Práctica 2.2. Sistema de Ficheros

Objetivos

En esta práctica se revisan las funciones del sistema básicas para manejar un sistema de ficheros, referentes a la creación de ficheros y directorios, duplicación de descriptores, obtención de información de ficheros o el uso de cerrojos.

Contenidos

Preparación del entorno para la práctica
Creación y atributos de ficheros
Redirecciones y duplicación de descriptores
Cerrojos de ficheros
Directorios

Preparación del entorno para la práctica

La realización de esta práctica únicamente requiere del entorno de desarrollo (compilador, editores y utilidades de depuración). Estas herramientas están disponibles en las máquinas virtuales de la asignatura y en la máquina física de los puestos del laboratorio.

Creación y atributos de ficheros

El inodo de un fichero guarda diferentes atributos de éste, como por ejemplo el propietario, permisos de acceso, tamaño o los tiempos de acceso, modificación y creación. En esta sección veremos las llamadas al sistema más importantes para consultar y fijar estos atributos así como las herramientas del sistema para su gestión.

Ejercicio 1. `ls(1)` muestra el contenido de directorios y los atributos básicos de los ficheros. Consultar la página de manual y estudiar el uso de las opciones `-a -l -d -h -i -R -1 -F` y `--color`. Estudiar el significado de la salida en cada caso.

-a, --all

do not ignore entries starting with .

-A, --almost-all

do not list implied . and ..

--author

with -l, print the author of each file

-b, --escape

print C-style escapes for nongraphic characters

--block-size=SIZE

scale sizes by SIZE before printing them; e.g.,

'--block-size=M' prints sizes in units of 1,048,576

bytes; see SIZE format below

Ejercicio 2. El *modo* de un fichero es `<tipo><rwx_propietario><rwx_grupo><rwx_resto>`:

- `tipo`: `-` fichero ordinario; `d` directorio; `l` enlace; `c` dispositivo carácter; `b` dispositivo bloque; `p` FIFO; `s` socket
- `rwx`: `r` lectura (4); `w` escritura (2); `x` ejecución (1)

Comprobar los permisos de algunos directorios (con `ls -ld`).

```
ls -ld <nombre_fichero>
```

Ejercicio 3. Los permisos se pueden otorgar de forma selectiva usando la notación octal o la simbólica. Ejemplo, probar las siguientes órdenes (equivalentes):

- `chmod 540 fichero`
- `chmod u=rx,g=r,o= fichero`

¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas? Consultar la página de manual `chmod(1)` para ver otras formas de fijar los permisos (p.ej. los operadores + y -).

chmod 645 <fichero>
chmod u+rw, g+r, o+r-w<fichero>

Ejercicio 4. Crear un directorio y quitar los permisos de ejecución para usuario, grupo y otros. Intentar cambiar al directorio.

mkdir directorio
chmod 666 midirectorio
Para poder cambiar de directorio es necesario tener permisos de ejecución

Ejercicio 5. Escribir un programa que, usando `open(2)`, cree un fichero con los permisos `rw-r--r-x`. Comprobar el resultado y las características del fichero con `ls(1)`.

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

int main() {
        int filedesc = open("file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0645);
        if(filedesc == -1) {
                printf("Error:%s\n", strerror(errno));
                return EXIT_FAILURE;
        } else {
                return EXIT_SUCCESS;
        }
}
```

gcc -g -Wall practica.c -o practica

Ejercicio 6. Cuando se crea un fichero, los permisos por defecto se derivan de la máscara de usuario (*umask*). El comando interno de la *shell* `umask` permite consultar y fijar esta máscara. Usando este comando, fijar la máscara de forma que los nuevos ficheros no tengan permiso de escritura para el grupo y no tengan ningún permiso para otros. Comprobar el funcionamiento con `touch(1)`, `mkdir(1)` y `ls(1)`.

umask 0727 → 111 - 010 - 111
touch archivo
ls -ld archivo

