Capitolo 11

Realizzazione del file system

OBIETTIVI

- **Descrizione dei dettagli realizzativi di file system locali e strutture di directory.**
- **Analisi della realizzazione di file system remoti.**
- **Esame dell'allocazione dei blocchi e dei pro e contro degli algoritmi per la gestione dello spazio libero.**

Come illustrato nel Capitolo 10, il file system fornisce il meccanismo per la memorizzazione e l'accesso al contenuto dei file, compresi dati e programmi. Il file system risiede permanentemente nella *memoria secondaria*, progettata per contenere in modo permanente grandi quantità di dati. Questo capitolo riguarda principalmente i problemi connessi alla memorizzazione e all'accesso ai file nel più comune mezzo di memoria secondaria, il disco. Si esaminano varie modalità d'uso dei file, l'allocazione dello spazio dei dischi, il recupero dello spazio liberato, la registrazione delle locazioni dei dati, e l'interfaccia di altri componenti del sistema operativo alla memoria secondaria. Nel corso della trattazione si considerano anche i problemi riguardanti le prestazioni.

11.1 Struttura del file system

I dischi costituiscono la maggior parte della memoria secondaria in cui si conserva il file system. Hanno due caratteristiche importanti che ne fanno un mezzo conveniente per la memorizzazione dei file:

1. si possono riscrivere localmente; si può leggere un blocco dal disco, modificarlo e quindi scriverlo nella stessa posizione;
2. è possibile accedere direttamente a qualsiasi blocco di informazioni del disco, quindi risulta semplice accedere a qualsiasi file, sia in modo sequenziale sia in modo diretto, e passare da un file all'altro spostando le testine di lettura e scrittura e attendendo la rotazione del disco.

La struttura dei dischi è analizzata in modo particolareggiato nel Capitolo 12.

Anziché trasferire un byte alla volta, per migliorare l'efficienza dell'I/O, i trasferimenti tra memoria centrale e dischi si eseguono per *blocchi*. Ciascun blocco è composto da uno o più settori. Secondo l'unità a disco, la dimensione dei settori è compresa tra 32 byte e 4096 byte; di solito è pari a 512 byte.

Per fornire un efficiente e conveniente accesso al disco, il sistema operativo fa uso di uno o più **file system** che consentono di memorizzare, individuare e recuperare facilmente i dati. Un file system presenta due problemi di progettazione piuttosto diversi. Il primo riguarda la definizione dell'aspetto del file system agli occhi dell'utente. Questo compito implica la definizione di un file e dei suoi attributi, delle operazioni permesse su un file e della struttura delle directory per l'organizzazione dei file. Il secondo riguarda la creazione di algoritmi e strutture dati che permettano di far corrispondere il file system logico ai dispositivi fisici di memoria secondaria.

Lo stesso file system è generalmente composto da molti livelli distinti. La struttura illustrata nella Figura 11.1 è un esempio di struttura stratificata. Ogni livello si serve delle funzioni dei livelli inferiori per crearne di nuove impiegate dai livelli superiori.

Il livello più basso, il *controllo dell'I/O*, costituito dai **driver dei dispositivi** e dai gestori dei segnali d'interruzione, si occupa del trasferimento delle informazioni tra memoria centrale e memoria secondaria. Un driver di dispositivo si può concepire come un traduttore che riceva comandi ad alto livello, come "recupera il blocco 123", e che emette specifiche istruzioni di basso livello per i dispositivi, usate dal controllore che fa da interfaccia tra i dispositivi di I/O e il resto del sistema. Un driver di dispositivo di solito scrive specifiche configurazioni di bit in specifiche locazioni della memoria del controllore di I/O per indicare quali azioni il dispositivo di I/O debba compiere, e in quali locazioni. I dettagli dei driver dei dispositivi e le strutture per l'I/O sono trattati nel Capitolo 13.

Il **file system di base** deve soltanto inviare dei generici comandi all'appropriato driver di dispositivo per leggere e scrivere blocchi fisici nel disco. Ogni blocco fisico si identifica col suo indirizzo numerico nel disco, ad esempio unità 1, cilindro 73, superficie 2, settore 10. Questo strato gestisce inoltre buffer di memoria e le cache che conservano vari blocchi dei file system, delle directory e dei dati. Un blocco viene allocato nel buffer prima che possa verificarsi il trasferimento di un blocco del disco. Quando il buffer è pieno, il gestore del buffer deve recuperare più spazio di memoria per il buffer oppure deve liberare spazio nel buffer per permettere il completamento di un I/O richiesto. Le cache servono a conservare metadati di file system usati frequentemente, in modo da migliorare le prestazioni. La gestione dei loro contenuti è quindi un punto critico per conseguire prestazioni ottimali del sistema.

Il **modulo di organizzazione dei file** è a conoscenza dei file e dei loro blocchi logici, così come dei blocchi fisici dei dischi. Conoscendo il tipo di allocazione dei file usato e la locazione dei file, può tradurre gli indirizzi dei blocchi logici, che il file system di base deve trasferire, negli indirizzi dei blocchi fisici. I blocchi logici di ciascun file sono numerati da 0 (o 1) a *n*; i

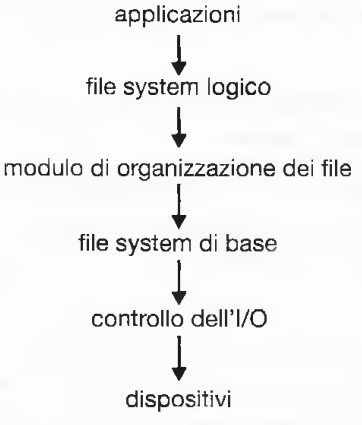


Figura 11.1 File system stratificato.

blocchi fisici contenenti tali dati di solito non corrispondono ai numeri dei blocchi logici; per individuare ciascun blocco è quindi necessaria una traduzione. Il modulo di organizzazione dei file comprende anche il gestore dello spazio libero, che registra i blocchi non assegnati e li mette a disposizione del modulo di organizzazione dei file quando sono richiesti.

Infine, il **file system logico** gestisce i metadati; si tratta di tutte le strutture del file system, eccetto gli effettivi dati (il contenuto dei file). Il file system logico gestisce la struttura della directory per fornire al modulo di organizzazione dei file le informazioni di cui necessita, dato un nome simbolico di file. Mantiene le strutture di file tramite i **blocchi di controllo dei file** (*file control block*, FCB), contenenti informazioni sui file, come la proprietà, i permessi, e la posizione del contenuto. Come si discute nel Capitolo 10 e Capitolo 14, il file system logico è responsabile anche della protezione e della sicurezza.

Nei file system stratificati la duplicazione di codice è ridotta al minimo. Il controllo dell'I/O e, talvolta, il codice di base del file system, possono essere comuni a numerosi file system, che poi gestiscono il file system logico e i moduli per l'organizzazione dei file secondo le proprie esigenze. Sfortunatamente, la stratificazione può comportare un maggior overhead del sistema operativo, che può generare un conseguente decadimento delle prestazioni. L'utilizzo della stratificazione e le scelte sul numero di strati da impiegare e sulle loro funzionalità rappresentano una grande sfida per la progettazione di nuovi sistemi.

Esistono svariati tipi di file system al giorno d'oggi, e non è raro che i sistemi operativi ne prevedano più d'uno. Molti CD-ROM, a esempio, sono scritti nel formato ISO 9660, uno standard concordato dai produttori di CD-ROM. Oltre ai file system dei supporti rimovibili, ciascun sistema operativo possiede un file system, o più di uno, basato sui dischi. UNIX adotta il **File System UNIX** (UFS), che si fonda a sua volta sul File System Berkeley Fast (FFS). Windows NT, 2000 e XP adottano i formati FAT, FAT32 e NTFS (o File System di Windows NT), così come i formati per CD-ROM, DVD e dischetti. Sebbene Linux possa funzionare con più di quaranta file system diversi, quello standard è noto come **file system esteso**, le cui versioni maggiormente diffuse sono ext2 ed ext3. Esistono anche file system distribuiti, ossia tali che un file system del server è montato da uno o più client in una rete.

La ricerca relativa ai file system continua a essere un'area attiva della progettazione e dell'implementazione dei sistemi operativi. Google ha progettato un proprio file system per soddisfare esigenze di memorizzazione e recupero dati specifiche dell'azienda. Un altro progetto interessante è il file system FUSE, che garantisce flessibilità nell'utilizzo del sistema grazie all'implementazione e all'esecuzione di file system a livello utente invece che a livello del codice del kernel. Gli utenti di FUSE possono aggiungere nuovi file system a numerosi sistemi operativi e utilizzarli per gestire i propri file.

11.2 Realizzazione del file system

Per permettere ai processi di richiedere l'accesso al contenuto dei file, i sistemi operativi offrono le chiamate di sistema `open()` e `close()`. In questo paragrafo si approfondiscono le strutture dati e le operazioni usate per realizzare le funzioni del file system.

11.2.1 Introduzione

Per realizzare un file system si usano parecchie strutture dati, sia nei dischi sia in memoria. Queste strutture variano secondo il sistema operativo e il file system, ma esistono dei princìpi generali. Nei dischi, il file system tiene informazioni su come eseguire l'avviamento di

un sistema operativo memorizzato nei dischi stessi, il numero totale di blocchi, il numero e la locazione dei blocchi liberi, la struttura delle directory e i singoli file. Molte di loro sono analizzate in modo particolareggiato nel seguito di questo capitolo.

Fra le strutture presenti nei dischi ci sono le seguenti.

- Il **blocco di controllo dell'avviamento** (*boot control block*), contenente le informazioni necessarie al sistema per l'avviamento di un sistema operativo da quel volume; se il disco non contiene un sistema operativo, tale blocco è vuoto. Di solito è il primo blocco di un volume. Nell'UFS, si chiama **blocco d'avviamento** (*boot block*); nell'NTFS, **settore d'avviamento della partizione** (*partition boot sector*).
- I **blocchi di controllo dei volumi** (*volume control block*); ciascuno di loro contiene i dettagli riguardanti il relativo volume (o partizione), come il numero e la dimensione dei blocchi nel disco, il contatore dei blocchi liberi e i relativi puntatori, il contatore dei blocchi di controllo dei file liberi e i relativi puntatori. Nell'UFS si chiama **superblocco**; nell'NTFS si chiama **tabella principale dei file** (*master file table*, MFT).
- Le **strutture delle directory** (una per file system) usate per organizzare i file. Nel caso dell'UFS comprendono i nomi dei file e i numeri di **inode** associati. Nel caso dell'NTFS sono memorizzate nella **tabella principale dei file** (*master file table*).
- I **blocchi di controllo di file** (FCB), contenenti molti dettagli dei file, compresi i permessi d'accesso ai relativi file, i proprietari, le dimensioni e le locazioni dei blocchi di dati. Nell'UFS si chiamano *inode*; nell'NTFS, queste informazioni sono memorizzate all'interno della tabella principale dei file, che si serve di una struttura di base di dati relazionale, con una riga per ciascun file.

Le informazioni tenute in memoria servono sia per la gestione del file system sia per migliorare le prestazioni attraverso l'uso di cache. I dati si caricano al momento del montaggio e si eliminano allo smontaggio. Le strutture che vi possono essere incluse sono di diverso tipo:

- la tabella di montaggio interna alla memoria che contiene informazioni relative a ciascun volume montato;
- la struttura della directory, tenuta in memoria, contenente le informazioni relative a tutte le directory cui i processi hanno avuto accesso di recente (per le directory che costituiscono dei punti di montaggio, può essere presente un puntatore alla tabella dei volumi);
- la tabella generale dei file aperti, contenente una copia del blocco di controllo del file per ciascun file aperto, insieme con altre informazioni;
- la tabella dei file aperti per ciascun processo, contenente un puntatore all'appropriato elemento della tabella generale dei file aperti, insieme con altre informazioni;
- i buffer conservano blocchi del file system durante la loro lettura o scrittura sul disco.

Le applicazioni, per creare un nuovo file, eseguono una chiamata al file system logico, il quale conosce il formato della struttura della directory. Esso crea e alloca un nuovo FCB. (In alternativa, nel caso dei file system che creano tutti i blocchi di controllo al momento della loro installazione, esso alloca semplicemente un blocco di controllo libero.) Il sistema carica quindi la directory appropriata in memoria, la aggiorna con il nome del nuovo file e con il blocco di controllo associato, e la scrive nuovamente sul disco. Una tipica struttura di blocco di controllo dei file (FCB) è illustrata nella Figura 11.2.

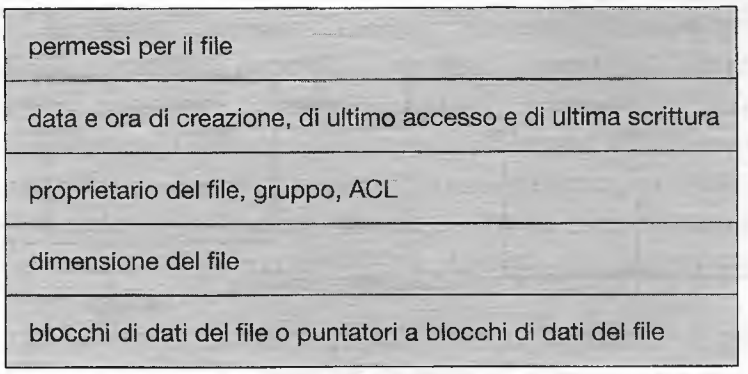


Figura 11.2 Tipico blocco di controllo dei file.

Alcuni sistemi operativi, compreso UNIX, trattano le directory esattamente come i file, distinguendole con un campo per il tipo che indica che si tratta di una directory. Altri, tra cui il sistema operativo Windows NT, dispongono di chiamate di sistema distinte per i file e le directory e trattano le directory come entità separate dai file. Indipendentemente da questioni strutturali, il file system logico può basarsi sul modulo che si occupa dell'organizzazione dei file per far corrispondere l'I/O su directory a numeri di blocchi di disco, che poi si passano al file system di base e al sistema per il controllo dell'I/O.

Una volta creato un file, per essere usato per operazioni di I/O deve essere *aperto*. La chiamata di sistema `open()` passa un nome di file al file system. Per controllare se il file sia già in uso da parte di qualche processo, la chiamata `open()` dapprima esamina la tabella dei file aperti in tutto il sistema; in caso affermativo, aggiunge un elemento alla tabella dei file aperti del processo (per ogni processo che stia usando il file) che punta alla tabella dei file aperti in tutto il sistema. L'algoritmo può eliminare significativi rallentamenti. Una volta aperto il file, se ne ricerca il nome all'interno della directory. Alcune porzioni della struttura delle directory sono di solito tenute in memoria per accelerare le operazioni sulle directory. Una volta trovato il file, si copia l'FCB nella tabella generale dei file aperti, tenuta in memoria. Questa tabella non solo contiene l'FCB, ma tiene anche traccia del numero di processi che in quel momento hanno il file aperto.

Successivamente, si crea un elemento nella tabella dei file aperti del processo con un puntatore alla tabella generale e con alcuni altri campi. Questi altri campi possono comprendere un puntatore alla posizione corrente nel file (per successive operazioni `read()` o `write()`) e il tipo d'accesso richiesto all'apertura del file. La `open()` riporta un puntatore all'elemento appropriato nella tabella dei file aperti del processo, sicché tutte le operazioni sul file si svolgeranno usando questo puntatore. Il nome del file potrebbe non essere contenuto nella tabella dei file aperti, visto che, una volta che l'FCB appropriato è stato individuato nei dischi, il sistema non ne ha bisogno. Tuttavia, potrebbe venir memorizzato in una cache per risparmiare tempo sulle aperture successive dello stesso file. Il nome dato all'elemento della tabella è **descrittore di file** (*file descriptor*) in UNIX, e **handle del file** in Windows. Finché un file non viene chiuso, tutte le operazioni si compiono sulla tabella dei file aperti usando questo elemento.

Quando un processo chiude il file, si cancella il relativo elemento nella tabella dei file aperti del processo e si decrementa il contatore associato al file nella tabella generale. Se tutti i processi che avevano aperto il file lo hanno chiuso, si riscrive l'informazione aggiornata sul file nella struttura delle directory nei dischi e si cancella il relativo elemento nella tabella generale dei file aperti.

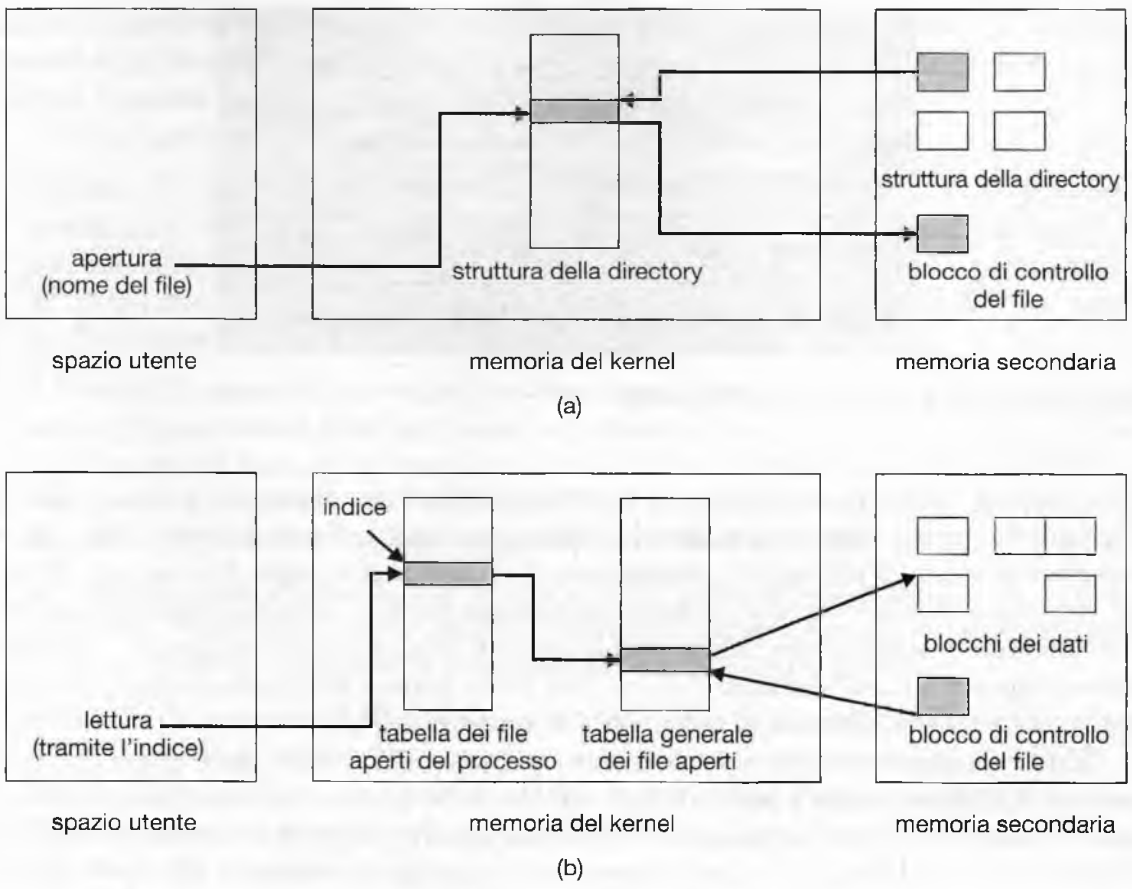


Figura 11.3 Strutture del file system che si mantengono nella memoria; (a) apertura di file; (b) lettura di file.

