Il file system di UNIX

Dal punto di vista interno, un file system è una struttura dati su disco (volume). Il disco è a sua volta un insieme di blocchi indirizzabili direttamente.

Il disco è diviso logicamente in quattro aree

1. **area di bootstrap**: che coincide con il blocco 0. Contiene le informazioni per il caricamento iniziale del sistema operativo.

2. **superblocco**: che corrisponde al blocco 1. Contiene informazioni necessarie per la gestione dell'intero volume.

3. la lista dei descrittori dei file (la **i-list**). Contiene tutti i descrittori dei file presenti sul volume. Ogni descrittore è chiamato **i-node**, ed è contraddistinto dal suo numero d'ordine (chiamato **i-number**). L'i-number, insieme al nome dell'unità su cui il volume è montato identifica univocamente il file nel sistema.

4. **l'area dei blocchi di dati**: è costituita dai blocchi rimanenti del volume e riguarda il contenuto dei file.

Contenuto dell'I-node

1. **proprietari**: sono due campi contenenti i numeri UID e GID

2. **tipo**: indica il tipo del file. Per esempio, può essere "d" (directory), "b" (file speciale per device a blocchi), etc.

3. **permessi di accesso**: è una sequenza di 12 bit così strutturati:
   a) i primi 3 bit costituiscono i diritti del proprietario
   b) i successivi si riferiscono ai diritti del gruppo del proprietario e al resto degli utenti
   c) il decimo e l'undicesimo bit (setuid, SUID, e setgid, SGID) permettono un cambio momentaneo delle priorità di un file eseguibile durante la sua esecuzione.
   d) l'ultimo bit (detto sticky bit), se abilitato, permette al codice del programma di rimanere in memoria centrale anche se nessun processo fa più riferimento ad esso. Viene usato per programmi du uso frequente (gli editor, i compilatori, etc).

4. **dimensione**: il numero di byte occupati dal file

5. **storia degli accessi**: contiene le date dell'ultima modifica al file, all'ultima modifica edll'i-node, etc.

6. **tabella di indirizzi dei blocchi di dati**: anche se da un punto di vista dell'utente un file è una sequenza ininterrotta di byte, fisicamente è memorizzato in vari blocchi non necessariamente contigui. Questa tabella contiene 13 elementi che rappresentano gli indirizzi di questi blocchi.

Indirizzamento dei blocchi di dati

UNIX mantiene una tabella di 13 elementi contenenti gli indirizzi dei blocchi allocati. Di questi, solamenente i **primi dieci** bit conducono <u>direttamente</u> ai primi 10 blocchi del file.

L'**undicesimo** elemento della tabella contiene l'indirizzo di un blocco ausiliario, il cui contenuto è costituito da un insieme di indirizzi, ciascuno dei quali conduce ad un blocco di dati del file.

Se la dimensione del file supera il numero di blocchi indirizzabili nel modo precedente, occorre arrivare al secondo livello di indirizzamento utilizzando il **dodicesimo** elemento della tabella.
Questo individua un blocco ausiliario, il cui contenuto conduce ad altri blocchi ausiliari. Ciascuno di questi conduce ai blocchi dei dati del file.

Il **tredicesimo** elemento della tabella ha lo stesso significato dei primi due, ma con un ulteriore livello di indirizzamento.

Area super-block

Quest'area contiene le principali informazioni che riguardano le altre aree del volume e costituisce il più importante punto di riferimento per la gestione delle loro strutture dati.

Più precisamente, i dati contenuti in quest'area rappresentano:

1. la dimensione del volume
2. la quantità di blocchi liberi nel volume
3. la lista degli indirizzi dei blocchi liberi (free-list)
4. l'indice del prossimo elemento da leggere nella free-list
5. la dimensione della i-list
6. la quantità degli i-node liberi (free-i-node list)
7. l'indice del prossimo elemento da leggere nella free-i-node list

Primitive di I/O

Le primitive di I/O di UNIX possono essere divise nelle seguenti classi:

1. utilizzo di un file esistente: open, read, write, lseek, close

2. creazione di un file: creat

3. utilizzo dell'i-node: chdir, chmod, stat

4. manipolazione del descrittore di un file: dup

5. manipolazione dell'albero

Strutture dati

Per la gestione delle primitive di I/O, il sistema mantiene in memoria centrale le seguenti tabelle:

1. **tabella dei descrittori dei file aperti da un processo** (tabella privata del file). Ogni elemento della tabella rappresenta un file aperto e contiene, fra l'altro,un puntatore ad un elemento della seguente tabella.

2. **tabella globale dei file aperti**, unica per tutto il sistema. Riguarda tutti i file aperti dai vari processi. Ogni suo elemento rappresenta un file e contiene (tra l'altro) la modalità con cui è stato aperto e la posizione in cui dovrà essere effettuata la prossima operazione.
   Oltre a questo, in ogni elemento della tabella è memorizzato un puntatore ad un in-core i-node relativo al file in esame.

