Lezione 11

Interfaccia del file system

Sistemi operativi

29 maggio 2012

Marco Cesati

System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

L'interfaccia del file system:

1. Il sistema di archiviazione
2. I file e la loro gestione
3. Le directory

La memoria secondaria

La memoria secondaria è un sistema di memorizzazione presente nella maggior parte dei calcolatori elettronici

Principali differenze rispetto alla memoria primaria:

- Maggiore capacità
  - migliaia di GB contro decine di GB
- Persistenza
  - dati conservati in assenza di alimentazione elettrica
- Prestazioni inferiori
  - tempi di accesso e trasferimento più alti
- Assenza di istruzioni macchina per l'accesso ai dati
  - uso di coprocessori dedicati all'I/O

Le caratteristiche hardware della memoria secondaria impongono al SO l'utilizzo di determinati meccanismi e politiche di gestione

Tipologie di memoria secondaria

- Dischi magnetici di grande capacità (hard disk)
- Dischi magnetici di piccola capacità (floppy disk)
- Memoria di tipo flash (tipicamente su bus USB)
- Dischi a stato solido (SSD)
- Nastri magnetici
- Dischi ottici (CD-ROM, DVD)
- Sistemi di memorizzazione ridondante (RAID, Redundant Array of Independent Disks)
- Sistemi di memorizzazione su rete (SAN, Storage Area Network)

È compito del SO:

- gestire in modo specifico ciascuna tipologia di hardware
- offrire ai programmi applicativi una visione astratta ed unificata della memoria secondaria

File system

I SO tipicamente includono un insieme di programmi dedicati alla gestione della memoria secondaria:

I *sistemi di archiviazione* o *file system*

- semplificano la programmazione degli accessi al disco tramite astrazioni quali “file” e “directory”
- consento l’accesso “trasparente” a file di tipo predefinito
- gestiscono i dispositivi di memoria secondaria assegnando e rilasciando spazio disco sulla base delle operazioni richieste dalle applicazioni in esecuzione
- implementano meccanismi di protezione e sicurezza per l’accesso ai file da parte degli utenti
- realizzano in modo automatico e trasparente copie di sicurezza (*backup*) dei dati

Ciascun SO definisce un proprio sistema di archiviazione nativo, ma è generalmente in grado di utilizzare anche dispositivi basati su file system di SO differenti

I file

Un *file* (*archivio*) è un insieme di informazioni correlate tra loro individuato da un nome unico all'interno del sistema

- Un file è tipicamente (ma non necessariamente) registrato in memoria secondaria persistente
- Il nome del file è una sequenza di caratteri alfanumerici, con caratteristiche che dipendono dal file system:
  - L'insieme di caratteri utilizzabili
  - Equivalenza di lettere maiuscole e minuscole
  - Lunghezza massima
- Esistono diversi tipi di file:
  - File regolari
  - Directory
  - Alias o link simbolici
  - File associati a dispositivi hardware
  - File per canali di comunicazione tra processi
    - Socket (derivati da Unix BSD)
    - Pipe con nome (o FIFO) nei sistemi della famiglia Unix

Tipo e contenuto di un file

Talvolta si fa un po' di confusione tra tipo e contenuto di un file

- Il tipo di file rappresenta essenzialmente lo scopo per il quale il file esiste, e dunque la modalità con cui il kernel del SO lo gestisce
- Il contenuto di un file (di tipo regolare) rappresenta la tipologia dei dati memorizzati
- In alcuni SO il contenuto determina l'applicazione che il SO esegue in risposta ad una azione dell'utente sul file

Spesso il kernel di un SO gestisce allo stesso modo tutte le operazioni sui file di tipo regolare, a prescindere dal contenuto

Nei file system della famiglia Unix il contenuto non è memorizzato tra gli attributi del file

Esempio: l'estensione del nome del file

In alcuni SO il nome del file è utilizzato anche per codificare il suo contenuto

Caso tipico: l'estensione del file nei file system FAT (MS DOS) e NTFS (Windows NT)