Alcuni sistemi complicano ulteriormente lo schema descritto, usando il file system come interfaccia per altri aspetti del sistema, come la comunicazione in rete. Ad esempio, nell'UFS, la tabella generale dei file aperti contiene gli *inode* e altre informazioni su file e directory, ma contiene anche informazioni simili per le connessioni di rete e i dispositivi. In questo modo si può usare un unico meccanismo per molteplici fini.

Le questioni concernenti l'uso delle cache per queste strutture non vanno però trascurate; usando questo schema, tutta l'informazione su un file aperto, eccetto i suoi effettivi blocchi di dati, è tenuta in memoria. Il sistema UNIX BSD è noto per il suo uso di cache ovunque sia possibile risparmiare su operazioni di I/O nei dischi. La sua frequenza media di successi nella cache, pari all'85 per cento, dimostra l'utilità di queste tecniche.

La Figura 11.3 riassume le strutture che si usano nella realizzazione di un file system.

11.2.2 Partizioni e montaggio

Un disco si può configurare in vari modi, secondo il sistema operativo che lo gestisce. Si può suddividere in più partizioni, oppure un volume può comprendere più partizioni su molteplici dischi. Qui trattiamo il primo caso, mentre il secondo, che si può considerare un caso particolare di organizzazione RAID, è trattato nel Paragrafo 12.7.

Ciascuna partizione è priva di struttura logica (*raw partition*) se non contiene alcun file system. Se nessun file system è appropriato, si usa un **disco privo di struttura logica** (*raw disk*). Il sistema operativo UNIX impiega una partizione priva di struttura per l'area d'avvicendamento dei processi; per questo scopo usa un formato specifico. Allo stesso modo alcu-

ni sistemi di gestione di basi di dati usano dischi privi di un'ordinaria struttura logica e formattano i dati secondo le proprie necessità. Un disco privo di struttura logica può anche contenere informazioni necessarie per sistemi RAID di gestione dei dischi, ad esempio le mappe di bit che indicano quali blocchi sono duplicati in altri dischi, e quali sono stati modificati e si devono aggiornare negli altri dischi. Analogamente, può contenere una piccola base di dati di informazioni sulla configurazione RAID, ad esempio, quali dischi appartengono a ciascun insieme RAID. Il Paragrafo 12.5.1 affronta altri aspetti concernenti l'uso dei dischi privi di struttura logica.

Le informazioni relative all'avviamento del sistema si possono registrare in un'apposita partizione, che anche in questo caso ha un proprio formato, poiché nella fase d'avviamento il sistema non ha ancora caricato i driver di dispositivo del file system e quindi non può interpretarne il formato. Questa partizione consiste piuttosto in una serie sequenziale di blocchi, che si carica in memoria come un'immagine. L'esecuzione dell'immagine comincia a una locazione prefissata, ad esempio il primo byte. L'immagine d'avviamento può contenere più informazioni di quelle che servono per un singolo sistema operativo. I PC, ad esempio, e altri sistemi si possono configurare per l'installazione di più sistemi operativi (*dual-booted*). In questo caso l'area d'avviamento può contenere un modulo, detto caricatore d'avviamento (*boot loader*), capace di interpretare diversi file system e diversi sistemi operativi. Una volta caricato, può avviare uno dei sistemi operativi disponibili nei dischi. Ciascun disco può avere più partizioni, ognuna contenente un diverso tipo di file system e un sistema operativo differente.

Nella fase di caricamento del sistema operativo, si esegue il montaggio della **partizione radice** (*root partition*), che contiene il kernel del sistema operativo e in alcuni casi altri file di sistema. Secondo il sistema operativo, il montaggio degli altri volumi avviene automaticamente in questa fase oppure si può compiere successivamente in modo esplicito. Durante l'operazione di montaggio, il sistema verifica che il dispositivo contenga un file system valido chiedendo al dispositivo di leggere la directory di dispositivo e verificando che la directory abbia il formato corretto. Se così non fosse, è necessaria una verifica della coerenza della partizione e una eventuale correzione, con o senza l'intervento dell'utente. Infine, il sistema annota nella struttura della **tabella di montaggio** in memoria che un file system è stato montato insieme al tipo di file system. I dettagli di questa funzione dipendono dal sistema operativo. I sistemi basati sui sistemi operativi Microsoft Windows eseguono il montaggio di ogni volume in uno spazio di nomi separato, identificato da una lettera seguita dai due punti (`:`). Ad esempio, per memorizzare che un file system è stato montato in `F:`, il sistema operativo introduce un puntatore al file system in un campo della struttura del dispositivo corrispondente a `F:`. Quando un processo specifica la lettera dell'unità, il sistema operativo trova il puntatore al file system appropriato e attraversa la struttura delle directory in quel dispositivo per trovare lo specifico file o directory. Le recenti versioni di Windows permettono il montaggio di un file system in qualsiasi punto all'interno della struttura della directory esistente.

In UNIX, l'operazione di montaggio di un file system si può compiere in qualsiasi directory. Questa funzione si realizza impostando un flag nella copia dell'*inode* tenuta in memoria di quella directory, che segnala che la directory è un punto di montaggio. Un campo dell'*inode* punta a un elemento nella tabella di montaggio, che indica quale dispositivo è montato in quella posizione. L'elemento della tabella di montaggio contiene un puntatore al superblocco del file system in quel dispositivo. Questo schema permette al sistema operativo di attraversare facilmente la propria struttura della directory, passando da un file system all'altro secondo le necessità.

11.2.3 File system virtuali

Nel paragrafo precedente si sottolinea il fatto che i sistemi operativi moderni devono gestire contemporaneamente tipi di file system diversi. Per capire come si può realizzare questa funzione occorre tuttavia considerare il modo in cui un sistema operativo può consentire l'integrazione di diversi tipi di file system in un'unica struttura della directory, così da permettere agli utenti di spostarsi senza problemi da un tipo di file system all'altro, mentre percorrono lo spazio del file system complessivo.

Un metodo ovvio ma non ottimale per realizzare più tipi di file system è scrivere procedure di gestione di file e directory separate per ciascun tipo di file system. Al contrario, la maggior parte dei sistemi operativi, compreso UNIX, impiega tecniche orientate agli oggetti per semplificare e organizzare in maniera modulare la soluzione. L'uso di queste tecniche rende possibile la realizzazione, nella stessa struttura, di tipi di file system molto diversi tra loro, compresi i file system di rete, come l'NFS. Gli utenti possono accedere a file contenuti in diversi file system nei dischi locali, o anche in file system disponibili tramite la rete.

Per isolare le funzioni di base delle chiamate di sistema dai dettagli di realizzazione si adoperano apposite strutture dati. In questo modo la realizzazione del file system si articola in tre strati principali, riportati in modo schematico nella Figura 11.4. Il primo strato è l'interfaccia del file system, basata sulle chiamate di sistema `open()`, `read()`, `write()` e `close()` e sui descrittori di file.

Il secondo strato si chiama strato del **file system virtuale** (*virtual file system*, VFS) e svolge due funzioni importanti.

1. Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme. Nello stesso calcolatore possono coesistere più interfacce VFS, che permettono un accesso trasparente a diversi tipi di file system montati localmente.

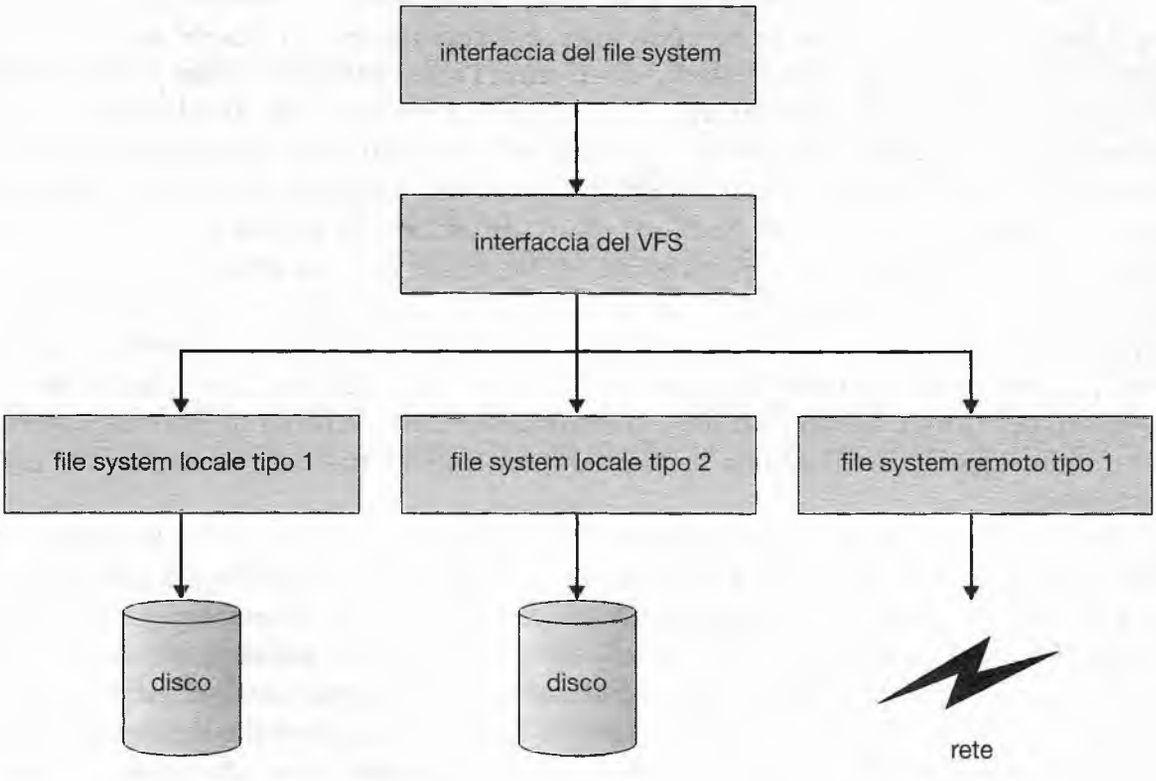


Figura 11.4 Schema di un file system virtuale.

2. Permette la rappresentazione univoca di un file su tutta la rete. Il VFS è basato su una struttura di rappresentazione dei file detta **vnode** che contiene un indicatore numerico unico per tutta la rete per ciascun file. (Gli *inode* di UNIX sono unici solo all'interno di un singolo file system.) Tale unicità per tutta la rete è richiesta per la gestione del file system di rete. Il kernel contiene una struttura *vnode* per ciascun nodo attivo, sia che si tratti di un file sia che si tratti di una directory.

Quindi, il VFS distingue i file locali da quelli remoti, e distingue i file locali secondo i relativi tipi di file system.

Il VFS attiva le operazioni specifiche del file system per gestire le richieste locali secondo i tipi di file system, e invoca le procedure del protocollo NFS per le richieste remote. Gli handle del file si costruiscono secondo i *vnode* relativi e s'inviano a queste procedure come argomenti. Lo strato che realizza il protocollo NFS è il più basso dell'architettura.

Esaminiamo succintamente l'architettura VFS di Linux. I quattro tipi più importanti di oggetti definiti in questo sistema sono:

- l'oggetto **inode**, che rappresenta il singolo file;
- l'oggetto **file**, che rappresenta un file aperto;
- l'oggetto **superblock**, che rappresenta un intero file system;
- l'oggetto **dentry**, che rappresenta il singolo elemento della directory.

Per ognuno di questi tipi, VFS specifica un insieme di operazioni da implementare. Ciascun oggetto di uno di questi tipi contiene un puntatore a una tabella di funzioni; questa, alla sua volta, contiene gli indirizzi delle effettive funzioni che implementano le operazioni richieste dalla specifica dell'oggetto. Per esempio, una versione semplificata della API di alcune delle operazioni dell'oggetto file è:

- `int open(...)` – apre il file.
- `ssize_t read(...)` – legge dal file.
- `ssize_t write(...)` – scrive sul file.
- `int mmap(...)` – mappa il file in memoria.

Ogni implementazione dell'oggetto file per uno specifico tipo di file deve implementare tutte le funzioni specificate nella definizione dell'oggetto file. (La definizione completa dell'oggetto file si trova nella struttura `struct file_operations`, nel file `/usr/include/linux/fs.h` .)

Questo schema permette a VFS di eseguire operazioni su uno degli oggetti in questione invocando l'appropriata funzione della tabella delle funzioni dell'oggetto, senza dover conoscere i dettagli dello specifico oggetto. Per esempio, VFS non sa, né vuol sapere, se un certo inode rappresenti un file su disco, un file di directory o un file remoto. L'implementazione appropriata dell'operazione `read()` per l'oggetto in questione è comunque reperibile sempre nel medesimo punto all'interno della tabella delle funzioni: invocando tale implementazione, VFS può disinteressarsi dei dettagli legati all'effettiva lettura dei dati.

11.3 Realizzazione delle directory

La selezione degli algoritmi di allocazione e degli algoritmi di gestione delle directory ha un grande effetto sull'efficienza, le prestazioni e l'affidabilità del file system. Per tale ragione è necessario comprendere i vari aspetti di questi algoritmi, trattati di seguito.

11.3.1 Lista lineare

Il più semplice metodo di realizzazione di una directory è basato sull'uso di una lista lineare contenente i nomi dei file con puntatori ai blocchi di dati. Questo metodo è di facile programmazione, ma la sua esecuzione è onerosa in termini di tempo. Per creare un nuovo file occorre prima esaminare la directory per essere sicuri che non esista già un file con lo stesso nome, quindi aggiungere un nuovo elemento alla fine della directory. Per cancellare un file occorre cercare nella directory il file con quel nome, quindi rilasciare lo spazio che gli era assegnato. Esistono vari metodi per riutilizzare un elemento della directory: si può contrassegnare l'elemento come non usato (attribuendogli un nome speciale, come un nome vuoto, oppure un bit d'uso in ogni elemento), oppure può essere aggiunto a una lista di elementi di directory liberi; una terza possibilità prevede la copiatura dell'ultimo elemento della directory in una locazione liberata e la diminuzione della lunghezza della directory. Per ridurre il tempo di cancellazione di un file si può usare anche una lista concatenata.

Il vero svantaggio dato da una lista lineare di elementi di directory è dato dalla ricerca lineare di un file. Le informazioni sulla directory vengono usate frequentemente, e gli utenti avvertirebbero una gestione lenta dell'accesso a tali informazioni. In effetti, molti sistemi operativi impiegano una cache per memorizzare le informazioni sulla directory usata più recentemente. La presenza nella cache delle informazioni richieste ne evita la continua rilettura dai dischi. Una lista ordinata permette una ricerca binaria e riduce il tempo medio di ricerca, tuttavia il requisito dell'ordinamento può complicare la creazione e la cancellazione di file, poiché, per tenere ordinata la lista, può essere necessario spostare quantità notevoli di informazioni sulla directory. In questo caso, può essere d'aiuto una struttura dati più raffinata, come un B-albero. Un vantaggio della lista ordinata è che consente di produrre l'elenco ordinato del contenuto della directory senza una fase d'ordinamento separata.

11.3.2 Tabella hash

Un'altra struttura dati che si usa per realizzare le directory è la **tabella hash**. In questo metodo una lista lineare contiene gli elementi di directory, ma si usa anche una struttura dati hash. La tabella hash riceve un valore calcolato usando come operando il nome del file e riporta un puntatore al nome del file nella lista lineare. Questa struttura dati può diminuire notevolmente il tempo di ricerca nella directory. L'inserimento e la cancellazione sono abbastanza semplici, anche se occorre prendere provvedimenti per evitare **collisioni**, cioè situazioni in cui da due nomi di file si ottiene un riferimento alla stessa locazione.

Le maggiori difficoltà legate a una tabella hash sono la sua dimensione, che in genere è fissa, e la dipendenza della funzione hash da tale dimensione. Si supponga, ad esempio, di realizzare una tabella hash di 64 elementi; la funzione hash converte i nomi di file in interi da 0 a 63, ad esempio, usando il resto di una divisione per 64. Per creare in un secondo tempo un sessantacinquesimo file occorre allungare la tabella hash della directory, ad esempio fino a 128 elementi. Occorre quindi una nuova funzione hash per associare i nomi di file all'intervallo compreso tra 0 e 127, e gli elementi esistenti nella directory si devono riorganizzare in modo da riflettere i loro nuovi valori della funzione hash.

Alternativamente, ciascun elemento della tabella hash, anziché un singolo valore, può essere una lista concatenata; ciò consente di risolvere le collisioni aggiungendovi il nuovo elemento. Le ricerche vengono alquanto rallentate, poiché la ricerca per nome può richiedere l'attraversamento della lista concatenata degli elementi in collisione della tabella hash; probabilmente tale metodo è comunque più veloce di una ricerca lineare nell'intera directory.

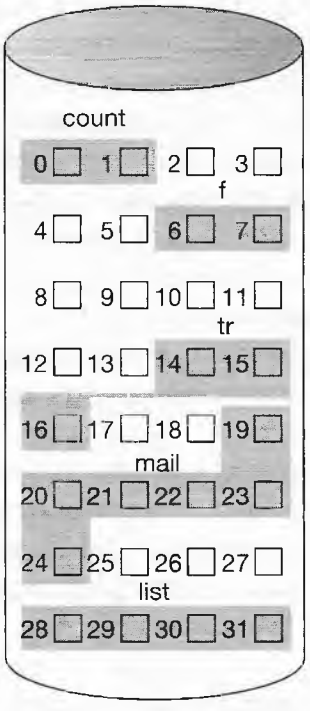
11.4 Metodi di allocazione

La natura ad accesso diretto dei dischi permette una certa flessibilità nella realizzazione dei file. In quasi tutti i casi, molti file si memorizzano nello stesso disco. Il problema principale consiste dunque nell'allocare lo spazio per questi file in modo che lo spazio nel disco sia usato efficientemente e l'accesso ai file sia rapido. Esistono tre metodi principali per l'allocazione dello spazio di un disco; può essere infatti contigua, concatenata o indicizzata. Ciascuno di questi metodi presenta vantaggi e svantaggi. Alcuni sistemi, come l'RDOS di Data General per la linea di calcolatori Nova, dispongono di tutti e tre i metodi. Generalmente, però, un sistema usa un unico metodo per tutti i file all'interno di un tipo di file system.

11.4.1 Allocazione contigua

Per usare il metodo di **allocazione contigua**, ogni file deve occupare un insieme di blocchi contigui del disco. Gli indirizzi del disco definiscono un ordinamento lineare nel disco stesso. Con questo ordinamento l'accesso al blocco $b + 1$ dopo il blocco b non richiede normalmente alcuno spostamento della testina. Se la testina deve essere spostata dall'ultimo settore di un cilindro al primo settore del cilindro successivo lo spostamento avviene su una sola traccia. Quindi, il numero dei posizionamenti (*seek*) richiesti per accedere a file il cui spazio è allocato in modo contiguo è trascurabile, così com'è trascurabile il tempo di ricerca (*seek time*), quando quest'ultimo è necessario. Il sistema operativo IBM VM/CMS usa l'allocazione contigua poiché consente tali buone prestazioni.

L'allocazione contigua dello spazio per un file è definita dall'indirizzo del primo blocco (inteso come numero di blocco) e dalla lunghezza (espressa in blocchi). Se il file è lungo n blocchi e comincia dalla locazione b, allora occupa i blocchi b, $b + 1$, $b + 2$, ..., $b + n - 1$. L'elemento di directory per ciascun file indica l'indirizzo del blocco d'inizio e la lunghezza dell'area assegnata per questo file (Figura 11.5).



directory

file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figura 11.5 Allocazione contigua dello spazio dei dischi.

