Struttura interna del sistema operativo Linux

4. Il file system

A cura di:
Giuseppe Pozzi Donatella Sciuto

DEI, Politecnico di Milano

giuseppe.pozzi/donatella.sciuto@polimi.it

- versione del 21 marzo 2003 -

Sommario

- Introduzione al file system;
- file ed attributi;
- operazioni sui file;
- meccanismi di protezione;
- prestazioni di un file system;
- struttura di un volume;
- strutture dati del file system;
- un esempio riassuntivo (non banale: differenze tra Unix System V e Linux Red Hat 7.2.).

A cosa serve la memoria di massa

- Conservare programmi e dati quando il calcolatore non è alimentato (memoria non volatile o persistente).
- Memorizzare i dati generati da un processo dopo la sua terminazione e consentirne l'uso ad altri processi (memoria a "lunga vita").
- Archiviare grandi quantità di dati e informazioni.
- Consentire a diversi processi di accedere agli stessi dati contemporaneamente (accesso concorrente).
- File: unità di archiviazione in memoria di massa.

Introduzione

- Le istruzioni di Input/Output sono definite a livello di sistema operativo.
- A livello di sistema operativo le istruzioni di I/O lavorano con un'astrazione: il file.
- Per capire come sono implementate le istruzioni di I/O è necessario comprendere come sono organizzati e memorizzati i file: il file system.

Introduzione

- File system: componente del sistema operativo che realizza i servizi di gestione dei file.
- File system svolge le sue funzioni accedendo ai dischi tramite l'invocazione di routine messe a disposizione dal gestore del disco (disk driver).
- Il driver del disco fornisce una interfaccia di accesso ai dati su disco nascondendone le caratteristiche fisiche.
- Il disco è rappresentato da un dispositivo logico, chiamato volume composto da un array di blocchi.
- Il blocco è una porzione di disco costituita da un numero intero di byte che viene trasferita in memoria con un'unica operazione.

File

- Il concetto di *file* offre una visione logica omogenea delle informazioni memorizzate.
- La visione non dipende dal tipo di dispositivo fisico su cui le informazioni vengono memorizzate.
- Un *file* è costituito da:
  - un insieme di informazioni omogenee;
  - un nome simbolico;
  - un insieme di attributi.
- Un *file* può contenere:
  - dati;
  - programmi;
  - riferimenti.

Attributi (i)

- Ad un file sono associati alcuni attributi che ne descrivono alcune caratteristiche:
  - *Nome:*
    - E' un nome simbolico univoco nella sua directory con cui ci si riferisce ad esso.
  - *Tipo:*
    - Definisce il tipo dei dati contenuti (estensione del nome, opzionale).
  - *Locazione:*
    - E' un puntatore alla posizione fisica sul dispositivo.

Attributi (ii)

- *Dimensione:*
  - Dimensione dei dati espressa in bytes o blocchi.
- *Protezione:*
  - Definisce le politiche di gestione degli accessi.
- *Ora e Data:*
  - Indicano il momento della creazione, ultima modifica o ultimo accesso.
- *Proprietario:*
  - Indica il nome dell'utente che ha creato il file.

Operazioni

- Sui file possono essere compiute diverse operazioni.
- Le operazioni vengono svolte attraverso richieste di servizi al sistema operativo:
  - creazione;
  - apertura;
  - scrittura;
  - lettura;
  - riposizionamento;
  - cancellazione.

Operazioni

- Le primitive per operare sui file nei vari linguaggi (ad es. nel linguaggio C vi sono `fopen, fread, fwrite, fclose, fputs, fgets` etc.) invocano i servizi messi a disposizione dal sistema operativo ed illustrati di seguito.
- Attenzione: fopen e simili possono avere comportamenti differenti da un sistema operativo ad un altro (ad es. Unix System V e Linux Red Hat 7.2).

Operazioni

Creazione

- Viene aggiunto un nuovo file al file system
- Le operazioni richieste sono:
  - allocazione;
  - creazione del nuovo descrittore del file;
  - aggiunta del descrittore al file system.
- Utilizzo:
  - int creat (char *nomefile, int permessi);
  - permessi: gestisce i permessi di accesso.

Operazioni

Apertura

- Sulla base del nome individua la posizione del file sul disco.
- Copia il descrittore del file nella tabella dei file aperti.
- Restituisce un intero, esito dell'operazione.
- Utilizzo:
  - int open (char *nomefile, int tipo, int permessi);
  - permessi: gestisce i permessi di accesso;
  - tipo: lettura, scrittura, lettura e scrittura:
    - O_RDONLY: The file is open for reading only.
    - O_WRONLY: The file is open for writing only.
    - O_RDWR: The file is open for reading and writing.

Operazioni

Scrittura

- Aggiunge dati ad un file già creato.
- Per scrivere dati su un file è necessario fornire:
  - l'intero descrittore del file (ottenuto tramite open o creat);
  - i dati da scrivere.
- Utilizzo:
  - int scritti = write(int fd, char buffer[], int numero_byte);
  - restituisce il numero di caratteri effettivamente scritti.
- Il file system mantiene un puntatore alla posizione in cui deve essere effettuata la scrittura successiva.

Operazioni

Lettura

- Preleva dati da un file già creato.
- Per leggere dati da un file è necessario fornire:
  - l'intero descrittore del file (ottenuto tramite open);
  - un puntatore alla zona di memoria destinata a contenere i dati.
- Utilizzo:
  - int letti = read(int fd, char buffer[], int numero_byte);
  - restituisce il numero di caratteri effettivamente letti.
- Il file system mantiene un puntatore alla posizione in cui deve essere effettuata la lettura successiva.

