File e Directory

capitolo 4 - stevens

Vitiello Autilia, PhD Student
Facoltà di Scienze MM.FF.NN.
Università degli Studi di Salerno
vitiello@dia.unisa.it
http://www.dia.unisa.it/dottorandi/avitiello/

FUNZIONI STAT, FSTAT, LSTAT

```
#include <sys/types.h>
#include <sys/stat.h>

int stat (const char *pathname, struct stat *buf);
int fstat (int fd, struct stat *buf);
int lstat (const char *pathname, struct stat *buf);
```

- Descrizione: danno informazioni sul file preso come primo argomento
- Restituiscono: 0 se OK

  -1 in caso di errore

FUNZIONI STAT, FSTAT, LSTAT - 2

- **`stat`**: fornisce una struttura di info relative al file dato come primo argomento

- **`fstat`**: come prima, ma il file cui si riferisce è già aperto e quindi prende il file descriptor

- **`lstat`**: le info ottenute sono relative al link simbolico (e non al file a cui esso riferisce)

Struct stat

```
struct stat {
  mode_t   st_mode;     /* file type & mode (permissions) */
  ino_t    st_ino;      /* i-node number (serial number) */
  dev_t    st_dev;      /* device number (filesystem) */
  dev_t    st_rdev;     /* device number for special files */
  nlink_t  st_nlink;    /* number of links */
  uid_t    st_uid;      /* user ID of owner */
  gid_t    st_gid;      /* group ID of owner */
  off_t    st_size;     /* size in bytes, for regular files */
  time_t   st_atime;    /* time of last access */
  time_t   st_mtime;    /* time of last modification */
  time_t   st_ctime;    /* time of last file status change */
  long     st_blksize;  /* best I/O block size */
  long     st_blocks;   /* number of 512-byte blocks allocated */
};
```

Tipi di dati di sistema primitivi definiti in <sys/types>

Tipi di File

- ***Regular file*** = dal punto di vista del kernel un file regolare contiene testo oppure è binario;
- ***Directory file*** = contiene nomi e puntatori ad altri file;
- ***Character special file*** = usato per individuare alcuni dispositivi del sistema;
- ***Block special file*** = usato per individuare i dischi;
- ***Pipe e FIFO*** = usati per la comunicazione tra processi;
- ***Symbolic link*** = un tipo di file che punta ad un altro file;
- ***Socket*** = usato in per la comunicazione in rete tra processi.

Macro per tipi di file

- Le macro seguenti sono funzioni booleane che aiutano ad identificare il tipo di un file verificando ciò che è contenuto nel campo **`st_mode`** della struttura `stat` del file

Macro	Type of file
`S_ISREG()`	regular file
`S_ISDIR()`	directory file
`S_ISCHR()`	character special file
`S_ISBLK()`	block special file
`S_ISFIFO()`	pipe or FIFO
`S_ISLNK()`	symbolic link (not in POSIX.1 or SVR4)
`S_ISSOCK()`	socket (not in POSIX.1 or SVR4)

ESEMPIO

```
#include      <sys/types.h>
#include      <sys/stat>
#include      "ourhdr.h"

int main(int argc, char *argv[])
{
 int  i;   struct stat   buf;

 for (i=1;i<argc;i++){
  printf("%s:", argv[i]);
  if (lstat(argv[i],&buf) <0) {
              err_ret("lstat error");
              continue;
      }
  if (S_ISREG(buf.st_mode))  printf("regular");
  else if (S_ISDIR(buf.st_mode)) printf("directory");
  else if (S_ISCHR(buf.st_mode)) printf("character special");
  else if (S_ISBLK(buf.st_mode)) printf("block special");
       ……….
    }
   exit(0);
}
```

ID dei Processi

- Il campo **`st_uid (st_gid)`** della struttura `stat` contiene l'ID dell'utente (gruppo) possessore del file.

- Ogni processo ha degli ID associati:
  - ***real id***: chi siamo realmente presi dal file */etc/passwd* al login time;
  - ***effective id*** : determina i permessi di accesso ai file;
  - ***saved set id***: contengono copie dell'effective id quando è eseguito un programma (exec);

SET-USER-ID & SET-GROUP-ID

- quando un programma è eseguito normalmente *effective id* è uguale al *real id*

- ...ma si può settare un flag speciale ***set-user-id*** nel campo `st_mode` che fa si che il processo sia eseguito con effective user id = proprietario (`st_uid`) del file

- Allo stesso modo si può settare il flag ***set-group-id*** nel campo `st_mode` che fa si che il processo sia eseguito con effective group id= group id (`st_gid`) del file
- Questi bit possono essere testati usando le costanti **`S_ISUID`** e **`S_ISGID`**