Accedere a un file il cui spazio è assegnato in modo contiguo è facile. Quando si usa un accesso sequenziale, il file system memorizza l'indirizzo dell'ultimo blocco cui è stato fatto riferimento e, se è necessario, legge il blocco successivo. Nel caso di un accesso diretto al blocco i di un file che comincia al blocco b si può accedere immediatamente al blocco $b + i$. Quindi, sia l'accesso sequenziale sia quello diretto si possono gestire con l'allocazione contigua.

L'allocazione contigua presenta però alcuni problemi; una difficoltà riguarda l'individuazione dello spazio per un nuovo file. La realizzazione del sistema di gestione dello spazio libero, illustrata nel Paragrafo 11.5, determina il modo in cui tale compito viene eseguito. Si può usare ogni sistema di gestione, anche se alcuni sono più lenti di altri.

Il problema dell'allocazione contigua dello spazio dei dischi si può considerare un'applicazione particolare del problema generale dell'**allocazione dinamica della memoria**, trattato nel Paragrafo 8.3; il problema generale è, infatti, quello di soddisfare una richiesta di dimensione n data una lista di buchi liberi. I più comuni criteri di scelta di un buco libero da un insieme di buchi disponibili sono quelli del primo buco abbastanza grande (*first-fit*) e del più piccolo tra i buchi abbastanza grandi (*best-fit*). Simulazioni hanno dimostrato che questi due criteri sono più efficienti di quello di scelta del buco più grande (*worst-fit*) sia in termini di tempo sia d'uso della memoria. Nessuno dei due è palesemente migliore rispetto all'uso della memoria, ma la scelta del primo buco abbastanza grande è generalmente più rapida.

Questi algoritmi soffrono della **frammentazione esterna**: assegnando e liberando lo spazio per i file, lo spazio libero dei dischi viene frammentato in tanti piccoli pezzi. La frammentazione esterna si ha ogniqualvolta lo spazio libero è suddiviso in pezzi, e diviene un problema quando il più grande di tali pezzi contigui non è sufficiente a soddisfare una richiesta; la memoria viene frammentata in tanti buchi, nessuno dei quali è abbastanza grande da contenere i dati. Secondo la capacità dei dischi e la dimensione media dei file, la frammentazione esterna può essere un problema più o meno grave.

Una strategia per prevenire la perdita di una quantità significativa di spazio sul disco a causa della frammentazione esterna consiste nel copiare un intero file system su un altro disco o nastro. Quindi si liberava completamente il primo disco creando un ampio spazio libero contiguo. La procedura provvedeva poi a copiare nuovamente i file nel disco, assegnando tale spazio contiguo. Questo schema **compatta** efficacemente tutto lo spazio libero in uno spazio contiguo, risolvendo il problema della frammentazione. Il costo di questa compattazione è rappresentato dal tempo necessario, ed è particolarmente pesante per i dischi di grande capacità che impiegano l'allocazione contigua; in essi, compattare lo spazio può richiedere ore e può essere necessario eseguire tale operazione settimanalmente. Alcuni sistemi richiedono l'esecuzione **non in linea** (*off-line*) di questa funzionalità, ossia con il file system non montato. Durante questo "periodo morto" (*down time*) il normale funzionamento del sistema non è possibile, quindi tale compattazione va evitata a tutti i costi per i calcolatori in attività. Molti sistemi moderni sono invece in grado di eseguire la deframmentazione **in linea** (*on-line*), ossia durante il loro normale funzionamento, a prezzo, però, di una sostanziale diminuzione delle prestazioni.

Un altro problema che riguarda l'allocazione contigua è la determinazione della quantità di spazio necessaria per un file. Quando si crea un file, occorre trovare e allocare lo spazio di cui necessita. Esiste il problema di conoscere la dimensione del file da creare; in alcuni casi questa dimensione si può stabilire in modo abbastanza semplice, ad esempio quando si copia un file esistente; in generale, tuttavia, non è facile stimare la dimensione di un file che deve contenere dati emessi da un programma.

Se un file riceve poco spazio, può essere impossibile estenderlo: soprattutto nel caso in cui si adoperi il criterio di allocazione del più piccolo tra i buchi abbastanza grandi, lo spazio oltre le due estremità del file può essere già in uso, quindi non è possibile ampliare il fi-

le in modo contiguo. Esistono allora due possibilità. La prima è che il programma utente si possa terminare con un idoneo messaggio d'errore. L'utente deve allora allocare più spazio ed eseguire di nuovo il programma. Queste esecuzioni ripetute possono essere onerose; per prevenire tale circostanza, normalmente l'utente sovrastima la quantità di spazio necessaria, sprecandone parecchio. L'altra possibilità consiste nel trovare un buco più grande, copiare il contenuto del file nel nuovo spazio e rilasciare lo spazio precedente. Queste operazioni si possono ripetere finché esiste spazio, anche se ciò può far perdere tempo. In questo caso tuttavia non è necessario informare esplicitamente l'utente su che cosa stia succedendo; anche se più lentamente, il sistema prosegue le attività nonostante il problema.

Anche se si conosce in anticipo la quantità di spazio necessaria per un file, l'allocazione preventiva può in ogni modo essere inefficiente. A un file che cresce lentamente in un periodo di tempo lungo (mesi o anni) si deve allocare spazio sufficiente per la sua dimensione finale, anche se molto di quello spazio può rimanere inutilizzato per parecchio tempo. Il file ha perciò un'estesa frammentazione interna.

Per ridurre al minimo questi inconvenienti, alcuni sistemi operativi fanno uso di uno schema di allocazione contigua modificato: inizialmente si assegna una porzione di spazio contiguo, e se questa non è abbastanza grande si aggiunge un'altra porzione di spazio, un'**estensione**. La locazione dei blocchi dei file si registra come una locazione e un numero dei blocchi, insieme con l'indirizzo del primo blocco dell'estensione seguente. In alcuni sistemi il proprietario del file può impostare la dimensione dell'estensione, e tale possibilità, se il proprietario è impreciso, può causare inefficienze. La frammentazione interna può ancora essere un problema se le estensioni sono troppo grandi; si possono presentare problemi dovuti alla frammentazione esterna quando si assegnano e si rilasciano estensioni di dimensione variabile. Il file system commerciale Veritas impiega le estensioni per ottimizzare le prestazioni; è un sostituto ad alte prestazioni dell'ordinario UFS di UNIX.

11.4.2 Allocazione concatenata

L'**allocazione concatenata** risolve tutti i problemi dell'allocazione contigua. Con questo tipo di allocazione, infatti, ogni file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso. La directory contiene un puntatore al primo e all'ultimo blocco del file. Ad esempio, un file di cinque blocchi può cominciare dal blocco 9, continuare al blocco 16, quindi al blocco 1, al blocco 10 e infine terminare al blocco 25 (Figura 11.6). Ogni blocco contiene un puntatore al blocco successivo. Questi puntatori non sono disponibili all'utente, quindi se ogni blocco è formato di 512 byte e un indirizzo del disco (il puntatore) richiede 4 byte, l'utente vede blocchi di 508 byte.

Per creare un nuovo file si crea semplicemente un nuovo elemento nella directory. Con l'allocazione concatenata, ogni elemento della directory ha un puntatore al primo blocco del file. Questo puntatore s'inizializza a `nil` (valore del puntatore di fine lista) a indicare un file vuoto; anche il campo della dimensione s'imposta a 0. Un'operazione di scrittura nel file determina la ricerca di un blocco libero attraverso il sistema di gestione dello spazio libero, la scrittura in tale blocco, e la concatenazione di tale blocco alla fine del file. Per leggere un file occorre semplicemente leggere i blocchi seguendo i puntatori da un blocco all'altro. Con l'allocazione concatenata non esiste frammentazione esterna e per soddisfare una richiesta si può usare qualsiasi blocco libero della lista. Inoltre non è necessario dichiarare la dimensione di un file al momento della sua creazione. Un file può continuare a crescere finché sono disponibili blocchi liberi, di conseguenza non è mai necessario compattare lo spazio del disco.

L'allocazione concatenata presenta comunque alcuni svantaggi. Il problema principale riguarda il fatto che può essere usata in modo efficiente solo per i file ad accesso sequenzia-

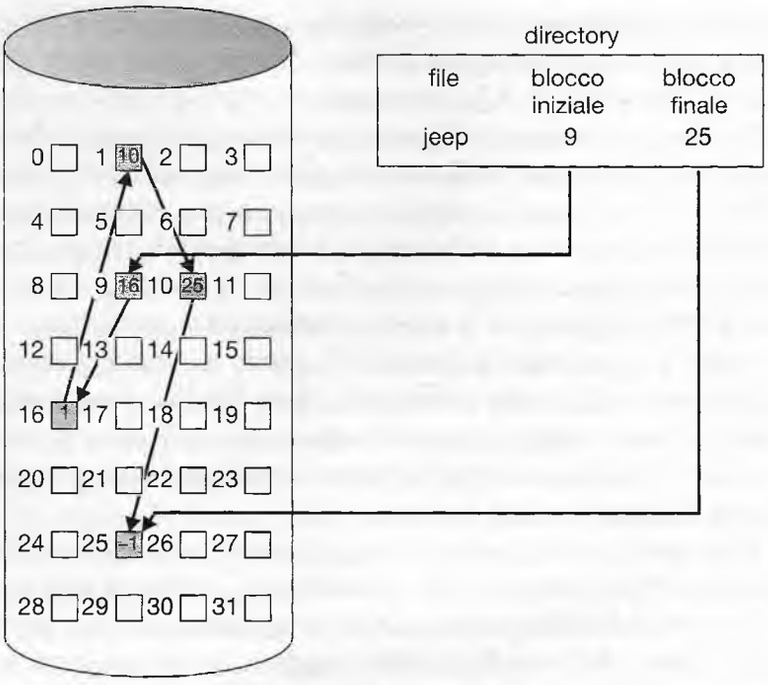


Figura 11.6 Allocazione concatenata dello spazio dei dischi.

le. Per trovare l'*i*-esimo blocco di un file occorre partire dall'inizio del file e seguire i puntatori finché non si raggiunge l'*i*-esimo blocco. Ogni accesso a un puntatore implica una lettura del disco, e talvolta un posizionamento della testina. Di conseguenza, per file il cui spazio è assegnato in modo concatenato, la funzione d'accesso diretto è inefficiente.

Un altro svantaggio dell'allocazione concatenata riguarda lo spazio richiesto per i puntatori. Se un puntatore richiede 4 byte di un blocco di 512 byte, allora lo 0,78 per cento del disco è usato per i puntatori anziché per le informazioni: ogni file richiede un po' più spazio di quanto ne richiederebbe altrimenti.

La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi contigui in **cluster** (gruppi di blocchi), e nell'allocare i cluster anziché i blocchi. Ad esempio, il file system può definire cluster di 4 blocchi e operare nel disco soltanto per unità di cluster. Così i puntatori usano una quantità di spazio di disco che si riduce in modo proporzionale al numero di cluster. Questo metodo permette che la corrispondenza tra blocchi logici e blocchi fisici rimanga semplice, ma migliora la produttività del disco: si hanno meno posizionamenti della testina del disco e diminuisce lo spazio necessario per l'allocazione dei blocchi e la gestione della lista dei blocchi liberi. Il costo di questo metodo è dato da un incremento della frammentazione interna, poiché se un cluster è parzialmente pieno si spreca più spazio di quanto se ne sprecherebbe con un solo blocco parzialmente pieno. I cluster si possono usare per ottimizzare l'accesso ai dischi in molti altri algoritmi, quindi s'impiegano nella maggior parte dei sistemi operativi.

Un altro problema riguarda l'affidabilità. Poiché i file sono tenuti insieme da puntatori sparsi per tutto il disco, s'immagini che cosa accadrebbe se un puntatore andasse perduto o danneggiato. Un errore di programmazione del sistema operativo oppure un errore di un'unità a disco potrebbero causare il prelevamento del puntatore errato. Questo errore, a sua volta, potrebbe causare il collegamento alla lista dei blocchi liberi oppure a un altro file. Una soluzione parziale a tale problema consiste nell'usare liste doppiamente concatenate oppure nel memorizzare il nome del file e il relativo numero di blocco in ogni blocco; questi schemi però sono ancora più onerosi per ogni file.

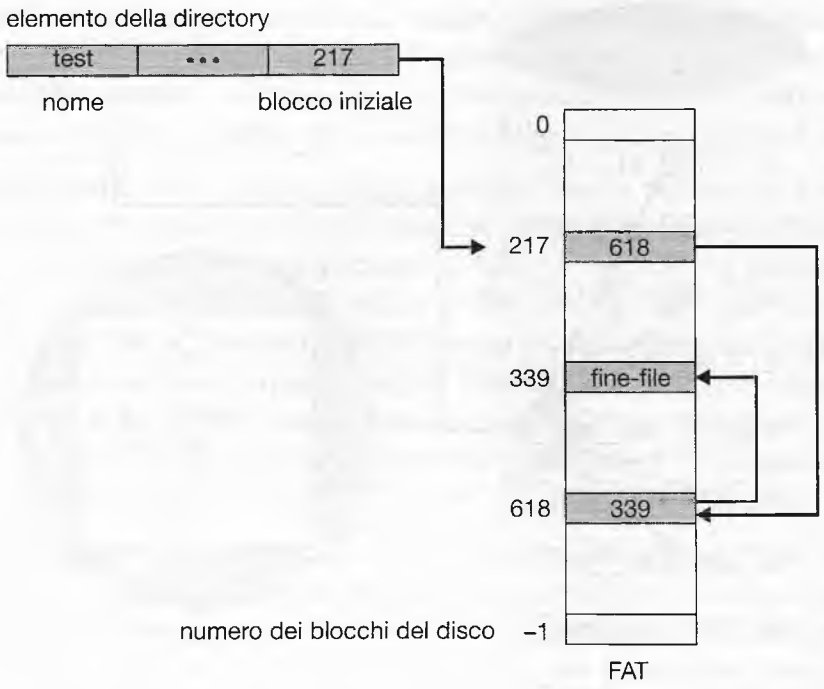


Figura 11.7 Tabella di allocazione dei file.

Una variante importante del metodo di allocazione concatenata consiste nell'uso della **tabella di allocazione dei file** (*file allocation table*, FAT). Tale metodo di allocazione, semplice ma efficiente, dello spazio dei dischi è usato nei sistemi operativi MS-DOS e OS/2. Per contenere tale tabella si riserva una sezione del disco all'inizio di ciascun volume; la FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco; si usa essenzialmente come una lista concatenata. L'elemento di directory contiene il numero del primo blocco del file. L'elemento della tabella indicizzato da quel numero di blocco contiene a sua volta il numero del blocco successivo del file. Questa catena continua fino all'ultimo blocco, che ha come elemento della tabella un valore speciale di fine del file. I blocchi inutilizzati sono indicati nella tabella da un valore 0. L'allocazione di un nuovo blocco a un file implica semplicemente la localizzazione del primo elemento della tabella con valore 0 e la sostituzione del valore di fine del file precedente con l'indirizzo del nuovo blocco; lo 0 è quindi sostituito con il valore di fine del file. Un esempio esplicativo di tale metodo è dato dalla struttura della FAT della Figura 11.7, dove il file in questione è formato dai blocchi 217, 618 e 339.

Lo schema di allocazione basato sulla FAT, se non si usa una cache, può causare un significativo numero di posizionamenti della testina. La testina del disco deve spostarsi all'inizio del volume per leggere la FAT e trovare la locazione del blocco in questione, quindi raggiungere la locazione del blocco stesso; nel caso peggiore sono necessari ambedue i movimenti per ciascun blocco. Un vantaggio è dato dall'ottimizzazione del tempo d'accesso diretto, poiché la testina del disco può trovare la locazione di ogni blocco leggendo le informazioni contenute nella FAT.

11.4.3 Allocazione indicizzata

L'allocazione concatenata risolve il problema della frammentazione esterna e quello della dichiarazione delle dimensioni dei file, entrambi presenti nell'allocazione contigua. Tuttavia, in mancanza di una FAT, l'allocazione concatenata non è in grado di sostenere un efficiente accesso diretto, poiché i puntatori ai blocchi sono sparsi, con i blocchi stessi, per tutto il di-

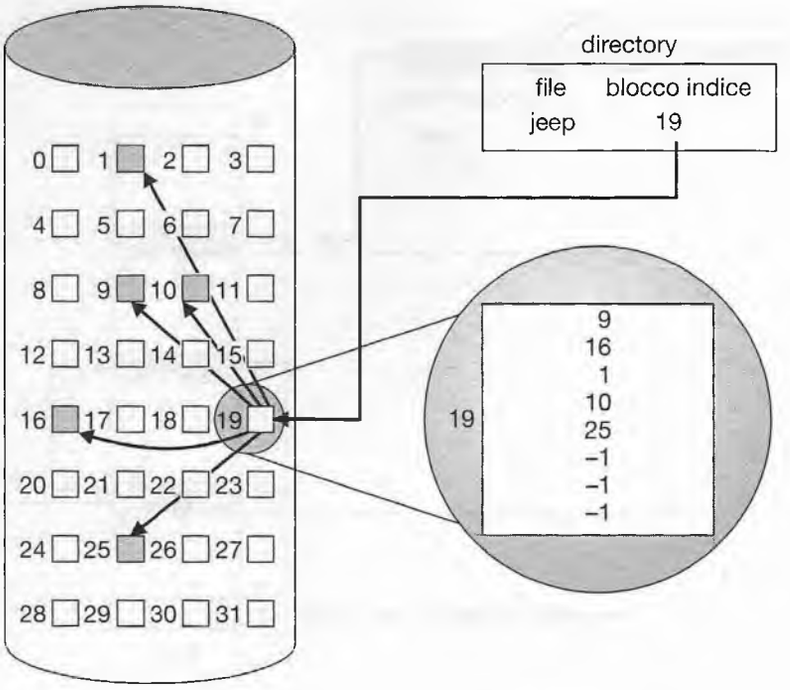


Figura 11.8 Allocazione indicizzata dello spazio dei dischi.

sco e si devono recuperare in ordine. L'**allocazione indicizzata** risolve questo problema, raggruppando tutti i puntatori in una sola locazione: il **blocco indice**.

Ogni file ha il proprio blocco indice: si tratta di un array d'indirizzi di blocchi del disco. L'*i*-esimo elemento del blocco indice punta all'*i*-esimo blocco del file. La directory contiene l'indirizzo del blocco indice, com'è illustrato nella Figura 11.8. Per individuare e leggere l'*i*-esimo blocco occorre usare il puntatore che si trova nell'*i*-esimo elemento del blocco indice, per poi localizzare e leggere il blocco desiderato. Questo schema è simile a quello della paginazione descritto nel Paragrafo 8.4.

Una volta creato il file, tutti i puntatori del blocco indice sono impostati a `nil`. Quando si scrive l'*i*-esimo blocco per la prima volta, il gestore dei blocchi liberi fornisce un blocco; l'indirizzo di questo blocco viene inserito nell'*i*-esimo elemento del blocco indice. Poiché ogni blocco libero del disco può soddisfare una richiesta di maggiore spazio, l'allocazione indicizzata consente l'accesso diretto senza soffrire di frammentazione esterna.

Lo spazio aggiuntivo richiesto dai puntatori del blocco indice è generalmente maggiore dello spazio aggiuntivo necessario per l'allocazione concatenata. Si consideri il comune caso di un file con uno o due blocchi; con l'allocazione concatenata si perde il solo spazio di un puntatore per blocco, complessivamente uno o due puntatori; con l'allocazione indicizzata occorre allocare un intero blocco indice, anche se solo uno o due puntatori sono diversi da `nil`.

Questo punto solleva la questione della dimensione del blocco indice. Ogni file deve avere un blocco indice, quindi è auspicabile che questo sia quanto più piccolo è possibile; ma se il blocco indice è troppo piccolo non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni, quindi è necessario disporre di un meccanismo per gestire questa situazione.