Operazioni

Riposizionamento

- Sposta la posizione dei puntatori di lettura e di scrittura.
- Le operazioni consentite dipendono dal tipo di accesso al file.
- Spesso viene mantenuto dal file system un solo puntatore valido per lettura e per scrittura.
- Utilizzo:
  - long lseek(int fd, long offset, int riferimento);
  - modifica la posizione corrente e la restituisce come long.

Operazioni

Riposizionamento

- riferimento = 0: la nuova posizione è calcolata come inizio file + offset;
- riferimento = 1: la nuova posizione è calcolata come posizione corrente precedente + offset;
- riferimento = 2: la nuova posizione è calcolata come fine file + offset.

- Esempi:
  - lseek(fd, 0L, 1): restituisce la posizione attuale senza modificarla;
  - lseek(fd, 0L, 0): posiziona all'inizio del file;
  - lseek(fd, 0L, 2): posiziona alla fine del file.

Operazioni

Duplicazione

- Ottengo un nuovo descrittore associato ad un file già a aperto con la open.
  - Aggiunge una riga nella tabella dei file aperti da quel processo.
- Utlizzo:
  - fd1=open(…);
  - fd2=dup(fd1);
  - la posizione corrente è comunque unica per i due o più descrittori.

Operazioni

Chiusura

- Sulla base del nome o di un identificatore localizza il descrittore del file nella tabella dei file aperti.
- Elimina il descrittore dalla tabella dei file aperti dal processo.
- Elimina il descrittore dalla tabella globale dei file aperti **se non vi sono altri riferimenti**.
- Utilizzo:
  - int close(int fd);
  - restituisce l'esito dell'operazione di chiusura.

Operazioni

Cancellazione

- Elimina un file
- Per eliminare un file è necessario specificarne il nome.
- Le azioni necessarie sono:
  - Deallocazione dello spazio sul dispositivo fisico
  - Aggiunta dello spazio deallocato alla lista dello spazio disponibile sul dispositivo
  - Rimozione del descrittore del file dal file system.
- Utilzzo:
  - int unlink(char *nome);

Operazioni

- Le operazioni descritte richiedono l'accesso ad un file.
- Il file system deve cercare sul dispositivo fisico il file.
- I file in uso si dicono *aperti.*

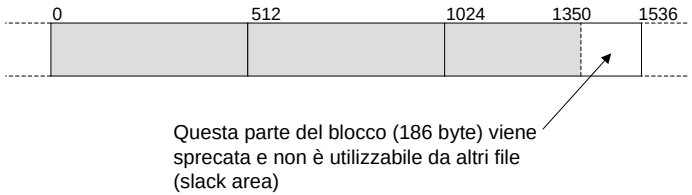
Per rendere più efficiente la ricerca, il file system mantiene una tabella dei file aperti.

Struttura del file system

- *Struttura logica:*
  - I dati sono organizzati in unità logiche di lunghezza fissa ma arbitraria dette *record*
  - Nel sistema UNIX un *record* è un byte
- *Struttura fisica:*
  - I dati sono organizzati in unità fisiche di lunghezza fissa e dipendente dal dispositivo dette *blocchi*
  - Dimensioni tipiche dei blocchi di unità a disco rigido: da 32 a 4096 byte, tipicamente 512
- La struttura logica e fisica sono in genere differenti.

Struttura

- Tale differenza, unitamente alla dimensione fissata dei blocchi provoca uno spreco di spazio
- Esempio: file di 1350 byte e disco con blocchi da 512 byte



- Questo fenomeno è noto come *frammentazione interna*

File system

- I moderni dispositivi di memorizzazione di massa consentono di salvare milioni di file
  - Necessaria una strutturazione gerarchica
- Spesso sorge la necessità di condividere un file o un gruppo di file tra più utenti
- Un file system è organizzato in:
  - *Partizioni:* Contengono insiemi di file correlati
  - *Directory:* Una partizione è suddivisa in directory. Le directory contengono informazioni sui file e fungono da indice.
  - *File:* Contengono effettivamente i dati o i programmi

Directory

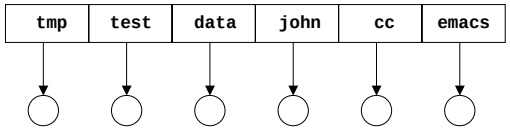
- Contiene informazioni sui file:
  - attributi;
  - posizione;
  - proprietario.
- Directory è un file di proprietà del sistema operativo.
- Fornisce una corrispondenza tra nomi di file e file stessi.

Directory

- Sulle directory, così come sui file, è possibile compiere alcune operazioni:
  - *Ricerca di un file:* Sulla base di un nome o una espressione regolare consente di recuperare le informazioni su uno o più file;
  - *Creazione di un file:* Alla directory viene aggiunto un elemento che raccoglie le informazioni sul nuovo file;
  - *Rimozione di un file:* Eliminazione di un descrittore di file. Il descrittore deve essre prima individuato tramite una operazione di ricerca;
  - *Elenco dei file:* Produce l'elenco dei nomi ed eventualmente altre informazioni relative ai file memorizzati;
  - *Rinomina di un file:* Modifica il nome di un file già presente nella directory. Il descrittore deve essre prima individuato tramite una operazione di ricerca.

