, tutti i file da condividere si possono inserire in una directory comune. Ciascuna directory utente di tutti i membri del gruppo contiene questa directory di file condivisi in forma di sottodirectory. Anche nel caso di un singolo utente, il metodo di gestione dei file di tale utente può richiedere che alcuni file siano inseriti in più sottodirectory. Ad esempio, un programma scritto per un progetto particolare deve trovarsi sia nella directory di tutti i programmi sia nella directory di quel progetto.

I file e le sottodirectory condivisi si possono realizzare in molti modi. Un metodo diffuso, esemplificato da molti tra i 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. Ad esempio, un collegamento si può realizzare come un nome di percorso assoluto o relativo. Quando si fa riferimento a un file, si compie una ricerca nella directory, l'elemento cercato è contrassegnato come collegamento e riporta il nome di percorso del file, o della directory, reale. Quindi si *risolve* il collegamento usando il nome di percorso per localizzare il file reale. I collegamenti si identificano facilmente tramite il loro formato nell'ele-

mento della directory (o, nei sistemi che gestiscono i tipi, dal tipo speciale), e sono infatti chiamati puntatori indiretti. Durante l'attraversamento degli alberi delle directory il sistema operativo ignora questi collegamenti, preservandone così la struttura aciclica.

Un altro comune metodo per la realizzazione dei file condivisi prevede semplicemente 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, invece, la copia e l'originale sono resi indistinguibili: sorge allora il problema di mantenere la coerenza se il file viene modificato.

Una struttura della directory a grafo aciclico è più flessibile di una semplice struttura ad albero, ma anche più complessa. Si devono prendere in considerazione parecchi problemi. Un file può avere più nomi di percorso assoluti, quindi nomi diversi possono riferirsi allo stesso file. Questa situazione è simile al problema dell'uso degli *alias* nei linguaggi di programmazione. Quando si percorre tutto il file system – per trovare un file, per raccogliere dati statistici su tutti i file o per fare le copie di riserva di tutti i file – il problema diviene più grave poiché le strutture condivise non si devono attraversare più di una volta.

Un altro problema riguarda la cancellazione, poiché è necessario stabilire in quali casi è possibile allocare e riutilizzare lo spazio allocato a un file condiviso. Una possibilità prevede che a ogni operazione di cancellazione segua l'immediata rimozione del file; quest'azione può però lasciare puntatori a un file che ormai non esiste più. Sarebbe ancora più grave se i puntatori contenessero gli indirizzi effettivi del disco e lo spazio fosse poi riutilizzato per altri file, poiché i puntatori potrebbero puntare nel mezzo di questi altri file.

In un sistema dove la condivisione è realizzata da collegamenti simbolici la gestione di questa situazione è relativamente semplice. La cancellazione di un collegamento non influisce sul file originale, poiché si rimuove solo il collegamento. Se si cancella il file, si libera lo spazio corrispondente lasciando in sospeso il collegamento; a questo punto è possibile cercare tutti questi collegamenti e rimuoverli, ma se in ogni file non esiste una lista dei collegamenti associati al file stesso questa ricerca può essere abbastanza onerosa. In alternativa, si possono lasciare i collegamenti finché non si tenta di usarli, quindi si "scopre" che il file con il nome dato dal collegamento non esiste e non si riesce a determinare il collegamento rispetto al nome; l'accesso è trattato proprio come qualsiasi altro nome di file irregolare. In questo caso, il progettista del sistema deve decidere attentamente cosa si debba fare quando si cancella un file e si crea un altro file con lo stesso nome, prima che sia stato usato un collegamento simbolico al file originario. In UNIX, quando si cancella un file, i collegamenti simbolici restano, è l'utente che deve rendersi conto che il file originale è scomparso o è stato sostituito. Nella famiglia di sistemi operativi Microsoft Windows si segue lo stesso criterio.

Un altro tipo di approccio prevede la conservazione del file finché non siano stati cancellati tutti i riferimenti a esso. In questo caso è necessario disporre di un meccanismo che determini la cancellazione dell'ultimo riferimento a quel file; è possibile tenere una lista di tutti i riferimenti a un file (elementi di directory o collegamenti simbolici). Quando si crea un collegamento, oppure una copia dell'elemento della directory, si aggiunge un nuovo elemento alla lista dei riferimenti al file; quando si cancella un collegamento oppure un elemento della directory, si elimina dalla lista l'elemento corrispondente. Quando la sua lista di riferimenti è vuota, il file viene cancellato.

Questo metodo presenta, però, un problema: la dimensione della lista dei riferimenti al file può essere variabile e potenzialmente grande. Tuttavia, non è realmente necessario mantenere l'intera lista, è sufficiente un contatore del *numero* di riferimenti. Un nuovo collegamento o un nuovo elemento della directory incrementa il numero dei riferimenti; la

cancellazione di un collegamento o un elemento decrementa questo numero. Quando il contatore è uguale a 0 si può cancellare il file, poiché non ci sono più riferimenti a tale file. Il sistema operativo UNIX usa questo metodo per i collegamenti non simbolici, o **collegamenti effettivi** (*hard link*); il contatore dei riferimenti è tenuto nel blocco di controllo del file o *inode*. Impedendo che si facciano più riferimenti a una directory, si può mantenere una struttura a grafo aciclico.

Per evitare questi problemi alcuni sistemi non consentono la condivisione delle directory né i collegamenti. Nell'MS-DOS, ad esempio, la struttura della directory è una struttura ad albero anziché a grafo aciclico.

10.3.7 Directory con struttura a grafo generale

Un serio problema connesso all'uso di una struttura a grafo aciclico consiste nell'assicurare che non vi siano cicli. Iniziando con una directory a due livelli e permettendo agli utenti di creare sottodirectory si crea una directory con struttura ad albero. Aggiungendo nuovi file e nuove sottodirectory alla directory con struttura ad albero, la natura di quest'ultima persiste. Tuttavia, quando si aggiungono dei collegamenti a una directory con struttura ad albero, tale struttura si trasforma in una semplice struttura a grafo, come quella illustrata nella Figura 10.12.

Il vantaggio principale di un grafo aciclico è dato dalla semplicità degli algoritmi necessari per attraversarlo e per determinare quando non ci siano più riferimenti a un file. È preferibile evitare un duplice attraversamento di sezioni condivise di un grafo aciclico, soprattutto per motivi di prestazioni. Se un file particolare è stato appena cercato in una sottodirectory condivisa, ma non è stato trovato, è preferibile evitare una seconda ricerca nella stessa sottodirectory, che costituirebbe solo una perdita di tempo.

Se si permette che nella directory esistano cicli, è preferibile evitare una duplice ricerca di un elemento, per motivi di correttezza e di prestazioni. Un algoritmo mal progettato potrebbe causare un ciclo infinito di ricerca. Una soluzione è quella di limitare arbitrariamente il numero di directory cui accedere durante una ricerca.