- **Schema concatenato.** Un blocco indice è formato normalmente di un solo blocco di disco; perciò, ciascun blocco indice può essere letto e scritto esattamente con un'operazione. Per permettere la presenza di lunghi file è possibile collegare tra loro parecchi

blocchi indice. Ad esempio, un blocco indice può contenere una piccola intestazione in cui sono riportati il nome del file e l'insieme dei primi 100 indirizzi del blocco di disco. L'indirizzo successivo, vale a dire l'ultima parola del blocco indice, è `nil` (per un file piccolo) oppure è un puntatore a un altro blocco indice (per un file lungo).

- **Indice a più livelli.** Una variante della rappresentazione concatenata consiste nell'impiego di un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello che, a loro volta, puntano ai blocchi dei file. Per accedere a un blocco, il sistema operativo usa l'indice di primo livello, con il quale individua il blocco indice di secondo livello, e con esso trova il blocco di dati richiesto. Questo metodo può continuare fino a un terzo o quarto livello, secondo la massima dimensione desiderata del file. Con blocchi di 4096 byte si possono memorizzare 1024 puntatori di 4 byte in un blocco indice. Due livelli di indici consentono 1.048.576 blocchi di dati, che permettono di avere file sino a 4 GB.
- **Schema combinato.** Un'altra possibilità, è la soluzione adottata nell'UFS, consistente nel tenere i primi 15 puntatori del blocco indice nell'*inode* del file. I primi 12 di questi 15 puntatori puntano a **blocchi diretti**, cioè contengono direttamente gli indirizzi di blocchi contenenti dati del file. Quindi, i dati per piccoli file (non più di 12 blocchi) non richiedono un blocco indice distinto. Se la dimensione dei blocchi è di 4 KB, è possibile accedere direttamente fino a 48 KB di dati. Gli altri tre puntatori puntano a **blocchi indiretti.** Il primo puntatore di blocco indiretto è l'indirizzo di un **blocco indiretto singolo**; si tratta di un blocco indice che non contiene dati, ma indirizzi di blocchi che contengono dati. Quindi c'è un puntatore di **blocco indiretto doppio** contenente l'indirizzo di un blocco che a sua volta contiene gli indirizzi di blocchi contenenti puntatori agli effettivi blocchi di dati. L'ultimo puntatore contiene l'indirizzo di un **blocco indiretto triplo.** Con questo metodo, il numero dei blocchi che si può allocare a un file supera la quantità di spazio che possono indirizzare i puntatori a file di 4 byte usati da molti sistemi operativi. Un puntatore a file di 32 bit consente di arrivare a soli 2^{32} byte, 4 GB. Molte versioni del sistema operativo UNIX, tra le quali Solaris e l'IBM AIX ora gestiscono puntatori a file sino a 64 bit. Puntatori di questa dimensione permettono di avere file e file system di dimensioni dell'ordine dei terabyte. Un *inode* UNIX è mostrato nella Figura 11.9.

Gli schemi d'allocazione indicizzata soffrono di alcuni dei problemi di prestazioni dell'allocazione concatenata. In particolare, i blocchi indice si possono caricare in memoria, ma i blocchi dei dati possono essere sparsi per un'intero volume.

11.4.4 Prestazioni

I metodi d'allocazione presentati hanno diversi livelli di efficienza di memorizzazione e differenti tempi d'accesso ai blocchi di dati; entrambi i fattori sono importanti nella scelta del metodo o dei metodi d'allocazione più adatti da impiegare in un sistema operativo.

Prima di scegliere un metodo di allocazione, è necessario determinare il modo in cui si usano i sistemi: un sistema con una prevalenza di accessi sequenziali farà uso di un metodo differente da quello di un sistema con una prevalenza di accessi diretti.

Per qualsiasi tipo d'accesso, l'allocazione contigua richiede un solo accesso per ottenere un blocco. Poiché è facile tenere l'indirizzo iniziale del file in memoria, si può calcolare immediatamente l'indirizzo del disco dell'*i*-esimo blocco, oppure del blocco successivo, e leggerlo direttamente.

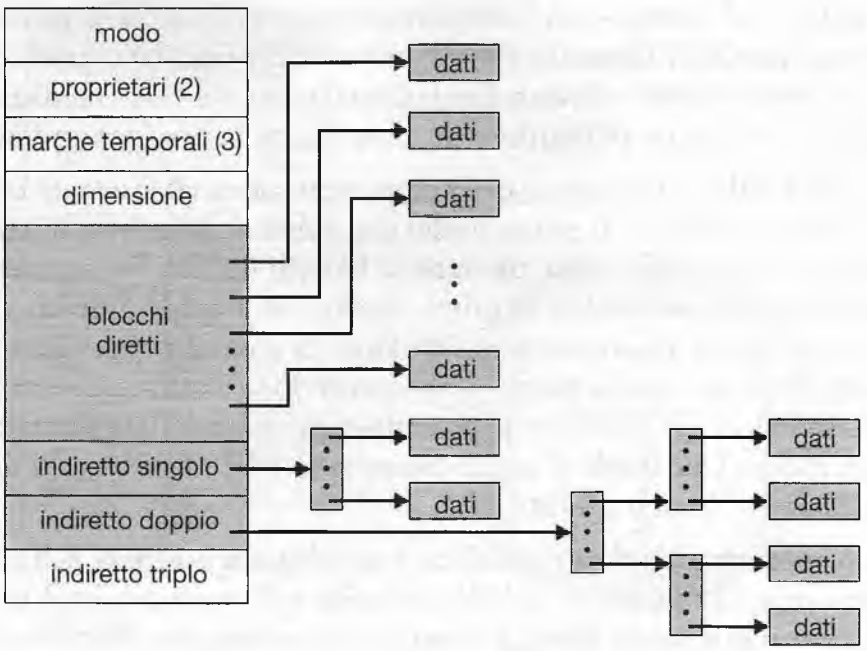


Figura 11.9 *Inode* di UNIX.

Con l'allocazione concatenata si può tenere in memoria anche l'indirizzo del blocco successivo e leggerlo direttamente. Questo metodo è valido per l'accesso sequenziale mentre, per quel che riguarda l'accesso diretto, un accesso all'*i*-esimo blocco può richiedere *i* letture del disco. Questo spiega perché l'allocazione concatenata non si dovrebbe usare per un'applicazione che richiede accessi diretti.

Da tutto ciò segue che alcuni sistemi gestiscono i file ad accesso diretto usando l'allocazione contigua, e i file ad accesso sequenziale tramite l'allocazione concatenata. Per questi sistemi, il tipo d'accesso si deve dichiarare al momento della creazione del file. Un file creato per l'accesso sequenziale è un file concatenato e non si può usare per l'accesso diretto. Un file creato per l'accesso diretto è contiguo e consente entrambi i tipi d'accesso, purché se ne dichiari la lunghezza massima al momento della sua creazione. In questo caso, il sistema operativo deve avere strutture dati idonee e algoritmi capaci di gestire *entrambi* i metodi di allocazione. I file si possono convertire da un tipo all'altro creando un nuovo file del tipo desiderato, nel quale si copia il contenuto del vecchio file; quest'ultimo si può quindi cancellare e il nuovo file rinominare.

L'allocazione indicizzata è più complessa. Se il blocco indice è già in memoria, l'accesso può essere diretto. Tuttavia, per tenere il blocco indice in memoria occorre una quantità di spazio considerevole. Se questo spazio di memoria non è disponibile, occorre leggere prima il blocco indice e quindi il blocco di dati desiderato. Per un indice a due livelli possono essere necessarie due letture del blocco indice. Se un file è estremamente grande, per compiere l'accesso a un blocco che si trovi vicino alla fine del file, prima di leggere il blocco dei dati occorre leggere tutti i blocchi indice per seguire la catena dei puntatori. Quindi le prestazioni dell'allocazione indicizzata dipendono dalla struttura dell'indice, dalla dimensione del file e dalla posizione del blocco desiderato.

Alcuni sistemi combinano l'allocazione contigua con l'allocazione indicizzata, usando quella contigua per i file piccoli (fino a tre o quattro blocchi) e passando automaticamente a quella indicizzata per i file grandi. Poiché generalmente i file sono piccoli, e in questo caso l'allocazione contigua è efficiente, le prestazioni medie possono risultare abbastanza buone.

Nel 1991, ad esempio, la versione di UNIX di Sun Microsystems è stata modificata per migliorare le prestazioni dell'algoritmo di allocazione del file system. Alcune misure delle prestazioni hanno mostrato che la massima produttività del disco in una tipica stazione di lavoro (SPARCstation1, da 12 MIPS) richiedeva il 50 per cento d'impegno della CPU e produceva un'ampiezza di banda di soli 1,5 MB al secondo (56 KB era la massima dimensione del trasferimento per DMA di allora). Per migliorare le prestazioni, la Sun ha apportato alcune modifiche per allocare lo spazio in cluster di 56 KB ogniqualvolta fosse possibile. Questo metodo di allocazione riduceva la frammentazione esterna e i tempi di ricerca e di latenza. Inoltre le procedure di lettura dei dischi sono state ottimizzate per leggere in questi grandi cluster. La struttura dell'*inode* non è stata modificata. Queste modifiche insieme con l'uso della lettura anticipata e del rilascio indietro (discussi nel Paragrafo 11.6.2) hanno prodotto una diminuzione del 25 per cento dell'impegno della CPU e una produttività notevolmente migliorata.

Sono possibili e si usano effettivamente anche molte altre ottimizzazioni. Data la disparità tra velocità della CPU e velocità dei dischi, non è irragionevole aggiungere al sistema operativo migliaia di istruzioni solo per risparmiare alcuni movimenti della testina. Con il passare del tempo tale disparità aumenta a tal punto che, per ottimizzare i movimenti della testina, si possono ragionevolmente usare centinaia di migliaia di istruzioni.

11.5 Gestione dello spazio libero

Poiché la quantità di spazio dei dischi è limitata, è necessario riutilizzare lo spazio lasciato dai file cancellati per scrivere, se è possibile, nuovi file (i dischi ottici a una sola scrittura permettono una sola scrittura in qualsiasi settore e quindi il riutilizzo è fisicamente impossibile). Per tener traccia dello spazio libero in un disco, il sistema conserva una **lista dello spazio libero**; vi sono registrati tutti gli spazi *liberi*, cioè non allocati ad alcun file o directory. Per creare un file occorre cercare nella lista dello spazio libero la quantità di spazio necessaria e assegnarla al nuovo file, quindi rimuovere questo spazio dalla lista. Quando si cancella un file, si aggiungono alla lista dello spazio libero i blocchi di disco a esso assegnati. A dispetto del suo nome, la lista dello spazio libero potrebbe non essere realizzata come una lista, come vedremo più avanti.

11.5.1 Vettore di bit

Spesso la lista dello spazio libero si realizza come una **mappa di bit**, o **vettore di bit**. Ogni blocco è rappresentato da un bit: se il blocco è libero, il bit è 1, se il blocco è assegnato il bit è 0.

Si consideri, ad esempio, un disco dove i blocchi 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 e 27 sono liberi e gli altri sono allocati. La mappa di bit dello spazio libero è la seguente:

001111001111110001100000011100000...

I vantaggi principali che derivano da questo metodo sono la sua relativa semplicità ed efficienza nel trovare il primo blocco libero o n blocchi liberi consecutivi nel disco; in effetti, molti calcolatori forniscono istruzioni di manipolazione dei bit utilizzabili con efficacia a tale scopo. Ad esempio, la famiglia di CPU Intel a partire dall'80386 e la famiglia di CPU Motorola a partire dal 68020 (rispettivamente, nei PC e nei vecchi Macintosh) hanno istruzioni che riportano lo scostamento del primo bit con valore 1 contenuto in una parola. Infatti il sistema operativo Apple Macintosh usava il metodo del vettore di bit per allocare lo spa-

zio dei dischi. Per trovare il primo blocco libero, il sistema operativo Macintosh controllava in modo sequenziale ogni parola nella mappa di bit per verificare che il valore non fosse 0, poiché una parola con valore 0 ha tutti i bit a 0 e rappresenta un insieme di blocchi assegnati. La prima parola non 0 viene scandita alla ricerca del primo bit 1, che indica la locazione del primo blocco libero. Il numero del blocco è dato dalla seguente espressione:

(*numero di bit per parola*) × (*numero di parole di valore 0*) + *scostamento del primo bit* 1.

Anche in questo caso le caratteristiche dell'architettura guidano le funzioni del sistema operativo. Sfortunatamente, i vettori di bit sono efficienti solo se tutto il vettore è mantenuto in memoria centrale, e viene di tanto in tanto scritto in memoria secondaria allo scopo di consentire eventuali operazioni di ripristino; è possibile tenere il vettore in memoria centrale se i dischi sono piccoli, come quelli usati nei microcalcolatori; tale soluzione non è applicabile ai dischi più grandi. Un disco di 1,3 GB con blocchi di 512 byte richiederebbe una mappa di bit di oltre 332 KB per tenere traccia dei suoi blocchi liberi. Il raggruppamento di quattro blocchi riduce questo numero a 83 KB per disco. Un disco di 1 TB con blocchi di 4 KB richiede 32 MB per memorizzare la propria mappa di bit. Dato che la dimensione del disco è in costante crescita, i problemi legati ai vettori di bit continueranno ad aggravarsi. Un file system di 1 PB richiederà una mappa di bit di 32 GB solo per gestire il suo spazio libero.

11.5.2 Lista concatenata

Un altro metodo di gestione degli spazi liberi consiste nel collegarli tutti, tenere un puntatore al primo di questi in una speciale locazione del disco e caricarlo in memoria. Questo primo blocco contiene un puntatore al successivo blocco libero, e così via. Facciamo riferimento al nostro esempio precedente (Paragrafo 11.5.1) nel quale i blocchi 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 e 27 erano liberi, mentre i restanti blocchi erano allocati. In questa situazione dovremmo mantenere un puntatore al blocco 2, trattandosi del primo blocco libero. Il blocco 2 conterrebbe un puntatore al blocco 3, che punterebbe al blocco 4, che punterebbe al blocco 5, il quale punterebbe a sua volta al blocco 8, e così via (Figura 11.10). Questo schema non è efficiente; per attraversare la lista è infatti necessario leggere ogni blocco, con un notevole tempo di I/O. Fortunatamente la necessità di attraversare la lista dello spazio libero non è frequente. Di solito il sistema operativo ha semplicemente bisogno di un blocco libero perché possa assegnarlo a un file, quindi si usa il primo blocco della lista. Il metodo che fa uso della FAT include il conteggio dei blocchi liberi nella struttura dati per l'allocazione; non è necessario un metodo separato.

11.5.3 Raggruppamento

Una possibile modifica del metodo della lista dello spazio libero prevede la memorizzazione degli indirizzi di n blocchi liberi nel primo di questi. I primi $n - 1$ di questi blocchi sono effettivamente liberi; l'ultimo blocco contiene gli indirizzi di altri n blocchi liberi, e così via. L'importanza di questo metodo, diversamente dall'ordinaria lista concatenata, è data dalla possibilità di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi.

11.5.4 Conteggio

Un altro orientamento sfrutta il fatto che, generalmente, più blocchi contigui si possono allocare o liberare contemporaneamente, soprattutto quando lo spazio viene allocato usando l'algoritmo di allocazione contigua o attraverso l'uso di cluster. Quindi, anziché tenere una lista di n indirizzi liberi, è sufficiente tenere l'indirizzo del primo blocco libero e il numero

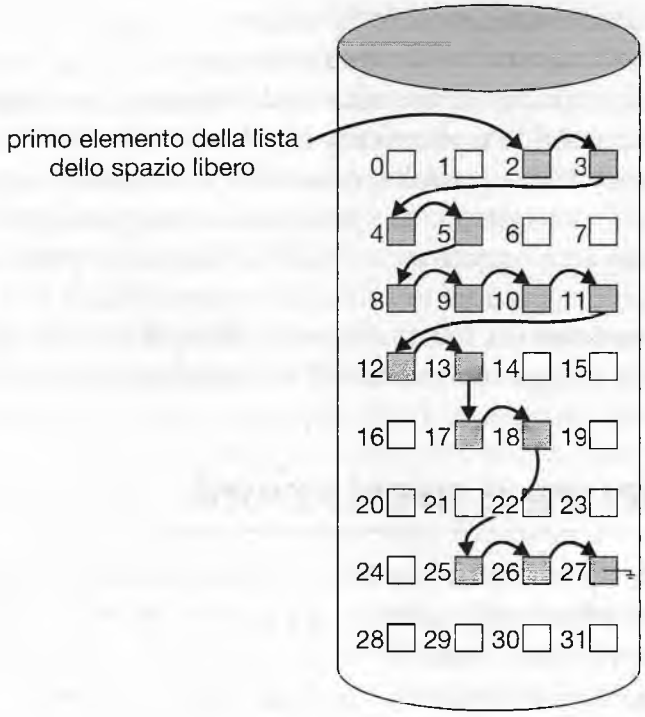


Figura 11.10 Lista concatenata degli spazi liberi su disco.

n di blocchi liberi contigui che seguono il primo blocco. Ogni elemento della lista dello spazio libero è formato da un indirizzo del disco e un contatore. Anche se ogni elemento richiede più spazio di quanto ne richieda un semplice indirizzo del disco, se il contatore è generalmente maggiore di 1 la lista complessiva è più corta. Tenete presente che questo metodo, il tracciamento dello spazio libero, è simile al metodo generale per allocare i blocchi. Queste voci possono essere salvate in un B-albero o in una lista concatenata, in modo da permettere controlli, inserzioni e cancellazioni efficienti.

11.5.5 Mappe di spazio

Il file system ZFS di Sun è stato progettato per contenere un gran numero di file, directory e persino di file system (in ZFS è possibile creare gerarchie di file system). Se non progettate e implementate adeguatamente, le strutture di dati risultanti possono essere grandi e inefficienti. Su queste scale, i metadati I/O possono avere un impatto notevole sulle prestazioni. Notate ad esempio che, se la lista dello spazio libero è implementata come una mappa di bit, le mappe di bit devono essere modificate sia quando i blocchi vengono allocati sia quando vengono liberati. Liberare 1 GB di dati su un disco di 1 TB potrebbe comportare l'aggiornamento di migliaia di blocchi di mappe di bit, perché quei blocchi di dati potrebbero essere sparpagliati sull'intero disco.

Nel suo algoritmo di gestione dello spazio libero ZFS utilizza una combinazione di tecniche per controllare la dimensione delle strutture di dati e minimizzare l'I/O necessario a gestire quelle strutture. Per prima cosa, ZFS crea **metalastre** (*metaslab*) per dividere lo spazio sul dispositivo in parti che abbiano una dimensione gestibile. Un dato volume potrebbe contenere centinaia di metalastre. Ogni metalastra è associata a una mappa di spazio. ZFS utilizza l'algoritmo di conteggio per memorizzare informazioni riguardanti i blocchi liberi. Piuttosto che scrivere strutture di calcolo su disco, impiega le tecniche dei file system con registrazione delle modifiche per registrarle. La mappa di spazio è un registro di tutte le attivi-

tà del blocco (allocazione e liberazione), in ordine cronologico, in formato di conteggio. Quando ZFS decide di allocare o liberare spazio su una metalastra, carica la mappa di spazio associata nella memoria in una struttura ad albero bilanciato (per operazioni molto efficienti), indicizzata da scostamenti, e replica il log in tale struttura.

