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: No ignorar entradas que empiezan por .
-l: Muestra la lista con toda la información.
-d: Muestra la lista de directorios
-h: Si se usa junto con -l, muestra los tamaños en formato K,M,G.
-i: Muestra el inodo de cada fichero.
-R: Muestra subdirectorios de forma recursiva.
-1: Muestra un fichero por línea
-F: clasifica la salida (con */=>@|).
--color: Aplica color a la salida de ls en función del tipo de fichero.

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`).

```
[cursoredes@localhost Pr2]$ ls -ld
drwxrwxr-x 2 cursoredes cursoredes 20 Nov 21 11:05 .
```

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 u=rw, g=r, o=rx fichero
chmod 645 fichero
```

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

```
[cursoredes@localhost Pr2]$ mkdir PruebaDir
[cursoredes@localhost Pr2]$ chmod u-x,g-x,o-x PruebaDir
[cursoredes@localhost Pr2]$ ls -l
total 4
-rw-r--r-- 1 root      root      8 Nov 21 11:05 prueba
drw-rw-r-- 2 cursoredes cursoredes 6 Nov 21 11:11 PruebaDir
[cursoredes@localhost Pr2]$ cd PruebaDir/
bash: cd: PruebaDir/: Permission denied
```

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 <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv){
   if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }
   int fd = open(argv[1], O_CREAT, 0645);
   if(fd==-1){
      printf("Error al crear el archivo.\n");
      exit(1);
   }
   return 0;
}
```

```
[cursoredes@localhost Pr2]$ ./ej5 fichprueba

[cursoredes@localhost Pr2]$ ls -l
total 20
-rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
-rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
-rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:31 fichprueba
-rw-r--r-- 1 root      root         8 Nov 21 11:05 prueba
drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
```

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)`.

```
[cursoredes@localhost Pr2]$ umask 027
[cursoredes@localhost Pr2]$ umask
0027

[cursoredes@localhost Pr2]$ ./ej5 fichprueba2
[cursoredes@localhost Pr2]$ ls -l
total 20
-rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
-rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
-rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
-rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba2
-rw-r--r-- 1 root      root         8 Nov 21 11:05 prueba
drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir

Fichprueba2 no tiene ningún permiso en otros.
```

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 <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv){
   if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }
```

```
   mode_t mask = umask(0027);
   int fd = open(argv[1], O_CREAT, 0645);
   if(fd==-1){
      printf("Error al crear el archivo.\n");
      exit(1);
   }

  return 0;
}
```

Haciendo umask en la shell:
[cursoredes@localhost Pr2]$ umask
0002

No ha cambiado

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 Pr2]$ stat prueba
  File: ‘prueba’
  Size: 8               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 3577006     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-11-21 11:05:50.769633114 +0100
Modify: 2022-11-21 11:05:50.770632671 +0100
Change: 2022-11-21 11:05:50.770632671 +0100
 Birth: -
```

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`?

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

int main(int argc, char **argv){
   if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }
   struct stat buffstat;
```

```
   int st = stat(argv[1], &buffstat);
   if(st==-1){
      printf("Error en stat.\n");
      exit(1);
   }
   printf("Major: %i\n", major(buffstat.st_dev));
   printf("Minor: %i\n", minor(buffstat.st_dev));

   printf("I-node number: %li\n", buffstat.st_ino);

   if(S_ISREG(buffstat.st_mode)){
      printf("Es un fichero regular.\n");
   }
   if(S_ISDIR(buffstat.st_mode)){
      printf("Es un directorio.\n");
   }
   if(S_ISLNK(buffstat.st_mode)){
      printf("Es un enlace simbolico.\n");
   }

   struct tm *time = localtime(&buffstat.st_atim);
   printf("Ultimo acceso a fichero: %i:%li\n", time->tm_hour,time->tm_min);


   return 0;
}

time_t st_mtime;   Última modificación
time_t st_ctime;   Último cambio de estado (afecta a la información del i-nodo).
```

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 Pr2]$ ln -s prueba pruebasim
[cursoredes@localhost Pr2]$ ls -li
total 52
  309700 -rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
  309699 -rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
  161297 -rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 21 11:50 ej7
  309703 -rw-r--r-- 1 root      root       404 Nov 21 11:50 ej7.c
  309705 -rwxrwxr-x 1 cursoredes cursoredes 8824 Nov 21 12:24 ej9
  309704 -rw-r--r-- 1 root      root       940 Nov 21 12:24 ej9.c
  309701 -rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
  309702 -rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:51 fichprueba2
 3577006 -rw-r--r-- 1 root      root        11 Nov 21 12:22 prueba
17302769 drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
  309706 lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 12:27 pruebasim -> prueba


Coinciden los i-nodos
```