Directory a singolo livello

- Tutti i file sono contenuti in una sola directory
- Struttura molto semplice ma presenta alcuni problemi:
  - con molti file è difficile garantire che tutti i nomi siano diversi;
  - con molti file le dimensioni della directory diventano grandi a discapito del tempo di accesso in quanto le ricerche sono più complesse;
  - utenti differenti possono accedere agli stessi file.



Directory a due livelli

- Si hanno due livelli:
  - directory principale (*root*): contiene un elenco di directory, una per ogni utente;
  - directory utente: contiene i file di un singolo utente (*home* directory dell'utente).
- I singoli utenti vedono e gestiscono solo i file nella propria home directory ed in eventuali sottodirettori.
- La gestione della root directory è affidata ad un amministratore del sistema (*root*, *administrator*, *superuser*).

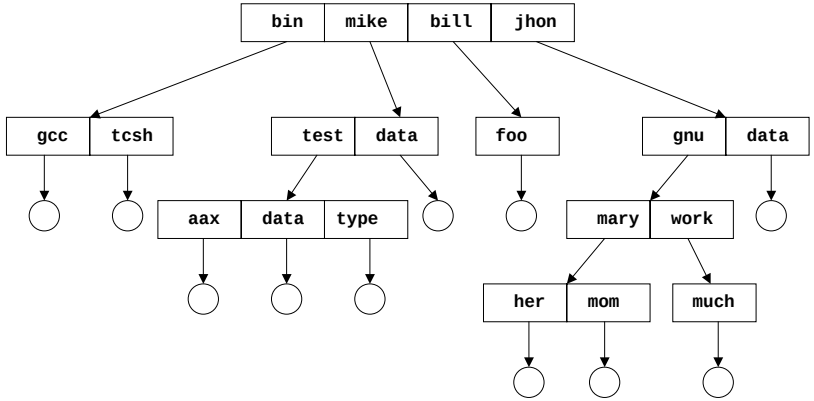
Directory ad albero

- E' una estensione del caso precedente ad un numero arbitrario di livelli.
- Per semplificare l'accesso ai file viene definito il concetto di *current working directory* (*cwd*) cioè di directory corrente.
- Un utente accede ai file non di cwd attraverso il *path name*, cioè il percorso completo a partire dalla radice oppure a partire dal direttorio corrente indicando quali altri direttori si devono attraversare.
- I file vengono cercati nella directory corrente o utilizzanto il path name se specificato.

Directory a più livelli

- Se un file non è nella directory corrente:
  - si usa il suo path name. In alternativa,
  - si cambia la directory corrente.
- Il sistema operativo offre alcuni comandi per gestire la *cwd:*
  - *cd <dir>:* Change Directory. La directory *dir* diviene la cwd
  - *pwd:* Print Working Directory. Mostra il nome della cwd

Esempio di directory ad albero

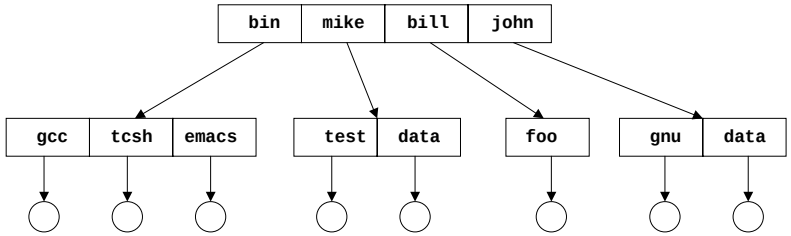


Directory a più livelli

- Quando un utente richiede l'accesso ad un file, il file system ne cerca il nome nella directory corrente.
- Per accedere ai file di altri utenti si utilizza il *path name*.
- Spesso gli applicativi sono accessibili a tutti gli utenti: a tale scopo esiste una directory specifica (*/bin*).

Directory a più livelli

- I file eseguibili vengono cercati nelle directory definite dalla variabile di ambiente PATH.



Link e unlink

- Per associare un nuovo nome ad un file già esistente:
  - int link(char *nome, char *nuovo_nome)
- Per rimuovere un nome associato ad un file:
  - int unlink(char *nome);
- Se non vi sono ulteriori riferimenti al file, il file viene definitivamente rimosso.

Protezione

- I dati memorizzati nei file di un file system necessitano di protezione.
- Protezione da *danni fisici:*
  - malfunzionamenti dei dispoitivi;
  - danni meccanici e/o elettrici;
  - **soluzione**: *backup* e *mirroring*.
- Protezione da *accessi impropri:*
  - riservatezza;
  - modifica o eliminazione accidentale di dati importanti;
  - **soluzione**: definizione di una *politica di accesso*.

⇨ ***Protezione****: definizione di una politica di accesso e relativa implementazione.*

Protezione: Tipi di accesso

- Alcune banali politiche di accesso sono:
  - *Ogni utente accede solo ai propri file:* si tratta di una scelta limitante, ad esempio per i gruppi di lavoro
  - *Ogni utente accede a tutti i file:* è assente una politica di accesso
- La soluzione consiste nell'*accesso controllato*

Regole di accesso ai file

- Si definiscono *regole di accesso* ai file sulla base di:
  - Identità e gruppo di lavoro dell'utente
  - Proprietà dei file
- Tali regole dipendono dal tipo di operazione richiesta:
  - Lettura
  - Scrittura o eliminazione
  - Esecuzione o lista
  - Aggiunta

Protezione: Liste di accesso

- L'accesso e le operazioni consentite dipendono dalla identità dell'utente
- Ad ogni file è associata una *lista di accesso* che indica quali *operazioni* sono consentite a quali *utenti*
- Quando una operazione viene richiesta il sistema operativo controlla la lista di accesso per verificare se:
  - il richiedente è contemplato;
  - il richiedente ha il permesso di compiere quel tipo di operazione.