La mappa di spazio all'interno della memoria è un'accurata rappresentazione dello spazio allocato e libero nella metalastra. ZFS condensa la mappa il più possibile combinando blocchi liberi contigui in una singola voce. Infine la lista dello spazio libero viene aggiornata sul disco come parte delle operazioni orientate alle transazioni di ZFS. Durante la fase di raccolta e classificazione possono verificarsi ancora richieste di blocchi, che vengono soddisfatte dal registro. In sostanza, il registro, insieme all'albero bilanciato, *costituiscono* la lista libera.

11.6 Efficienza e prestazioni

Dopo avere descritto le opzioni di allocazione dei blocchi e di gestione delle directory, è possibile considerare i loro effetti sulle prestazioni e l'efficienza d'uso dei dischi. I dischi tendono di solito a essere il principale collo di bottiglia per le prestazioni di un sistema, essendo i più lenti tra i componenti più rilevanti di un calcolatore. In questo paragrafo si considerano diverse tecniche utili per migliorare l'efficienza e le prestazioni della memoria secondaria.

11.6.1 Efficienza

L'uso efficiente di un disco dipende fortemente dagli algoritmi usati per l'allocazione del disco e la gestione delle directory. Ad esempio, gli *inode* di UNIX sono assegnati preventivamente in un volume. Anche un disco "vuoto" impiega una certa percentuale del suo spazio per gli *inode*. D'altra parte, l'allocazione preventiva degli *inode* e la loro distribuzione nel volume migliorano le prestazioni del file system. Queste migliori prestazioni sono il risultato degli algoritmi di allocazione e di gestione dei blocchi liberi adottati da UNIX, i quali cercano di mantenere i blocchi di dati di un file vicini al blocco che ne contiene l'*inode* allo scopo di ridurre il tempo di ricerca.

Come ulteriore esempio, si consideri lo schema che fa uso dei cluster presentato nel Paragrafo 11.4, che migliora le prestazioni di ricerca in un file e trasferimento di un file al costo di una maggiore frammentazione interna. Per ridurre questa frammentazione, il BSD UNIX varia la dimensione del cluster al crescere della dimensione del file. Cluster più grandi si usano dove possono essere riempiti, mentre cluster più piccoli si usano per file di piccole dimensioni e per l'ultimo cluster di un file.

Si devono tenere in considerazione anche il tipo di dati normalmente contenuti in un elemento di una directory (o di un *inode*). Di solito si memorizza la *data dell'ultima scrittura* per fornire informazioni all'utente e per determinare se per il file occorre la creazione o l'aggiornamento di una copia di riserva. Alcuni sistemi mantengono anche la *data dell'ultimo accesso* per consentire all'utente di risalire all'ultima volta che un file è stato letto. Per mantenere queste informazioni, ogniqualvolta si legge un file, si deve aggiornare un campo della directory. Questa modifica richiede la lettura nella memoria del blocco, la modifica della sezione e la riscrittura del blocco nel disco, poiché sui dischi si può operare solamente per blocchi (o cluster). Quindi, ogni volta che si apre un file per la lettura, si deve leggere e scrivere anche l'elemento della directory a esso associato. Ciò può essere inefficiente per file cui si accede frequentemente, quindi nella fase della progettazione del file system è necessario confrontare i benefici con i costi rispetto alle prestazioni. In generale, è necessario considerare l'influenza sull'efficienza e sulle prestazioni di *ogni* informazione che si vuole associare a un file.

A titolo d'esempio, si consideri come l'efficienza sia influenzata dalle dimensioni dei puntatori usati per l'accesso ai dati. La maggior parte dei sistemi usa puntatori di 16 o 32 bit ovunque all'interno del sistema operativo. Questa dimensione limita la lunghezza di un file a 2^{16} (64 KB) o 2^{32} byte (4 GB). Alcuni sistemi impiegano puntatori di 64 bit per portare il limite a 2^{64} byte, un numero effettivamente molto grande. Comunque, i puntatori di 64 bit richiedono più spazio per la loro memorizzazione e di conseguenza fanno sì che i metodi di allocazione e di gestione dello spazio libero (liste concatenate, indici, e così via) impieghino più spazio.

Una delle difficoltà nella scelta della dimensione dei puntatori, o di qualsiasi altra dimensione di allocazione fissa all'interno di un sistema operativo, è la pianificazione degli effetti provocati dal cambiamento della tecnologia. Basti considerare che il primo IBM PC XT aveva un disco di 10 MB e che il file system dell'MS-DOS poteva gestire solamente 32 MB. (Ciascun elemento della FAT era di 12 bit e puntava a un cluster di 8 KB.) Con la crescita della capacità dei dischi, i dischi più grandi si dovevano suddividere in partizioni di 32 MB, poiché il file system non poteva tener traccia di blocchi disposti oltre i 32 MB. Quando divennero comuni dischi di capacità superiore ai 100 MB, si dovettero modificare le strutture dati e gli algoritmi usati dall'MS-DOS per gestire i dischi in modo da consentire file system più grandi. (La dimensione di ciascun elemento della FAT fu portata a 16 bit e più tardi a 32 bit.) La decisione iniziale fu presa per motivi di efficienza; tuttavia, con l'avvento della Versione 4 dell'MS-DOS milioni di utenti si trovarono a disagio quando dovettero passare ai nuovi, più grandi file system. Il file system ZFS di Sun adotta puntatori di 128 bit, che in teoria non dovrebbero mai necessitare di un'estensione. (La minima massa di un dispositivo in grado di archiviare 2^{128} byte tramite memorizzazione al livello atomico è di circa 272 trilioni di chilogrammi.)

Come altro esempio, si consideri l'evoluzione del sistema operativo Solaris di Sun Microsystems. Originariamente molte strutture dati avevano lunghezza fissa ed erano assegnate all'avviamento del sistema. Queste strutture comprendevano la tabella dei processi e quella dei file aperti. Una volta riempite queste tabelle, non si potevano più creare nuovi processi o aprire nuovi file, sicché il sistema non riusciva nel proprio compito di fornire un servizio agli utenti. L'unico modo di aumentare le dimensioni di queste tabelle era la ricompilazione del kernel e il riavvio del sistema. Dall'uscita di Solaris 2 quasi tutte le strutture dati del kernel si assegnano in modo dinamico, eliminando questi limiti artificiali alle prestazioni del sistema. Naturalmente, gli algoritmi che manipolano queste tabelle sono ora più complessi e il sistema operativo è un po' più lento dovendo allocare e rilasciare dinamicamente gli elementi delle tabelle, ma questo è il prezzo da pagare per la generalizzazione delle funzioni.

11.6.2 Prestazioni

Dopo aver scelto gli algoritmi fondamentali del file system le prestazioni possono essere migliorate in diversi modi. Come si osserva nel Capitolo 13, alcuni controllori di unità a disco contengono una quantità di memoria locale sufficiente per la creazione di una **cache** interna al controllore sufficientemente grande da memorizzare un'intera traccia del disco alla volta. Eseguito il posizionamento della testina, si legge la traccia nella cache del controllore del disco a partire dal settore sotto cui si viene a trovare la testina (riducendo il tempo di latenza). Il controllore trasferisce quindi al sistema operativo tutte le richieste di settori. Quando i blocchi sono trasferiti dal controllore del disco alla memoria centrale, il sistema operativo ha la possibilità di inserirli in una propria cache nella memoria centrale.

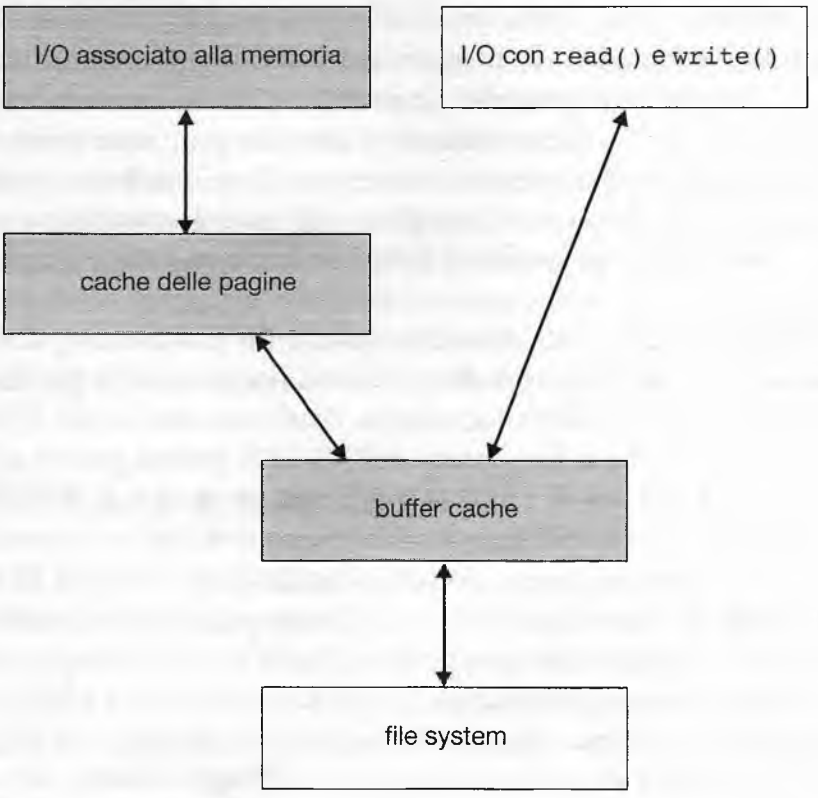


Figura 11.11 I/O senza una buffer cache unificata.

Alcuni sistemi riservano una sezione separata della memoria centrale come **cache del disco**, dove tenere i blocchi in previsione di un loro riutilizzo entro breve tempo. Altri sistemi impiegano una **cache delle pagine** per i file; si tratta di una soluzione che impiega tecniche di memoria virtuale per la gestione dei dati dei file come pagine anziché come blocchi di file system; l'uso degli indirizzi virtuali è molto più efficiente dell'uso dei blocchi fisici di disco. Diversi sistemi, compreso Solaris, Linux, Windows NT, 2000 e XP, usano le cache delle pagine, sia per le pagine relative ai processi sia per i dati dei file. Questo metodo è noto come **memoria virtuale unificata**.

Alcune versioni di UNIX e Linux prevedono la cosiddetta **buffer cache unificata**. Per illustrarne i vantaggi si considerino le due possibilità di aprire un file e accedervi: l'uso della mappatura dei file in memoria (Paragrafo 9.7) e l'uso delle ordinarie chiamate di sistema `read()` e `write()`. Senza una buffer cache unificata, si verifica una situazione simile a quella illustrata nella Figura 11.11. In questo caso, le chiamate di sistema `read()` e `write()` passano attraverso la buffer cache. La chiamata con associazione alla memoria richiede l'uso di due cache, la cache delle pagine e la buffer cache. L'associazione alla memoria prevede la lettura dei blocchi di disco dal file system e la loro memorizzazione nella buffer cache. Poiché il sistema di memoria virtuale non può interfacciarsi con la buffer cache, si deve copiare nella cache delle pagine il contenuto del file presente nella buffer cache. Questa situazione è nota come **double caching** proprio perché i dati del file system richiedono un doppio passaggio di cache. Non solo ciò comporta uno spreco di memoria, ma anche uno spreco di cicli della CPU e di I/O dovuti a un ulteriore trasferimento di dati nella memoria del sistema. Inoltre, eventuali incoerenze tra le due cache possono generare errori nella memorizzazione dei dati nei file. Con una buffer cache unificata, sia l'associazione alla memoria sia le chiamate di sistema `read()` e `write()` usano la stessa cache delle pagine,

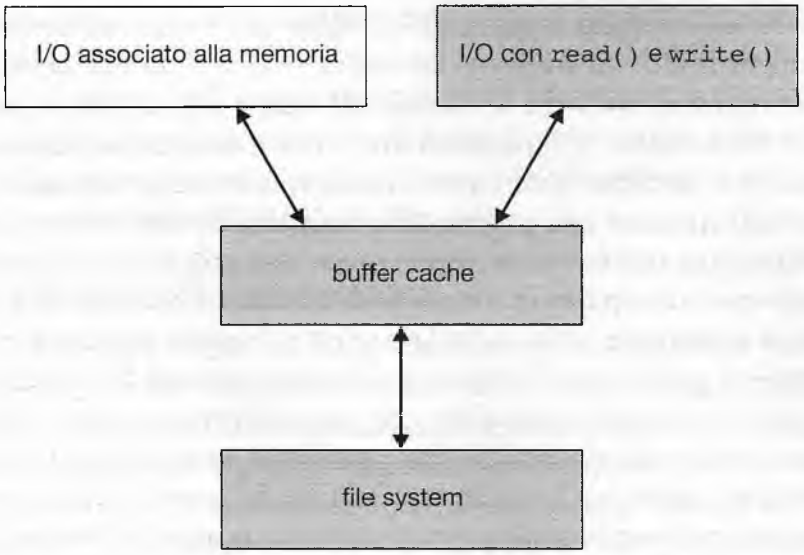


Figura 11.12 I/O con una buffer cache unificata.

con il vantaggio di evitare il double caching e di permettere al sistema di memoria virtuale di gestire dati del file system. La Figura 11.12 illustra l'uso della buffer cache unificata.

Indipendentemente dalla gestione delle cache per blocchi di disco oppure per pagine (o per entrambi), l'algoritmo LRU è in generale ragionevole per la sostituzione dei blocchi o delle pagine. Tuttavia, l'evoluzione degli algoritmi di gestione delle cache delle pagine usati dal sistema operativo Solaris rivela le difficoltà nella scelta di un algoritmo ottimale. Tale sistema operativo permette ai processi e alla cache delle pagine di condividere la memoria inutilizzata; prima della versione 2.5.1, non si facevano distinzioni tra l'allocazione delle pagine a un processo o alla cache delle pagine, con la conseguenza che un sistema che eseguiva molte operazioni di I/O usava la maggior parte della memoria disponibile per la cache delle pagine. A causa dell'alta frequenza delle operazioni di I/O, quando la memoria libera diventa troppo esigua, il modulo di scansione delle pagine (Paragrafo 9.10.2) sottrae pagine ai processi anziché alla cache delle pagine. In Solaris 2.6 e in Solaris 7 è stata realizzata in forma opzionale la tecnica di *paginazione con priorità*, secondo la quale il modulo di scansione delle pagine dà la priorità alle pagine dei processi rispetto a quelle della cache delle pagine. Il sistema operativo Solaris 8 ha aggiunto un limite prefissato tra pagine dei processi e cache delle pagine per il file system, impedendo a ciascun meccanismo di sottrarre totalmente la memoria all'altro. Con Solaris 9 e 10 sono stati nuovamente modificati gli algoritmi per migliorare l'uso della memoria e contenere il fenomeno della paginazione degenere.

Ci sono altri aspetti che possono influenzare le prestazioni di I/O, come quelli che riguardano la necessità di scritture sincrone o asincrone. Le **scritture sincrone** avvengono nell'ordine in cui le riceve il sottosistema per la gestione del disco e non subiscono la memorizzazione transitoria. Quindi la procedura chiamante prima di proseguire deve attendere che i dati raggiungano l'unità a disco. Nella maggior parte dei casi si usano **scritture asincrone**. Nelle scritture asincrone si memorizzano i dati nella cache e si restituisce immediatamente il controllo alla procedura chiamante. Le scritture dei metadati, tra le altre, possono essere sincrone. I sistemi operativi spesso includono un flag nella chiamata di sistema `open()` per permettere a un processo di richiedere che le operazioni di scrittura si eseguano in modo sincrono. I sistemi di gestione delle basi di dati ad esempio usano questa funzione per realizzare le transazioni atomiche, in modo da assicurare che i dati raggiungano la memoria stabile nell'ordine richiesto.

Alcuni sistemi ottimizzano la cache delle pagine adottando, secondo il tipo d'accesso ai file, differenti algoritmi di sostituzione. Le pagine relative a un file da leggere o scrivere in modo sequenziale non si dovrebbero sostituire nell'ordine LRU, infatti la pagina usata più di recente sarà usata nuovamente per ultima, o forse mai. Gli accessi sequenziali si potrebbero invece ottimizzare con tecniche note come rilascio indietro e lettura anticipata. Il **rilascio indietro** (*free-behind*) rimuove una pagina dalla memoria di transito non appena si verifica una richiesta della pagina successiva; le pagine precedenti con tutta probabilità non saranno più usate e quindi sprecano spazio in memoria di transito. Con la **lettura anticipata** (*read-ahead*) si leggono e si mettono nella cache la pagina richiesta e parecchie pagine successive: è probabile che queste pagine siano richieste una volta terminata l'elaborazione della pagina corrente. Il recupero di questi dati dal disco con un unico trasferimento e la memorizzazione nella cache consentono di risparmiare una quantità di tempo considerevole. La presenza nel controllore di una cache per le tracce non elimina la necessità di adottare la tecnica di lettura anticipata in un sistema multiprogrammato, ciò a causa dell'elevata latenza e del sovraccarico determinato dai tanti piccoli trasferimenti dalla cache per le tracce alla memoria centrale; il ricorso alla *lettura anticipata* è vantaggioso.

La cache delle pagine, il file system e i driver del disco interagiscono in modi interessanti. Quando i dati vengono scritti su un file del disco, le pagine sono memorizzate nella cache, che qui funge da buffer, mentre il driver del disco ordina la propria coda di dati in uscita in base all'indirizzo sul disco. Queste due azioni consentono al driver del disco di minimizzare gli spostamenti della testina del disco e fanno sì che la scrittura dei dati rispetti i tempi ottimali per la rotazione del disco. A meno che le scritture richieste siano sincrone, un processo, per scrivere sul disco, scrive direttamente nella cache: il sistema trasferisce infatti i dati su disco, in maniera asincrona, quando lo ritiene opportuno. Dal punto di vista del processo utente, le scritture sembreranno estremamente rapide. Durante la lettura, l'I/O a blocchi procede a qualche lettura anticipata; ma, in realtà, le scritture si giovano della modalità asincrona molto più delle letture. Per grandi quantità di dati, quindi, la scrittura su disco tramite il file system è spesso più veloce della lettura, contrariamente a ciò che suggerirebbe l'intuizione.

11.7 Ripristino

Poiché i file e le directory sono mantenuti sia in memoria centrale sia nei dischi, è necessario aver cura di assicurare che il verificarsi di un malfunzionamento nel sistema non comporti la perdita di dati o la loro incoerenza. in questo paragrafo vedremo anche come un sistema può essere ripristinato in seguito a un malfunzionamento.

Un crollo del sistema può causare incoerenze tra le strutture dati del file system su disco, come le strutture delle directory, i puntatori ai blocchi liberi e i puntatori agli FCB liberi. Molti file system applicano delle modifiche direttamente a queste strutture. Operazioni comuni come la creazione di un file possono comportare molti cambiamenti strutturali all'interno del file system di un disco. Le strutture delle directory vengono modificate, gli FCB e i blocchi di dati allocati e i contatori liberi per tutti questi blocchi diminuiti. Quando queste modifiche sono interrotte da un crollo del sistema, ne possono derivare incoerenze tra le strutture. Ad esempio, il contatore degli FCB liberi potrebbe indicare che un FCB è stato allocato, ma la struttura della directory potrebbe non avere un puntatore a quel FCB. L'utilizzo della cache che i sistemi operativi adottano per ottimizzare le prestazioni di I/O aggrava questo problema. Alcuni cambiamenti potrebbero andare direttamente sul disco, mentre gli

altri possono finire nella cache. Se i cambiamenti nella cache non raggiungono il disco prima che si verifichi un crollo, è possibile che la situazione peggiori ulteriormente.

Inoltre, anche i bachi nell'implementazione del file system, i controllori del disco, e persino le applicazioni utente possono indurre errori nel file system. I file system hanno svariati metodi per affrontare queste circostanze, a seconda delle strutture dati e degli algoritmi del file system. Ci occuperemo adesso di questi temi.