Un problema analogo si presenta al momento di stabilire quando sia possibile cancellare un file. Come con le strutture delle directory a grafo aciclico, la presenza di uno 0 nel

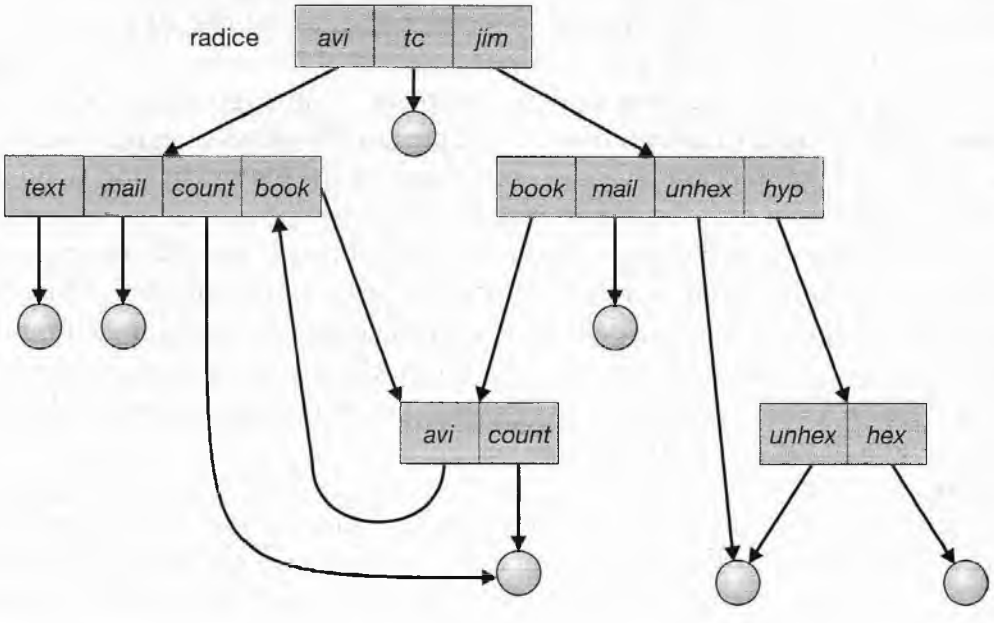


Figura 10.12 Directory a grafo generale.

contatore dei riferimenti significa che non esistono più riferimenti al file o alla directory, e quindi il file può essere cancellato. Tuttavia, se esistono cicli, è possibile che il contatore dei riferimenti possa essere non nullo, anche se non è più possibile far riferimento a una directory o a un file. Questa anomalia è dovuta alla possibilità di autoriferimento (ciclo) nella struttura delle directory. In questo caso è generalmente necessario usare un metodo di "ripulitura" (*garbage collection*) per stabilire quando sia stato cancellato l'ultimo riferimento e quando sia possibile riallocare lo spazio dei dischi. Tale metodo implica l'attraversamento del file system, durante il quale si contrassegna tutto ciò che è accessibile; in un secondo passaggio si raccoglie in un elenco di blocchi liberi tutto ciò che non è contrassegnato. Una procedura di marcatura analoga è utilizzabile per assicurare che un attraversamento o una ricerca coprano tutto quel che si trova nel file system una sola volta. L'applicazione di questo metodo a un file system basato su dischi richiede però molto tempo, perciò viene tentata solo di rado.

Inoltre, poiché è necessaria solo a causa della presenza dei cicli, è molto più conveniente lavorare con una struttura a grafo aciclico. La difficoltà consiste nell'evitare i cicli quando si aggiungono nuovi collegamenti alla struttura. Per sapere quando un nuovo collegamento ha completato un ciclo si possono impiegare gli algoritmi che permettono di individuare la presenza di cicli nei grafi. Dal punto di vista del calcolo, però, questi algoritmi sono onerosi, soprattutto quando il grafo si trova in memoria secondaria. Nel caso particolare di directory e collegamenti, un semplice algoritmo prevede di evitare i collegamenti durante l'attraversamento delle directory: si evitano così i cicli senza alcun carico ulteriore.

10.4 Montaggio di un file system

Così come si deve *aprire* un file per poterlo usare, per essere reso accessibile ai processi di un sistema, un file system deve essere *montato*. La struttura delle directory può ad esempio essere composta di volumi, che devono essere montati affinché siano disponibili nello spazio dei nomi di un file system.

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. Alcuni sistemi operativi richiedono un file system prefissato, mentre altri ispezionano le strutture del dispositivo e determinano il tipo del file system presente. Di solito, un punto di montaggio è una directory vuota cui sarà agganciato il file system che deve essere montato. Ad esempio, in un sistema UNIX, un file system contenente le directory iniziali degli utenti si potrebbe montare come `/home`; quindi la directory iniziale dell'utente *gianna* avrebbe il percorso `/home/gianna`. Se lo stesso file system si montasse come `/users` il percorso per quella directory sarebbe `/users/gianna`.

Il passo successivo consiste nella verifica da parte del sistema operativo della validità del file system contenuto nel dispositivo. La verifica si compie chiedendo al driver del dispositivo di leggere la directory di dispositivo e 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. Questo schema permette al sistema operativo di attraversare la sua struttura della directory, passando da un file system all'altro secondo le necessità.

Per illustrare l'operazione di montaggio, si consideri il file system rappresentato nella Figura 10.13, in cui i triangoli rappresentano sottoalberi di directory rilevanti. La Figura 10.13(a), mostra un file system esistente, mentre nella Figura 10.13(b) è raffigurato un volume non ancora montato che risiede in `/device/dsk`. A questo punto, si può accedere

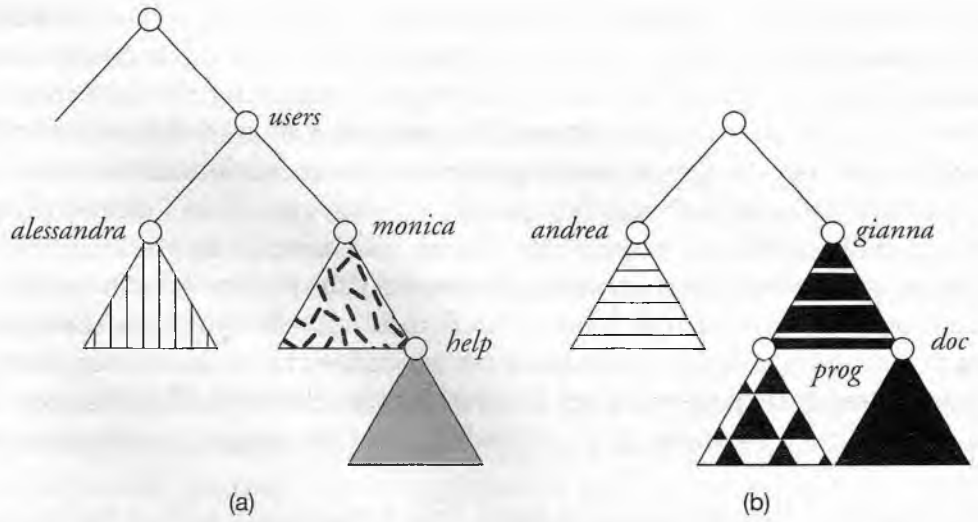


Figura 10.13 File system; (a) esistente; (b) volume non montato.

solo ai file del file system esistente. Nella Figura 10.14 si possono vedere gli effetti dell'operazione di montaggio del volume residente in `/device/dsk` al punto di montaggio `/users`. Se si smonta il volume, il file system ritorna alla situazione rappresentata nella Figura 10.13.

Per rendere più chiare le loro funzioni, sistemi operativi impongono una semantica a queste operazioni. Ad esempio, 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 finché non si *smonti* il file system, concludendone l'uso e permettendo l'accesso ai file originariamente presenti in tale directory. Come ulteriore esempio, un sistema potrebbe permettere il montaggio ripetuto dello stesso file system in diversi punti di montaggio, o potrebbe imporre una sola possibilità di montaggio per ciascun file system.

Si considerino le azioni del sistema operativo Macintosh. Ogni volta che il sistema rileva per la prima volta un disco (i dischi sono rilevati nella fase d'avviamento, mentre i dischetti quando sono inseriti nella relativa unità), il sistema operativo Macintosh cerca un file system nel dispositivo. Se ne trova uno, esegue automaticamente il montaggio del file system a livello della radice, aggiungendo un'icona di cartella sulla scrivania virtuale, etichettata con il nome del file system (secondo quel che è memorizzato nella directory di dispositivo). A questo punto l'utente può selezionare l'icona con il mouse e quindi vedere il contenuto del nuovo file system appena montato. L'OS X di Mac si comporta in modo si-

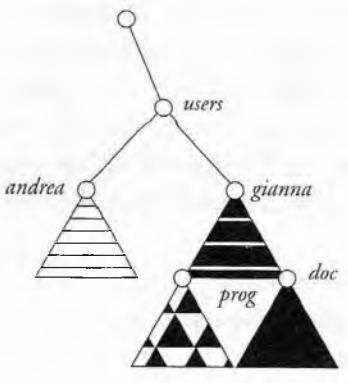


Figura 10.14 Punto di montaggio.

mile a UNIX BSD, sul quale si basa. Tutti i file system vengono montati sotto la directory/Volumes. L'interfaccia grafica di Mac OS X nasconde questa caratteristica e mostra i file system come se fossero tutti montati al livello della radice.