Liste di accesso

- La soluzione basata su liste di accesso presenta alcuni svantaggi:
  - le liste di accesso possono essere di dimensioni notevoli;
  - le liste devono essere create e mantenute per ogni file;
  - il tempo di accesso ad un file si allunga.

Protezione: UNIX

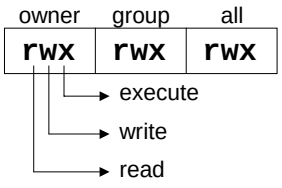
- Soluzione semplificata implementata in UNIX
- Utenti identificati in base a:
  - *Username:* Identificativo dell'utente
  - *Group:* Identificativo di gruppo, condiviso da più utenti
- Utenti raggruppati in tre classi:
  - *Owner:* Solo il proprietario del file
  - *Group:* Solo i membri del gruppo del proprietario del file
  - *All:* Tutti gli utenti

Protezione: UNIX

- Le operazioni su file sono raggruppate in tre classi:
  - *Read:* Lettura, copia
  - *Write:* Scrittura, modifica, eliminazione
  - *Execute:* Esecuzione
- Ad ogni file sono associati:
  - Owner
  - Group
  - Control access list

Protezione: UNIX

- La *control access list* è formata da tre gruppi di bit:
  - Ogni gruppo di bit si riferisce ad una delle tre classi di utenti
  - Ogni bit del gruppo si riferisce ad una delle tre operazioni



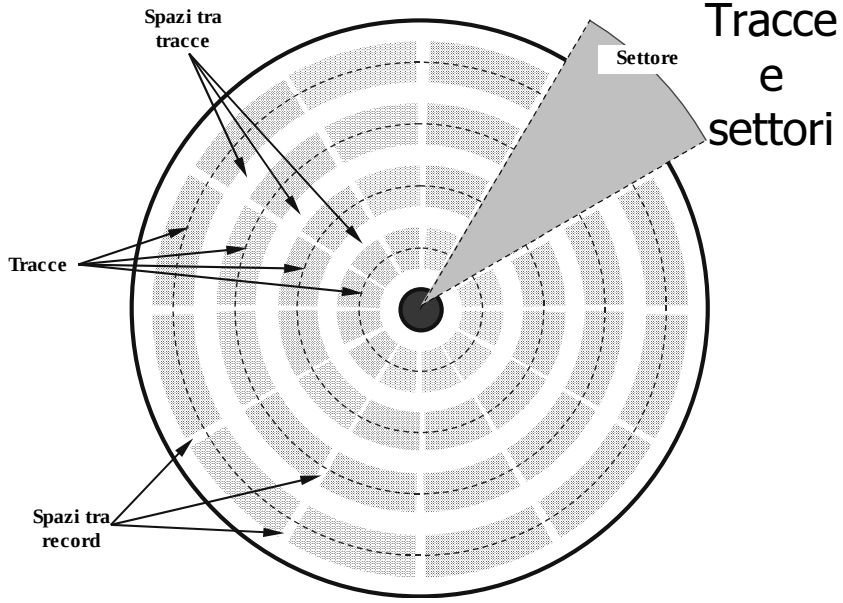
owner	group	all
rwx	rwx	rwx
111	101	101
7	5	5

Periferiche e file speciali

- In Unix le periferiche sono viste come file speciali, simulate come file nel direttorio /dev.
- Sulle periferiche è possibile eseguire open, read e write (purchè ne abbia senso), close ma non creat.
- Non ha senso leggere dal video o scrivere su una tastiera.
- Ogni programma dispone già di 3 descrittori di file: stdin, stdout, stderr. I 3 descrittori possono essere eventualmente ridiretti verso altri file.