11.7.1 Verifica della coerenza

Quale che sia la causa degli errori, un file system deve prima scoprire i problemi e poi correggerli. Per scoprire gli errori vengono esaminati tutti i metadati su ogni file system per verificare la coerenza del sistema. Sfortunatamente, questo procedimento richiederà diversi minuti, o addirittura delle ore, e avverrà tutte le volte che il sistema si avvia. In alternativa, un file system può registrare il suo stato all'interno dei metadati del file system. All'inizio di ogni serie di modifiche dei metadati è impostato un bit di stato per indicare che i metadati sono in stato di modifica. Se tutti gli aggiornamenti dei metadati si completano con successo, il file system può azzerare quel bit. Se tuttavia il bit dello stato rimane impostato, entra in funzione un verificatore della coerenza.

Il **verificatore della coerenza** – un programma di sistema come `fsck` in UNIX o `chkdsk` in Windows – confronta i dati delle directory con quelli contenuti nei blocchi dei dischi, tentando di correggere ogni incoerenza. Gli algoritmi di allocazione e di gestione dello spazio libero determinano il genere di problemi che questo programma può riconoscere e con quanto successo riuscirà a risolverli. Ad esempio, se si adotta uno schema di allocazione concatenata con un puntatore da ciascun blocco al successivo, si può ricostruire l'intero file e ricreare il corrispondente elemento nella directory analizzando i blocchi di dati. Diversamente, la perdita di un elemento di una directory in un sistema ad allocazione indicizzata potrebbe essere disastrosa, poiché ogni blocco di dati non contiene alcuna informazione sugli altri blocchi di dati. Per questo motivo, UNIX gestisce tramite cache gli elementi delle directory per le letture, mentre qualsiasi operazione di scrittura di dati che produca l'allocazione di spazio, o altre modifiche dei metadati, è svolta in modo sincrono, prima della scrittura dei corrispondenti blocchi di dati. Naturalmente nuovi problemi possono ancora insorgere se una scrittura sincrona viene interrotta da un crollo di sistema.

11.7.2 File system con registrazione delle modifiche

Spesso nell'informatica si adottano algoritmi e tecnologie anche in aree diverse da quelle per le quali sono stati progettati. È il caso degli algoritmi per il ripristino, sviluppati nell'area dei sistemi di gestione delle basi di dati, basati sulla registrazione delle modifiche, descritti nel Paragrafo 6.9.2. Questi algoritmi sono stati applicati con successo al problema della verifica della coerenza, realizzando i **file system orientati alle transazioni e basati sulla registrazione delle modifiche** (*log-based transaction-oriented file system*), noti anche come **file system annotati** (*journaling file system*).

Si osservi che l'approccio della verifica della coerenza esaminato precedentemente permette in sostanza alle strutture di esibire incoerenze successivamente corrette grazie al ripristino. Questa strategia comporta tuttavia alcuni problemi. Per esempio, l'incoerenza potrebbe rivelarsi irreparabile. Inoltre, il verificatore della coerenza potrebbe non essere in grado di ripristinare le strutture, con una conseguente perdita di file o addirittura di intere directory. Ancora, il verificatore della coerenza potrebbe richiedere l'intervento umano per risolvere i conflitti, il che causa inconvenienti: in mancanza di assistenza da parte di qualcuno, il sistema potrebbe essere inutilizzabile finché una persona non gli indichi come procedere. Infine,

il verificatore della coerenza sottrae risorse al sistema: per controllare un terabyte di dati possono essere necessarie molte ore.

La soluzione a questo problema consiste nell'applicare agli aggiornamenti dei metadati relativi al file system metodi di ripristino basati sulla registrazione delle modifiche. Sia il file system NTFS sia il Veritas usano questo metodo, che è anche opzionale rispetto al file system UFS nel Solaris 7 e nelle versioni successive. In realtà, sta diventando un metodo comune in molti sistemi operativi.

Fondamentalmente, tutte le modifiche dei metadati si annotano in modo sequenziale in un file di registrazione, detto *log*. Ogni insieme di operazioni che esegue uno specifico compito si chiama **transazione**. Una volta che le modifiche sono riportate nel file di registrazione, le operazioni si considerano portate a termine con successo (*committed*) e la chiamata di sistema può restituire il controllo al processo utente, permettendogli di proseguire la sua esecuzione. Nel frattempo, si applicano alle effettive strutture del file system le operazioni scritte nel log, e man mano che si eseguono si aggiorna un puntatore che indica quali azioni sono state completate e quali sono ancora incomplete. Quando un'intera transazione è stata completata, se ne rimuovono le annotazioni dal log, che è in realtà un buffer circolare. I **buffer circolari** scrivono fino alla fine dello spazio disponibile, e poi ricominciano dall'inizio, sovrascrivendo i vecchi contenuti. Naturalmente, si devono prendere delle misure per evitare che dati non ancora salvati siano sovrascritti. Il log si potrebbe mantenere in una sezione separata del file system, o anche in un disco separato. È più efficiente, anche se è più complesso, averlo sotto testine di lettura e scrittura separate, poiché si riducono le situazioni di contesa della testina e i tempi di ricerca (*seek time*).

Se si verifica un crollo del sistema, nel log ci potranno essere zero o più transazioni. Le transazioni presenti non sono mai state ultimate nel file system, anche se il sistema operativo le definisce portate a termine con successo, e quindi si devono completare. Le transazioni si possono eseguire a partire dalla posizione corrente del puntatore fino al completamento, e le strutture del file system rimangono coerenti. L'unico problema che si può presentare è il caso in cui una transazione sia fallita (*aborted*), cioè non sia stata dichiarata terminata con successo prima del crollo del sistema. In questo caso, si devono annullare tutti i cambiamenti che erano stati applicati al file system dalla transazione, di nuovo mantenendo la coerenza del file system. Questo ripristino è tutto ciò che è necessario fare dopo un crollo del sistema, eliminando tutti i problemi concernenti la verifica della coerenza.

Un vantaggio indiretto dell'uso dell'annotazione in un disco degli aggiornamenti dei metadati è che gli aggiornamenti sono molto più rapidi di quelli che si applicano direttamente alle strutture dati nei dischi. La ragione di questo miglioramento sta nel vantaggio, dal punto di vista delle prestazioni, dell'I/O ad accesso sequenziale rispetto a quello ad accesso diretto. Le onerose operazioni di scrittura dei metadati ad accesso diretto e sincrono si sostituiscono con molto meno gravose operazioni di scrittura sincrone ma ad accesso sequenziale nell'area di registrazione delle modifiche di un file system con annotazione delle modifiche. I cambiamenti determinati da quelle operazioni si riportano successivamente in modo asincrono nelle strutture appropriate nei dischi attraverso operazioni di scrittura ad accesso diretto. Il risultato complessivo è un significativo guadagno in termini di prestazioni per le operazioni orientate ai metadati, come la creazione e la cancellazione dei file.

11.7.3 Altre soluzioni

Un'altra alternativa alla verifica della coerenza è impiegata dal file system WAFL di Network Appliance e da ZFS di Sun. Entrambi i sistemi non sovrascrivono mai i blocchi con nuovi dati; al contrario, una transazione scrive tutti i cambiamenti di dati e metadati su nuovi

blocchi. Quando la transazione viene completata, le strutture di metadati che puntavano alla vecchia versione di questi blocchi sono aggiornate in modo da puntare ai nuovi. Il file system può quindi rimuovere i vecchi puntatori e i vecchi blocchi per renderli nuovamente disponibili. Se i vecchi puntatori e i vecchi blocchi vengono mantenuti, viene creata una **snapshot** (*istantanea*), cioè un'immagine del file system prima dell'ultimo aggiornamento. Questa soluzione non dovrebbe richiedere una verifica della coerenza se l'aggiornamento del puntatore è eseguito automaticamente. Ciononostante, il WAFL possiede comunque un controllore della coerenza, perché alcuni tipi di malfunzionamento possono ancora causare un errore nei metadati (per dettagli sul sistema WAFL si veda il Paragrafo 11.9).

In merito alla coerenza del disco lo ZFS di Sun presenta un approccio ancora più innovativo. ZFS non sovrascrive mai i blocchi, come WAFL, ma è in grado di andare oltre fornendo un riassunto delle verifiche su tutti i metadati e i blocchi di dati. Questa soluzione (se combinata con RAID) assicura che i dati siano sempre corretti. ZFS non possiede quindi un controllore della coerenza. (Maggiori dettagli su ZFS sono presenti nel Paragrafo 12.7.6.)

11.7.4 Copie di riserva e recupero dei dati

Poiché si possono verificare malfunzionamenti e perdite di dati anche nei dischi magnetici, è necessario preoccuparsene e provvedere affinché i dati non vadano persi definitivamente. A questo scopo si possono usare programmi di sistema che consentano di fare delle **copie di riserva** (*backup*) dei dati residenti nei dischi in altri dispositivi di registrazione di dati, come dischetti, nastri magnetici, dischi ottici o hard disk supplementari. Il ripristino della situazione antecedente la perdita di un singolo file, o del contenuto di un intero disco, richiederà il **recupero** (*restore*) dei dati dalle copie di riserva.

Al fine di ridurre al minimo la quantità di dati da copiare, è possibile sfruttare le informazioni contenute nell'elemento della directory associato a ogni file. Ad esempio, se il programma di creazione delle copie di riserva sa quando è stata eseguita l'ultima copia di riserva di un file, e se la data di ultima scrittura di quel file, registrata nella directory, indica che il file da quel momento non ha subìto variazioni, non sarà necessario copiare nuovamente il file. Quella che segue è una tipica sequenza di gestione delle copie di riserva.

- **Giorno 1.** Copiatura nel supporto di backup delle copie di riserva di tutti i file contenuti nel disco; detta **copiatura completa**.
- **Giorno 2.** Copiatura su un altro supporto di tutti i file modificati dal Giorno 1; si tratta di una **copiatura incrementale**.
- **Giorno 3.** Copiatura su un altro supporto di tutti i file modificati dal Giorno 2.
  - .
  - .
  - .
- **Giorno *n*.** Copiatura su un altro supporto di tutti i file modificati dal Giorno $n - 1$. Ritorno al Giorno 1.

Il nuovo ciclo può comportare la scrittura delle nuove copie di riserva nel primo insieme di supporti di backup, oppure in un nuovo insieme; in questo modo si ha la possibilità di recuperare il contenuto dell'intero disco iniziando le operazioni di recupero dalla copia di riserva completa e proseguendo con le copie di riserva incrementali. Naturalmente, più grande è n, maggiore sarà il numero di nastri o dischi da leggere per un completo recupero. Un ulteriore vantaggio di questo ciclo di creazione di copie di riserva è la possibilità di recupe-

rare qualsiasi file accidentalmente cancellato durante il ciclo, recuperandolo dalle copie del giorno precedente. La lunghezza del ciclo è un compromesso tra la quantità di supporti per le copie di riserva necessari e il numero di giorni addietro da cui si può compiere un'operazione di recupero. La lunghezza del ciclo è un compromesso fra la quantità di spazio richiesta dalle copie di riserva e il numero di giorni addietro da cui si può compiere un'operazione di recupero. Una possibilità per diminuire la quantità di nastri che è necessario leggere per portare a termine il ripristino è di eseguire inizialmente una copia di riserva completa, per poi eseguire copie dei soli file modificati nei giorni successivi. In questo modo, il ripristino può fondarsi sull'ultima copia incrementale, insieme all'ultima copia completa, senza necessità di altre copie incrementali. Qui il compromesso da tenere a mente è che il numero di file modificati aumenta giornalmente: ogni nuova copia incrementale, quindi, richiede più spazio.

Un utente potrebbe accorgersi dopo molto tempo che si è perso o si è danneggiato un particolare file. Per questa ragione si è soliti pianificare di tanto in tanto una copiatura completa che sarà conservata in modo permanente; quindi il supporto che la contiene non sarà riutilizzato. È inoltre opportuno conservare tali copie di riserva permanenti lontano dalle copie ordinarie per proteggerle dai vari pericoli, ad esempio un incendio che può distruggere il calcolatore e tutte le copie di riserva. Se il ciclo di creazione delle copie di riserva prevede il reimpiego dei mezzi che le contengono, è anche necessario aver cura di non usarli troppe volte: se dovessero logorarsi, potrebbe essere impossibile ripristinare i dati in essi contenuti.

11.8 NFS

I file system di rete sono di particolare interesse; in genere integrano l'intera struttura della directory e l'interfaccia del sistema client. L'NFS è un buon esempio di file system di rete client-server ampiamente usato e ben realizzato, che in questo paragrafo s'impiega per esplorare i particolari che caratterizzano la realizzazione dei file system di rete.

L'NFS è sia una realizzazione sia una definizione di un sistema per l'accesso a file remoti attraverso LAN, o anche WAN. Fa parte dell'ONC+, adottato dalla maggior parte dei distributori del sistema operativo UNIX e in alcuni sistemi operativi per PC. La versione qui descritta fa parte del sistema operativo Solaris, che è a sua volta una versione modificata dello UNIX SVR4, delle stazioni di lavoro Sun e altre architetture. Esso usa i protocolli TCP/IP o i protocolli UDP/IP (secondo la rete di comunicazione). La definizione e realizzazione dell'NFS, nella sua descrizione fornita in questa sede, si accavallano: per ogni descrizione dettagliata si fa riferimento alla versione realizzata dalla Sun; mentre ogni descrizione sufficientemente generale vale anche per la sua definizione.

Ci sono numerose versioni di NFS, l'ultima delle quali è la Versione 4. Esamineremo qui la Versione 3, attualmente la più utilizzata.

11.8.1 Generalità

Nel contesto dell'NFS si considera un insieme di stazioni di lavoro interconnesse come un insieme di calcolatori indipendenti con file system indipendenti. Lo scopo è quello di permettere un certo grado di condivisione tra i file system, su richiesta esplicita, in modo trasparente. La condivisione è basata su una relazione client-server. Un calcolatore può essere, come spesso accade, sia un client sia un server. La condivisione è ammessa tra ogni coppia di calcolatori, anziché essere limitata ai calcolatori aventi la specifica funzione di server. Per as-

sicurare l'indipendenza dei calcolatori, la condivisione di un file system remoto ha effetto esclusivamente sul calcolatore client.

Affinché una directory remota sia accessibile in modo trasparente a un calcolatore particolare, ad esempio C_1, un client di quel calcolatore deve prima eseguire un'operazione di montaggio. La semantica dell'operazione consiste nel fatto che una directory remota si monta in corrispondenza di una directory di un file system locale. Una volta completata l'operazione di montaggio, la directory montata assume l'aspetto di un sottoalbero integrante del file system locale, e sostituisce il sottoalbero che discende dalla directory locale; questa, a sua volta, rappresenta la radice della directory appena montata. La directory remota si specifica come argomento dell'operazione di montaggio in modo esplicito; si deve fornire la locazione (o il nome del calcolatore) della directory remota. Tuttavia, da questo momento in poi gli utenti del calcolatore C_1 possono accedere ai file della directory remota in modo del tutto trasparente.

Per illustrare il montaggio dei file system, si consideri il file system riportato nella Figura 11.13, dove i triangoli rappresentano i sopraccitati sottoalberi di directory. La figura illustra tre file system di calcolatori indipendenti chiamati U, S_1 e S_2. A questo punto, in ogni calcolatore si può accedere solo a file locali. Nella Figura 11.14(a) sono riportati gli effetti del montaggio di `S1:/usr/shared` in `U:/usr/local`. In questa figura è illustrata la visione che gli utenti di U hanno del loro file system. Occorre osservare che, una volta completato il montaggio, essi possono accedere a qualsiasi file che si trovi nella directory `dir1`, ad esempio usando il prefisso `/usr/local/dir1` in U. La directory originale `/usr/local` di quel calcolatore non è più visibile.

Potenzialmente ogni file system o ogni directory in un file system, nel rispetto dei diritti d'accesso, si possono montare in modo remoto in cima a qualsiasi directory locale. Le stazioni di lavoro prive di dischi possono montare i propri file system prelevandoli dai server.

In alcune versioni dell'NFS sono permessi anche i montaggi a cascata; ciò significa che un file system si può montare in corrispondenza di un altro file system montato in modo remoto. Un calcolatore è tuttavia sottoposto ai soli montaggi da esso richiesti.

Montando un file system remoto, il client non acquisisce l'accesso ai file system che erano montati sopra il primo; così, il meccanismo di montaggio non ha la proprietà transitiva. Nella Figura 11.14(b) sono riportati i montaggi a cascata relativi all'esempio precedente. Nella figura è riportato il risultato del montaggio di `S2:/usr/dir2` in `U:/usr/local/dir1`, che è già stato montato in modo remoto da S_1. Gli utenti di U possono ac-

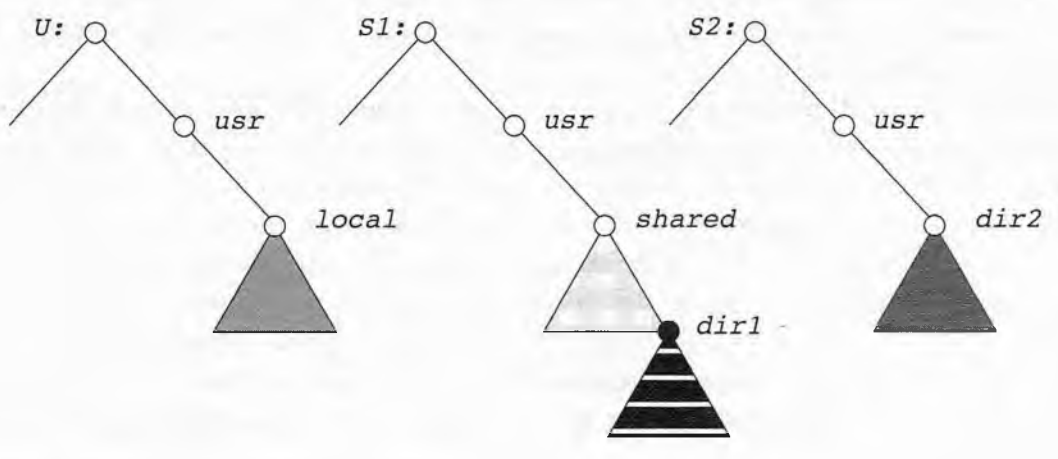


Figura 11.13 Tre file system indipendenti.

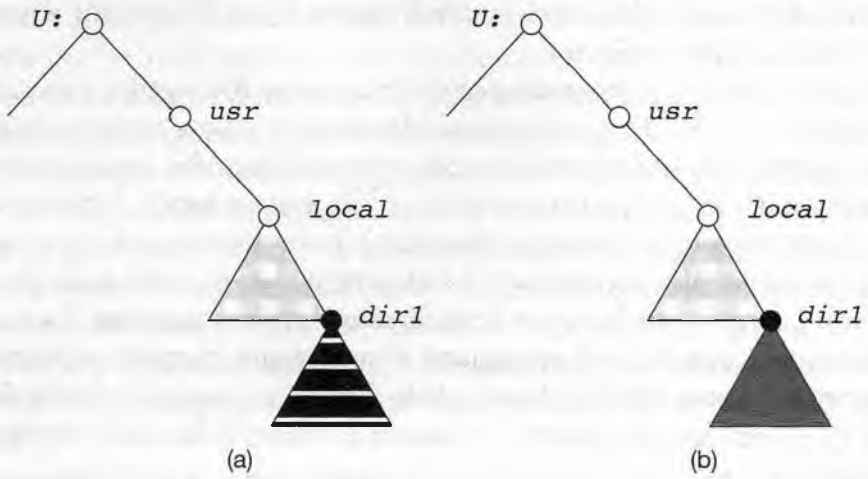


Figura 11.14 Montaggio nell'NFS; (a) montaggio; (b) montaggi a cascata.

cedere ai file di `dir2` usando il prefisso `/usr/local/dir1`. Se si monta un file system condiviso in corrispondenza delle directory iniziali degli utenti di tutti i calcolatori di una rete, un utente può aprire una sessione in qualsiasi stazione di lavoro e prelevare il proprio ambiente di lavoro iniziale. Questa proprietà permette la **mobilità dell'utente**.