La famiglia di sistemi operativi Microsoft Windows (95, 98, NT, 2000, 2003, XP, Vista) mantiene una struttura della directory a due livelli estesa, con una lettera di unità associata a dispositivi e volumi, che hanno una struttura della directory a grafo generale associata a una lettera di unità. Il percorso completo per uno specifico file è quindi della forma `lettera_di_unità:\percorso\file`. Le versioni più recenti di Windows permettono di montare un file system su qualunque punto dell'albero delle directory, esattamente come UNIX. I sistemi Windows rilevano automaticamente tutti i dispositivi e montano tutti i file system rilevati all'avvio della macchina. In altri sistemi, per esempio UNIX, occorre montare esplicitamente i file system. Un file di configurazione del sistema contiene una lista di dispositivi (e relativi punti di montaggio) da montare automaticamente all'avvio, ma altri montaggi possono essere eseguiti manualmente.

Le questioni riguardanti il montaggio di file system sono approfondite nel Paragrafo 11.2.2.

10.5 Condivisione di file

Nei paragrafi precedenti sono state presentate le motivazioni alla base della necessità di condivisione dei file, oltre ad alcune difficoltà che si incontrano nel permettere che diversi utenti possano condividere file. Tale possibilità è particolarmente utile per utenti che vogliano collaborare e per ridurre le risorse richieste per raggiungere un certo obiettivo di calcolo. Quindi, nonostante le difficoltà inerenti alla condivisione, i sistemi operativi orientati agli utenti devono soddisfare questa esigenza.

In questo paragrafo si considerano diversi aspetti della condivisione dei file, innanzitutto l'aspetto relativo alla molteplicità degli utenti e ai diversi metodi di condivisione possibili. Una volta che più utenti possono condividere file, l'obiettivo diventa estendere la condivisione a più file system, compresi i file system remoti. Infine, ci possono essere diverse interpretazioni delle azioni conflittuali intraprese su file condivisi. Ad esempio, se più utenti stanno scrivendo nello stesso file, ci si può chiedere se il sistema dovrebbe permettere tutte le operazioni di scrittura, oppure dovrebbe proteggere le azioni di ciascun utente da quelle degli altri.

10.5.1 Utenti multipli

Se un sistema operativo permette l'uso del sistema da parte di più utenti, diventano particolarmente rilevanti i problemi relativi alla condivisione dei file, alla loro identificazione tramite nomi e alla loro protezione. Data una struttura della directory che permette la condivisione di file da parte degli utenti, il sistema deve poter mediare questa condivisione. Il sistema può permettere a ogni utente, in modo predefinito, di accedere ai file degli altri utenti, oppure può richiedere che un utente debba esplicitamente concedere i permessi di accesso ai file. Questi aspetti sono alla base dei temi del controllo degli accessi e della protezione, trattati nel Paragrafo 10.6.

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. Sebbene storicamente siano stati proposti molti metodi per la realizzazione di questi meccanismi, la maggior parte dei sistemi ha adottato – relativamente a ciascun file o directory –

i concetti di *proprietario* (o utente) e *gruppo*. Il proprietario è l'utente che può cambiare gli attributi di un file o directory, concedere l'accesso e che, in generale, ha il maggior controllo sul file o directory. L'attributo di gruppo di un file si usa per definire il sottoinsieme di utenti autorizzati a condividere l'accesso al file. Ad esempio, il proprietario di un file in un sistema UNIX può fare qualsiasi operazione sul file, mentre i membri del gruppo possono compiere un sottoinsieme di queste operazioni e il resto degli utenti un altro sottoinsieme. Il proprietario del file può definire l'esatto insieme di operazioni che i membri del gruppo e gli altri utenti possono eseguire. Maggiori dettagli sugli attributi che regolano i permessi sono trattati nel paragrafo successivo.

Gli identificatori del gruppo e del proprietario di un certo file o directory (ID) sono memorizzati insieme con gli altri attributi del file. Quando un utente richiede di compiere un'operazione su un file, per verificare se l'utente richiedente è il proprietario del file si può confrontare l'identificatore utente con l'attributo che identifica il proprietario. Analogamente si confrontano gli identificatori di gruppo. Ne risultano i permessi che l'utente ha sul file, e che il sistema considera per consentire o impedire l'operazione richiesta.

Molti sistemi operativi hanno più file system locali, inclusi volumi di un unico disco, o più volumi in diversi dischi connessi al sistema. In questi casi, la verifica degli identificatori e il confronto dei permessi si possono fare facilmente dopo l'operazione di montaggio dei file system.

10.5.2 File system remoti

L'avvento delle reti (Capitolo 16) ha permesso la comunicazione tra calcolatori separati da grandi distanze. Le reti permettono la condivisione di risorse sparse nell'area di un *campus* universitario o addirittura in diversi luoghi del mondo. Un'ovvia risorsa da condividere sono i dati, nella forma di file.

I metodi con i quali i file si condividono in una rete sono cambiati molto, seguendo l'evoluzione della tecnologia delle reti e dei file. Un primo metodo consiste nel trasferimento dei file richiesto in modo esplicito dagli utenti, attraverso programmi come l'`ftp`. 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, il **World Wide Web**, è, da un certo punto di vista, il contrario del primo. Per accedere ai file remoti si usa un programma di consultazione (*browser*), e operazioni distinte – essenzialmente un involucro (*wrapper*) per l'`ftp` – per trasferirli.

L'`ftp` si usa sia per l'accesso anonimo sia per quello autenticato. L'**accesso anonimo** permette di trasferire file senza essere utenti accreditati nel sistema remoto. Il World Wide Web usa quasi esclusivamente lo scambio di file anonimo. Un DFS comporta un'integrazione molto più stretta tra il calcolatore che accede ai file remoti e il calcolatore che fornisce i file. Tale integrazione incrementa la complessità, illustrata in questo paragrafo.

10.5.2.1 Modello client-server

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 i file si chiama *server*, mentre il calcolatore che richiede l'accesso ai file si chiama *client*. La relazione tra client e server è piuttosto comune tra i calcolatori di una rete. In generale, il server dichiara che determinate risorse sono disponibili ai client, specificando esattamente quali (in questo caso, quali file) ed esattamente a quali client. Un server può gestire richieste provenienti da più client, e un client può accedere a più server, secondo i dettagli del particolare sistema client-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 perché può avvenire facilmente tramite i relativi nomi simbolici di rete, o tramite altri identificatori, si può altrettanto facilmente **ingannare** un server imitando l'identificatore di un client accreditato (si tratta del cosiddetto *spoofing*). Un client non accreditato o privo di determinate autorizzazioni può ingannare un server presentandosi con l'identità di un altro client che possiede le autorizzazioni che il client impostore vuole ottenere. Tra le soluzioni più sicure ci sono quelle che prevedono l'autenticazione reciproca dei client e dei server tramite chiavi di cifratura. Sfortunatamente, l'introduzione di tecniche per la sicurezza introduce nuovi problemi, ad esempio la necessità della compatibilità tra client e server (si devono impiegare gli stessi algoritmi di cifratura) e dello scambio sicuro delle chiavi di cifratura (l'intercettazione delle chiavi può permettere accessi non autorizzati). Questi problemi sono sufficientemente difficili da far sì che nella maggioranza dei casi si usino metodi di autenticazione insicuri.