```
[cursoredes@localhost ~]$ umask 0727
[cursoredes@localhost ~]$ umask
0727
[cursoredes@localhost ~]$ touch archivo
[cursoredes@localhost ~]$ ls -ld archivo
----r----- 1 cursoredes cursoredes 0 Nov 21 10:00 archivo
[cursoredes@localhost ~]$
```

Ejercicio 7. Modificar el ejercicio 5 para que, antes de crear el fichero, se fije la máscara igual que en el ejercicio 6. Comprobar el resultado con `ls(1)`. Comprobar que la máscara del proceso padre (la *shell*) no cambia.

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

int main() {
        umask(0727);
        int filedesc = open("file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0645);
        if(filedesc == -1) {
                printf("Error:%s\n", strerror(errno));
                return EXIT_FAILURE;
        } else {
                return EXIT_SUCCESS;
        }
}
```

Ejercicio 8. `ls(1)` puede mostrar el inodo con la opción `-i`. El resto de información del inodo puede obtenerse usando `stat(1)`. Consultar las opciones del comando y comprobar su funcionamiento.

```
[cursoredes@localhost ~]$ ls -i
50520877 archivo    193044 Documents  17280582 Music     51384715 practica    51068183 Public     51068185 Videos
  193043 Desktop  17280581 Downloads  33729703 Pictures  53012963 practica.c  33729702 Templates
[cursoredes@localhost ~]$ stat
stat: missing operand
Try 'stat --help' for more information.
[cursoredes@localhost ~]$ stat archivo
  File: ‘archivo’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 50520877    Links: 1
Access: (0040/----r-----)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 10:00:10.277887318 +0100
Modify: 2022-11-21 10:00:10.277887318 +0100
Change: 2022-11-21 10:00:10.277887318 +0100
 Birth: -
[cursoredes@localhost ~]$
```

Ejercicio 9. Escribir un programa que emule el comportamiento de `stat(1)` y muestre:

- El número *major* y *minor* asociado al dispositivo.
- El número de inodo del fichero.
- El tipo de fichero (directorio, enlace simbólico o fichero ordinario).
- La hora en la que se accedió el fichero por última vez. ¿Qué diferencia hay entre `st_mtime` y `st_ctime`?

La diferencia es que st_ctime muestra la última vez que el inodo de un archivo fue cambiado, es decir, sus permisos, su nombre, etcétera… y st_mtime muestra la última vez que el contenido del archivo fue cambiado.

```
#include <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
int main(int argc, char *argv[]) {
        if(argc != 2)  exit(EXIT_FAILURE);
 const char* filename = argv[1];
 struct stat buf;
int res;
 char buffer [80];
 struct tm* timeinfo;
 char string [24] = "Fichero";
 res = lstat(filename, &buf);
        if(res == -1) {
                printf("Error:", strerror(errno));
        }
printf("Major number: %i Minor number: %i\n", major(buf.st_rdev), minor(buf.st_rdev));
printf("Numero de i-nodo: %i\n", buf.st_ino);
        if(S_ISDIR(buf.st_mode) != 0) strcpy(string, "Directorio");
        if(S_ISLNK(buf.st_mode) != 0) strcpy(string, "Enlace simbolico");
        if(S_ISREG(buf.st_mode) != 0) strcpy(string, "Fichero ordinario");
 printf("Tipo archivo: %s\n", string);
 timeinfo = localtime(&buf.st_atime);
 strftime(buffer, sizeof(buffer), "Ultimo acceso: %T", timeinfo);
 puts(buffer);
 return 0;
}
```

Ejercicio 10. Los enlaces se crean con `ln(1)`:

- Con la opción `-s`, se crea un enlace simbólico. Crear un enlace simbólico a un fichero ordinario y otro a un directorio. Comprobar el resultado con `ls -l` y `ls -i`. Determinar el inodo de cada fichero.

```
[cursoredes@localhost ~]$ ls
archivo  Documents  file.txt  Pictures  practica.c  stat    Templates
Desktop  Downloads  Music     practica  Public      stat.c  Videos
[cursoredes@localhost ~]$ ln -s file.txt enlacesimbolicoaarchivo
[cursoredes@localhost ~]$ ln -s Music/ enlacesimbolicoadirectorio
[cursoredes@localhost ~]$ ls -l
total 32
----r----- 1 cursoredes cursoredes    0 Nov 21 10:00 archivo
drwxr-xr-x 3 cursoredes cursoredes  140 Sep  7 12:45 Desktop
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Documents
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Downloads
lrwxrwxrwx 1 cursoredes cursoredes    8 Nov 21 11:10 enlacesimbolicoaarchivo -> file.txt
lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 11:12 enlacesimbolicoadirectorio -> Music/
----r----- 1 cursoredes cursoredes    0 Nov 21 10:22 file.txt
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Music
drwxr-xr-x 2 cursoredes cursoredes  147 Sep 22  2018 Pictures
-rwxrwxrwx 1 cursoredes cursoredes 8664 Nov 21 10:22 practica
-rwxrwxrwx 1 cursoredes cursoredes  345 Nov 21 10:07 practica.c
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Public
-rwxrwxrwx 1 cursoredes cursoredes 8992 Nov 21 10:43 stat
-rw-rw-r-- 1 cursoredes cursoredes 1001 Nov 21 10:43 stat.c
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Templates
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Videos
[cursoredes@localhost ~]$ ls -i
50520877 archivo    51385538 enlacesimbolicoaarchivo     33729703 Pictures    51385537 stat
  193043 Desktop    51385541 enlacesimbolicoadirectorio  51384715 practica    51385540 stat.c
  193044 Documents  53012964 file.txt                    53012963 practica.c  33729702 Templates
17280581 Downloads  17280582 Music                       51068183 Public      51068185 Videos
[cursoredes@localhost ~]$
```

- Repetir el apartado anterior con enlaces rígidos. Determinar los inodos de los ficheros y las propiedades con stat (observar el atributo número de enlaces).

No se puede crear un enlace rígido a un directorio ya que puede vincularse a un padre de sí mismo, lo que crea un bucle del sistema de archivos.

```
[cursoredes@localhost ~]$ ln  file.txt enlacerigidoaarchivo
[cursoredes@localhost ~]$ ln Music/ enlacerigidoadirectorio
ln: 'Music/': hard link not allowed for directory
[cursoredes@localhost ~]$ mkdir carpeta
[cursoredes@localhost ~]$ ls
archivo  Documents            enlacesimbolicoaarchivo     Music     practica.c  stat.c
carpeta  Downloads            enlacesimbolicoadirectorio  Pictures  Public      Templates
Desktop  enlacerigidoaarchivo  file.txt                    practica  stat        Videos
[cursoredes@localhost ~]$ ln carpeta/ enlacerigidoadirectorio
ln: 'carpeta/': hard link not allowed for directory
[cursoredes@localhost ~]$ ln carpeta enlacerigidoadirectorio
ln: 'carpeta': hard link not allowed for directory
[cursoredes@localhost ~]$ ls -l
total 32
----r----- 1 cursoredes cursoredes    0 Nov 21 10:00 archivo
drwxrwxrwx 2 cursoredes cursoredes    6 Nov 21 11:18 carpeta
drwxr-xr-x 3 cursoredes cursoredes  140 Sep  7 12:45 Desktop
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Documents
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Downloads
----r----- 2 cursoredes cursoredes    0 Nov 21 10:22 enlacerigidoaarchivo
lrwxrwxrwx 1 cursoredes cursoredes    8 Nov 21 11:10 enlacesimbolicoaarchivo -> file.txt
lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 11:12 enlacesimbolicoadirectorio -> Music/
----r----- 2 cursoredes cursoredes    0 Nov 21 10:22 file.txt
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Music
drwxr-xr-x 2 cursoredes cursoredes  147 Sep 22  2018 Pictures
-rwxrwxrwx 1 cursoredes cursoredes 8664 Nov 21 10:22 practica
-rwxrwxrwx 1 cursoredes cursoredes  345 Nov 21 10:07 practica.c
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Public
-rwxrwxrwx 1 cursoredes cursoredes 8992 Nov 21 10:43 stat
-rw-rw-r-- 1 cursoredes cursoredes 1001 Nov 21 10:43 stat.c
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Templates
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Videos
[cursoredes@localhost ~]$ ls -i
50520877 archivo    53012964 enlacerigidoaarchivo        33729703 Pictures    51385540 stat.c
51385542 carpeta    51385538 enlacesimbolicoaarchivo     51384715 practica    33729702 Templates
  193043 Desktop    51385541 enlacesimbolicoadirectorio  53012963 practica.c  51068185 Videos
  193044 Documents  53012964 file.txt                    51068183 Public
17280581 Downloads  17280582 Music                       51385537 stat
```

```
[cursoredes@localhost ~]$ stat enlacerigidoaarchivo
  File: ‘enlacerigidoaarchivo’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 53012964    Links: 2
Access: (0040/----r-----)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 10:22:47.389091570 +0100
Modify: 2022-11-21 10:22:47.389091570 +0100
Change: 2022-11-21 11:17:28.118727917 +0100
 Birth: -
[cursoredes@localhost ~]$ stat file.txt
  File: ‘file.txt’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 53012964    Links: 2
Access: (0040/----r-----)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 10:22:47.389091570 +0100
Modify: 2022-11-21 10:22:47.389091570 +0100
Change: 2022-11-21 11:17:28.118727917 +0100
 Birth: -
```