Uno degli scopi nella progettazione dell'NFS era quello di operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete. La definizione dell'NFS è indipendente da questi mezzi e quindi incoraggia altre realizzazioni. Questa indipendenza si ottiene usando primitive RPC costruite su un protocollo di rappresentazione esterna dei dati (*external data representation*, XDR) usato tra due interfacce indipendenti dalla realizzazione. Quindi, se il sistema è formato da calcolatori e file system eterogenei adeguatamente interfacciati all'NFS, si possono montare file system di diversi tipi, sia localmente sia in modo remoto.

La definizione dell'NFS distingue tra i servizi offerti da un meccanismo di montaggio e gli effettivi servizi d'accesso ai file remoti. Di conseguenza, per questi servizi si definiscono due protocolli distinti: un protocollo di montaggio e un protocollo per gli accessi ai file remoti, il **protocollo** NFS. I protocolli sono definiti come insiemi di RPC, gli elementi di base usati per realizzare, in modo trasparente, l'accesso remoto ai file.

11.8.2 Protocollo di montaggio

Il **protocollo di montaggio** stabilisce la connessione logica iniziale tra un server e un client. Nella versione di Sun Microsystems ogni calcolatore ha un processo server, esterno al kernel, che esegue le funzioni del protocollo.

Un'operazione di montaggio comprende il nome della directory remota da montare e il nome del calcolatore server in cui tale directory è memorizzata. La richiesta di montaggio si associa alla RPC corrispondente e s'invia al server di montaggio in esecuzione nello specifico calcolatore server. Il server conserva una **lista di esportazione** (la `/etc/dfs/dfstab` nel sistema Solaris, modificabile soltanto da un *superuser*) che specifica i file system locali esportati per il montaggio e i nomi dei calcolatori a cui tale operazione è permessa. La lista può comprendere anche i diritti d'accesso, come la sola scrittura. Per semplificare la manutenzione delle liste di esportazione e delle tabelle di montaggio, si può usare uno schema distribuito di nominazione per contenere queste informazioni e renderle disponibili agli appropriati client.

Occorre ricordare che qualsiasi directory all'interno di un file system esportato si può montare in modo remoto da un calcolatore accreditato. Di conseguenza, un'unità compo-

nente è rappresentata da una directory di questo tipo. Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client un **handle del file** da usare come chiave per ulteriori accessi ai file che si trovano all'interno del file system montato. L'handle del file contiene tutte le informazioni di cui ha bisogno il server per gestire un proprio file. Nei termini dell'ambiente UNIX, l'handle del file è composto da un identificatore di file system e da un numero di *inode* per identificare la directory montata all'interno del file system esportato.

Il server contiene anche una lista dei calcolatori client e delle corrispondenti directory correntemente montate. Questa lista si usa soprattutto per scopi amministrativi, ad esempio per informare i client che un server sta andando fuori servizio. L'aggiunta o la cancellazione di elementi da questa lista sono gli unici modi in cui il protocollo dell'operazione di montaggio può modificare lo stato del server.

Generalmente un sistema ha una configurazione di montaggio predefinita che si stabilisce nella fase d'avviamento (`/etc/vfstab` in Solaris); tale configurazione si può comunque modificare. Oltre alla procedura di montaggio effettiva, il protocollo di montaggio comprende numerose altre procedure, come lo smontaggio e la restituzione della lista d'esportazione.

11.8.3 Protocollo NFS

Il protocollo NFS offre un insieme di RPC per operazioni su file remoti che svolgono le seguenti operazioni:

- ricerca di un file in una directory;
- lettura di un insieme di elementi di una directory;
- manipolazione di collegamenti e di directory;
- accesso ad attributi di file;
- lettura e scrittura di file.

Queste procedure si possono invocare soltanto dopo aver stabilito un handle del file per la directory montata in modo remoto.

L'omissione delle operazioni `open()` e `close()` è intenzionale. Una caratteristica importante dei server NFS è l'*assenza dell'informazione di stato*. I server non conservano informazioni sui loro client da un accesso all'altro. Non esistono parallelismi con la tabella dei file aperti o le strutture di file di UNIX da parte del server, quindi ogni richiesta deve fornire un insieme completo di argomenti, tra cui un identificatore unico di file e uno scostamento assoluto all'interno del file per svolgere le operazioni appropriate. La struttura che ne deriva è robusta; non si devono prendere misure speciali per ripristinare un server dopo un guasto. Per tale ragione, le operazioni sui file devono essere idempotenti. Ciascuna richiesta dell'NFS ha un numero di sequenza, che permette al server di determinare la duplicazione o la mancanza di richieste.

La presenza della suddetta lista di client sembra violare la proprietà dell'assenza di informazione di stato del server. Tuttavia, essa non è essenziale ai fini del corretto funzionamento del client o del server, quindi non è necessario recuperare tale lista dopo il crollo di un server; tale lista può contenere anche dati incoerenti e si considera come un semplice suggerimento.

Un'ulteriore implicazione della filosofia dei server senza informazione di stato e un risultato della sincronia di una RPC consiste nel fatto che i dati modificati, tra cui blocchi indiretti e di stato, si devono riscrivere nei dischi del server prima che i risultati siano riporta-

ti al client. Un client può cioè ricorrere a cache per i blocchi di scrittura, ma quando li invia al server, assume che abbiano raggiunto i dischi del server, che deve scrivere tutti i dati dell'NFS in modo sincrono. In questo modo il crollo di un server e il successivo ripristino saranno invisibili al client; tutti i blocchi che il server gestisce per il client resteranno intatti. La conseguente perdita di prestazioni può essere rilevante poiché si perdono i vantaggi derivanti dall'impiego di una cache. Le prestazioni si possono incrementare impiegando dispositivi di memoria secondaria con una propria cache non volatile (di solito si tratta di memorie alimentate da una batteria), Il controllore del disco riporta che la scrittura nel disco è avvenuta quando la scrittura è avvenuta nella cache non volatile. Essenzialmente, il calcolatore "vede" una scrittura sincrona molto rapida. Questi blocchi restano intatti anche dopo un crollo del sistema, e periodicamente vengono trasferiti da tale memoria stabile al disco.

Di una singola chiamata di procedura di scrittura dell'NFS sono garantite l'atomicità e la non interferenza con altre chiamate di scrittura nello stesso file. Tuttavia, il protocollo NFS non fornisce meccanismi di controllo della concorrenza, e poiché ogni chiamata di scrittura o lettura dell'NFS può contenere non più di 8 KB di dati e i pacchetti UDP sono limitati a 1500 byte, può essere necessario dividere una chiamata di sistema `write()` in diverse RPC di scrittura; quindi, due utenti che scrivono nello stesso file remoto possono riscontrare interferenze nei loro dati. Poiché la gestione di meccanismi di bloccaggio richiede informazioni di stato, si richiede che un servizio esterno all'NFS debba fornire tali meccanismi, come nel caso di Solaris. Gli utenti sanno che per coordinare l'accesso ai file condivisi devono usare meccanismi che sono oltre la portata dell'NFS.

L'NFS è integrato nel sistema operativo tramite un VFS. Per illustrarne l'architettura si può accennare al modo in cui si gestisce un'operazione su un file remoto già aperto (Figura 11.15). Il client inizia l'operazione con un'ordinaria chiamata di sistema. Lo strato del sistema operativo fa corrispondere tale chiamata di sistema a un'operazione del VFS sull'opportuno *vnode*. Lo strato del VFS identifica il file come remoto e invoca l'opportuna procedura dell'NFS. Avviene quindi una chiamata RPC allo strato di servizio dell'NFS nel server remoto. Tale chiamata si reintroduce nello strato del VFS del sistema remoto, che riconosce essere locale e invoca l'appropriata operazione del file system. Questo cammino si ripercorre al contrario per restituire il risultato. Un vantaggio di tale architettura è che il client e il server sono identici; così un calcolatore può essere un client, un server o entrambi. L'effettivo servizio su ciascun server è eseguito da thread del kernel.

11.8.4 Traduzione dei nomi di percorso

La **traduzione dei nomi di percorso** (*path-name translation*) del protocollo NFS prevede l'analisi sintattica di ogni percorso – per esempio `/usr/local/dir1/file.txt` – al fine di estrarre i nomi delle singole directory – nell'esempio: (1) `usr`, (2) `local` e (3) `dir1`. La traduzione dei nomi di percorso si compie suddividendo il percorso stesso in nomi componenti ed eseguendo una chiamata `lookup()` dell'NFS separata per ogni coppia formata da un nome componente e un *vnode* di directory. Quando s'incontra un punto di montaggio, la ricerca di un componente causa una RPC separata al server. Questo schema di attraversamento del nome di percorso è costoso ma necessario, poiché ogni client ha un'unica configurazione del proprio spazio di nomi logico, dettata dai montaggi che ha eseguito. Sarebbe stato molto più efficiente consegnare un nome di percorso a un server e ricevere un *vnode* d'arrivo una volta incontrato un punto di montaggio, tuttavia dovunque può essere presente un altro punto di montaggio per un particolare client sconosciuto al server senza informazione di stato.

Una cache per la ricerca dei nomi delle directory, nel sito del client, conserva i *vnode* per i nomi delle directory remote; in questo modo si accelerano i riferimenti ai file con lo

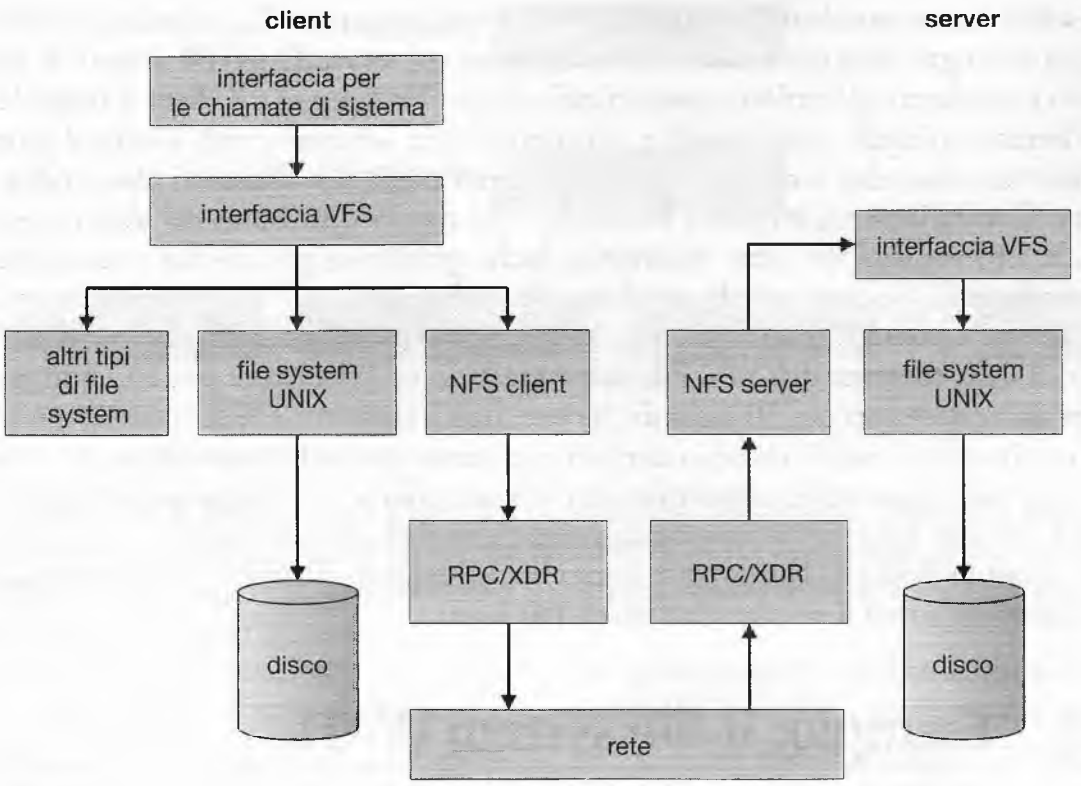


Figura 11.15 Schema dell'architettura dell'NFS.

stesso nome di percorso iniziale. Se gli attributi restituiti dal server non corrispondono agli attributi del *vnode* nella cache, si scarta il contenuto della cache della directory.

Occorre ricordare che nell'NFS è permesso il montaggio di un file system remoto in cima a un altro file system remoto già montato; si tratta del *montaggio a cascata*. Tuttavia, un server non può agire come intermediario tra un client e un altro server. Al contrario, un client deve stabilire un collegamento diretto client-server con il secondo server, montando direttamente la directory richiesta. Quando un client ha un montaggio a cascata, nel caso di un attraversamento di un nome di percorso può essere coinvolto più di un server. Tuttavia, ogni ricerca di componente si compie tra il client originale e alcuni server, perciò quando un client fa una ricerca in una directory sulla quale il server ha montato un file system, il client vede la directory sottostante e non la directory montata.

11.8.5 Funzionamento remoto

Eccetto che per l'apertura e la chiusura dei file, tra le normali chiamate di sistema di UNIX per operazioni su file e le RPC del protocollo NFS esiste una corrispondenza quasi da uno a uno. Quindi, un'operazione remota su un file si può tradurre direttamente nella RPC corrispondente. Dal punto di vista concettuale l'NFS aderisce al paradigma del servizio remoto, ma in pratica si usano tecniche di memorizzazione transitoria e cache per migliorare le prestazioni. Non c'è una corrispondenza diretta tra un'operazione remota e una RPC; le RPC prelevano blocchi e attributi del file che memorizzano localmente nelle cache. Le successive operazioni remote usano i dati nella cache, soggetti a vincoli di coerenza.

Esistono due cache: la cache degli attributi dei file (informazioni sugli *inode*) e la cache dei blocchi di file. Su un file aperto, il kernel fa un controllo rispetto al server remoto per stabilire se deve prelevare o riconvalidare gli attributi nella cache: i blocchi di file nella cache si

usano solo se i corrispondenti attributi nella cache sono aggiornati. La cache degli attributi viene aggiornata ogni volta che arrivano nuovi attributi dal server. Dopo 60 secondi si scartano, in modo predefinito, gli attributi presenti nella cache. Tra il server e il client si usano le tecniche di lettura anticipata (*read-ahead*) e scrittura differita (*delayed-write*). Finché il server non ha confermato che i dati sono stati scritti nei dischi, i client non liberano i blocchi di scrittura differita. Contrariamente a quanto accade nel file system distribuito del sistema operativo Sprite, la scrittura differita viene mantenuta anche quando si apre un file in concorrenza in modi conflittuali. Ne deriva che la semantica UNIX (Paragrafo 10.5.3.1) non si conserva.

Mettere a punto il sistema per migliorare le prestazioni rende difficile caratterizzare la semantica della coerenza dell'NFS. File nuovi creati in un calcolatore possono non essere visibili in altri calcolatori per 30 secondi. Inoltre, non è stabilito se le scritture eseguite in un file in un sito siano visibili anche in altri siti che hanno aperto lo stesso file per la lettura. Le nuove aperture di quel file consentono di osservare solo le modifiche già inviate al server, quindi l'NFS non fornisce né una stretta emulazione della semantica UNIX, né la semantica delle sessioni di Andrew. Nonostante questi inconvenienti, l'utilità e le alte prestazioni del meccanismo ne fanno il sistema distribuito più usato.

11.9 Esempio: il file system WAFL

L'I/O da e verso il disco si riflette significativamente sulle prestazioni del sistema. Di conseguenza, i progettisti devono esercitare grande cura sia nella progettazione sia nell'implementazione del file system. Alcuni file system sono concepiti per finalità generali, ossia sono in grado di offrire prestazioni accettabili e funzionalità adatte a file che differiscono per tipo e dimensione, e a carichi di I/O diversi. Altri sono ottimizzati per compiti specifici, nel tentativo di fornire prestazioni migliori dei sistemi a carattere generale per alcune applicazioni dedicate. Un'ottimizzazione di questo genere è rappresentata dal file system WAFL di Network Appliance. WAFL, acronimo di *write-anywhere file layout* ("modello di file per la scrittura ovunque"), è un file system potente ed elegante, ottimizzato per le scritture casuali.

È utilizzato in esclusiva dai file server di rete prodotti da Network Appliance, coerentemente con la sua vocazione di file system distribuito. Benché sia stato originariamente progettato per i soli NFS e CFS, consente l'invio di file ai client tramite i protocolli NFS, CIFS, `ftp` e `http`. Quando molti client contattano un file server attraverso questi protocolli, le richieste di letture casuali, e ancor di più quelle relative a scritture casuali, aumentano sensibilmente. I protocolli NFS e CIFS trattengono in una cache i dati provenienti dalle operazioni in lettura: per i file server, dunque, la scrittura assume massima importanza.

L'impiego del WAFL presuppone che i file server dispongano di una cache NVRAM per le scritture. I progettisti del WAFL hanno sfruttato il vantaggio di lavorare su un'architettura specifica, con una cache per la memorizzazione stabile dei dati, al fine di ottimizzare il file system per l'I/O ad accesso casuale. Dal momento che è stato concepito espressamente per un'applicazione, uno dei principi che hanno ispirato il WAFL è la facilità d'uso. I suoi autori, inoltre, hanno aggiunto una nuova funzionalità di duplicazione istantanea, per creare a più riprese, come vedremo, diverse copie a sola lettura del file system.

Il file system, pur essendo simile al Berkeley Fast, presenta varie modifiche; è basato sui blocchi e usa gli inode per la descrizione dei file. Ciascun inode contiene 16 puntatori ad altrettanti blocchi (o blocchi indiretti), che appartengono al file descritto dall'inode. Ciascun file system possiede un inode radice. Come si vede nella Figura 11.16, tutti i metadati sono custoditi all'interno di file: un file per gli inode, un altro per la mappa dei blocchi li-

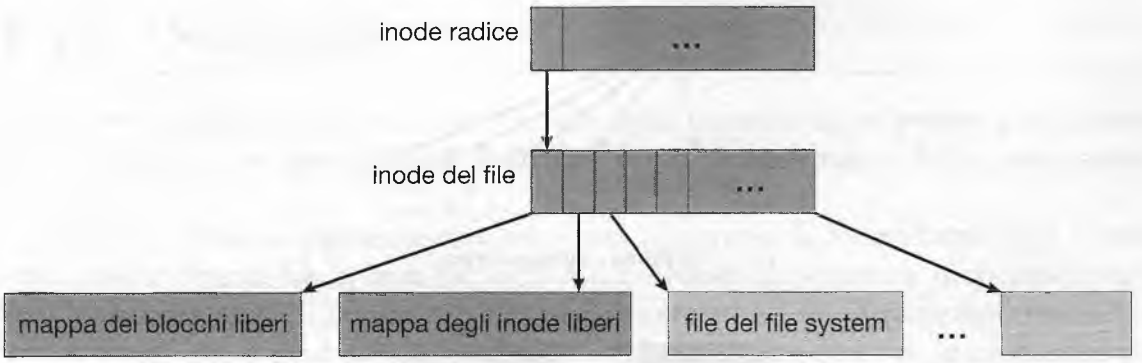


Figura 11.16 File system WAFL.