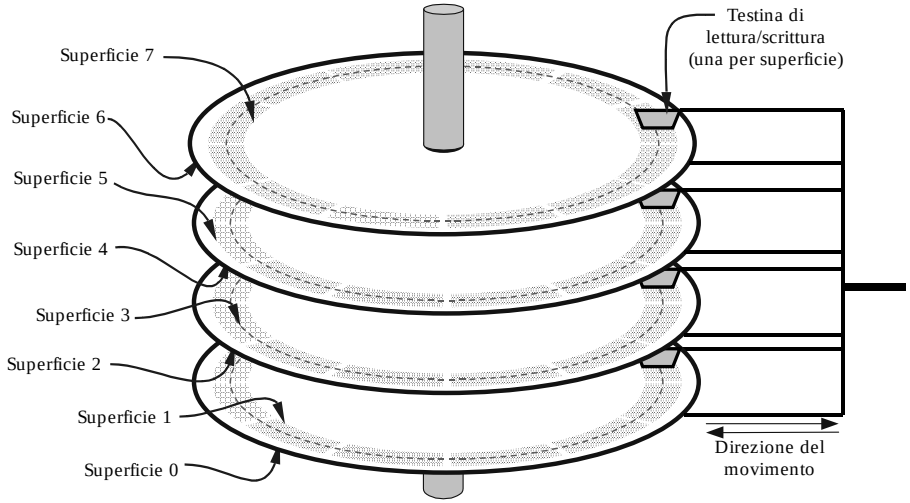
Tracce e settori

- **Traccia (track)**: sequenza circolare di bit scritta mentre il disco compie una rotazione completa:
  - la larghezza di una traccia dipende dalla dimensione della testina e dall'accuratezza con cui la si può posizionare; la densità radiale va da 800 a 2000 tracce per centimetro (5-10 µm per traccia);
  - tra una traccia e l'altra c'è un piccolo spazio di separazione (**gap**).
- **Settore (sector)**: parte di una traccia corrispondente a un settore circolare del disco:
  - un settore contiene 512 byte di dati, preceduti da un preambolo, e seguiti da un codice di correzione degli errori;
  - la densità lineare è di circa 50-100kbit per cm (0.1-0.2 µm per bit);
  - tra settori consecutivi si trova un piccolo spazio **(intersector gap).**
- **Formattazione**: operazione che predispone tracce e settori per la lettura/scrittura.



Tracce e settori

Schema di un Hard Disk



Le tracce in grigio formano un "cilindro"

Prestazioni dei dischi

- **Tempo di acceso** (ms o 10^{-3}s)
  - **Seek time**
    - la testina deve arrivare alla traccia giusta;
    - dipende dalla meccanica (5-15 ms, 1 per tracce adiacenti).
  - **Latency**
    - il disco deve ruotare fino a portare il dato nella posizione giusta;
    - dipende dalla velocità di rotazione (5400-10800 RPM ➠ 2.7-5.4ms).
- **Transfer Rate** (MBps)
  - **Velocità di trasferimento del disco**
    - dipende dalla densità di registrazione e dalla velocità di rotazione;
    - un settore di 512 byte richiede fra 25 e 100 µsec (5-20 MB/sec).
  - **Velocità di trasferimento del sistema di controllo**
    - SCSI vs. EIDE

Prestazioni di un file system

- Caratteristiche del software:
  - organizzazione dei buffer (quanti/quali dati vengono conservati in RAM);
  - velocità/efficienza del driver logico;
  - organizzazione dei blocchi.
- Caratteristiche dell'hardware:
  - velocità del disco (throughput, rotazione, seek);
  - velocità del driver fisico (e.g. SCSI vs EIDE);

Volume

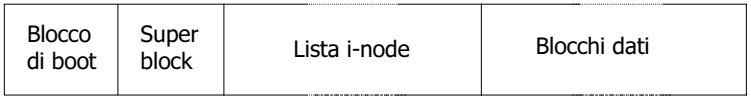
- UNIX gestisce i file system a livello logico considerandoli come dispositivi logici (volumi) identificati da un numero.
- La conversione tra indirizzi del dispositivo logico (file system) e indirizzi fisici è realizzata dal driver del disco.
- Volume composto da una sequenza di blocchi logici, di dimensioni fisse pari a un multiplo di 512 byte.

Ricerca del byte nel file

Ricerca del byte 520:

- Blocco = 520 div 512
- Offset nel blocco = 520 % 512 (resto della divisione) - 1

Struttura del volume



- Blocco 0 o blocco di boostrap: contenuto tipicamente nel primo settore contiene il codice di inizializzazione del sistema operativo;
- Superblock: descrive lo stato del file system (es. dimensioni, spazio libero...);
- Lista degli i-node: dimensione definita in fase di configurazione del sistema operativo, accessibile dalla tabella degli i-node (index-node).

Contenuto del superblock

- Dimensione del volume;
- numero di blocchi liberi sul volume;
- parte iniziale della lista dei blocchi liberi;
- numero di elementi della lista dei blocchi liberi;
- numero di elementi della tabella degli i-node;
- numero di i-node liberi;
- lista dei primi i-node liberi;
- altre informazioni ausiliarie.

Allocazione: UNIX *i-node*

- Il sistema operativo UNIX utilizza lo schema di allocazione indicizzata combinata, realizzato mediante una tabella i cui elementi sono chiamati *i-node (index-node)*.
- Ogni i-node contiene la lista degli attributi e degli indirizzi dei blocchi di disco a cui sono associati i blocchi del file.
- I primi indirizzi dei blocchi di disco sono memorizzati nell'i-node stesso.

Implementazione di file e directory in UNIX

- Ogni file o directory ha associato un i-node
- i-node: 64 byte di informazioni
- i-node dei file in un disco sono memorizzati in sequenza numerica all'inizio del disco o di ogni cilindro
- Dato il numero dell'i-node UNIX può localizzare la sua posizione calcolandone l'indirizzo su disco

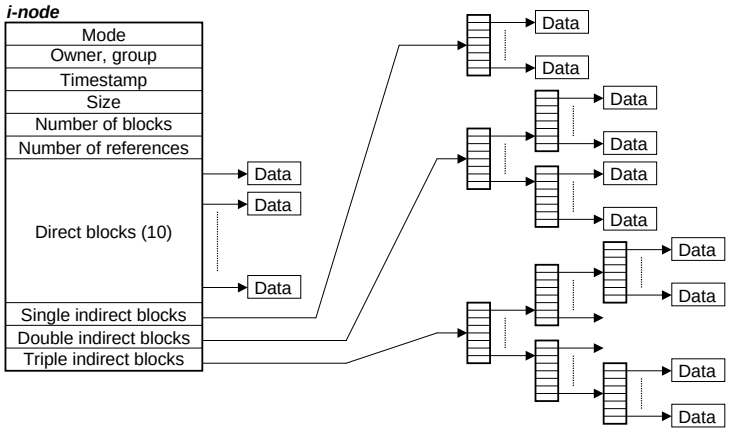
Informazioni nell'i-node

- Tipo di file, bit di protezione (9 rwx) e altri bit
- Numero di link al file
- Proprietario e gruppo del proprietario
- Lunghezza del file in byte
- 13 indirizzi su disco (blocchi su disco contenenti il file)
- Data e ora in cui il file è stato letto e scritto per l'ultima volta
- Data e ora dell'ultima modifica dell'i-node