- ¿Qué sucede cuando se borra uno de los enlaces rígidos? ¿Qué sucede si se borra uno de los enlaces simbólicos? ¿Y si se borra el fichero original?

Cuando se borra un enlace rígido simplemente se le quita un enlace al archivo original. Si se borra uno de los enlaces simbólicos igual. Si se borra el archivo original se queda el enlace simbólico sin nada a lo que dirigirse, y el rígido se separa como una copia.

```
[cursoredes@localhost ~]$ rm enlacerigidoaarchivo
rm: remove write-protected regular empty file ‘enlacerigidoaarchivo’?
[cursoredes@localhost ~]$ rm enlacerigidoaarchivo
rm: remove write-protected regular empty file ‘enlacerigidoaarchivo’? yes
[cursoredes@localhost ~]$ ls -i
50520877 archivo    51385538 enlacesimbolicoaarchivo     51384715 practica    33729702 Templates
51385542 carpeta    51385541 enlacesimbolicoadirectorio  53012963 practica.c  51068185 Videos
  193043 Desktop    53012964 file.txt                    51068183 Public
  193044 Documents  17280582 Music                       51385537 stat
17280581 Downloads  33729703 Pictures                    51385540 stat.c
[cursoredes@localhost ~]$ stat file.txt
  File: ‘file.txt’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 53012964    Links: 1
Access: (0040/----r-----)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 10:22:47.389091570 +0100
Modify: 2022-11-21 10:22:47.389091570 +0100
Change: 2022-11-21 11:25:06.703816213 +0100
 Birth: -
[cursoredes@localhost ~]$ rm enlacesimbolicoadirectorio
[cursoredes@localhost ~]$ ls -i
50520877 archivo    17280581 Downloads               33729703 Pictures    51385537 stat
51385542 carpeta    51385538 enlacesimbolicoaarchivo 51384715 practica    51385540 stat.c
  193043 Desktop    53012964 file.txt                53012963 practica.c  33729702 Templates
  193044 Documents  17280582 Music                   51068183 Public      51068185 Videos
[cursoredes@localhost ~]$ rm file.txt
rm: remove write-protected regular empty file ‘file.txt’? yes
[cursoredes@localhost ~]$ ls -i
50520877 archivo    17280581 Downloads               51384715 practica    51385540 stat.c
51385542 carpeta    51385538 enlacesimbolicoaarchivo 53012963 practica.c  33729702 Templates
  193043 Desktop    17280582 Music                   51068183 Public      51068185 Videos
  193044 Documents  33729703 Pictures                51385537 stat
[cursoredes@localhost ~]$
```

```
[cursoredes@localhost ~]$ ln -s file.txt enlacesim
[cursoredes@localhost ~]$ ln file.txt enlacerig
[cursoredes@localhost ~]$ ls -i
50520877 archivo    51385541 enlacerig               33729703 Pictures    51385540 stat.c
51385542 carpeta    51385539 enlacesim               51384715 practica    33729702 Templates
  193043 Desktop    51385538 enlacesimbolicoaarchivo 53012963 practica.c  51068185 Videos
  193044 Documents  51385541 file.txt                51068183 Public
17280581 Downloads  17280582 Music                   51385537 stat
[cursoredes@localhost ~]$ rm file.txt
[cursoredes@localhost ~]$ ls -i
50520877 archivo    51385541 enlacerig               51384715 practica    33729702 Templates
51385542 carpeta    51385539 enlacesim               53012963 practica.c  51068185 Videos
  193043 Desktop    51385538 enlacesimbolicoaarchivo 51068183 Public
  193044 Documents  17280582 Music                   51385537 stat
17280581 Downloads  33729703 Pictures                51385540 stat.c
[cursoredes@localhost ~]$
```

