Laboratorio di Sistemi Operativi

II Semestre - Marzo/Giugno 2008
Matricole congr. 0 mod 3

File & Directory

stat,fstat e 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 1° argomento

Restituiscono: 0 se OK
-1 in caso di errore

funzioni stat, fstat, lstat

- stat: fornisce una struttura di info relative al file del 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 si riferisce)

funzioni stat, fstat, lstat

- per tutte: bisogna fornire un puntatore ad una struttura (chiamata "stat") che viene poi riempita durante lo svolgimento della funzione.
- Tipico utilizzatore di tali funzioni è il comando shell ls –l

fornisce informazioni circa un file dato come argomento

```
bash>  ls -l  file.c
-rwxrw-rw-  1 cicalese   14441 Mar 18 16:35 file.c
```

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>

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)

tipi di file

- Regular file = dal punto di vista del kernel un file regolare contiene testo op. è binario
- Directory file = contiene nomi e puntatori ad altri file; solo il kernel può scriverci
- Character special file= usato per individuare alcuni dispositivi del sistema. Es: /dev/tty (la tastiera)
- Block special file = usato per individuare i dischi . Es:/dev/hda1
- Pipe e FIFO = usati per la comunicazione tra processi
- Symbolic link = un tipo di file che punta ad un altro file
- Socket = usato per la comunicazione in rete tra processi


```
#include      <sys/types.h>
#include      <sys/stat.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) {
              printf("lstat error\n");
              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 u/g ID, effective u/g ID, saved set-u/g-ID
  - Normalmente effective user ID coincide con real user ID

ID dei processi

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

Set-User-ID & Set-Group-ID

- quando un programma è eseguito normalmente effective=real
- ...ma si può settare un flag speciale nel campo **st_mode** che fa si che il processo sia eseguito con effective=proprietario (o group) del file eseguibile
- Questi bit possono essere testati usando le costanti S_ISUID e S_ISGID

Set User-ID

pippo.doc è un file di pippo

pippo può modificare pippo.doc usando *scrivi*? SI!

scrivi è un word-processor di pippo che può essere usato da tutti

l'utente pluto può modificare pippo.doc usando *scrivi* di pippo? NO! tranne se ....
scrivi ha il set-user-id flag settato

esercizi

1. scrivere un programma che testa se un file ha il flag *set-user-id* settato
   1. ricorda che i flag set-u/g-ID sono nel campo **st_mode**
   2. Hint: AND con le costanti S_ISUID e S_ISGID per testarli
2. inserire un nuovo utente ed implementare esempio precedente

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.)

Accesso ai file

- 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

1. eff. uid =0 --> accesso libero
2. eff. uid = owner ID
   - accesso in accordo ai permessi
3. eff. gid = group ID
   - accesso in accordo ai permessi
4. accesso in accordo ai permessi di *other*

- Queste verifiche sono eseguite esattamente in questo ordine

?

- Se si vuole dare accesso a tutti, quali permessi bisogna settare?

Nuovi file e directory

- quando si creano nuovi file, l'uid è settato come l'effective ID del processo che sta creando il file
- il gid è il group ID della directory nel quale il file è creato oppure il gid del processo

come è la situazione in Linux?

access

```
#include <unistd.h>

int access (const char *pathname, int mode);
```

Descrizione: verifica se il real ID ha accesso al file 1° argomento nella modalità specificata da *mode*

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

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission
F_OK	test for existence of file

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

int main(int argc, char *argv[])
{
        if (argc != 2)
         { printf("usage: a.out <pathname>"); exit(0);}

        if (access(argv[1], R_OK) < 0)
                printf("access error for %s", argv[1]);
        else
                printf("read access OK\n");

        if (open(argv[1], O_RDONLY) < 0)
                printf("open error for %s", argv[1]);
        else
                printf("open for reading OK\n");
        exit(0);
}
```



```
$ ls -l  a.out
$ -rwxrwxr-x 1 cicalese 1234 jan 18 08:48 a.out
$ a.out a.out
read access OK
open for reading OK
$ ls -l  prova
$-rw-r----- 1 rescigno 1234 jan 18 15:48 prova
$a.out  prova
access error for prova: Permission denied
open error for prova: Permission denied
$su
# chown rescigno a.out
# chmod u+s a.out
# ls -l a.out
# -rwsrwxr-x 1 rescigno 1234 jan 18 08:48 a.out
# exit
$ a.out prova
access error for prova: Permission denied
open for reading OK
```