Laboratorio di Sistemi Operativi

Marzo-Giugno 2008
Matr. Congr. 0 mod 3

File & Directory
(2)

umask

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

mode_t umask (mode_t cmask);
```

Descrizione: setta la maschera di creazione per l'accesso ad un file

Restituisce: la maschera di creazione precedente

(nota che non restituisce valori di errori)

umask

- Viene usato ogni volta che il processo crea un nuovo file o directory, secondo il seguente criterio:
  - setta la maschera di creazione (*cmask*)
  - comando: umask
  - alla creazione del file viene fatto l'AND tra la maschera negata e il *mode* della creazione del file

umask: 022 (--- -w- -w-)	000 010 010	
negaz.: 755 (rwx r-x r-x)	111 101 101	AND
creat("pippo.txt", 665)	110 110 101	
rw- r– r-x	110 100 101	

?

- Qual è il valore di default per la maschera di di creazione degli accessi ai file?

```
#include      <sys/types.h>
#include      <sys/stat.h>
#include      <fcntl.h>
/*esempio di utilizzo di umask*/
int main(void)
{
  umask(0);
  if(creat("foo",S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|
                                        S_IROTH|S_IWOTH)<0)
       printf("creat error for foo");

   umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH); /* 066 */
   if (creat("bar",S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|
                                    S_IROTH|S_IWOTH)<0)
       printf("creat error for bar");
   exit(0);
}
```


Risultati programma precedente

- rw- rw- rw- foo
- rw- --- --- bar

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 1° argomento
Restituiscono: 0 se OK, -1 in caso di errore

chmod *e* fchmod

- Per cambiare i permessi, l'effective uid del processo deve essere uguale all'owner del file, o il processo deve avere i permessi di root/superuser
- 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	4000
S_ISGID	set-group-ID on execution	2000
S_ISVTX	saved-text (sticky bit)	1000
S_IRWXU	read, write, and execute by user (owner)	700
S_IRUSR	read by user (owner)	400
S_IWUSR	write by user (owner)	200
S_IXUSR	execute by user (owner)	100
S_IRWXG	read, write, and execute by group	070
S_IRGRP	read by group	040
S_IWGRP	write by group	020
S_IXGRP	execute by group	010
S_IRWXO	read, write, and execute by other (world)	007
S_IROTH	read by other (world)	004
S_IWOTH	write by other (world)	002
S_IXOTH	execute by other (world)	001


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

int main(void)
{
    struct stat statbuf;

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

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

    exit(0);
}
```

?

- ...e se invece si vuole settare il group-execute bit?

```
chmod("foo", (statbuf.st_mode | S_IXGRP))
```


sticky bit

- ereditato da versioni "vecchie" di Unix
- se settato, una copia del programma viene salvata nella *swap area* così che la prossima volta che viene lanciato, è caricato in memoria più velocemente
- ora, con la memoria virtuale, non si usa più
- solo il superuser lo può modificare per un file regolare

sticky bit per directory

- se settato, un file in questa directory può essere *removed*, o *renamed* solo se:
  - l'utente ha il permesso di scrittura nella directory, ed
  - è vera almeno una delle seguenti condizioni
    1. è proprietario del file
    2. è proprietario della directory
    3. è *root*

Situazione Tipica: directory come /tmp

chown

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

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

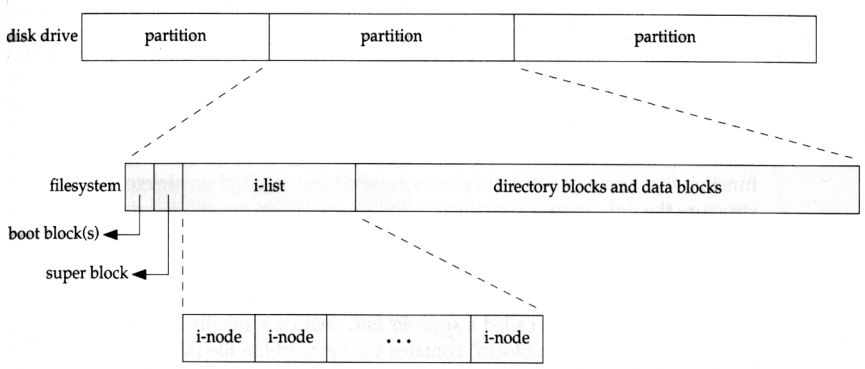
Descrizione: cambiano il proprietario ed il gruppo del file 1° argomento e li settano uguale a *owner* e *group*