Nel caso di UNIX e del suo file system di rete (*network file system*, NFS), l'autenticazione avviene, ordinariamente, tramite le informazioni di connessione relative al client. In questo schema, gli identificatori (ID) che identificano l'utente devono coincidere nel client e nel server; diversamente, il server non può determinare i diritti d'accesso ai file. Si consideri ad esempio un utente con un identificatore uguale a 1000 nel client e a 2000 nel server. Una richiesta per uno specifico file dal client al server non potrà essere gestita correttamente, perché il server cercherà di determinare se l'utente 1000 ha i permessi d'accesso al file, invece di usare il *reale* identificatore dell'utente che è 2000. L'accesso sarà concesso o negato secondo un'informazione di autenticazione sbagliata. Il server deve fidarsi del client e assumere che quest'ultimo gli presenti l'identificatore corretto. I protocolli NFS permettono relazioni da molti a molti; cioè più server possono fornire file a più client. Infatti, un calcolatore può comportarsi sia da server per altri client NFS, sia da client di altri server NFS.

Una volta montato il file system remoto, le richieste delle operazioni su file sono inviate al server, attraverso la rete, per conto dell'utente, usando il protocollo DFS. Normalmente, una richiesta di apertura di file si invia insieme con l'identificatore dell'utente richiedente. Il server quindi applica i normali controlli d'accesso per determinare se l'utente ha le credenziali per accedere al file nel modo richiesto; se tali controlli hanno esito positivo, riporta un handle ("maniglia") d'accesso al file (*file handle*) all'applicazione client, che la usa per eseguire sul file operazioni di lettura, scrittura e altro. Il client chiude il file quando non deve più accedervi. Il sistema operativo può applicare una semantica simile a quella adottata per il montaggio di un file system locale, oppure una semantica diversa.

10.5.2.2 Sistemi informativi distribuiti

Per semplificare la gestione dei servizi client-server, i **sistemi informativi distribuiti**, noti anche come **servizi di nominazione distribuiti**, sono stati concepiti per fornire un accesso unificato alle informazioni necessarie per il calcolo remoto. 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 (compreso il World Wide Web). Prima che s'inventasse il DNS e che si diffondesse capillarmente nella rete, si scambiavano tra i calcolatori, per posta elettronica o `ftp`, file contenenti le stesse informazioni. Questo metodo non poteva adattarsi dinamicamente all'aumento delle dimensioni della rete Internet. Il DNS è trattato ulteriormente nel Paragrafo 16.5.1.

Altri sistemi informativi distribuiti forniscono uno spazio identificato da *nome utente*/*parola d'ordine*/*identificatore utente*/*identificatore di gruppo* per un servizio distribuito. I sistemi UNIX hanno adottato un'ampia varietà di metodi per l'informazione distribuita. Sun Microsystems ha introdotto il sistema *yellow pages* (poi ribattezzato *network information*

service, NIS), adottato da gran parte dell'industria. Questo servizio centralizza la memorizzazione dei nomi degli utenti e dei calcolatori, delle informazioni sulle stampanti e altro. Sfortunatamente, usa metodi di autenticazione insicuri, ad esempio l'invio di parole d'ordine dell'utente non cifrate (*in chiaro*) e l'identificazione dei calcolatori attraverso gli indirizzi IP. Il sistema NIS+ di Sun è una versione del NIS più sicura, ma è anche molto più complessa e non ha avuto una gran diffusione.

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 (nome dell'utente e parola d'ordine) 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. Affinché questa autenticazione sia valida, i nomi utente devono essere uguali nel calcolatore client e nel server (come per l'NFS). Per fornire un unico spazio di nomi per gli utenti, Microsoft usa due strutture di nominazione distribuita: i **domìni** costituiscono la vecchia tecnologia di nominazione; la nuova tecnologia, disponibile nei sistemi operativi Windows 2000 e Windows XP, è l'**active directory**. Una volta impostata, la funzione di nominazione è usata per autenticare gli utenti da tutti i client e da tutti i server.

L'industria si sta orientando verso il protocollo LDAP (*lightweight directory-access protocol*) come meccanismo sicuro per la nominazione distribuita. Lo stesso *active directory* è basato sull'LDAP. Sun Microsystems permette l'uso del protocollo LDAP per l'autenticazione degli utenti e per altri servizi di ricerca di informazioni a livello dell'intero sistema, ad esempio tutte le stampanti disponibili. È pensabile che, se la convergenza sull'uso dell'LDAP avrà successo, un'organizzazione potrà usare una singola directory LDAP distribuita per memorizzare le informazioni su tutti gli utenti e le risorse di tutti i calcolatori dell'organizzazione stessa. Si avrebbe un **unico punto d'accesso sicuro** per gli utenti, che inserirebbero una sola volta le proprie informazioni di autenticazione per avere accesso a tutti i calcolatori dell'organizzazione. Questa soluzione semplificherebbe anche i compiti degli amministratori di sistema, concentrando in un unico punto informazioni ora sparse in vari file in ciascun sistema o in diversi servizi di informazione distribuiti.

10.5.2.3 Malfunzionamenti

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 (chiamate collettivamente **metadati**), malfunzionamenti dei controllori dei dischi, problemi ai cavi di connessione o agli adattatori. Anche il comportamento involontario degli utenti o dell'amministratore di sistema può causare la perdita di file, d'intere directory o addirittura la cancellazione di volumi. Molte di queste cause di malfunzionamento portano al crollo del sistema (*crash*), all'emissione di una condizione d'errore e alla necessità di un intervento umano per risolvere il problema.

L'uso di file system remoti implica maggiori possibilità di malfunzionamenti; a causa della complessità dei sistemi di rete e della necessità di interazioni tra calcolatori remoti, i problemi che possono interferire con il corretto funzionamento dei file system remoti sono infatti molto più numerosi. Nel caso delle reti, si possano verificare interruzioni del collegamento tra due calcolatori, dovute a malfunzionamenti o a improprie configurazioni dell'architettura, oppure a questioni relative alla realizzazione degli aspetti di rete in uno dei calcolatori coinvolti. Sebbene alcune reti includano meccanismi di tolleranza ai guasti, compresi cammini multipli tra ogni coppia di calcolatori, altre non li prevedono. Qualsiasi malfunzionamento potrebbe interrompere il flusso dei comandi del DFS.

Si consideri un client mentre usa un file system remoto. Il client ha il file system remoto montato nel proprio file system e potrebbe avere qualche file remoto aperto; tra le va-

rie attività potrebbe richiedere elenchi dei file nelle directory remote per aprire quelli necessari, svolgere operazioni di lettura e scrittura e chiudere i file. Si consideri ora un malfunzionamento della rete, un crollo del server remoto, oppure anche uno spegnimento programmato di quel server, tali da determinare l'inaccessibilità del file system remoto. Questo scenario è piuttosto comune, quindi il client non dovrebbe comportarsi come nel caso di una perdita del file system locale. Piuttosto, il sistema dovrebbe terminare tutte le operazioni sul server non più raggiungibile, oppure posticiparle finché il server sarà nuovamente disponibile. Questa semantica di trattamento dei malfunzionamenti si definisce e si realizza come parte del protocollo di un file system remoto. La terminazione di tutte le operazioni può portare alla perdita di dati (e della pazienza) da parte degli utenti. La maggior parte dei protocolli DFS impone o permette la posticipazione delle operazioni sul file system remoto, con la speranza che il calcolatore remoto diventi nuovamente disponibile in breve tempo.