`.exe`, `.com`	file eseguibile
`.obj`	file oggetto
`.c`, `.pas`	file sorgente
`.bat`	script per la shell di comandi
`.txt`	file di testo
`.dll`	libreria dinamica
`.zip`	archivio compresso
⋮	

L'estensione determina come il SO gestisce il file (tipicamente quale applicazione eseguire per manipolarlo)

Nei SO della famiglia Unix l'estensione del file non ha un formato fisso ed è usata unicamente come ausilio per gli utenti

Attributi dei file

Oltre al nome, il file system può memorizzare una serie di attributi per ciascun file:

- Identificatore unico all'interno del file system (generalmente un numero)
- Tipo di file e eventualmente contenuto del file
- Dimensione del file, ossia numero di byte o record che lo compongono
- Posizione dei dati corrispondenti al file sul supporto di memorizzazione
- Utente proprietario e diritti di accesso per proteggere il file
- Timestamp (data e ora) per la creazione, ultima modifica ed ultimo accesso al file

Perché si usa un "identificatore" oltre al "nome"?

Il nome consente di identificare un file in modo non ambiguo, tuttavia lo stesso file potrebbe essere associato a più nomi

Struttura dei file

Generalmente le informazioni all'interno di un file possono essere:

- Non strutturate: sequenza di caratteri o valori numerici
- Sequenza di record:
  - linee di caratteri
  - vettori scalari di lunghezza fissa o variabile
  - record di lunghezza fissa o variabile
- Struttura gerarchica, con sezioni in diverso formato
  - archivi di file
  - file eseguibili
  - immagini grafiche

La struttura di un file di tipo regolare è legata al suo contenuto

Nelle applicazioni commerciali spesso non esiste una netta distinzione tra i dati memorizzati e la struttura (ossia il formato) dei file che memorizzano tali dati

Esempio: i file in Mac OS X

Il file system HFS+ utilizzato nei SO della famiglia Mac OS X permette di definire diversi *flussi* di dati all'interno di un file:

- data fork: i dati principali del file
- resource fork: codice macchina, icone grafiche, immagini, layout delle finestre, messaggi di testo, ...
- info: gli attributi del file, tra cui il tipo ed un identificatore dell'applicazione che ha creato il file

Il campo identificatore dell'applicazione permette al SO di lanciare l'applicazione quando l'utente fa un doppio "click" sull'icona del file

In effetti HFS+ consente di avere un numero arbitrario di fork per un singolo file, ma questa caratteristica non è molto utilizzata per problemi di compatibilità con i comandi di sistema che operano sui file

Operazioni sui file

Ogni SO deve consentire alcune operazioni fondamentali relative ai file:

- Creazione del file
  - Assegnazione di un tipo, nome e identificatore
  - Allocazione dello spazio sul dispositivo di memorizzazione
- Apertura e chiusura del file
- Scrittura e lettura del file
  - Con accesso sequenziale, aggiornamento della posizione corrente del file
- Ricerca o riposizionamento per l'accesso diretto al file
- Cancellazione del file
- Ridimensionamento del file (modifica della lunghezza)

Altre operazioni possono essere definite in base alle precedenti, ad esempio:

- Copia di un file
- Rinomina di un file

Gestione dei file in un SO

Poiché gli accessi alla memoria secondaria sono costosi, il kernel del SO non può ricostruire la posizione di un file a partire dal suo nome per ogni operazione eseguita su di esso

