Lezione 12

Realizzazione del file system

Sistemi operativi

5 giugno 2012

Marco Cesati

System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

Organizzazione e realizzazione dei file system

1. Protezione dei file
2. Montaggio di un file system
3. Componenti del SO per gestire i file system
4. Il file system virtuale
5. Il file system su disco
6. L'allocazione dei blocchi

Realizzazione del file system

Marco Cesati

Schema della lezione
Protezione
Montaggio
Componenti del SO
File system virtuale
File system su disco
Allocazione dei blocchi

Protezione dei file

I meccanismi di protezione dei file sono essenziali per

- la robustezza del sistema (bug e guasti con effetto limitato)
- la riservatezza dei dati degli utenti

I metodi più generali per definire la protezione dei file sono basati sulla *matrice di accesso*: per ogni file ed ogni attore (utente, processo, o componente), la matrice indica le operazioni consentite

Implementazione reali sono basate su:

- *Access Control Lists* (*ACL*): associate a ciascun file e contenenti le autorizzazioni
- *Bit di accesso*: associati a ciascun file, per poche operazioni e pochi attori
- sofisticati programmi del kernel che implementano un controllo d'accesso obbligatorio per tutti i componenti del sistema, quindi anche per i file (esempio: NSA's SE Linux)

Esempio: bit di accesso per file regolare di Unix

```
$ ls -l fileA
-rwxr-xr-- 2 caio users 1046 2011-06-12 16:33 fileA
```

- `fileA`: il nome del file
- `16:33`: ora ultima modifica
- `2011-06-12`: data ultima modifica
- `1046`: dimensione del file
- `users`: gruppo di utenti proprietario
- `caio`: utente proprietario
- `2`: contatore di riferimento per hard link
- `-rwxr-xr--`: tipo file e diritti di accesso
  - `-`: file regolare
  - `rwx`: lettura, scrittura ed esecuzione per utente proprietario
  - `r-x`: lettura ed esecuzione per gruppo proprietario
  - `r--`: lettura per tutti gli altri

Esempio: bit di accesso per directory di Unix

```
$ ls -ld dirA
drwxr-x--x 5 caio users 4096 2011-06-12 16:33 dirA
```

- `dirA`: il nome della directory
- `16:33`: ora ultima modifica
- `2011-06-12`: data ultima modifica
- `4096`: dimensione della directory (voci e attributi)
- `users`: gruppo di utenti proprietario
- `caio`: utente proprietario
- `5`: numero di directory figlie + 2 ("`.`" e "`..`")
- `drwxr-x--x`: tipo file e diritti di accesso
  - `d`: file di tipo directory
  - `rwx`: listing, modifica e attraversamento per utente proprietario
  - `r-x`: listing e attraversamento per gruppo proprietario
  - `--x`: attraversamento per tutti gli altri

Partizioni dei dischi

Ogni disco rigido è tipicamente suddiviso in diverse zone:

- *Settore di bootstrap* o MBR, Master Boot Record: contiene un piccolo programma caricato dal firmware in avvio
  - Ha lo scopo di caricare in memoria un altro programma (*bootloader secondario*) che a propria volta caricherà in memoria il kernel del SO
- *Tabella delle partizioni*: una struttura di dati che elenca tutte le partizioni del disco, insieme alla loro posizione, dimensione e contenuto
- *Partizione*: zona contigua del disco utilizzata per memorizzare un file system, un'area di swap, i dati di un DBMS, ...

Le limitazioni intrinseche del SO MS-DOS hanno causato l'introduzione di complicazioni quali i concetti di *partizione di boot* e di *partizioni primarie, estese e logiche*

Un disco può anche essere utilizzato senza alcuna struttura logica interna (*raw disk*)

Montaggio di un file system

- Un file system è generalmente memorizzato su un intero disco oppure su una partizione di un disco
- Ciascun file system definisce una propria directory radice, attraverso la quale è possibile raggiungere tutti gli altri file
- In generale nel SO sono utilizzati diversi file system allo stesso tempo
- Nei SO derivati dall'MS-DOS, il file system è identificato da una lettera di unità: `C:\UNADIRECTORY\UNFILE`
- Nei SO della famiglia Unix l'albero delle directory è unico e ciascun file system è montato su di esso
  - Il *punto di montaggio* (*mount point*) è una directory il contenuto è completamente nascosto finché il file system non è smontato
  - È possibile anche montare lo stesso file system in modo che appaia in diversi punti dell'albero di sistema
  - I nomi completi dei file possono contenere diversi mount point, ma questo è trasparente per utenti e programmi
  - Esempio: `/mnt/floppy/fileA`