Per realizzare questo tipo di recupero dai malfunzionamenti è necessario mantenere alcune **informazioni di stato** sia sui client sia sui server. Se sia i client sia i server 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**. Sostanzialmente, assume che una richiesta di un client per la lettura o scrittura di un file non sia avvenuta, sempre che il file system non sia stato montato in modo remoto e il file in questione aperto prima della richiesta. Il protocollo NFS trasferisce tutte le informazioni necessarie per localizzare il file appropriato e per svolgere l'operazione richiesta sul file. Allo stesso modo, non tiene traccia di quali client abbiano montato i propri volumi esportati, assumendo anche in questo caso che, se perviene una richiesta, debba essere legittima. Sebbene questo metodo senza stato renda l'NFS tollerante ai guasti e sia piuttosto facile da realizzare, lo rende insicuro. Un server NFS potrebbe ad esempio permettere richieste contraffatte di lettura o scrittura da un client non autorizzato anche se la necessaria operazione di montaggio e il controllo dei permessi richiesti non sono avvenuti. Questioni del genere sono regolamentate dallo standard industriale NFS versione 4, in cui NFS è dotato di stato per migliorare le proprie funzionalità, prestazioni e sicurezza.

10.5.3 Semantica della coerenza

La **semantica della coerenza** è un importante criterio per la valutazione di qualsiasi file system che consenta la condivisione dei file. Si tratta di una caratterizzazione del sistema che specifica la semantica delle operazioni in cui più utenti accedono contemporaneamente a un file condiviso. In particolare, questa semantica deve specificare quando le modifiche ai dati apportate da un utente possano essere osservate da altri utenti. La semantica è tipicamente realizzata come codice facente parte del codice del file system.

La semantica della coerenza è direttamente correlata agli algoritmi di sincronizzazione dei processi del Capitolo 6. Tuttavia, a causa delle lunghe latenze e delle basse velocità di trasferimento dei dischi e delle reti, i complessi algoritmi descritti in tale capitolo di solito non s'impiegano per l'I/O su file. Ad esempio, l'esecuzione di una transazione atomica su dischi remoti può coinvolgere molte comunicazioni di rete e molte letture e scritture nei dischi. I sistemi che tentano una così completa serie di funzioni tendono ad avere scarse prestazioni. Una realizzazione riuscita di semantica della condivisione si trova nel file system del sistema Andrew.

Nella trattazione seguente si suppone che una serie d'accessi, cioè letture e scritture, tentati da un utente allo stesso file sia sempre compresa tra una coppia di operazioni `open()` e `close()`. Tale serie d'accessi si chiama **sessione di file.** Per illustrare questo concetto si descrivono sommariamente qui di seguito alcuni esempi di semantica della coerenza.

10.5.3.1 Semantica UNIX

Il file system di UNIX, descritto nel Capitolo 17, usa la seguente semantica della coerenza.

- Le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno aperto contemporaneamente lo stesso file.
- Esiste un metodo di condivisione in cui gli utenti condividono il puntatore alla locazione corrente nel file, quindi l'avanzamento del puntatore da parte di un utente influisce su tutti gli utenti che condividono il file. In questo caso il file ha una singola immagine e tutti gli accessi si alternano (intercalandosi) a prescindere dalla loro origine.

Nella semantica UNIX un file è associato a una singola immagine fisica, accessibile come una risorsa esclusiva. La contesa di quest'immagine singola determina il differimento dei processi utenti.

10.5.3.2 Semantica delle sessioni

Il file system Andrew (*Andrew file system*, AFS), descritto nel Capitolo 17, usa la seguente semantica della coerenza:

- le scritture in un file aperto da un utente non sono visibili immediatamente ad altri utenti che hanno aperto contemporaneamente lo stesso file;
- una volta chiuso il file, le modifiche apportate sono visibili solo nelle sessioni che iniziano successivamente. Le istanze del file già aperte non riportano queste modifiche.

Secondo questa semantica, un file può essere temporaneamente associato a parecchie immagini, probabilmente diverse. Di conseguenza, più utenti possono eseguire accessi concorrenti di lettura o scrittura sulla rispettiva immagine del file senza subire ritardi. Non si impone quasi alcun vincolo nella gestione degli accessi.

10.5.3.3 Semantica dei file condivisi immutabili

Un altro metodo è quello dei **file condivisi immutabili**; una volta che un file è stato dichiarato *condiviso* dal suo creatore, non può essere modificato. Un file immutabile presenta due caratteristiche chiave: il suo nome non si può riutilizzare e il suo contenuto non può essere modificato. Quindi il nome di un file immutabile indica che i contenuti di quel file sono fissi. Come si descrive nel Capitolo 17, la realizzazione di questa semantica in un sistema distribuito è semplice; infatti la condivisione è molto disciplinata poiché consente la sola lettura.

10.6 Protezione

La salvaguardia delle informazioni contenute in un sistema di calcolo dai danni fisici (la questione della *affidabilità*) e da accessi impropri (la questione della *protezione*) è fondamentale.

Generalmente l'affidabilità è assicurata da più copie dei file. Molti calcolatori hanno programmi di sistema che copiano i file dai dischi ai nastri a intervalli regolari, ad esempio una volta al giorno, alla settimana o al mese; quest'operazione di copiatura può essere automatica, o controllata dall'intervento di un operatore. Lo scopo è quello di conservare copie di riserva utili nei casi in cui il file system andasse distrutto a causa di problemi dei dispositivi: errori di lettura o scrittura, cali o cadute della tensione elettrica, rotture delle testine, sporcizia, temperature estreme, atti vandalici, ecc. I file possono inoltre essere cancellati ac-

cidentalmente, e anche errori di programmazione possono causare la perdita del contenuto dei file. L'affidabilità è trattata con maggiori dettagli nel Capitolo 12.

La protezione si può ottenere in molti modi. Per un piccolo sistema monoutente, la protezione si ottiene rimovendo fisicamente i dischetti e chiudendoli in un cassetto della scrivania oppure in un armadietto. In un sistema multiutente sono necessari altri metodi.

10.6.1 Tipi d'accesso

La necessità di proteggere i file deriva direttamente dalla possibilità di accedervi. I sistemi che non permettono l'accesso ai file di altri utenti non richiedono protezione; quindi si può ottenere una completa protezione proibendo l'accesso. In alternativa si può permettere un accesso totalmente libero senza alcuna protezione. Questi orientamenti sono entrambi eccessivi, quindi non si possono applicare in generale; ciò che serve in realtà è un **accesso controllato.**

Il controllo offerto dai meccanismi di protezione si ottiene limitando i possibili tipi d'accesso. Gli accessi si permettono o si negano secondo diversi fattori, innanzitutto i tipi d'accesso richiesti. Si possono controllare distinte operazioni.

- **Lettura.** Lettura da file.
- **Scrittura.** Scrittura o riscrittura di file.
- **Esecuzione.** Caricamento di file in memoria ed esecuzione.
- **Aggiunta.** Scrittura di nuove informazioni in coda ai file.
- **Cancellazione.** Cancellazione di file e liberazione del relativo spazio per un possibile riutilizzo.
- **Elencazione.** Elencazione del nome e degli attributi dei file.

Si possono controllare anche altre operazioni, come ridenominazione, copiatura o modifica dei file. Tuttavia, in molti sistemi queste funzioni di livello superiore si possono realizzare tramite un programma di sistema che compie alcune chiamate di sistema di livello inferiore, quindi è sufficiente garantire la protezione a livello inferiore. Ad esempio, la copiatura di un file si può realizzare semplicemente con una sequenza di richieste di lettura; in questo caso un utente con accesso per la lettura di un file può richiederne la copiatura, la stampa o altro.

Sono stati proposti molti meccanismi di protezione. Come sempre, ogni meccanismo presenta vantaggi e svantaggi, e deve essere appropriato alla particolare applicazione che richiede la protezione. Un piccolo calcolatore usato soltanto da pochi membri di un gruppo di ricerca non richiede la stessa protezione del sistema di calcolo di una grande società, usato per operazioni di ricerca, finanza e per il lavoro del personale. Il problema della protezione è trattato nel Capitolo 14.

10.6.2 Controllo degli accessi

Il problema della protezione comunemente si affronta rendendo l'accesso dipendente dall'identità dell'utente. Più utenti possono richiedere diversi tipi d'accesso a un file o a una directory. 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 e directory; in tale lista sono specificati i nomi degli utenti e i relativi tipi d'accesso consentiti. Quando un utente richiede un accesso a un file specifico il sistema operativo esamina la lista di controllo degli accessi associata a quel file; se tale utente è presente nella lista si autorizza l'accesso, altrimenti si verifica una violazione della protezione e si nega l'accesso al file.