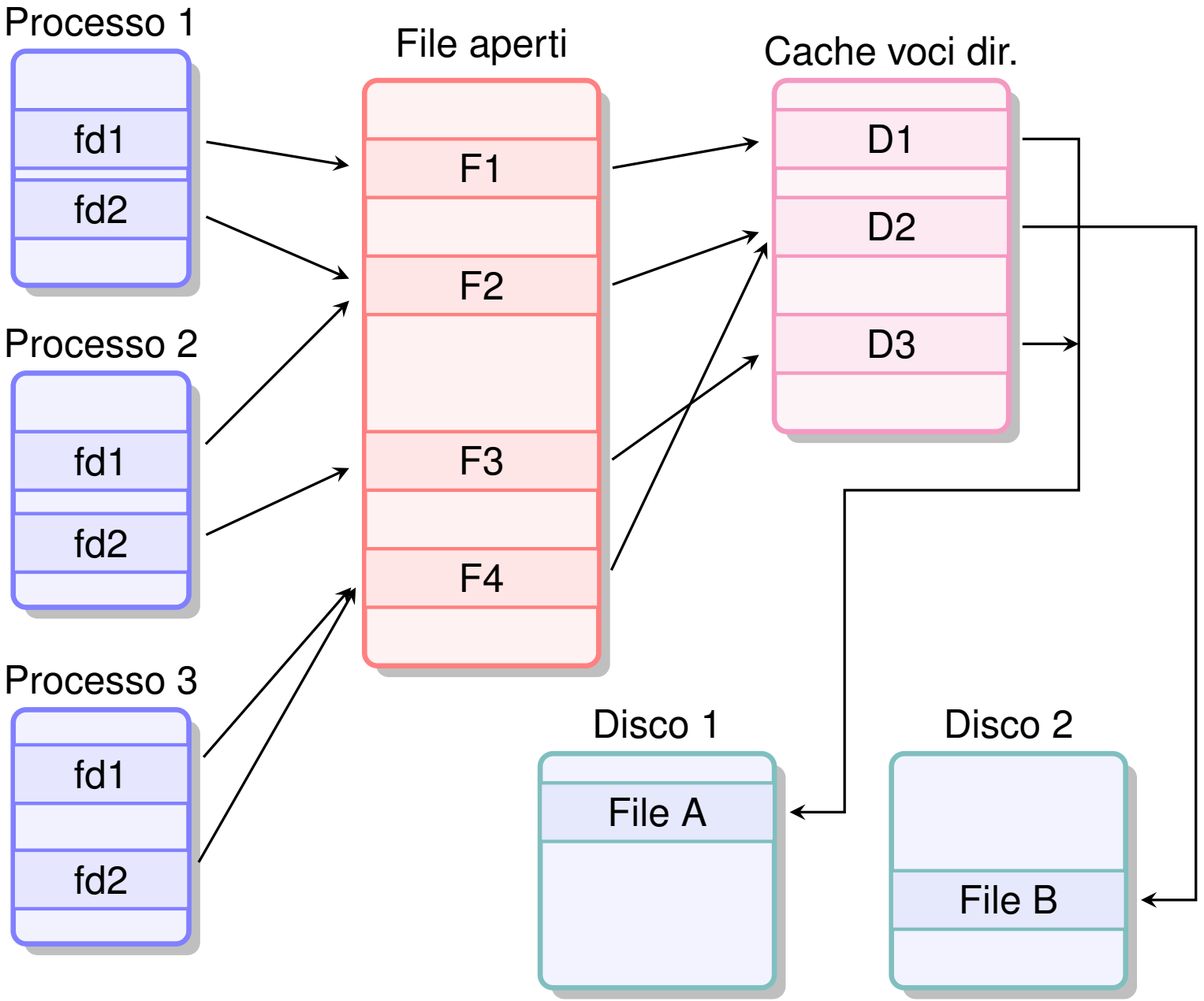
L'apertura del file consente di salvare in una struttura di dati del kernel la posizione sul disco e le altre informazioni necessarie per operare sul file

Tipicamente viene utilizzato uno schema con una struttura di dati a più livelli, che include:

- La tabella dei file aperti, definita per l'intero sistema, contiene una voce per ciascun file aperto nel sistema
- Per ogni processo, una tabella di descrittori di file contenente una voce per ciascun file aperto dal processo

In pratica le strutture di dati per gestire i file aperti sono sofisticate perché debbono tenere in considerazione la condivisione dei file aperti, la memorizzazione dei risultati delle ricerche dei file, la duplicazione dei descrittori, . . .