UNIX *i-node*

- Per file di dimensioni maggiori l'i-node contiene l'indirizzo di un blocco del disco chiamato single indirect block che a sua volta contiene altri indirizzi di blocchi del disco.
- Se questo non è sufficiente l'i-node mette a disposizione un altro indirizzo, chiamato double indirect block che contiene l'indirizzo di un blocco che a sua volta contiene una lista di single indirect block.
- Ognuno di questi blocchi punta a sua volta a qualche centinaia (128 di solito) di single indirect block.
- Esiste anche un triple indirect block nel caso la doppia indicizzazione non sia sufficiente.

File system UNIX



Gestione dello spazio libero

- All'atto della creazione e scrittura di un nuovo file è necessario individuare sul disco il primo blocco libero
- Una soluzione consiste nell'uso un *vettore di bit* in cui:
  - La posizione del bit indica il numero del blocco
  - Se il bit vale 0 il blocco è già utilizzato
  - Se il bit vale 1 il blocco è disponibile

Allocazione degli spazi liberi

- Individuazione del primo blocco libero :
  - Si scorrono le parole (di b bit) fino alla prima diversa da zero
  - Sia k il numero di parole uguali a zero
  - Si trova l'offset d del primo bit con valore 1
- L'indirizzo n del blocco è dato da:

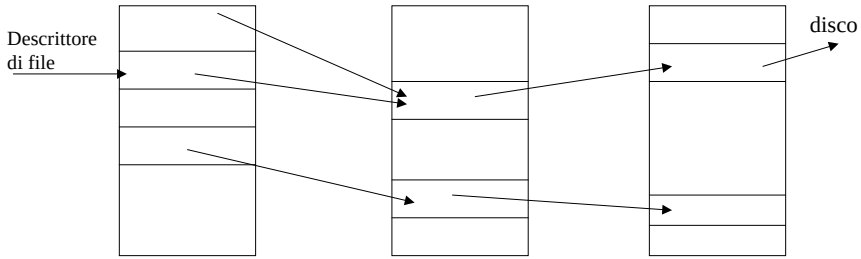
$$n = k \times b + d$$

Gestione dello spazio libero

- Soluzione alternativa: utilizzo di una *lista concatenata.*
- In questo schema:
  - la posizione del primo blocco disponibile è memorizzata in una zona riservata del disco;
  - ogni blocco disponibile contiene un riferimento al blocco disponibile successivo.
- Questa soluzione è migliore della precedente per dischi di grandi dimensioni.

Strutture dati del file system

UNIX dispone di tre tabelle per la gestione dei file



Strutture dati del file system

- Tabella dei descrittori di file utente:
  - tabella associata ad ogni processo utente contenente una riga per ogni file aperto dal processo con l'indirizzo della riga della tabella globale dei file aperti relativa al file.
- Tabella globale dei file aperti:
  - tabella del sistema operativo che contiene una riga per ogni file aperto nel sistema;
  - ogni riga contiene l'indirizzo del corrispondente i-node nella tabella degli i-node, l'indicatore della posizione corrente del file e il contatore al numero di riferimenti da parte dei processi a questo file.
- Tabella degli i-node:
  - le righe contengono la copia in memoria degli i-node del volume per maggiore efficienza nei riferimenti.

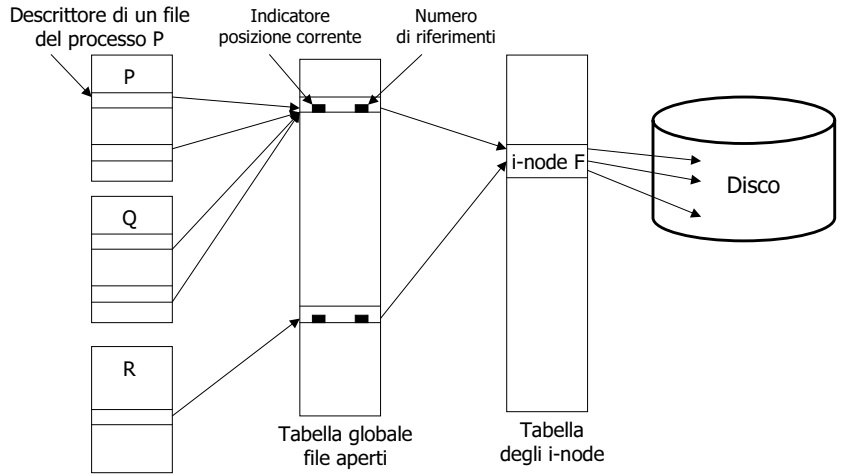
Strutture dai del file system

- Dup:
  - si generano due descrittori nella medesima tabella dei file aperti da quel processo e che puntano allo stesso file;
  - la posizione corrente è unica.
- Fork:
  - vengono duplicate anche le tabelle dei file aperti da un processo.
  - la posizione corrente è unica.
- Apertura dello stesso file da parte di 2 processi:
  - si aggiunge una riga nella tabella globale dei file aperti;
  - ogni processo ha una propria posizione corrente del file.