Questo sistema ha il vantaggio di permettere complessi metodi d'accesso. Il problema maggiore delle liste di controllo degli accessi è la loro lunghezza: per permettere a tutti di leggere un file, la lista deve contenere tutti gli utenti con accesso per la lettura. Questa tecnica comporta due inconvenienti:

- la costruzione di una lista di questo tipo può essere un compito noioso e non gratificante, soprattutto se la lista degli utenti del sistema non è già nota;
- l'elemento della directory, precedentemente di dimensione fissa, richiede di essere di dimensione variabile, quindi anche la gestione dello spazio è più complicata.

Questi problemi si possono risolvere introducendo una versione condensata della lista di controllo degli accessi.

Per condensarne la lunghezza, molti sistemi raggruppano gli utenti di ogni file in tre classi distinte.

- **Proprietario**. È l'utente che ha creato il file.
- **Gruppo**. Si tratta di un insieme di utenti che condividono il file e richiedono tipi di accesso simili.
- **Universo**. Tutti gli altri utenti del sistema.

Il più comune orientamento recente prevede la combinazione delle liste di controllo degli accessi con lo schema di controllo degli accessi per proprietario, gruppo e universo (più facile da realizzare). Il sistema operativo Solaris 2.6 (e le sue versioni successive) impiega le tre categorie d'accesso in modo predefinito ma, se si vuole una maggiore selettività del controllo degli accessi, permette l'attribuzione di liste di controllo degli accessi a specifici file e directory.

Si consideri, ad esempio, una persona, Donatella, che sta scrivendo un nuovo libro. Donatella ha assunto tre studenti, Giulia, Paolo e Carlo, per aiutarla a lavorare al progetto. Il testo del libro è tenuto in un file chiamato `libro`. La protezione associata a tale file prevede quel che segue:

- Donatella può compiere tutte le operazioni sul file;
- Giulia, Paolo e Carlo possono solo leggere e scrivere il file ma non possono cancellarlo;
- tutti gli altri utenti possono leggere, ma non scrivere, il file (Donatella ha interesse che il libro sia letto dal maggior numero possibile di persone, in modo da ottenere pareri competenti).

Per ottenere tale protezione si deve creare un nuovo gruppo composto da Giulia, Paolo e Carlo, e si deve associare il nome del gruppo, ad esempio `testo`, al file `libro` con i diritti d'accesso conformi al criterio descritto.

Si consideri un ospite cui Donatella vorrebbe concedere un accesso temporaneo al Capitolo 1. L'ospite non si può aggregare al gruppo `testo` poiché ciò gli darebbe accesso a tutti i capitoli, e considerato che i file possono essere in un solo gruppo, non si può associare un altro gruppo al Capitolo 1. L'aggiunta della funzione delle liste di controllo degli accessi permette di inserire l'ospite nella lista di controllo degli accessi del Capitolo 1.

Affinché questo schema funzioni correttamente, è necessario uno stretto controllo dei permessi e delle liste di controllo degli accessi, fattibile in diversi modi. Nel sistema UNIX ad esempio solo un utente con compiti di gestione (o un *superuser*) può creare e modificare i gruppi, quindi questo controllo si ottiene con la partecipazione umana. Nel sistema VMS, il proprietario del file può creare e modificare tale lista. Le liste di controllo degli accessi sono trattate anche nel Paragrafo 14.5.2.

Per definire la protezione, data questa più limitata classificazione, occorrono solo tre campi. Ogni campo è formato di un insieme di bit, ciascuno dei quali permette o impedisce l'accesso che gli è associato. Nel sistema UNIX, ad esempio, sono definiti tre campi di tre bit ciascuno: `rwx`, dove `r` controlla l'accesso per la lettura, `w` quello per la scrittura e `x` per l'esecuzione. Un campo distinto è riservato al proprietario del file, al gruppo proprietario e a tutti gli altri utenti. In questo schema, per registrare le informazioni di protezione sono necessari nove bit per file. Così, nell'esempio, i campi di protezione per il file `libro` s'impostano come segue: per Donatella, il proprietario, tutti e tre i bit; per il gruppo `testo` i bit `r` e `w`; per l'universo il solo bit `r`.

Nel combinare i due metodi si presenta una difficoltà nell'interfaccia utente; gli utenti devono poter indicare l'impostazione opzionale dei permessi nella lista di controllo degli accessi per un file. Nell'esempio di Solaris, un segno "+" aggiunge un permesso d'accesso:

```
19 -rw--r--r--+ 1 alessandra gruppo 130 May 25 22:13 file1
```

Una specifica serie di comandi `setfacl` e `getfacl` si usa per gestire le liste di controllo degli accessi.

Gli utenti di Windows XP, in genere, gestiscono le liste di controllo degli accessi tramite un'interfaccia grafica. La Figura 10.15 mostra la finestra dei permessi di un file del file

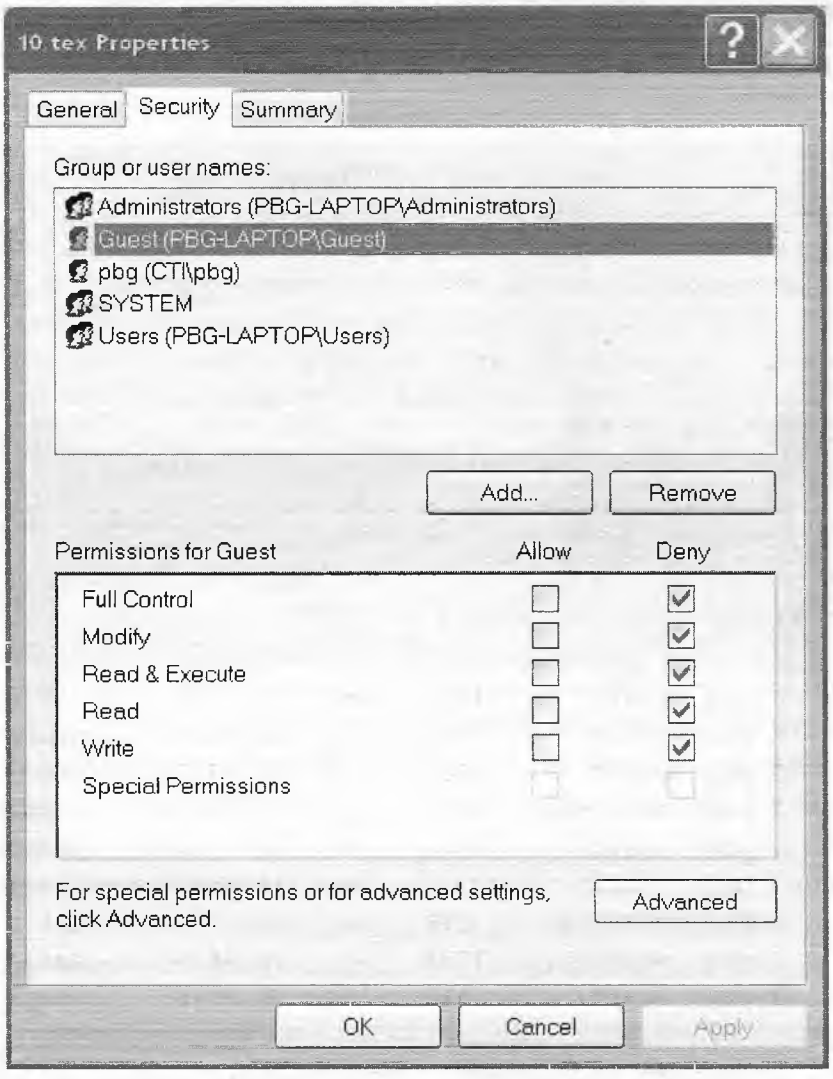


Figura 10.15 Gestione della lista di controllo degli accessi in Windows XP

system NTFS di Windows XP. In questo esempio si vieta esplicitamente all'utente Guest l'accesso al file `10.tex`.