Esempio di strutture di dati per i file



Esempio di strutture di dati per i file (2)

- Il descrittore fd2 del processo P1 ed il descrittore fd1 del processo P2 fanno riferimento allo stesso file aperto F2
  - I processi P1 e P2 accedono al file B con la stessa modalità e con un unico puntatore alla posizione corrente
- I descrittori fd1 e fd2 del processo P3 fanno riferimento allo stesso file aperto F4
  - Tipicamente un descrittore di file è stato “rediretto” sull’altro
- Le voci F2 e F4 della tabella dei file aperti fanno riferimento alla stessa voce di directory D2
  - Lo stesso nome di file è stato aperto due volte, tipicamente da processi e con modalità d’accesso differenti
- Le voci di directory D1 e D3 fanno riferimento allo stesso file su disco
  - Lo stesso file è accessibile per mezzo di due nomi differenti

Accesso sequenziale

- Per ciascun file aperto il SO mantiene un puntatore alla posizione corrente all'interno del file
  - Alcuni SO mantengono due posizioni, una per le letture e l'altra per le scritture
- Gli accessi al file sono relativi alla posizione corrente ed aggiornano automaticamente la posizione corrente
- È possibile modificare esplicitamente la posizione corrente
  - Talvolta è possibile solo ritornare all'inizio del file
- È l'unica modalità di accesso ai file consentita per i dispositivi di memoria a nastro magnetico

Quali sono i vantaggi dell'accesso sequenziale?

- Semplicità: non è necessario indicare la posizione nelle chiamate di sistema per accedere al file
- Efficienza: il SO può predire le successive richieste di accesso e programmare in anticipo il trasferimento dei dati

Accesso diretto

- Il file viene considerato come una sequenza numerata di elementi (blocchi o record)
  - Per motivi di efficienza gli elementi contegono almeno qualche centinaio di byte
- Ciascuna richiesta di accesso al file contiene il numero dell'elemento
  - Assoluto: indice rispetto all'inizio del file
  - Relativo: indice rispetto all'ultimo elemento acceduto
- Realizzabile anche con API ottimizzate per l'accesso sequenziale eseguendo prima di ogni accesso una chiamata di sistema che modifica la posizione corrente
  - Esempio: `lseek()` nei sistemi Unix

Quali sono i vantaggi dell'accesso diretto?

- È indispensabile per accedere rapidamente a singoli dati contenuti in file molto grandi (ad es.: basi di dati)
- Permette la creazione di file "sparsi", logicamente molto grandi ma con ridotta occupazione su disco

Accesso indicizzato

- Realizzato combinando tra loro l'accesso diretto con quello sequenziale
- Viene costruito un indice contenente le posizioni ai vari elementi di un file rispondenti ad un determinato criterio
  - Ad esempio: elementi caratterizzati da una chiave alfanumerica, ricerca sulla chiave resa efficiente dall'indice
- Ogni richiesta di accesso al file contiene una "chiave" che permette di analizzare rapidamente l'indice e determinare il numero dell'elemento contenente le informazioni volute
- Talvolta si utilizzano due indici gerarchici: il principale (in memoria centrale) dà la posizione in quello secondario (su disco), che a sua volta dà il numero dell'elemento del file
- Comune sui mainframe e per l'implementazione delle basi di dati
  - Esempio: ISAM (Indexed Sequential Access Method)

Directory

Nei primi SO le directory non esistevano: tutti i file erano identificati da un semplice nome alfanumerico

Quali svantaggi comporta non avere directory?

- Tutti i nomi dei file debbono essere differenti, anche quelli di utenti diversi
- Per cercare il nome di un file l'utente è costretto a navigare tra migliaia di nomi
- La procedura per aprire un file è lenta e costosa
- Utilizzare supporti di memorizzazione esterni come CD-ROM, DVD o memorie flash è scomodo ed inefficiente

La directory è un file di tipo particolare che consente di raggruppare ed organizzare i file in modo gerarchico

Nel sistema Unix originale la directory era un file di testo regolare che elencava i nomi dei file in essa "contenuti"

Sistema a due livelli di directory

- Il livello superiore è chiamato directory principale o MFD (Master File Directory)
- All'interno della directory principale è definita una directory secondaria per ciascun utente del sistema

Quali sono gli svantaggi di questo approccio?