Strutture dati del file system

- P apre il file F;
- P esegue una dup;
- P esegue una fork, generando Q;
- R apre lo stesso file F.

Strutture dati del file system



Apertura di un file

La richiesta di apertura di un file al sistema operativo richiede le seguenti operazioni:

- Riserva una riga nella tabella globale dei file aperti e pone il contatore dei riferimenti e l'indicatore di posizione a 1
- Riserva un riga nella tabella dei descrittori di file utente aperti del processo e vi scrive l'indice della riga della tabella globale dei file aperti
- Restituisce al processo un descrittore costituito dal numero della riga della tabella dei descrittori di file utente aperti
- Determina quale sia l'i-node corrispondente al file il cui path name è stato indicato nella richiesta e lo scrive nella riga della tabella globale dei file aperti allocata

Operazioni su file

- Lettura o scrittura:
  - Deve essere fornito il descrittore del file che permette di seguire i puntatori nelle tre tabelle e dall'inode identificare i dati nel file
- Chiusura di un file:
  - Il sistema oprativo libera la corrispondente riga dei descrittori di file utente aperti
  - Decrementa il contatore dei riferimenti nella tabella globale dei file e se =0, libera la riga

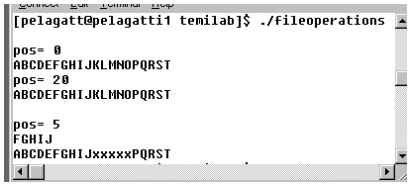
Un esempio (i)

```
/* esempio di uso delle operazioni LINUX sui file .*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
void main()
{     int fd1;
      int i,pos;
      char c;
                  /* apertura del file */
      fd1=open("fileprova", O_RDWR);
                  /* visualizza posizione corrente */
      printf("\npos= %ld \n", lseek(fd1,0,1));
                  /* scrittura del  file */
      for (i=0; i<20;i++)
      {           c=i + 65; /*caratteri ASCII a partire da A*/
                  write(fd1, &c, 1);
                  printf("%c",c);}
                  /* visualizza posizione corrente; riportala a 0 */
      printf("\npos= %ld \n", lseek(fd1,0,1));
      lseek(fd1,0,0);
```


Un esempio (ii)

```
                  /* lettura e visualizzazione del  file */
      for (i=0; i<20;i++)
      {           read(fd1, &c, 1);
                  printf("%c",c);}
      printf("\n");
                  /* posizionamento al byte 5 del file e stampa posiz.*/
      printf("\npos= %ld\n", lseek(fd1,5,0));
                  /*lettura di 5 caratteri */
      for (i=0; i<5;i++)
      {           read(fd1, &c, 1);
                  printf("%c",c);}
      printf("\n");
                  /*scrittura di 5 caratteri x*/
      c= 'x';
      for (i=0; i<5;i++)
      {           write(fd1, &c, 1);}
                  /* lettura del  file dall'inizio */
      lseek(fd1,0,0);
      for (i=0; i<20;i++)
      {           read(fd1, &c, 1);
                  printf("%c",c);}
      printf("\n");
                  /* chiusura file */
      close(fd1);
}
```

Esecuzione



Un secondo esempio (i)

```
/* vedi figura 8 pag. 180 libro Sistema operativo Linux e TCP/IP, G. Pelagatti, ediz. 2002 */

#include <stdio.h>
void main(int argc, char* argv[])
{
  int error, status;
  long posiz;
  pid_t pid, pid_figlio;
  FILE *fp;
  char *s1 = "Nel mezzo del cammin di nostra vita";
  char *s2 = "mi ritrovai in una selva oscura";
  char *s3 = "che la diritta via era smarrita";

  /* il processo padre apre il file */
  printf("Verifica con FOPEN\n\n");
  printf("padre: apro il file \n");
  fp=fopen("DivCom","w");
  error = (fp == 0);
  if (error!=0)
    {
    fprintf(stderr,"Errore di apertura del file\n");
    exit(1);
  }
```


Un secondo esempio (ii)

```
posiz = ftell(fp);
printf("padre: posizione corrente = %d\n",posiz);

/* il processo padre scrive nel file */
printf("padre: scrivo sul file\n");
fprintf(fp,"%s\n",s1);
fprintf(fp,"%s\n",s2);
fprintf(fp,"%s\n",s3);
posiz = ftell(fp);
printf("padre: posizione corrente = %d\n",posiz);

/* il padre genera un figlio. Il figlio eredita il file */
/* e la stessa posizione corrente del padre */
printf("padre: eseguo la fork\n");
pid = fork();
```