Un'altra difficoltà s'incontra nell'assegnazione delle precedenze quando ci sono conflitti tra i permessi e liste di controllo degli accessi. Ad esempio, se Andrea è in un gruppo di file, che ha il permesso di lettura, ma il file ha un lista di controllo degli accessi contenente i permessi di lettura e scrittura per Andrea, si pone il problema della concessione del permesso di scrittura. Nel sistema operativo Solaris si attribuiscono i permessi contenuti nelle liste di controllo degli accessi; sono più selettivi e non sono predefiniti. Si segue il principio che la maggiore specificità deve essere prioritaria.

10.6.3 Altri metodi di protezione

Un altro metodo di protezione consiste nell'associazione di una parola d'ordine (*password*) a ciascun file. Proprio come l'accesso al sistema di calcolo è spesso controllato da una parola d'ordine, anche l'accesso a ogni file può avere lo stesso tipo di protezione. Se le parole d'ordine sono scelte a caso e si cambiano spesso, questo schema può essere efficace, poiché limita l'accesso a un file agli utenti che conoscono la parola d'ordine. Questo presenta ha tuttavia diversi svantaggi: innanzitutto, il numero di parole d'ordine da ricordare può diventare molto alto, rendendo tale metodo impraticabile; secondariamente, se si impiega la stessa parola d'ordine per tutti i file, la sua scoperta li rende tutti accessibili. La protezione è basata sul principio del "o tutto o niente". Per risolvere questo problema alcuni sistemi, ad esempio il TOPS-20, permettono a un utente di associare una parola d'ordine a una directory anziché

PERMESSI IN UN SISTEMA UNIX

Nel sistema UNIX la protezione delle directory è gestita in modo simile alla protezione dei file. In altre parole, a ciascuna directory sono associati tre campi (proprietario, gruppo e universo), ciascuno composto dei tre bit `rwx`. Quindi, un utente può elencare il contenuto di una directory solamente se il bit `r` è inserito nel campo appropriato. Analogamente, un utente può modificare la propria directory corrente in un'altra directory (ad esempio `foo`) solo se è associato, nel campo appropriato, il bit di `foo`.

Nella Figura 10.16 è illustrato un esempio di elenco del contenuto di una directory nell'ambiente UNIX. Il primo campo descrive le protezioni di file e directory, il carattere `d` presente all'inizio del campo contraddistingue le directory; inoltre, l'elenco contiene il numero di collegamenti relativi al file, il nome del proprietario e del gruppo, la dimensione del file in byte, la data di creazione e infine il nome del file (con l'eventuale estensione).

```
-rw-rw-r--   1 pbg   staff      31200   set   3  08:30    intro.ps
drwx------   5 pbg   staff        512   lug   8  09:33    privato/
drwxrwxr-x   2 pbg   staff        512   lug   8  09:35    doc/
drwxrwx---   2 pbg   studente     512   ago   3  14:13    studente-prog/
-rw-r--r--   1 pbg   staff       9423   feb  24  2003     program.c
-rwxr-xr-x   1 pbg   staff      20471   feb  24  2003     program
drwx--x--x   4 pbg   facoltà      512   lug  31  10:31    lib/
drwx------   3 pbg   staff       1024   ago  29  06:52    mail/
drwxrwxrwx   3 pbg   staff        512   lug   8  09:35    test/
```

Figura 10.16 Esempio di elenco del contenuto di una directory.

a un singolo file. Il sistema operativo IBM VM/CMS consente di associare tre parole d'ordine ai minidischi: una per la lettura, una per la scrittura, e una per gli accessi multipli per la scrittura.

Alcuni sistemi operativi monoutente come MS-DOS e le versioni del sistema Macintosh prima di Mac OS X, non offrono grande protezione dei file. Nel caso in cui tali sistemi più antiquati siano posti in reti dove sia necessario condividere file, occorre aggiungere loro dei meccanismi di protezione. Progettare una caratteristica di un nuovo sistema operativo è quasi sempre più facile che aggiungerla a uno esistente. Inoltre, tali aggiornamenti sono di solito meno efficaci e si integrano meno bene nel sistema.

In una struttura della directory a più livelli è necessario proteggere non solo i singoli file, ma anche gruppi di file contenuti in directory; è cioè necessario disporre di un meccanismo per la protezione delle directory. Le operazioni riguardanti le directory da proteggere sono piuttosto diverse dalle operazioni sui file. Esse sono la creazione e la cancellazione dei file in una directory; probabilmente va anche controllata la possibilità, per un utente, di determinare l'esistenza di un file in una directory. Talvolta la conoscenza dell'esistenza di un file e del suo nome può essere di per sé significativa, perciò l'elencazione del contenuto di una directory dev'essere un'operazione protetta. Quindi, affinché un nome di percorso possa far riferimento a un file in una certa directory, all'utente deve essere consentito l'accesso sia al file sia alla directory. Nei sistemi dove i file possono avere numerosi nomi di percorso (come quelli con struttura a grafo aciclico o a grafo generale) un certo utente può avere diversi diritti d'accesso secondo il nome di percorso di cui fa uso.

10.7 Sommario

Un file è un tipo di dati astratto definito e realizzato dal sistema operativo. È una sequenza di elementi logici (o *record*), ciascuno dei quali può essere un byte, una riga di lunghezza fissa o variabile, oppure un elemento di dati più complesso. Il sistema operativo può gestire in modo specifico diversi tipi di elementi logici o può lasciare tale gestione al programma applicativo.

Il compito più importante del sistema operativo consiste nell'associare i file ai dispositivi fisici di memorizzazione, ad esempio dischi o nastri magnetici. Poiché le dimensioni dei blocchi fisici dei dispositivi possono non coincidere con quelle dei record logici, può essere necessario riunire un certo numero di record logici in modo da ottenere una dimensione pari a quella di un blocco fisico. Anche questo compito può essere gestito dal sistema operativo, oppure lasciato al programma applicativo.

Ciascun dispositivo di un file system conserva una tabella dei contenuti del volume o una directory di dispositivo che elenca la locazione dei file presenti nel dispositivo. Inoltre, è utile creare directory per permettere di organizzare i file. Una directory a livello singolo in un sistema multiutente causa problemi di nominazione, poiché ogni file deve avere un nome unico. Una directory a due livelli limita questo problema all'ambito relativo a ciascun utente creando una directory distinta per ciascun utente, che dispone di una directory contenente i suoi file. Le directory contengono la lista dei file che le costituiscono, insieme con informazioni a essi associate: locazione nei dischi, lunghezza, tipo, ora di creazione, ora dell'ultimo uso, e così via.

La naturale generalizzazione del concetto di directory a due livelli è la directory con struttura ad albero. Tale tipo di struttura permette a un utente di creare sottodirectory in cui organizzare i file. Le strutture delle directory a grafo aciclico permettono la condivisione di sottodirectory e file, ma complicano le funzioni di ricerca e cancellazione. Una struttura a

grafo generale permette la massima flessibilità nella condivisione dei file e delle directory, ma talvolta richiede operazioni di "ripulitura" (*garbage collection*) per recuperare lo spazio inutilizzato nei dischi.

I dischi sono suddivisi in uno o più volumi, ciascuno contenente un file system o privo di struttura. Per essere resi disponibili, i file system si possono montare nelle strutture di nominazione del sistema. Lo schema di nominazione varia da sistema a sistema. Una volta montati, i file all'interno del volume sono disponibili per l'uso. I file system si possono smontare per disabilitarne l'accesso o per attività di manutenzione.