Ejercicio 11. `link(2)` y `symlink(2)` crean enlaces rígidos y simbólicos, respectivamente. Escribir un programa que reciba una ruta a un fichero como argumento. Si la ruta es un fichero regular, creará un enlace simbólico y rígido con el mismo nombre terminado en `.sym` y `.hard`, respectivamente. Comprobar el resultado con `ls(1)`.

```
#include <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
#define MAX_SIZE 100
int main(int argc, char *argv[]) {
        if(argc != 2) {
        printf("Eespecifica una ruta\n");
        exit(EXIT_FAILURE);
        }
 char *path = argv[1];
 char *tmppath = strdup(path);
 char *tmppath2 = strdup(path);
 const char *newpath = strcat(tmppath, ".hard");
 const char *newpath2 = strcat(tmppath2, ".sym");
 struct stat buf;
 int res;
 res = lstat(path, &buf);
        if(res == -1) {
                printf("Error:\n", strerror(errno));
        } else {
                if(S_ISREG(buf.st_mode) != 0) { // Si es un fichero regular
                        int res2 = link(path, newpath);
                        if(res2 == -1) {
                                printf("error:%s\n", strerror(errno));
                        } else {
                                printf("Enlace rígido creado\n");
                        }
```

```
                        int res3 = symlink(path, newpath2);
                        if(res3 == -1) {
                                printf("error:%s\n", strerror(errno));
                        } else {
                                printf("Enlace simbólico creado\n");
                        }
                }
        }
 return 0;
}
```

```
[cursoredes@localhost ~]$ ./sym file.txt
Enlace rígido creado
Enlace simbólico creado
[cursoredes@localhost ~]$ ls -i
50520877 archivo     51385539 enlacesim                33729703 Pictures    51385544 sym
51385542 carpeta     51385538 enlacesimbolicoaarchivo  51384715 practica    51385543 sym.c
  193043 Desktop     51385545 file.txt                 53012963 practica.c  33729702 Templates
  193044 Documents   51385545 file.txt.hard            51068183 Public      51068185 Videos
17280581 Downloads   51385546 file.txt.sym             51385537 stat
51385541 enlacerig   17280582 Music                    51385540 stat.c
[cursoredes@localhost ~]$
```

Redirecciones y duplicación de descriptores

La *shell* proporciona operadores (>, >&, >>) que permiten redirigir un fichero a otro, ver los ejercicios propuestos en la práctica opcional. Esta funcionalidad se implementa mediante `dup(2)` y `dup2(2)`.

Ejercicio 12. Escribir un programa que redirija la salida estándar a un fichero cuya ruta se pasa como primer argumento. Probar haciendo que el programa escriba varias cadenas en la salida estándar.

```
#include <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
 if(argc != 2) {
   printf("Vuelve a ejecutar\n");
   exit(EXIT_FAILURE);
 }

 int filed = open(argv[1], O_RDWR | O_CREAT | O_TRUNC, 0645);
 dup2(filed, 1);
 dup2(filed, 2);
 printf("Mensajes salida estandar: prueba1 prueba2\n");
```

```
 close(filed);
 return 0;
}
```

Ejercicio 13. Modificar el programa anterior para que también redirija la salida estándar de error al fichero. Comprobar el funcionamiento incluyendo varias sentencias que impriman en ambos flujos. ¿Hay diferencia si las redirecciones se hacen en diferente orden? ¿Por qué `ls > dirlist 2>&1` es diferente a `ls 2>&1 > dirlist`?

```
#include <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
 if(argc != 2) {
   printf("Vuelve a ejecutar\n");
   exit(EXIT_FAILURE);
 }

 int filed = open(argv[1], O_RDWR | O_CREAT | O_TRUNC, 0645);
 dup2(filed, 1);
 dup2(filed, 2);
 printf("Mensajes salida estandar: prueba1 prueba2\n");
 printf("otralinea\n");
 perror("Mensajes salida error: perror");
```

```
 close(filed);

 return 0;

}
```

Cerrojos de ficheros

El sistema de ficheros ofrece cerrojos de ficheros consultivos.

Ejercicio 14. El estado y cerrojos de fichero en uso en el sistema se pueden consultar en el fichero `/proc/locks`. Estudiar el contenido de este fichero.