Esempio: Set User-ID

- pippo.doc è un file di pippo sul quale solo pippo può scrivere
- scrivi è un word-processor di pippo che può essere usato da tutti

- Domanda 1: pippo può modificare pippo.doc usando *scrivi?*
  - SI

- *Domanda 2:* l'utente pluto può modificare pippo.doc usando *scrivi* di pippo?
  - NO! Ameno che …. *scrivi* ha il set-user-id flag settato

PERMESSI DI ACCESSO AI FILE

- **`st_mode`** nella struttura `stat` include anche 9 bit che regolano i permessi di accesso al file cui esso si riferisce

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

Accesso ai File

- Gli ID dell'owner (user & group) sono proprietà di file infatti hanno un campo della `struct stat`.
- Gli effective ID (user & group) sono proprietà del processo che utilizza quel file (apri, chiudi, etc.)
- Per aprire un file (lettura o scrittura) bisogna avere permesso di esecuzione in tutte le directory contenute nel path assoluto del file;
- Per creare un file bisogna avere permessi di scrittura ed esecuzione nella directory che conterrà il file.

Algoritmo Di Accesso Ai File

- Se effective user id =0
  - accesso libero (superuser)
- Se effective user id = owner ID del file
  - accesso in accordo ai permessi del proprietario
- Se effective group id = group ID del file
  - accesso in accordo ai permessi del gruppo
- accesso in accordo ai permessi di *other*

- Queste verifiche sono eseguite esattamente in questo ordine
  - Quindi: se il processo è proprietario del file (caso 2) l'accesso è garantito o negato seguendo i permessi del proprietario, gli altri permessi (group e other) non saranno analizzati.

Nuovi file e directory

- quando si creano nuovi file, l'user id è settato come l'effective user ID del processo che sta creando il file.

- il group id è il group ID della directory nel quale il file è creato oppure il group id del processo.

CHMOD E FCHMOD

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod (const char *pathname, mode_t mode);
int fchmod (int fd, mode_t mode);
```

- Descrizione: cambiano i bit di permesso del file dato come primo argomento
- Restituiscono: 0 se OK, -1 in caso di errore

- Per cambiare i permessi, l'effective user id del processo deve essere uguale all'owner del file, o il processo deve avere i permessi di root.
- Il *mode* è specificato come l'OR bit a bit di costanti che rappresentano i vari permessi.

COSTANTI PER CHMOD

mode	Description
S_ISUID	set-user-ID on execution
S_ISGID	set-group-ID on execution
S_ISVTX	saved-text (sticky bit)
S_IRWXU	read, write, and execute by user (owner)
S_IRUSR	read by user (owner)
S_IWUSR	write by user (owner)
S_IXUSR	execute by user (owner)
S_IRWXG	read, write, and execute by group
S_IRGRP	read by group
S_IWGRP	write by group
S_IXGRP	execute by group
S_IRWXO	read, write, and execute by other (world)
S_IROTH	read by other (world)
S_IWOTH	write by other (world)
S_IXOTH	execute by other (world)

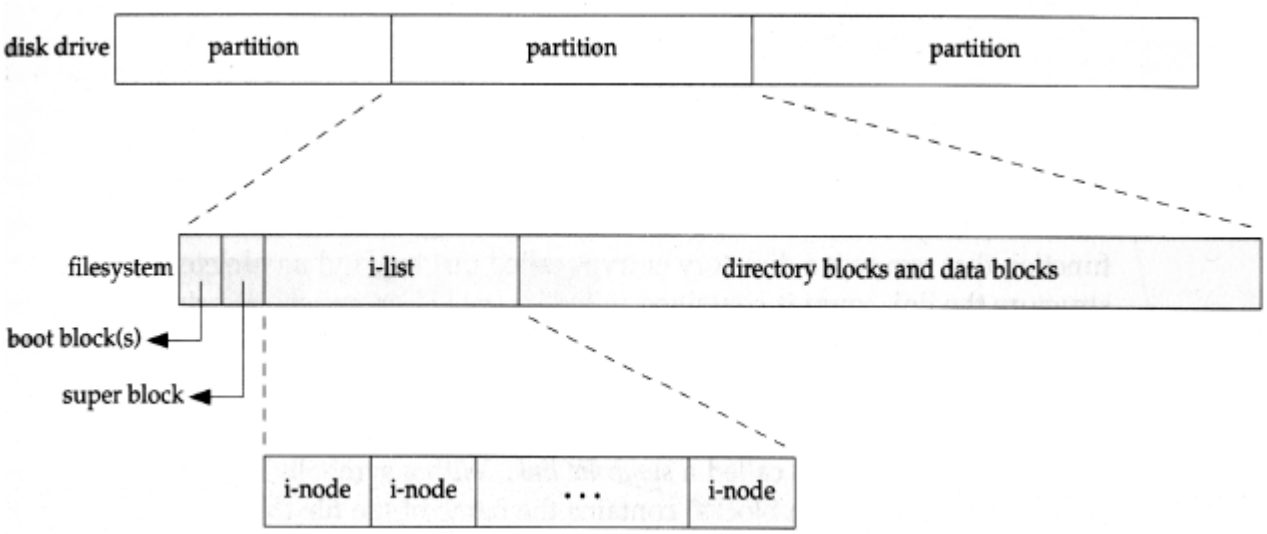
Esempio