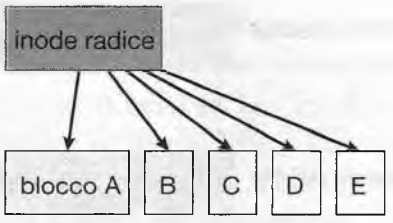
beri e un terzo per la mappa degli inode liberi. Poiché si tratta di file ordinari, i blocchi di dati non hanno un indirizzo predefinito, ma possono risiedere dovunque. Se un file system viene ampliato con l'aggiunta di dischi, esso aumenterà la lunghezza di questi file per i metadati in modo automatico.

Pertanto, i file system WAFL possono essere raffigurati come un albero di blocchi che si dipartono dall'inode radice; per catturarne un'**istantanea** (*snapshot*), WAFL crea una copia dell'inode radice. In seguito a ciò, ogni aggiornamento dei file o dei metadati occuperà blocchi nuovi, anziché sovrascrivere i relativi blocchi esistenti. Il nuovo inode radice punta ai metadati e ai dati nella loro versione aggiornata. Nello stesso tempo, il vecchio inode radice continua a puntare ai blocchi precedenti, non aggiornati, e in tal modo dà accesso a un'immagine del file system che ricalca esattamente il momento in cui era stato fotografato; lo spazio occupato sul disco per questa operazione è davvero esiguo. In sostanza, la copia richiede un supplemento di spazio sul disco equivalente ai soli blocchi che siano stati modificati dal momento in cui si è scattata l'istantanea.

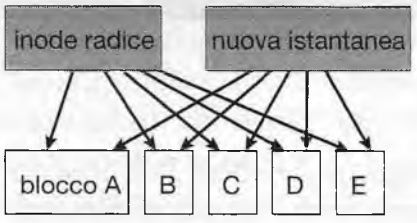
Una differenza di rilievo in confronto a file system più tradizionali è data dalla mappa dei blocchi liberi, che possiede più di un bit per blocco. Si tratta di una maschera di bit che prevede un bit impostato a uno a ogni istantanea che stia utilizzando il blocco. Quando tutte le istantanee che hanno utilizzato un blocco sono cancellate, la maschera di bit associata è formata da zeri, e il blocco può essere riutilizzato. I blocchi usati non sono mai sovrascritti, di modo che le scritture avvengono a gran velocità, visto che le operazioni in scrittura possono sfruttare il blocco libero più vicino alla posizione corrente della testina. Vi sono, nel WAFL, molti altri modi per ottimizzare le prestazioni.

Possono coesistere allo stesso tempo numerose istantanee: se ne può scattare una per ogni ora del giorno e ogni giorno del mese. Gli utenti abilitati, consultando queste istantanee, hanno accesso a ciascuno dei file per come appariva nei vari momenti in cui è stato fotografato. Questa funzionalità è anche utile per le copie di riserva, per le fasi di test, per gestire diverse versioni di un progetto, e altro ancora. Si tratta di un'implementazione molto efficiente, che non richiede neppure di duplicare con la copiatura su scrittura ciascun blocco di dati prima che sia modificato. Altri file system offrono la medesima funzionalità, ma spesso con minor efficienza. Le istantanee del WAFL sono descritte nella Figura 11.17.

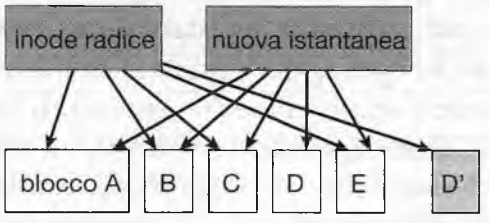
Le versioni più recenti del WAFL permettono istantanee di lettura e scrittura, note come **cloni**. Come le istantanee, anche i cloni sono efficienti, in quanto ne utilizzano le stesse tecniche. Una istantanea di sola lettura cattura lo stato del file system e un clone fa riferimento a quell'istantanea di sola lettura. Eventuali scritture sul clone sono memorizzate in nuovi blocchi e i puntatori del clone vengono aggiornati per fare riferimento ai nuovi blocchi. L'istantanea originale non è modificata, mantenendo così l'immagine del file system



(a) Prima dell'istantanea.



(b) Dopo l'istantanea, prima di modifiche ai blocchi.



(c) Dopo che il blocco D è stato modificato in D'.

Figura 11.17 Istantanee di WAFL.

prima dell'aggiornamento del clone. I cloni possono essere promossi alla sostituzione del file system originale e ciò comporta l'eliminazione dei vecchi puntatori e di ogni vecchio blocco a essi associato. I cloni sono utili per i controlli e gli aggiornamenti; la versione originale rimane infatti immutata e il clone viene cancellato quando il controllo è stato effettuato o quando l'aggiornamento fallisce.

Un'altra caratteristica che deriva naturalmente dall'implementazione del file system WAFL è la **replica**, ovvero la duplicazione e la sincronizzazione di un insieme di dati in un altro sistema attraverso una rete. Per prima cosa, l'istantanea di un file system WAFL viene duplicata in un altro sistema. Quando si scatta un'altra istantanea del sistema sorgente, per aggiornare il sistema remoto è sufficiente mandare tutti i blocchi contenuti nella nuova istantanea. Questi blocchi sono quelli che hanno subìto modifiche nell'intervallo di tempo tra lo scatto delle due istantanee. Il sistema remoto aggiunge questi blocchi al file system e aggiorna i propri puntatori. Il nuovo sistema è dunque un duplicato del sistema sorgente nel momento in cui è stata scattata la seconda istantanea. La ripetizione di questo processo fa sì che il sistema remoto sia (quasi) una copia aggiornata del primo sistema. Le repliche sono utili per il ripristino da eventuali crash. Nel caso in cui il primo sistema sia vittima di una perdita totale di dati, la maggior parte dei suoi dati sarebbe comunque disponibile sulla replica del sistema remoto.

Infine è opportuno ricordare che il file system ZFS di Sun supporta le istantanee, i cloni e un sistema di repliche altrettanto efficienti.

11.10 Sommario

Il file system risiede permanentemente in memoria secondaria, progettata per contenere, permanentemente, una grande quantità di dati. Il più comune mezzo di memoria secondaria è il disco.

I dischi si possono segmentare in partizioni, allo scopo di controllarne l'uso e consentire più, anche diversi, file system per ogni disco. Questi file system si montano in un file system logico per renderli disponibili all'uso. I file system spesso si realizzano secondo una struttura stratificata o modulare: i livelli più bassi hanno a che fare con le caratteristiche fisiche dei dispositivi di memorizzazione; i livelli più alti hanno a che fare con i nomi simbolici e le caratteristiche logiche dei file; i livelli intermedi fanno corrispondere le caratteristiche logiche dei file alle caratteristiche fisiche dei dispositivi.

Ogni tipo di file system può avere diverse strutture e algoritmi. Uno strato VFS consente ai livelli superiori di aver a che fare con ciascun tipo di file system in modo uniforme. Nella struttura della directory del sistema si possono integrare anche i file system remoti sui quali si agisce con le ordinarie chiamate di sistema tramite l'interfaccia del VFS.

Lo spazio dei dischi può essere allocato ai file in tre modi: allocazione contigua, concatenata e indicizzata. L'allocazione contigua può risentire di frammentazione esterna. I file con accesso diretto non si possono gestire con l'allocazione concatenata. L'allocazione indicizzata, infine, può richiedere un notevole carico per il proprio blocco indice. Tali algoritmi si possono ottimizzare in molti modi. Lo spazio contiguo si può allargare attraverso delle estensioni allo scopo di aumentare la flessibilità e ridurre la frammentazione esterna. L'allocazione indicizzata si può realizzare in cluster per incrementare la produttività e ridurre il numero di elementi dell'indice necessari. L'indicizzazione in cluster di grandi dimensioni è analoga all'allocazione contigua con estensioni.

I metodi di allocazione dello spazio libero influenzano anche l'efficienza d'uso dello spazio dei dischi, le prestazioni del file system e l'affidabilità della memoria secondaria. I metodi usati comprendono i vettori di bit e le liste concatenate. Le ottimizzazioni comprendono il raggruppamento, il conteggio e la FAT, che colloca la lista concatenata in un'area contigua.

Le procedure di gestione delle directory devono tener conto dell'efficienza, delle prestazioni e dell'affidabilità. La tabella hash è il metodo usato più spesso; è veloce ed efficiente. Sfortunatamente, il danneggiamento di una tabella o il crollo del sistema possono causare discordanze tra le informazioni contenute nelle directory e il contenuto del disco. Con un verificatore di coerenza si può porre rimedio ai danni. Gli strumenti di creazione di copie di riserva (*backup*) del sistema operativo consentono la copiatura nelle unità a nastro dei dati contenuti nei dischi allo scopo di poterli ripristinare in seguito a perdite dovute a malfunzionamenti dei dispositivi fisici, errori del sistema operativo, o a errori degli utenti.

I file system di rete, come l'NFS, usano un metodo client-server per permettere agli utenti di accedere a file e directory in calcolatori remoti come se fossero in file system locali. Si traducono le chiamate di sistema del client nei protocolli di rete, per poi ritradurle in operazioni del file system nel server. L'interconnessione in reti e gli accessi di più client costituiscono una sfida nei campi della coerenza dei dati e delle prestazioni.

A causa del ruolo fondamentale che i file system hanno nel funzionamento di un sistema, le loro prestazioni e affidabilità sono fondamentali. Tecniche come quelle che impiegano le cache e i file di log migliorano le prestazioni, mentre i log e le tecniche RAID migliorano l'affidabilità. Il sistema WAFL è un esempio di ottimizzazione delle prestazioni per rispondere a uno specifico carico di I/O.

Esercizi pratici

11.1 Prendete in considerazione un file costituito da 100 blocchi. Assumete che il blocco del file di controllo (e il blocco dell'indice, in caso di allocazione indicizzata) sia già in memoria. Calcolate quante operazioni di I/O del disco sono necessarie perché si realizzino strategie di allocazione contigue, concatenate e indicizzate (singolo livello), se, per un blocco, si mantengono le condizioni che seguono. Nel caso di allocazione contigua, assumete che non ci sia spazio di crescita all'inizio, ma solo alla fine. Assumete inoltre che il blocco di informazione da aggiungere sia salvato in memoria.

a. Il blocco viene aggiunto all'inizio.
b. Il blocco viene aggiunto al centro.
c. Il blocco viene aggiunto alla fine.
d. Il blocco viene rimosso dall'inizio.
e. Il blocco viene rimosso dal centro.
f. Il blocco viene rimosso dalla fine.

11.2 Quali problemi potrebbero verificarsi se un sistema permettesse di montare il file system simultaneamente in più di una posizione?

11.3 Perché la mappa di bit per l'allocazione dei file deve essere conservata nella memoria di massa, e non nella memoria principale?

11.4 Considerate un sistema che supporta le strategie di allocazione contigua, concatenata e indicizzata. Quali criteri dovrebbero essere impiegati per decidere quale strategia è migliore per un dato file?

11.5 Un problema dell'allocazione contigua consiste nel fatto che l'utente deve preallocare abbastanza spazio per ogni file. Se il file diventa più grande dello spazio che gli è stato allocato, devono essere intraprese delle azioni specifiche. Questo problema può essere risolto definendo una struttura di file che consiste in un'area inizialmente contigua (di una dimensione specificata.) Se un'area viene colmata, il sistema operativo definisce automaticamente un'area di overflow collegata all'area inizialmente contigua. Se l'area di overflow viene riempita, si alloca una seconda area di overflow. Paragonate questa implementazione con le versioni standard contigue e concatenate.

11.6 Come possono le cache contribuire a migliorare le prestazioni? Perché i sistemi non utilizzano un maggior numero di cache, oppure cache più grandi, se esse sono così utili?

11.7 Perché è vantaggioso per l'utente che il sistema operativo allochi dinamicamente le proprie tabelle interne? Quali sono le conseguenze negative in cui incorrono i sistemi operativi che si comportano in questo modo?

11.8 Spiegate come lo strato VSF permetta al sistema operativo di supportare facilmente diversi tipi di file system.

Esercizi

11.9 Considerate un file system che adopera un metodo di allocazione contigua modificato, comprensivo di estensioni. Un file contiene diverse estensioni, ognuna delle quali corrisponde a un insieme contiguo di blocchi. Un aspetto cruciale, per tali sistemi, è il grado di variabilità nella dimensione delle estensioni. Quali sono i vantaggi e gli svantaggi nel caso che:

a. tutte le estensioni siano della stessa grandezza, che è predeterminata;
b. le estensioni possono essere di qualunque misura e sono allocate dinamicamente;
c. le estensioni variano tra poche misure fisse, che sono predeterminate.

11.10 Quali vantaggi offre la variante dell'allocazione concatenata che utilizza una FAT per collegare i blocchi di un file?

11.11 Considerate un sistema che tenga traccia dello spazio libero in una lista apposita.

a. Supponete di perdere il puntatore alla lista. Il sistema è in grado di ricostruire la lista dello spazio libero? Argomentate la vostra risposta.
b. Considerate un file system simile a quello con allocazione indicizzata impiegato da UNIX. Quante operazioni di I/O del disco potrebbero essere necessarie per leggere i contenuti di un piccolo file locale posto in `/a/b/c`? Si assuma che nessuno dei blocchi del disco sia stato memorizzato nella cache.
c. Proponete una soluzione che impedisca la perdita del puntatore dovuta a un guasto della memoria.

11.12 In alcuni file system è possibile allocare lo spazio sul disco con diverse granularità. Un file system, ad esempio, può allocare 4 KB di spazio sul disco scegliendo un blocco unico da 4 KB oppure otto blocchi da 512 byte. In quale modo si può trarre vantaggio da questa flessibilità per migliorare le prestazioni? Quali modifiche si dovrebbero introdurre nel modello di gestione dello spazio libero per includervi questa caratteristica?

11.13 Discutete di come i tentativi di ottimizzazione delle prestazioni dei file system potrebbero generare difficoltà nella salvaguardia della coerenza dei sistemi, allorquando si verifichi un crash di sistema.

11.14 Considerate un file system in un disco con dimensioni dei blocchi logici e fisici di 512 byte. Supponete che le informazioni su ciascun file siano già in memoria. Per ciascuno dei tre metodi di allocazione (contigua, concatenata e indicizzata) fornite le risposte alle seguenti domande:

a. dite come in questo sistema si fanno corrispondere gli indirizzi logici agli indirizzi fisici (per l'allocazione indicizzata supponete che la lunghezza di un file sia sempre inferiore a 512 blocchi);
b. se l'ultimo accesso è stato fatto al blocco 10 (posizione corrente), dite quanti blocchi fisici si devono leggere dal disco per accedere al blocco logico 4.

11.15 Considerate un file system che utilizza degli inode per rappresentare i file. I blocchi del disco hanno una dimensione di 8 KB e un puntatore a un blocco del disco richiede 4 byte. Questo file system ha 12 blocchi diretti sul disco, ma anche blocchi indiretti singoli, doppi e tripli. Qual è la dimensione massima di file che può essere memorizzata nel sistema?

11.16 In un dispositivo di memoria si può eliminare la frammentazione ricompattando le informazioni. I tipici dispositivi a disco non dispongono di registri di rilocazione o registri di base (come quelli usati per compattare la memoria); in questa situazione dite come si possono rilocare i file. Fornite tre motivi per i quali la ricompattazione e la rilocazione dei file vengono spesso evitate.

11.17 In quali situazioni l'uso della memoria come disco RAM sarebbe più utile rispetto al suo uso come cache del disco?

11.18 Considerate l'integrazione seguente a un protocollo per l'accesso remoto ai file. Ciascun client gestisce una cache per i nomi, in cui memorizza le traduzioni del nome di un file nel corrispondente handle del file. Quali questioni dovrete tenere in considerazione nel realizzare la cache per i nomi?

11.19 Spiegate perché l'annotazione degli aggiornamenti ai metadati assicura il ripristino di un file system dopo un crollo del sistema.

11.20 Considerate il seguente schema di creazione di copie di riserva.

- **Giorno 1.** Copiatura nel supporto di backup delle copie di riserva di tutti i file contenuti nel disco.
- **Giorno 2.** Copiatura in un altro supporto di tutti i file modificati dal giorno 1.
- **Giorno 3.** Copiatura in un altro supporto di tutti i file modificati dal giorno 1.

Tale schema differisce dalla sequenza data nel Paragrafo 11.7.2 per il fatto che tutte le copiature riguardano tutti i file modificati dopo la prima copiatura completa. Dite quali sono i vantaggi di questo sistema rispetto a quello del Paragrafo 11.7.2 e se le operazioni di recupero sono più semplici o più difficili. Motivate le vostre risposte.

11.11 Note bibliografiche

Il sistema FAT dell'MS-DOS è spiegato in [Norton e Wilton 1988] e la descrizione del sistema OS/2 si trova in [Iacobucci 1988]. Questi sistemi operativi usano le famiglie di CPU Intel 8086 ([Intel 1985b], [Intel 1985a], [Intel 1986] e [Intel 1990]). I metodi di allocazione di IBM sono descritti in [Deitel 1990]. L'organizzazione interna del sistema BSD UNIX è ampiamente trattata in [McKusick et al. 1996]. [McVoy e Kleiman 1991] presentano ottimizzazioni di questi metodi realizzate per Solaris. Il file system di Google è descritto in [Ghemawat et al. 2003]. Si può trovare FUSE all'indirizzo http://fuse.sourceforge.net/.

L'allocazione dello spazio dei dischi per i file basata sul sistema buddy (gemellare) è trattata in [Koch 1987]. Uno schema di organizzazione dei file che garantisce il recupero dei dati con un solo accesso è trattato in [Larson e Kajla 1984]. I modelli di accesso ai file con registrazione per migliorare sia le prestazioni sia la coerenza sono stati presentati da [Rosenblum e Ousterhout 1991], [Seltzer et al. 1993], [Seltzer et al. 1995]. Gli algoritmi sugli alberi bilanciati sono discussi (insieme a molto altro materiale) in [Knuth 1998] e in [Cormen et al. 2001]. Il codice sorgente ZFS per le mappe di spazio è disponibile su http://src.opensolaris.org/source/xref/onnv/onnvgate/usr/src/uts/common/fs/zfs/space_map.c.

L'uso delle memorie cache nella gestione dei dischi è trattato da [McKeon 1985] e [Smith 1985]; l'uso della cache nel sistema operativo sperimentale Sprite, è descritto in [Nelson et al. 1988]. Analisi generali sulle tecnologie delle memorie di massa si trovano in [Chi 1982] e [Hoagland 1985]. L'opera di [Folk e Zoellick 1987] concerne le varie strutture di file. [Silvers 2000] analizza la realizzazione della cache delle pagine nel sistema operativo NetBSD.

[Sandberg et al. 1985], [Sandberg 1987], [Sun 1990] e [Callaghan 2000] trattano il file system di rete NFS. Le caratteristiche dei carichi di lavoro nei file system distribuiti sono studiate da [Baker et al. 1991]. Architetture basate sulla registrazione per file system di rete sono proposte in [Hartman e Ousterhout 1995], e da [Thekkath et al. 1997]. [Vahalia 1996] e [Mauro e McDougall 2001] descrivono l'NFS e il file system del sistema operativo UNIX, l'UFS. [Solomon 1998] descrive il file system NTFS del sistema operativo Windows NT. Il file system *Ext2* del sistema operativo Linux è stato trattato da [Bovet e Cesati 2002], mentre [Hitz et al. 1995] si è occupato del file system WAFL. Per approfondimenti riguardanti ZFS, si consulti http://www.opensolaris.org/os/community/ZFS/docs.