Due soli livelli non sono sufficienti:

- Gli utenti sono completamente isolati e non possono condividere file
  - Tutti i file di sistema debbono essere duplicati entro ogni directory utente
  - In alternativa, si deve complicare la procedura di ricerca dei file per includere anche una directory predefinita di sistema
- È difficile od impossibile gestire i dispositivi di memorizzazione esterni
- Ogni singolo utente non ha la possibilità di organizzare gerarchicamente i propri file

Sistema di directory con struttura ad albero

- Le directory sono organizzate gerarchicamente in una struttura ad albero
- La directory *radice* (*root*) contiene file ed altre directory
- Ciascuna directory può contenere altri file e altre directory
  - Nessun file può essere incluso in due directory
- Il nome completo del file (o percorso assoluto) è costituito dal nome di tutte le directory nel percorso dalla radice alla directory contenente il file, e dal nome del file stesso
  - Il separatore dei vari nomi è un carattere predefinito, ad esempio "/" (Unix) oppure "\" (MS-DOS, Windows)
- Ogni processo memorizza la posizione di una directory di lavoro detta directory corrente
  - È generalmente possibile accedere ai file nella directory corrente utilizzando soltanto il nome del file ("`fileA`")
  - È possibile anche utilizzare nomi di percorso relativi alla directory corrente ("`../fileB`")

Sistema di directory con struttura a grafo aciclico

- In generale il nome completo di un file permette di identificare senza ambiguità un file
- Nei SO della famiglia Unix, la corrispondenza tra nomi e file non è biunivoca: ad uno stesso file possono corrispondere più nomi (ma non viceversa!)
- Un file può quindi "apparire" in due diverse directory del sistema
- Se rappresentiamo con una freccia la relazione di inclusione (dalla directory al file contenuto), la presenza di file con più nomi porta alla creazione di grafi orientati

A cosa è dovuta la creazione di un ciclo nel grafo orientato?

Poiché per definizione la directory radice non è inclusa in nessun'altra directory, un ciclo è dovuto ad una directory con almeno due nomi (posizioni) nell'albero

Nei SO della famiglia Unix i file system hanno una struttura a grafo orientato aciclico

Gestione dei nomi di file multipli

Nei SO della famiglia Unix, il nome di un file è detto *hard link*

Si può creare un nuovo hard link per un file già esistente con

- La chiamata di sistema `link()`
- Il comando di sistema `ln`

Ciascun file dispone di un contatore di riferimento:

- Creando un nuovo nome per un file si incrementa il contatore
- Cancellando un (nome di) file, si decrementa il contatore
- Il file viene rimosso dal file system soltanto quando il contatore di riferimento si azzera

Sistema di directory con struttura a grafo generale

Perché in Unix si è scelto di non permettere cicli nella struttura di directory?

- Per motivi di efficienza il SO deve poter riconoscere quando si attraversa più volte la stessa directory per risolvere un nome di percorso
  - Il codice del file system è più lento e complesso
- Cancellando (il nome di) una directory, non è più possibile utilizzare il contatore di riferimento per decidere quando rimuoverla dal file system
  - Il SO deve eseguire una visita completa del grafo per determinare le cartelle non più raggiungibili dalla radice (*garbage collection*)

Soft link

Un soft link è un particolare tipo di file dei SO della famiglia Unix il cui contenuto è un nome di percorso di un altro file

Quando un soft link appare come componente di un nome di percorso, il SO automaticamente sostituisce il soft link con il percorso equivalente

Ad esempio: se "`b`" in "`/a`" è un soft link il cui contenuto è "`/c/d`", il percorso "`/a/b/e`" viene risolto in "`/c/d/e`"

Qual è lo svantaggio dei soft link?

A differenza degli hard link, risolvendo un percorso con soft link si possono incontrare cicli infiniti: ad es.: "`b`→`c`" e "`c`→`b`"

In pratica il SO pone un limite al numero di soft link attraversati per ciascun componente di un percorso