```
#include        <sys/types.h>
#include        <sys/stat.h>
#include        "ourhdr.h"
int main(void)
{
    struct stat statbuf;

    /* turn on set-group-ID and turn off group-execute */
    if (stat("foo", &statbuf) < 0)
         err_sys("stat error for foo");
    if (chmod("foo",(statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
         err_sys("chmod error for foo");

    /* set absolute mode to "rw-r--r--" */
    if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
         err_sys("chmod error for bar");

    exit(0);
}
```

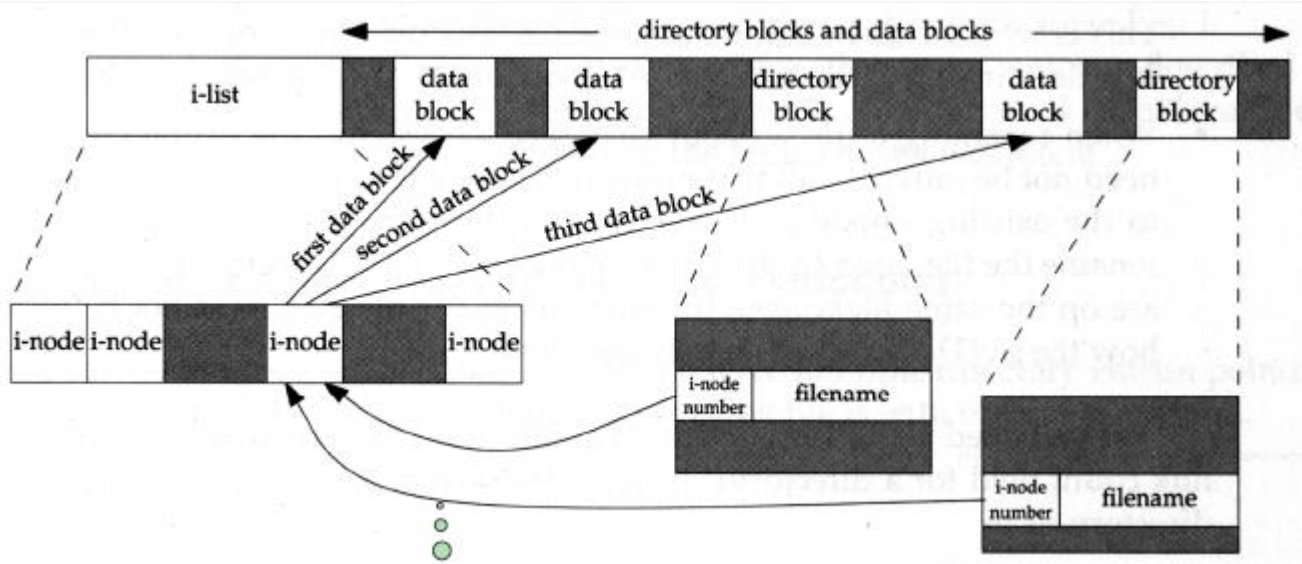
Dimensione Dei File

- La dimensione dei files (in bytes) è in **st_size** (della struttura `stat`)
- La dimensione del blocco utilizzato nelle operazioni di I/O è contenuto in **st_blksize**
- Il numero di blocchi da 512 byte allocati per il file è contenuto in **st_blocks**

FileSystem

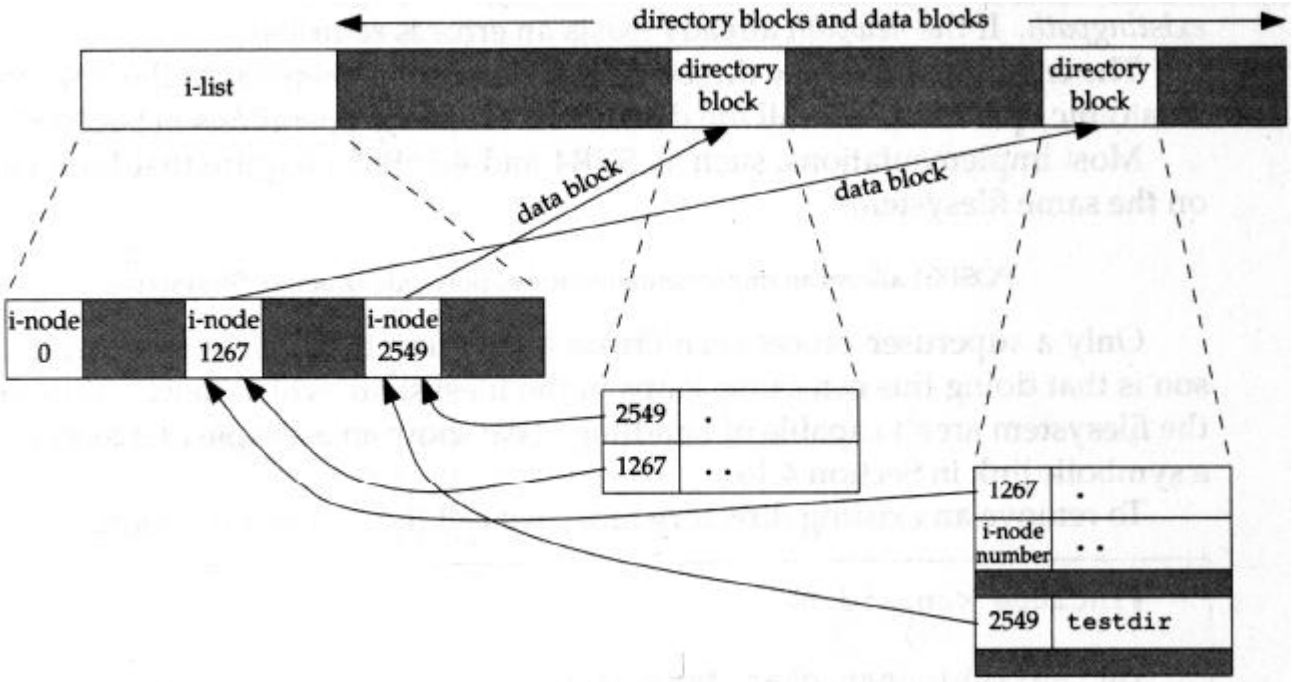

disk drive
partition
partition
partition
filesystem
i-list
directory blocks and data blocks
boot block(s)
super block
i-node
i-node
. . .
i-node

FileSystem - 2



- i-nodo contiene tutte le info che riguardano il file
  - tipo del file
  - bit di permesso
  - size del file
  - puntatori ai blocchi di dati del file
- directory block: è una lista di record aventi almeno due campi