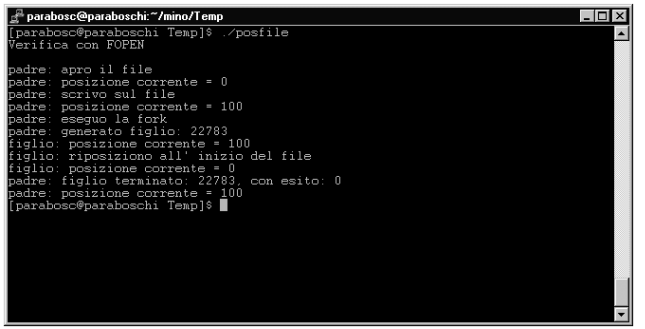

Un secondo esempio (iii)

```
if (pid == 0) {
    /*  nel figlio */
    /* legge posizione corrente, identica a quella del padre */
    posiz = ftell(fp);
    printf("figlio: posizione corrente = %d\n",posiz);
    /* il figlio si riposizioe all'inizio del file */
    printf("figlio: riposiziono all' inizio del file\n");
    error = fseek(fp, (long) 0, SEEK_SET);
    /* legge posizione corrente, non piu' identica a quella del padre */
    posiz = ftell(fp);
    printf("figlio: posizione corrente = %d\n",posiz);
    close(fp);
    /* chude il file e termina */
    exit(0);
    }
```

Un secondo esempio (iv)

```
 else {
     /* nel padre */
     printf("padre: generato figlio: %d\n",pid);
     /* legge la posizione corrente del file, dopo la */
     /* fseek del figlio */
     pid_figlio = wait(&status);
     printf("padre: figlio terminato: %d, con esito: %d\n",pid_figlio,status);
     posiz = ftell(fp);
     printf("padre: posizione corrente = %d\n",posiz);
  }
  /* chiude il file e termina */
  close(fp);
  exit(0);
}
```


Esecuzione su Linux



Un terzo esempio (i)

```
/* vedi figura 8 pag. 180 libro Sistema operativo Linux e TCP/IP, G. Pelagatti, ediz. 2002 */
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
void main(int argc, char* argv[])
{
   int error, status, file_desc;
   long posiz;
   pid_t pid, pid_figlio;
   char *s1 = "Nel mezzo del cammin di nostra vita";
   char *s2 = "mi ritrovai in una selva oscura";
   char *s3 = "che la diritta via era smarrita";

   /* il processo padre apre il file */
   printf("Verifica con utilizzo OPEN\n\n");
   printf("padre: apro il file \n");
   file_desc=open("DivCom",O_RDWR | O_CREAT, 0666);
   error = (file_desc == 0);
   if (error!=0)
     {
     fprintf(stderr,"Errore di apertura del file\n");
     exit(1);
   }
```


Un terzo esempio (ii)

```
posiz = lseek(file_desc, (long ) 0, 0);

printf("padre: posizione corrente = %d\n",posiz);

/* il processo padre scrive nel file */
printf("padre: scrivo sul file\n");
error = write(file_desc, s1, strlen(s1));
error = write(file_desc, "\n", 1);
error = write(file_desc, s2, strlen(s2));
error = write(file_desc, "\n", 1);
error = write(file_desc, s3, strlen(s3));
error = write(file_desc, "\n", 1);
posiz = lseek(file_desc, (long ) 0, 1);
printf("padre: posizione corrente = %d\n",posiz);

/* il padre genera un figlio. Il figlio eredita il file */
/* e la stessa posizione corrente del padre */
printf("padre: eseguo la fork\n");
pid = fork();
```

Un terzo esempio (iii)

```
if (pid == 0) {
    /*  nel figlio */
    /* legge posizione corrente, identica a quella del padre */
    posiz = lseek(file_desc, (long ) 0, 1);
    printf("figlio: posizione corrente = %d\n",posiz);
    /* il figlio si riposizioe all'inizio del file */
    printf("figlio: riposiziono all' inizio del file\n");
    error = lseek(file_desc, (long ) 0, 0);
    /* legge posizione corrente, non piu' identica a quella del padre */
    posiz = lseek(file_desc, (long ) 0, 1);
    printf("figlio: posizione corrente = %d\n",posiz);
    error = close(file_desc);
    /* chude il file e termina */
    exit(0);
    }
```

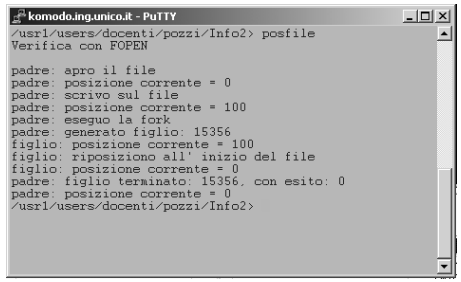

Un terzo esempio (iv)

```
else {
    /* nel padre */
    printf("padre: generato figlio: %d\n",pid);
    /* legge la posizione corrente del file, dopo la */
    /* fseek del figlio */
    pid_figlio = wait(&status);
    printf("padre: figlio terminato: %d, con esito: %d\n",pid_figlio,status);
    posiz = lseek(file_desc, (long ) 0, 1);
    printf("padre: posizione corrente = %d\n",posiz);
  }
  /* chiude il file e termina */
  error = close(file_desc);
  exit(0);
}
```

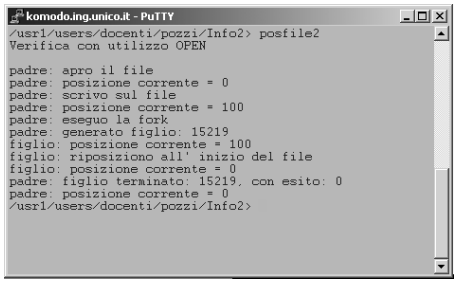

Esecuzione su Linux

n.b. Cygwin, Solaris si comportano come Linux

Esecuzione su Unix System V



Esecuzione su Unix System V



Bibliografia

- Pelagatti G., Sistema Operativo Linux e TCP/IP, Progetto Leonardo, Bologna, 2002 - capitolo 11.