3. **tabella degli in-core i-node**: anch'essa globale per il sistema. In essa sono mantenute tutte le informazioni necessarie al reperimento effettivo dei dati di tutti i file aperti. E' l'immagine di una parte degli i-node su disco

Primitiva open

Prima di poter accedere ad un file questo deve essere aperto mediante la primitiva:

fd = open(pathname, tipo, mod)

dove

- "pathname" rappresenta il nome simbolico del file
- "tipo" indica la modalità di apertura (lettura/scrittura)
- "mod" (indica la configurazione dei bit di protezione del file qualora si dovesse creare il file
- "fd" è il numero di descrittore del file

Ogni volta che un processo richiede l'apertura di un file, il sistema accede al corrispondente in-core i-node (se questo è presente), altrimenti preleva la sua copia dal disco.

Se il richiedente può accedere al file il sistema alloca un nuovo elemento nella tabella globale dei file aperti. Questo elemento contiene, oltra all'indirizzo che porta all'i-node interessato, un numero chiamato "offset" che rappresenta la posizione per il prossimo accesso. Questo numero sarà inizialmente 0 se il file è aperto in lettura/scrittura; sarà uguale alla dimensione del file se aperto in modalità APPEND.
Infine, il sistema alloca un nuovo elemento nella tabella privata del processo che ha richiesto la open.

L'allocazione del nuovo elemento nella tabella privata di file per un processo avviene cercando (a partire dall'elemento 0) il primo posto libero. L'indice dell'elemento trovato è il numero descrittore (fd).

E' importante notare che per uno stesso file vengono allocati tanti elementi sulle due tabelle di file quante sono le volte che il file viene aperto. Tutti questi processi convergono verso un unico in-core i-node.

Esempio

Processo A

```
fd1 = open("/usr/f1", O_RDONLY);
fd2 = open("f2", O_APPEND | O_RDONLY);
```

Processo B

```
fd6 = open("f3", O_WRONLY);
fd7 = open("/usr/f1", O_APPEND | O_WRONLY);
fd9 = open("f4", O_RDWR);
```

Primitiva creat

La creazione di un file può essere ottenuta con la primitiva

fd = creat(pathname, mod)

L'effetto di creat è quello di generare un nuovo file se pathname non corrisponde a nessun file esistente; altrimenti, il contenuto di pathname viene cancellato

Primitive read e write

Un file può essere letto tramite la primitiva:

totbyte = read(file_des, buf, nbyte)

dove

- "file_des" è il numero descrittore del file interessato
- "buf" è l'indirizzo di un'area dati del processo dove verranno depositati i dati letti
- "nbyte" è il numero di byte che si vogliono leggere
- "totbyte" è il numero di byte effettivamente letti

L'aggiornamento di un file può essere ottenuto con la primitiva

totbyte = write(file_des, buf, nbyte)

Primitiva lseek

La lettura di un file mediante read è sequenziale, a partire dal primo carattere del file stesso. Esiste la possibilità di accedere ad una posizione qualsiasi del file mediante la primitiva lseek:

new_pos = lseek(fd, offset,base)

La primitiva permette l'accesso alla posizione n-esima del file con numero di descrittore "fd", dove

n=base+offset

Il parametro base può assumere i seguenti valori:

0: inizio del file
1: fine del file
2: posizione corrente nel file

"offset" può anche assumere valori negativi.

Le operazioni effettuate dal sistema per l'esecuzione della lseek consistono nell'aggiornamento del campo "offset" dell'elemento associato a "fd" nella tabella dei file aperti

Primitiva dup

Il numero descrittore di un file aperto può essere duplicato dalla chiamata:

another_fd = dup(fd)

dove "another_fd" è il nuovo numero descrittore ottenuto, mentre "fd"è quello originale.

L'effetto della chiamata è quello di allocare il numero descrittore del primo elemento libero nella tabella privata dei file aperti, facendolo convergere allo stesso file puntato dal descrittore originale.

Ciò significa che i due descrittori punteranno ad un unico elemento della tabella globale (al contrario della open)

Esempio:

```
fd1 = open("/usr/f1", O_RDONLY);
fd2 = open("f2", O_WRONLY);
fd3 = open("/usr/f1", O_WRONLY);
fd4 = dup(fd2);
```

Primitiva close

Un file aperto da un processo può essere chiuso tramite la primitiva close:

close(fd)

dove "fd" è il numero descrittore del file.

La chiusura di un file determina la liberazione del corrispondente elemento nella tabella privata dei file aperti.

Se il campo "numero di riferimento" nella tabella globale è 1, l'elemento viene liberato. Le stesse operazioni vengono effettuate sull'in-core i-node.
Se invece è maggiore di 1, il campo viene decrementato (anche quello dell'in-core i-node).