Files e Directories

Franco Maria Nardini

stat(2)

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

int stat(const char *path, struct stat *sb);
int lstat(const char *path, struct stat *sb);
int fstat(int fd, struct stat *sb);
```

Returns: 0 if OK, -1 on error

- Famiglia di system call
- Ritornano gli attributi estesi del file riferito
  - in caso di link simbolico, `lstat(2)` ritorna attributi del link stesso
  - le altre, attributi del file puntato

stat(2)

```
struct stat {
    dev_t      st_dev;          /* device number (filesystem) */
    ino_t      st_ino;          /* i-node number (serial number) */
    mode_t     st_mode;         /* file type & mode (permissions) */
    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_blocks;       /* number of 512-byte* blocks allocated */
    long       st_blksize;      /* best I/O block size */
};
```

- `st_mode` codifica il tipo di file: regular, special, ecc.

UNIX file types

- Tipi di file
  - *regular file*: file comune, listati con un “-“ davanti
  - *directory*: file speciale più comune
  - *symbolic link*: reference ad un altro file. ricorda nulla? :)
  - *named pipe*: comunicazioni inter-processo (più avanti)
  - *socket*: comunicazioni inter-processo (più avanti)
  - *device file*: tutto è un file, anche un disco
    - device a caratteri: stream in input e output
    - device a blocchi: accesso diretto (dischi possono essere entrambe)
  - *door*: comunicazioni inter-processo (client-server)

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/04/still-
simple-ls.c

$ cc -Wall -o myls still-simple-ls.c

$ ./myls .

$ ln -s myls pippo

$ ./myls .
```

`struct stat`

- Ogni processo ha sei o più IDs associati
  - userID, groupID reali (utente loggato)
  - userID, groupID effettivi (associati al file)
  - set-user-ID, set-group-ID (salvati da **`exec`**)
- **`setuid(2)`** setta userID effettivo con **`st_uid`**
- **`setgid(2)`** setta groupID effettivo con **`st_gid`**
- **`st_uid`** e **`st_gid`** identificano sempre utente e gruppo proprietario

`struct stat`

- **`st_mode`** definisce anche i permessi di accesso al file:
  - S_IRUSR, S_IWUSR, S_IXUSR
  - S_IRGRP, S_IWGRP, S_IXGRP,
  - S_IROTH, S_IWOTH, S_IXOTH

struct stat

- Accesso al file:
  - per aprire un file, permessi di esecuzione su tutte le cartelle del path
  - per aprire un file in O_RDONLY e O_RDWR servono permessi in lettura
  - per aprire un file in O_WRONLY e O_RDWR servono permessi in scrittura
  - per l'uso di O_TRUNC servono permessi in scrittura
  - per creare un nuovo file, permessi di scrittura E esecuzione per la directory
  - per cancellare un file, permessi di scrittura E esecuzione per la directory, nessun requisito per il file
  - per eseguire un file (con la famiglia **`exec`**), servono permessi di esecuzione

access(2)

```
#include <unistd.h>

int access(const char *path, int mode);

                                        Returns: 0 if OK, -1 on error
```

- Verifica l'accessibilità del file sulla base dei userID/groupID reali.
- Consente a **`setuid(2)`** e **`setgid(2)`** di verificare se l'utente reale può accedere al file
- **`mode`** può essere un bitwise OR di:
  - R_OK, controllo permessi di lettura
  - W_OK, controllo permessi di scrittura
  - X_OK, controllo permessi di esecuzione
  - F_OK, test di esistenza del file

Esempio I

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/04/access.c

$ cc -Wall access.c

$ ./a.out /etc/passwd

$ ls -l /etc/passwd

$ ./a.out /etc/master.passwd

$ ls -l /etc/master.passwd
```

Esempio II

```
$ sudo chown root ./a.out

$ sudo chmod 4755 ./a.out

$ ./a.out /etc/passwd

$ ./a.out /etc/master.passwd
```

Permessi

- Il set di permessi è determinato (lista ordinata):
  - se userID effettivo == 0: consenti accesso
  - se userID effettivo == **`st_uid`**
    - se bit permesso utente settato: consenti accesso, altrimenti nega
  - se groupID effettivo == **`st_gid`**
    - set bit permesso gruppo settato: consenti accesso, altrimenti nega
  - se bit permesso “other” settato: consenti accesso, altrimenti nega

Nuovi File

- Nuovi file:
  - `st_uid` = userID effettivo
  - `st_gid`
    - groupID effettivo del processo
    - groupID della cartella che lo ospita

umask(2)

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

mode_t umask(mode_t numask);

                                    Returns: previous file mode creation mask
```

- Setta la maschera per la creazione dei file
  - i bit indicati nella maschera vengono messi a off nel file
- Ogni utente ha una maschera di default
  - se processo ha esigenze specifiche può modificare la maschera
  - la modifica tocca solo il processo

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/04/umask.c

$ cc -Wall umask.c

$ ./a.out

$ ls -l .
```

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/04/chmod.c

$ cc -Wall chmod.c

$ umask 077

$ touch foo

$ touch foo1

$ ./a.out
```

chown(2)

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);

                                        Returns: 0 if OK, -1 on error
```

- cambia **`st_uid`** e **`st_gid`** per un file
- *owner* e *group* possono assumere valore -1 per indicare che devono rimanere gli stessi
- non-superusers possono cambiare **`st_gid`** se:
  - userID effettivo == **`st_uid`** E
  - *owner* == userID del file E
  - *group* == groupID effettivo o supplementari
- puliscono i bit **`set_uid`** e **`set_gid`**

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/04/size.c

$ cc -Wall size.c

$ ./a.out ./size.c
```

Homework

- Del programma che legge il file dei primi 100 numeri, lo ordina e lo riscrive ordinato su un altro file.
  - verificare che rimuovendo il permesso in lettura per l'owner, non consente l'accesso anche se la lettura è consentita per il gruppo.
- studiare le pagine man delle funzioni analizzate.
- settare umask nella shell, creare nuovi file e verificare le conseguenze.