La condivisione dei file dipende dalla semantica definita dal sistema. I file possono avere più lettori, più scrittori, o limiti alla condivisione. I file system distribuiti consentono ai sistemi client di montare volumi o directory di server, man mano che vi accedono tramite una rete. I file system remoti pongono delle sfide in termini di affidabilità, prestazioni e sicurezza. I sistemi informativi distribuiti mantengono informazioni su utenti, calcolatori e accessi in modo che i client e i server condividano le informazioni di stato per la gestione dell'uso e degli accessi.

Poiché i file rappresentano il principale meccanismo di memorizzazione delle informazioni, è necessario che siano dotati di un sistema di protezione. Ogni tipo d'accesso ai file si può controllare separatamente: lettura, scrittura, esecuzione, aggiunta, elencazione del contenuto di directory, e così via. La protezione dei file si può ottenere con parole d'ordine, liste d'accesso, oppure tecniche appositamente predisposte per le situazioni specifiche.

Esercizi pratici

10.1 Alcuni sistemi cancellano automaticamente tutti i file utente quando un utente si disconnette oppure quando un lavoro termina, a meno che l'utente non richieda esplicitamente che questi vengano conservati; altri sistemi conservano tutti i file a meno che l'utente non li cancelli esplicitamente. Discutete i meriti di ciascun approccio.

10.2 Perché alcuni sistemi tengono traccia del tipo di un file, mentre altri lasciano questo compito all'utente e altri semplicemente non implementano tipi di file multipli? Quale sistema è "migliore"?

10.3 In modo simile, alcuni sistemi mettono a disposizione molte tipologie diverse di strutture per i dati di un file, mentre altri trattano solo un semplice flusso di byte. Quali sono i vantaggi e gli svantaggi di tali approcci?

10.4 È possibile simulare una struttura di directory multilivello con una struttura di directory a livello singolo nella quale possono essere usati nomi arbitrariamente lunghi? Se la risposta è affermativa, spiegate come ciò sia fattibile e confrontate questo schema con lo schema a directory multilivello. In caso di risposta negativa, spiegate che cosa impedisce il successo della simulazione. Come cambierebbe la vostra risposta se la lunghezza del nome dei file fosse limitata a sette caratteri?

10.5 Spiegate lo scopo delle operazioni `open()` e `close()`.

10.6 Fornite un esempio di applicazione nella quale si debba accedere ai dati in un file nell'ordine seguente:

a. sequenziale;

b. casuale.

10.7 In alcuni sistemi una sottodirectory può essere letta e scritta da un utente autorizzato, proprio come i file ordinari.

a. Descrivete i problemi di protezione che ne potrebbero derivare.

b. Suggerite uno schema per trattare ognuno di questi problemi di protezione.

10.8 Considerate un sistema che supporti 5.000 utenti. Supponete di voler permettere a 4.990 utenti di accedere a un dato file.

a. Come specifichereste questo schema di protezione in UNIX?

b. Potete suggerire un altro schema di protezione utilizzabile a questo scopo più efficacemente dello schema fornito da UNIX?

10.9 Alcuni ricercatori hanno suggerito che, invece di associare una lista di accesso a ogni file (dove la lista specifica quali utenti possono accedere al file e come), dovremmo avere *una lista di controllo degli utenti* associata a ogni utente (dove la lista specifica a quali file un utente può accedere e come). Discutete i meriti relativi a questi due schemi.

Esercizi

10.10 Considerate un file system in cui si può cancellare un file e reclamare il suo spazio di memoria secondaria mentre esistono ancora collegamenti (*link*) a esso. Dite quale problema si può presentare se si crea un nuovo file nella stessa area di memoria o con lo stesso nome di percorso assoluto. Spiegate come tali problemi siano evitabili.

10.11 La tabella dei file aperti registra le informazioni riguardanti i file aperti in quel momento. Il sistema operativo dovrebbe mantenere una tabella separata per ciascun utente oppure mantenere una tabella unica per tutti gli utenti con le indicazioni sui file a cui accedono in un certo momento? Se due diversi programmi o utenti eseguono accessi al medesimo file, questi dovrebbero comparire come accessi separati nella tabella dei file aperti?

10.12 Quali sono vantaggi e svantaggi di un sistema che fornisce lock obbligatori anziché lock consigliati, il cui utilizzo è rimesso al giudizio degli utenti?

10.13 Spiegate quali sono vantaggi e svantaggi della registrazione, tra gli attributi di un file, del nome del programma che crea il file stesso, come avviene nel sistema operativo Macintosh.

10.14 Alcuni sistemi aprono un file automaticamente quando ci si riferisce a esso per la prima volta, e lo chiudono al termine del lavoro. Illustrate vantaggi e svantaggi di questo schema, confrontandolo con quello tradizionale, in cui l'utente deve aprire e chiudere il file esplicitamente.

10.15 Qualora il sistema operativo dovesse apprendere che una certa applicazione accederà ai dati di un file in modo sequenziale, come potrebbe sfruttare questa informazione per migliorare le prestazioni?

10.16 Illustrate un'applicazione che potrebbe trarre vantaggio da un sistema operativo che offra l'accesso casuale ai file indicizzati.

10.17 Analizzate vantaggi e svantaggi di permettere collegamenti a file che oltrepassano i punti di montaggio (vale a dire, collegamenti che rimandano a file memorizzati in un volume differente).

10.18 Alcuni sistemi consentono la condivisione dei file usando una singola copia di ogni file; altri sistemi impiegano più copie, una per ciascun utente che condivide il file. Discutete i vantaggi di ciascun metodo.

10.19 Considerate vantaggi e svantaggi insiti nell'associare a file system remoti (memorizzati su file server) una semantica del fallimento diversa da quella prevista nei file system locali.

10.20 Quali sono le implicazioni derivanti dall'abbinare a file residenti su file system remoti la semantica della coerenza di UNIX per l'accesso condiviso?

10.8 Note bibliografiche

Analisi generali sui file system sono reperibili in [Grosshans 1986]; [Golden e Pechura 1986] descrive la struttura dei file system dei microcalcolatori. I sistemi di gestione delle basi di dati e le loro strutture di file sono ampiamente descritti in [Korth e Silberschatz 2001].

La struttura delle directory a più livelli è stata realizzata per la prima volta nel sistema MULTICS [Organick 1972]. Attualmente la maggior parte dei sistemi dispone di una struttura delle directory a più livelli, ad esempio Linux, [Bovet e Cesati 2002], Mac OS X (http://www.apple.com/macosx/), Solaris [McDougal e Mauro 2007] e tutte le versioni di Windows, incluso Windows 2000 [Russinovich e Solomon 2005].

Il Network File System (NFS), progettato dalla Sun Microsystems, permette di distribuire le strutture delle directory sui calcolatori di una rete. NFS è approfondito nei dettagli nel Capitolo 17. La versione 4 di NFS è descritta in RFC3505 (http://www.ietf.org/rfc/rfc3530.txt). Una trattazione generale sui file system in Solaris si può trovare nella guida di Sun *System Administration Guide: Devices and File Systems* (http://docs.sun.com/app/docs/doc/817-5093).

Inizialmente proposto da [Su 1982], DNS è passato attraverso diverse revisioni; [Mockapetris 1987] lo ha esteso con importanti funzionalità. Più recentemente, [Eastlake 1999] ha proposto estensioni relative alla sicurezza che permettano a DNS di detenere delle chiavi di sicurezza.

Il protocollo LDAP, noto anche come X.509, è un sottoinsieme del protocollo di directory distribuita X.500. È stato definito da [Yeong et al. 1995] e incorporato in molti sistemi operativi.

Sono in corso interessanti ricerche nell'area delle interfacce dei file system, in particolare per i problemi relativi alla nominazione dei file e ai loro attributi. Ad esempio, il sistema operativo Plan 9 dei Bell Laboratories (Lucent Technology) rende tutti gli oggetti simili a file system. In tal modo, per ottenere l'elenco dei processi nel sistema, un utente deve semplicemente leggere il contenuto della directory `/proc`; per veder che ore sono si deve leggere il file `/dev/time`.