Montaggio di un file system (2)

- Per montare / smontare un file system in Unix si utilizzano
  - le chiamata di sistema `mount()` e `umount()`
  - i comandi di sistema `mount` e `umount`
- Tipicamente in Unix vengono montati molti file system differenti, ad esempio:

Partizione	Mount point	
`/dev/sda1`	`/`	Radice
`/dev/sda2`	`/home`	Directory degli utenti
	`/sys`	File system virtuale
`/dev/sdb`	`/mnt/usbpen`	Memoria flash

"`/dev/sda1`" è il nome di un file di tipo dispositivo (device file) corrispondente alla prima partizione del primo disco SCSI o SATA

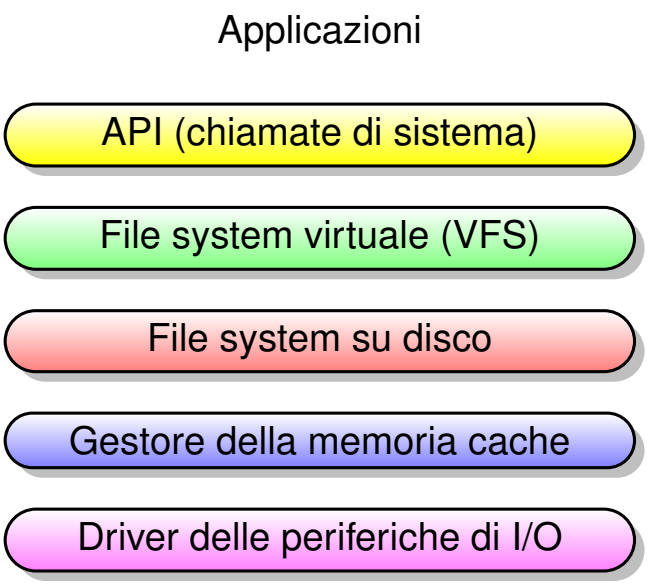
Implementazione dei file system

L'implementazione dei file system in un SO è di importanza cruciale:

- I file sono l'unità di informazione di base sia per il SO che per l'utente
- L'utente utilizza i file interagendo con il file system
- I dischi moderni hanno grande capacità (migliaia di terabyte) e quindi contengono un numero enorme di file
- Un SO può utilizzare un gran numero di file system di tipo differente

L'implementazione dei file system è uno dei fattori che maggiormente caratterizza un SO rispetto agli altri, sia in termini di metodo di utilizzo che di prestazioni

Componenti del SO per la gestione dei file system



Componenti del SO per la gestione dei file system (2)

File system virtuale (VFS)

Componente del SO che consente di

- nascondere i dettagli implementativi dei file system specifici utilizzati nel sistema
- assegnare identificatori univoci a tutti i file del sistema, a prescindere dal file system di appartenenza

File system su disco

Componente del SO che gestisce ed organizza i file in memoria secondaria

Gestore della memoria cache

Componente del SO che consente di utilizzare una porzione della memoria centrale come memoria "di transito" per i dati in memoria secondaria

File system virtuale

Il file system virtuale (VFS) è presente in tutti i SO che consentono di utilizzare file system di diverso tipo e natura

Consideriamo una chiamata di sistema `open()`:

- Identificatore del file passato alla `open()`:
  - con VFS, generale ed astratto ("*vnode*")
  - senza VFS, specifico per ogni file system e non univoco
- Implementazione della chiamata di sistema:
  - con VFS, generale ed unica per tutti i file system
  - senza VFS, diversa per ogni file system

Come si può realizzare una implementazione di una chiamata di sistema che funzioni per qualunque tipo di file system?

È necessario utilizzare una metodologia di programmazione che "nasconda" i dettagli implementativi che non debbono essere necessariamente noti a livello di interfaccia (API)

File system virtuale (2)

Un metodo comune per realizzare programmi indipendenti da dettagli di livello più basso è la programmazione ad oggetti

Un *oggetto* è una struttura di dati che definisce, oltre ai dati stessi, i metodi attraverso i quali interagire con i dati

Quindi per realizzare il VFS tramite oggetti è necessario scrivere il SO con un linguaggio di programmazione OO? ***No!***

La metodologia di programmazione non è legata in modo essenziale al linguaggio di programmazione utilizzato (ad esempio, C++ invece che C)