  - numero dell'i-nodo
  - nome del file

FileSystem: Esempio



Hard Link

- Ogni i-node ha un contatore di link che contiene il numero di directory entry che lo puntano
  - solo quando scende a zero, allora il file può essere cancellato

- il contatore è nella `struct stat` nel campo **`st_nlink`**
- questi tipi di link sono detti **hard link**
- non si possono fare hard link tra filesystem differenti

- hard link possono essere creati usando la funzione **`link`**
- solo un processo super-user può creare un nuovo link (usando link) che punti ad una directory
- **`unlink`** decrementa il contatore di link

LINK

```
#include <unistd.h>
int link (const char *path, const char *newpath);
```

- Descrizione: crea una nuova directory entry *newpath* che si riferisce a *path*

- Restituisce: 0 se OK,

  -1 in caso di errore (anche se `newpath` già esiste)

Link

- link crea un hard link aggiuntivo (con la corrispondente directory entry) ad un file esistente
- link crea automaticamente il nuovo link (e quindi la nuova directory entry) ed incrementa anche di uno il contatore dei link **`st_nlink`** (è una operazione atomica!)
- Il vecchio ed il nuovo link, riferendosi allo stesso i-node, condividono gli stessi diritti di accesso al "file" cui essi si riferiscono

UnLink