Ejercicio 15. Escribir un programa que intente bloquear un fichero usando `lockf(3)`:

- Si lo consigue, mostrará la hora actual y suspenderá su ejecución durante 10 segundos con `sleep(3)`. A continuación, desbloqueará el fichero, suspenderá su ejecución durante otros 10 segundos y terminará.
- Si no lo consigue, el programa mostrará el error con `perror(3)` y terminará.

```
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
int main(int argc, char *argv[]) {
   if(argc != 2) {
      printf("Especifica el fichero\n");
      exit(EXIT_FAILURE);
   }
   struct flock lock;
   time_t rawtime;
   struct tm* hora;
   char buffer [9];
   int fd;
   printf("Comprueba el estado del cerrojo...\n");
   fd = open(argv[1], O_WRONLY);
   memset(&lock, 0, sizeof(lock));
   fcntl(fd, F_GETLK, &lock);
   if(lock.l_type != F_UNLCK) {
      printf("El cerrojo está bloqueado\n");
      exit(EXIT_SUCCESS);
   } else {
      printf("Cerrojo desbloqueado\n");
      lock.l_type = F_WRLCK;
      memset(&lock, 0, sizeof(lock));
      fcntl(fd, F_SETLKW, &lock);
      time(&rawtime);
      hora = localtime(&rawtime);
      strftime(buffer, sizeof(buffer), "%T", hora);
```

```
        write(fd, buffer, 8);
        printf("Cerrojo adquirido\n");
        sleep(5);
        printf("Liberando cerrojo\n");
        lock.l_type = F_UNLCK;
        fcntl(fd, F_SETLKW, &lock);
    }
  close(fd);
  return 0;
}
```

***Ejercicio 16* (Opcional).** `flock(1)` proporciona funcionalidad de cerrojos antiguos BSD en guiones *shell*. Consultar la página de manual y el funcionamiento del comando.
man flock()

Directorios

Ejercicio 17. Escribir un programa que muestre el contenido de un directorio:

- El programa tiene un único argumento que es la ruta a un directorio. El programa debe comprobar la corrección del argumento.
- El programa recorrerá las entradas del directorio y escribirá su nombre de fichero. Además:
    - Si es un fichero regular y tiene permiso de ejecución para usuario, grupo u otros, escribirá el carácter `'*'` después del nombre.
    - Si es un directorio, escribirá el carácter `'/'` después del nombre
    - Si es un enlace simbólico, escribirá "`->`" y el nombre del fichero enlazado después del nombre. Usar `readlink(2)`.
- Al final de la lista, el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.

```
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <dirent.h>

int main(int argc, char *argv[]) {
    if(argc != 2) {
        printf("Especifica el directorio\n");
        exit(EXIT_FAILURE);
    }
    DIR *dir;
    struct dirent *dirp;
    char * dirname = argv[1];
    char * path;
    struct stat buf;
    char buffer[1024];
```

```
    int res;
    int size = 0;

     res = lstat(dirname, &buf);
    if(res == -1) {
        perror("Error");
        exit(EXIT_FAILURE);
    } else if(!S_ISDIR(buf.st_mode)) {
        printf("Debe apuntar a un directorio\n");
        exit(EXIT_FAILURE);
    }
     dir = opendir(argv[1]);

    while((dirp = readdir(dir)) != NULL) {
        if(dirp->d_type == DT_REG) {
            printf("%s\n", dirp->d_name);

                        = size + buf.st_size;
        }
         if(dirp->d_type == DT_DIR) {
            printf("%s/\n", dirp->d_name);
            size = size + buf.st_size;
        }
        if(dirp->d_type == DT_LNK) {
            path = strcat(dirname, "/");
            path = strcat(path, dirp->d_name);
            ssize_t tambuf = readlink(path, buffer, sizeof(buffer) - 1);
            if(tambuf != -1) {
                buffer[tambuf] = '\0';
                printf("%s -> %s\n", dirp->d_name, buffer);
                size = size + buf.st_size;
            } else perror("Fallo");
        }
        if(dirp->d_type == DT_BLK) {
            printf("%s*\n", dirp->d_name);
            size = size + buf.st_size;
        }
        }
    closedir(dir);

    size = size / 1024;
    printf("Tamaño total (KB): %i\n", size);

    return 0;
}
```