Esempio: il VFS di Linux

Il VFS del SO Linux definisce quattro tipi di oggetti ("classi") principali:

- Oggetto superblock: rappresenta un file system
- Oggetto inode: rappresenta un file "fisico"
- Oggetto file: rappresenta un "file aperto"
- Oggetto dentry: rappresenta una voce di una directory

Ciascuno di questi oggetti è costituito da una struttura del linguaggio C contenente sia dati che puntatori a funzioni (metodi)

L'implementazione dei metodi (ossia il valore assegnato al puntatore) è specifico per ciascun file system

Una chiamata di sistema del VFS:

- esegue operazioni di carattere generale (ad es., controllo dei diritti di accesso, allocazione degli oggetti, ecc.)
- invoca quando opportuno il metodo di un oggetto specifico

Esempio: il VFS di Linux (2)

Ad esempio la struttura C `struct file` include:

`f_count`	contatore di riferimento
`f_flags`	flag di apertura
`f_lock`	lock per accesso condiviso
`f_mode`	diritti di accesso
`f_path`	oggetto dentry del file
`f_pos`	posizione corrente nel file
`f_op`	puntatore ai metodi
`f_owner`	utente proprietario

⋮

`f_op` punta ad una tabella `struct file_operations`:

`lock`	applica un file lock
`llseek`	aggiorna la posizione corrente
`mmap`	crea una regione mappata sul file
`open`	operazioni per l'apertura del file
`read`	lettura dal file
`release`	ultima chiusura del file aperto
`write`	scrittura nel file

⋮

Il file system su disco

Consideriamo un qualunque file system su disco, ad esempio NTFS (MS Windows NT) oppure Ext3 (Linux): esso deve gestire ogni aspetto della registrazione delle informazioni in memoria secondaria

In particolare è necessario:

- Organizzare il file system stesso:
  - Posizione in memoria secondaria delle strutture di dati
  - Gestione dello spazio libero
  - Controlli di coerenza delle strutture di dati
- Rappresentare i file e le directory:
  - Gestione dei riferimenti ai file (nomi di percorso)
  - Gestione dei metadati (tipo e contenuto, diritti di accesso, dimensione, . . .)
  - Registrazione dei blocchi di dati del file

Le strutture di dati del file system sono memorizzate in memoria secondaria; tuttavia quando il file system è in uso le strutture di dati possono anche risiedere in memoria centrale

Il blocco di controllo del volume

La struttura di dati principale di un file system è il *blocco di controllo del volume* (VCB)

Contiene informazioni di tipo generale, ad esempio:

- La posizione in memoria secondaria delle altre strutture di dati del file system
- La dimensione del blocco (unità di allocazione e trasferimento dei dati)
- Il numero totale di blocchi del file system
- Il numero di blocchi utilizzati e liberi
- La data dell'ultimo montaggio del file system

Il blocco di controllo del volume è chiamato

- in Ext3: superblocco (superblock)
- in NTFS: MFT, o Master File Table

Il blocco di controllo del file

Ciascun file in un file system è descritto da una struttura di dati chiamata *blocco di controllo del file* (FCB)

Contiene informazioni generali sul file, ad esempio:

- Dimensione del file in byte ed in blocchi
- Utente e gruppo di utenti proprietari del file
- Diritti di accesso al file
- Data e ora di creazione e ultimi accessi in lettura e/o scrittura

Il blocco di controllo del file

- in Ext3 è memorizzato in un blocco chiamato *inode*
- in NTFS è memorizzato all'interno della MFT per mezzo di un record di un data base relazionale

Rappresentazione delle directory

Le directory possono essere assimilate a file contenenti una lista dei nomi (riferimenti) relativi a file e directory incluse

- I metadati relativi alle directory possono essere memorizzati esattamente come quelli relativi ai file regolari
- Ad esempio in Ext3 le directory sono file contenenti un vettore di strutture di dati, ciascuna delle quali rappresenta una voce

Tuttavia per ragioni di efficienza molti file system implementano le directory in modo radicalmente diverso dai file regolari

- Ad es. in NTFS le directory sono memorizzate nella MFT
- Le voci della directory possono essere organizzate come
  - lista lineare (semplice e rapida per directory con poche voci)
  - tabella hash
  - albero di ricerca (ad es.,B-tree)

Allocazione dei blocchi dei file

Un file system deve gestire la memorizzazione dei blocchi di dati dei file in modo da

- sfruttare al meglio lo spazio disponibile sul disco
- rendere l'accesso al file quanto più efficiente possibile