```
#include <unistd.h>
int unlink (const char *pathname);
```

- Descrizione: rimuove la directory entry specificata da *pathname* ed decrementa il contatore dei link del file cui il link si riferisce

- Restituisce: 0 se OK,
  -1 in caso di errore

UnLink

- `unlink` è consentita solo se si ha il permesso di scrittura ed esecuzione nella directory dove è presente la directory entry

- Se tutti i link ad un file sono stati rimossi e nessun processo ha ancora il file aperto, allora tutte le risorse allocate per il file vengono rimosse e non è più possibile accedere al file

- Se però uno o più processi hanno il file aperto quando l'ultimo link è stato rimosso, pur essendo il contatore dei link a 0 il file continua ad esistere e sarà rimosso solo quando tutti i riferimenti al file saranno chiusi

Link Simbolici

- hard link non possono attraversare file system differenti ed hard link a directory possono essere creati solo dal *superuser*

- I soft link, invece contraddicono entrambe le cose
  - sono dei puntatori indiretti ad un file

- Quando si usano funzioni che si riferiscono a file (open, read, stat, etc.), si deve sapere se seguono il link simbolico o no
  - Si: ci si riferisce al file puntato;
  - No: ci si riferisce al link;

Symbolic Link - Creazione

```
#include <unistd.h>
int symlink (const char *path, const char *sympath);
```

- Descrizione: crea un link simbolico *sympath* che punta a *path*

- Restituisce: 0 se OK
  -1 altrimenti

SYMLINK

- Quando `symlink` crea il link simbolico *`sympath`* verrà creata un directory entry nella directory cui ci si riferisce e tale entry avrà un suo proprio i-node

- Non è indispensabile che *`path`* esista quando il link simbolico è creato.

- Non è necessario che *`path`* e *`sympath`* risiedano nello stesso filesystem.

READLINK

```
#include <unistd.h>
int readlink (const char *pathname, char *buf,
  int bufsize);
```

- Descrizione: legge dal link simbolico dato come primo argomento e ne mette il contenuto in *buf* la cui taglia è *bufsize*

- Restituisce: il numero di byte letti se OK

  -1 in caso di errore

READLINK

- Legge il contenuto del link e non del file cui esso si riferisce
- Se la lunghezza del link simbolico è > *`bufsize`* viene dato l'errore
- combina insieme le funzioni di `open`, `read` e `close` sul link simbolico

I Tempi Dei File

- per ciascun file 3 tempi sono gestiti (essi sono presenti nella struttura *`stat`)*
  - `st_atime` = la data dell'ultimo accesso al file (`read`)
  - `st_mtime` = la data dell'ultima modifica al file (`write`)
  - `st_ctime` = la data dell'ultimo cambiamento apportato all'i-node (`chmod`, `chown`)

- i tempi di modifica di una directory sono relativi alla creazione o cancellazione dei suoi file non ad operazioni di lettura o scrittura nei suoi file.

Rename

```
#include <stdio.h>

int rename (const char *oldname, const
  char *newname)
```

- Descrizione: assegna un nuovo nome *newname* ad un file od ad una directory data come primo argomento
  - Restituisce: 0 se OK,

    -1 in caso di errore

MKDIR

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int mkdir (const char *pathname, mode_t mode);
```

- Descrizione: crea una directory i cui permessi di accesso vengono determinati da *mode*.

- Restituisce: 0 se OK,
  -1 in caso di errore

Mkdir

- La directory creata avrà come
  - owner ID = l'effective user ID del processo
  - group ID = il group ID della directory padre
  - vedi altre caratteristiche con man 2 mkdir
- La directory sarà vuota ad eccezione di . e ..

RMDIR

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int rmdir (const char *pathname);
```

- Descrizione: viene decrementato il numero di link al suo i-node; se esso =0 si libera la memoria solo se nessun processo ha quella directory aperta.

- Restituisce: 0 se OK,
  -1 in caso di errore

Leggere Directory

```
#include <dirent.h>

DIR *opendir(const char *pathname);
```

- ritorna NULL su errore

```
struct dirent *readdir(DIR *dp);
```

- ritorna NULL se non ci sono più elementi

```
struct dirent{
     ino_t d_ino; /* i-node number*/
     char d_name[256] /*filename*/
}
```

Esempio

```
#include      <dirent.h>

int main()
{
 DIR *dp;    struct dirent *item;

 if((dp = opendir(".")) == NULL)
       {printf("Errore"); exit(1);}

 item = readdir(dp);
 if(item != NULL)
   printf("Primo file: %s", item->d_name);


   return(0);
}
```

Esempio 2

```
#include      <dirent.h>

int main(int argc, char *argv[])
{
 DIR *dp;    struct dirent *item;

If(argc < 2)
   exit(0);
 if((dp = opendir(argv[1])) == NULL)
       {printf("Errore"); exit(1);}

 while((item = readdir(dp)) != NULL)
 {
   printf("file: %s", item->d_name);
 }
```