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

dimensione di 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**

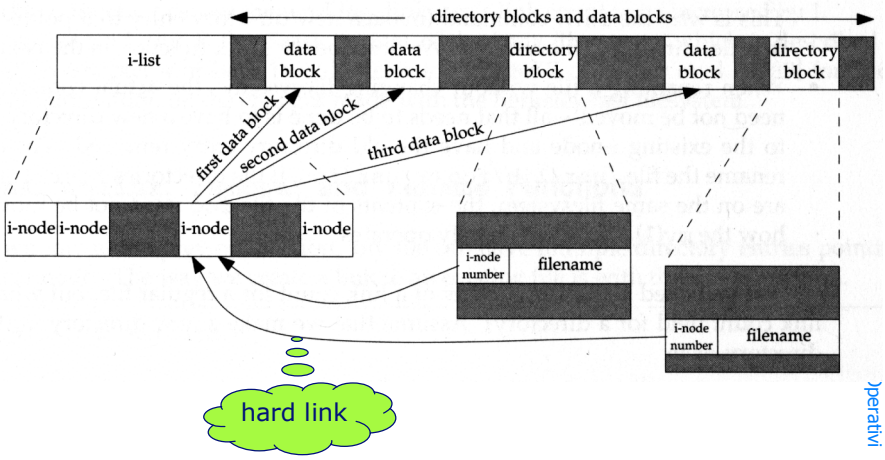
Filesystem



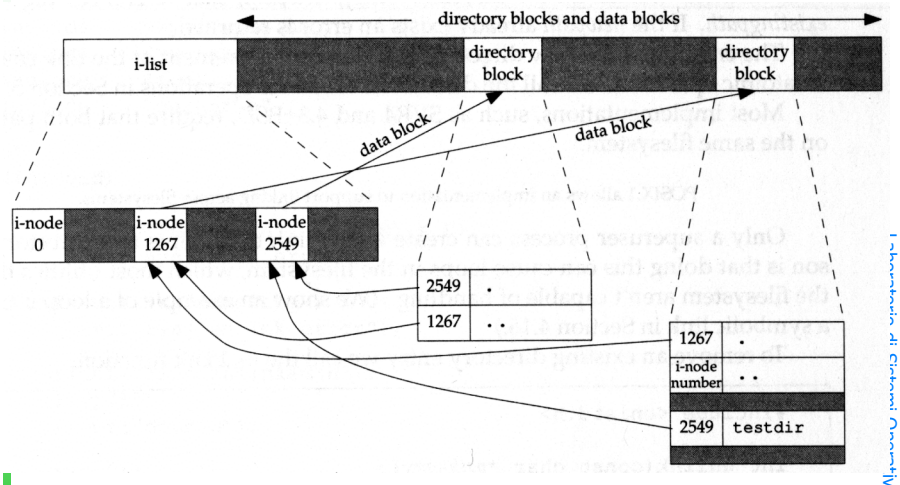
i-nodo

- 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 (i-node)



filesystem: esempio



filesystem: osservazioni

- 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 (i blocchi sono rilasciati...funzione unlink)
- il contatore è nella struct stat nel campo **st_nlink**
- questi tipi di link sono detti hard link
- gli altri sono soft link (link simbolici)
  - lib → usr/lib (7 caratteri di dati nel *data block*)
- non si possono fare hard link tra filesystem differenti

hard link

- un file può avere più di una directory entry che punta al suo i-node, cioè più hard link il cui numero è contenuto in **st_nlink**
- 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
- quando il contatore va a zero, il blocco è rilasciato

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

link simbolici

- hard link non possono *attraversare* file system differenti ed hard link a directory possono essere creati solo dal *superuser*
- Per i soft link, invece tali limitazioni non esistono
- 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 vero file
  - no → ci si riferisce al link
- vedere figura 4.17 (es: open si, lstat no)