Tre alternative principali per realizzare l'allocazione dei blocchi:

- Allocazione contigua: ciascun file occupa un insieme di blocchi contigui nel disco
- Allocazione concatenata: i blocchi di un file non sono contigui nel disco; ciascun blocco contiene un puntatore al blocco successivo
- Allocazione indicizzata: i blocchi di un file non sono contigui nel disco; i puntatori ai blocchi sono memorizzati in strutture di dati attestate nel FCB

Allocazione contigua

- I blocchi di un file system possono essere associati ad un indice progressivo detto *numero di blocco*
  - Conoscendo la dimensione del blocco e la posizione iniziale del file system sul disco si può ricavare dal numero di blocco i corrispondenti numeri di settore del disco
- Con uno schema di allocazione contigua i numeri di blocco dei dati di un file sono sempre progressivi
  - Nel FCB è sufficiente memorizzare il numero di blocco iniziale e la lunghezza del file

Qual è il vantaggio dell'allocazione contigua?

L'accesso sequenziale al file è semplice e rapido: è sufficiente un solo riposizionamento della testina del disco (*seek*)

Anche l'accesso diretto è facile da implementare, poiché l'associazione tra numero di blocco logico del file ed il numero di blocco del file system è immediata

Quali sono gli svantaggi dell'allocazione contigua?

Allocazione contigua (2)

Gli svantaggi dell'allocazione contigua:

- Quanto spazio allocare ad un file appena creato?
  - Se troppo poco: inefficienza se il file cresce di dimensioni (si deve copiare il file in una nuova posizione)
  - Se troppo grande: rischio di spreco dello spazio disco a causa della frammentazione interna
- Frammentazione esterna: la partizione potrebbe avere molti blocchi liberi, ma nessuna porzione contigua sufficientemente grande per memorizzare un certo file

Uno schema adottato in pratica utilizza le estensioni:

- Ad un nuovo file viene assegnato uno spazio contiguo predeterminato
- Se il file cresce viene aggiunta un'altra porzione di spazio contiguo (*estensione*)
- Esempi di FS con estensioni: Btrfs (Linux), Ext4 (Linux), HFS+ (Apple), JFS (AIX), NTFS (Windows), UDF (dischi ottici)

Allocazione concatenata

- Ogni file è composto da una lista concatenata di blocchi
- Ciascun blocco contiene (almeno) il numero del blocco successivo nella lista
- Il FCB contiene il numero del primo blocco, la lunghezza della lista e il numero dell'ultimo blocco

Qual è il vantaggio della allocazione concatenata?

Non esiste problema di frammentazione esterna, e lo spazio sprecato per la frammentazione interna è limitato

Quali sono gli svantaggi?

- Tempi di accesso sequenziale significativi (l'accesso ad ogni blocco richiede un riposizionamento della testina)
- Tempi di accesso diretto eccessivi (richiede una lettura dell'intera porzione di file che precede il blocco acceduto)
- Spazio non disponibile all'interno dei blocchi, e complicazione nella gestione delle letture e scritture
- Scarsa affidabilità: se un blocco si corrompe, da quel blocco il file non è non più accessibile

Tabella di allocazione dei file (FAT)

Variante del metodo di allocazione concatenata: utilizzo della tabella di allocazione dei file (FAT, File Allocation Table)

- Adottato nei file system dei SO MS-DOS e OS/2
- Una sezione della partizione è occupata dalla FAT
- La FAT è un vettore indicizzato dal numero di blocco del file system
- Ogni voce contiene
  - zero se il blocco è libero;
  - il numero del blocco successivo nel file, ovvero
  - un valore convenzionale EOF se il blocco è l'ultimo di un file
- Il numero del primo blocco di un file è memorizzato nella corrispondente voce di directory

Qual è il principale vantaggio della FAT rispetto alla allocazione concatenata classica?

Se il disco non è troppo grande la FAT può essere copiata una volta per tutte in memoria centrale, e ciò consente l'accesso diretto al singolo blocco di un file

Allocazione indicizzata

Nello schema della allocazione indicizzata i blocchi di dati di un file non sono contigui nel file system, ed i numeri di blocco sono memorizzati in una struttura di dati chiamata *indice*

La differenza con lo schema che utilizza la FAT è che

- la FAT è una struttura di dati globale per l'intero file system
- la struttura di dati indice è specifica per ciascun file

Quali sono i vantaggi della allocazione indicizzata?

- Rispetto alla allocazione contigua ha minori problemi di frammentazione esterna ed interna
- Rispetto alla allocazione concatenata consente di realizzare efficientemente l'accesso diretto
- Rispetto ai file system FAT è efficiente anche con dischi di grandi dimensioni

Quali sono gli svantaggi della allocazione indicizzata?

- L'occupazione di spazio dovuta ai numeri di blocco è maggiore (anche i file piccoli hanno un proprio indice)
- Gli accessi sequenziali sono lenti (rispetto all'alloc. seq.)

Allocazione indicizzata (2)

Poiché ogni file possiede un proprio indice, questo dovrebbe piccolo; al tempo stesso deve essere possibile gestire i file di grandi dimensione

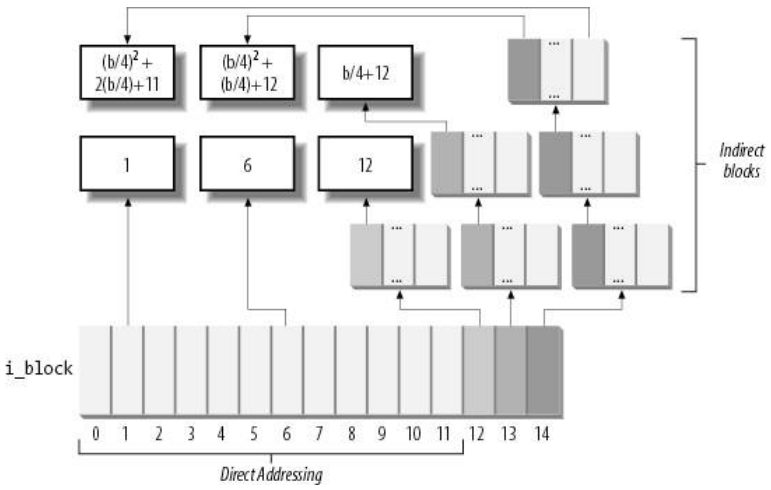
Tre possibili schemi per la struttura indice:

- Indice concatenato: l'indice è un singolo blocco, e contiene anche il puntatore ad un eventuale blocco indice successivo
- Indice a più livelli: un blocco indice contiene i numeri di blocco degli indici di secondo livello, e così via; l'ultimo livello contiene i numeri di blocco dei dati del file
- Indice combinato: il blocco indice contiene
  - un certo numero di blocchi di dati del file (*blocchi diretti*)
  - il numero di un *blocco indiretto*, contenente numeri di blocchi di dati
  - il numero di un *blocco indiretto doppio*, contenente numeri di blocchi indiretti
  - il numero di un *blocco indiretto triplo*, contenente numeri di blocchi indiretti doppi

Realizzazione del file system
Marco Cesati

Schema della lezione
Protezione
Montaggio
Componenti del SO
File system virtuale
File system su disco
Allocazione dei blocchi

Esempio: allocazione indicizzata in Ext3



Source: D.P. Bovet, M. Cesati, Understanding the Linux kernel, O'Reilly, 2005

- b: dim. blocco
- Numeri blocco da 4 byte
- Blocchi diretti: da 0 a 11

- Blocchi con singola indirezione: da 12 a $L + 11$ ($L = b/4$)
- Blocchi con doppia indirezione: da $L + 12$ a $L^2 + L + 11$
- Blocchi con tripla indirezione: da $L^2 + L + 12$ a $L^3 + L^2 + L + 11$
- Se $b = 4096$, $L = 1024$ e la massima dimensione di un file è di oltre 4 TB (in realtà in Ext3 la massima dim. di un file è 2 TB)

Gestione dello spazio libero

Tra i compiti del file system c'è quello di tenere traccia dei blocchi inutilizzati all'interno del disco (partizione)

Tra i possibili schemi vi sono:

- Vettore o mappa di bit: lo stato è rappresentato da un bit (libero o occupato); la ricerca di un blocco libero comporta la scansione della mappa di bit
  - Efficiente per dischi non grandi in cui la mappa di bit può essere mantenuta in memoria centrale
- Lista concatenata: il VCB memorizza il numero del primo blocco libero, e ciascun blocco libero memorizza il numero del successivo
  - Allocare diversi blocchi in una volta sola è poco efficiente
- Raggruppamento: come la lista concatenata, ma il primo blocco libero contiene i numeri di n blocchi liberi; l'ultimo di tali blocchi contiene i numeri di altri blocchi liberi, ecc.
- Conteggio: nei blocchi liberi viene memorizzata una struttura ordinata che descrive le aree libere contigue del disco o partizione