unlink

```
#include <unistd.h>

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

Descrizione: rimuove la "directory entry" specificata da *pathname* e se *pathname* è un hard link allora 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
- solo il superuser può rimuovere un hard link ad una directory

unlink

- 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

unlink

- Se pathname e' un link simbolico, unlink rimuove il link simbolico, non il file cui esso punta.

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

int main(void)
{
      if (open("tempfile", O_RDWR) < 0)
            printf("open error");

      if (unlink("tempfile") < 0)
      {printf("unlink error"); exit(0); }

      printf("file unlinked\n");
      sleep(15);
      printf("done\n");

      exit(0);
}
```



```
$ ls –l tempfile
-rw-r--r--  stevens   9240990 Jul 31 13:42 tempfile
$ df /home
Filesystem   kbytes      used     avail        use       mounted on
/dev/sdoh    282908     181979   72638      71%       /home
$ a.out &
1354
$ file unlinked
$ ls –l tempfile
tempfile not found
$ df /home
Filesystem   kbytes      used     avail        use       mounted on
/dev/sdoh    282908     181979   72638      71%       /home
$ done
$ df /home
Filesystem   kbytes      used     avail        use       mounted on
/dev/sdoh    282908     172939   81678      68%       /home
```

remove

```
#include <stdio.h>
int remove (const char *pathname);
```

Descrizione: rimuove il file specificato da *pathname*

Restituisce: 0 se OK,

-1 in caso di errore

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 1° argomento

Restituisce: 0 se OK,

-1 in caso di errore

similitudini

- per un file remove è identico a unlink
- per una directory è identico a rmdir

symlink

```
#include <unistd.h>

int symlink (const char *path, const char *sympath);
```

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

Resatituisce: 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

.....link simbolici

- Il file puntato da un link simbolico può non esistere

```
$ ln -s /no/such/file myfile
$ ls myfile
myfile
$ cat myfile
cat: myfile : No such file or directory
$ ls -l myfile
lrwxrwxrwx 1 stevens 13 Dec 6 07:27 myfile -> /no/such/file
```

- loop creati con link simbolici possono essere facilmente rimossi con unlink (non segue il link simbolico)

readlink

#include <unistd.h>

int readlink (const char **pathname*, char **buf*, int *bufsize*);

Descrizione: legge dal link simbolico 1° 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

- 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 files

- 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

utime

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

int utime (const char *pathname, const struct utimbuf *times);
```

Descrizione: cambia i tempi di accesso e modifica di *pathname*

Restituisce: 0 se OK,

-1 in caso di errore

utime

```
struct utimbuf {
        time_t    actime;   /*access time*/
        time_t    modtime;  /*modification time*/
}
```

- Se *times* = NULL allora entrambi i tempi sono settati ai tempi correnti
  - L' operazione viene effettuata se EFF_ID=OWNER_ID o il processo ha permesso di scrittura sul file
- Se *times* ≠ NULL allora i tempi sono settati ai valori presenti in times
  - L'operazione viene effettuata se EFF_ID=OWNER_ID o il processo e' ha privilegi da superuser

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* e dalla mode creation mask del processo (vedi umask)

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

mkdir

- La directory creata avrà come
  - owner ID = l'effective ID del processo
  - group ID = 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

Funzioni chdir, fchdir

#include <unistd.h>

int chdir (const char **pathname*);

int fchdir (int *fd*);

Descrizione: cambiano la cwd del processo chiamante a quella specificata come argomento

Restituiscono: 0 se OK
-1 in caso di errore

- Si noti che:
  - cwd è un attributo del processo
  - home directory è un attributo di una login name

getcwd

#include <unistd.h>

char *getcwd (char **buf*, size_t *size*);

Descrizione: ottiene in *buf* il path assoluto della cwd

Restituisce: *buf* se OK
NULL in caso di errore

funzioni trunc e ftrunc

- Se si vuole eliminare la parte finale di un file
- Svuotare un file mediante il flag O_TRUNC e' un caso particolare di "troncamento"

```
#include <unistd.h>
void trunc(const char *pathname, off_t length);
int ftrunc(int filedes, off_t length);
```

Troncano il file (esistente) ad una lunghezza di *length* bytes. Se il file e' piu' lungo la parte oltre *length* viene eliminata.

Se la taglia precedente era meno di length, la lunghezza del file viene aumentata

Restituiscono: 0 se OK, -1 altrimenti

esercizio

Ricorda che da shell si puo' creare un link simbolico con il comando **ln -s**.

Scrivere un programma che prende su linea di comando un nome di file, e se tale file e' un link simbolico il programma esegue unlink sul file cui tale link punta, altrimenti cancella il file il cui nome gli e' stato passato.

esercizio

Scrivere un programma che con un loop, crea una directory, entra in essa e crea una nuova directory, entra in essa e crea una nuova directory… e cosi' via, per 10 volte.