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

```
[cursoredes@localhost Pr2]$ touch prueba2
[cursoredes@localhost Pr2]$ ln prueba2 prueba2rig
[cursoredes@localhost Pr2]$ ls -li
total 52
  309700 -rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
  309699 -rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
  161297 -rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 21 11:50 ej7
  309703 -rw-r--r-- 1 root      root       404 Nov 21 11:50 ej7.c
  309705 -rwxrwxr-x 1 cursoredes cursoredes 8824 Nov 21 12:24 ej9
  309704 -rw-r--r-- 1 root      root       940 Nov 21 12:24 ej9.c
  309701 -rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
  309702 -rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:51 fichprueba2
 3577006 -rw-r--r-- 1 root      root        11 Nov 21 12:22 prueba
  309707 -rw-rw-r-- 2 cursoredes cursoredes    0 Nov 21 12:29 prueba2
  309707 -rw-rw-r-- 2 cursoredes cursoredes    0 Nov 21 12:29 prueba2rig
17302769 drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
  309706 lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 12:27 pruebasim -> prueba

[cursoredes@localhost Pr2]$ stat prueba2
  File: ‘prueba2’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 309707      Links: 2
Access: (0664/-rw-rw-r--)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 12:29:28.354360037 +0100
Modify: 2022-11-21 12:29:28.354360037 +0100
Change: 2022-11-21 12:30:33.876085576 +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?

```
[cursoredes@localhost Pr2]$ rm prueba2rig
[cursoredes@localhost Pr2]$ ls -l
total 52
-rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
-rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 21 11:50 ej7
-rw-r--r-- 1 root      root       404 Nov 21 11:50 ej7.c
-rwxrwxr-x 1 cursoredes cursoredes 8824 Nov 21 12:24 ej9
-rw-r--r-- 1 root      root       940 Nov 21 12:24 ej9.c
-rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
-rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:51 fichprueba2
-rw-r--r-- 1 root      root        11 Nov 21 12:22 prueba
-rw-rw-r-- 1 cursoredes cursoredes    0 Nov 21 12:29 prueba2
drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 12:27 pruebasim -> prueba
[cursoredes@localhost Pr2]$ rm pruebasim
```

```
[cursoredes@localhost Pr2]$ ls -l
total 52
-rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
-rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 21 11:50 ej7
-rw-r--r-- 1 root      root       404 Nov 21 11:50 ej7.c
-rwxrwxr-x 1 cursoredes cursoredes 8824 Nov 21 12:24 ej9
-rw-r--r-- 1 root      root       940 Nov 21 12:24 ej9.c
-rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
-rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:51 fichprueba2
-rw-r--r-- 1 root      root        11 Nov 21 12:22 prueba
-rw-rw-r-- 1 cursoredes cursoredes    0 Nov 21 12:29 prueba2
drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
[cursoredes@localhost Pr2]$ stat prueba2
  File: 'prueba2'
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 309707      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2022-11-21 12:29:28.354360037 +0100
Modify: 2022-11-21 12:29:28.354360037 +0100
Change: 2022-11-21 12:32:00.746025027 +0100
 Birth: -
```

Al borrar el rígido ha decrecido el número de enlaces del fichero original. Esto no ocurre al borrar el enlace simbólico. Si se borra el original al rígido no le pasa nada, y el simbólico pierde el i-nodo.

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/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc==1){
       printf("Error, faltan argumentos.\n");
       exit(1);
   }
   struct stat buffstat;
   int st = stat(argv[1], &buffstat);
   if(st==-1){
       printf("Error en stat.\n");
       exit(1);
```

```
    }
    if(!S_ISREG(buffstat.st_mode)){
        printf("No es fichero regular.\n");
        exit(1);
    }
    char *pathrig = malloc(sizeof(argv[1])+5);
    char *pathsim = malloc(sizeof(argv[1])+4);

    strcpy(pathrig, argv[1]);
    strcpy(pathsim, argv[1]);

    if(link(argv[1], strcat(pathrig, ".hard"))==-1){
        printf("Error al crear enlace rigido.\n");
        exit(1);
    }

    if(symlink(argv[1], strcat(pathsim, ".sym"))==-1){
        printf("Error al crear enlace simbolico.\n");
    }

    return 0;
}

[cursoredes@localhost Pr2]$ touch prueba
[cursoredes@localhost Pr2]$ ./ej11 prueba
[cursoredes@localhost Pr2]$ ls -l
total 64
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 21 16:58 ej11
-rw-r--r-- 1 root      root       955 Nov 21 16:58 ej11.c
-rwxrwxr-x 1 cursoredes cursoredes 8536 Nov 21 11:31 ej5
-rw-r--r-- 1 root      root       370 Nov 21 11:31 ej5.c
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 21 11:50 ej7
-rw-r--r-- 1 root      root       404 Nov 21 11:50 ej7.c
-rwxrwxr-x 1 cursoredes cursoredes 8824 Nov 21 12:24 ej9
-rw-r--r-- 1 root      root       940 Nov 21 12:24 ej9.c
-rw-r--r-x 1 cursoredes cursoredes    0 Nov 21 11:35 fichprueba
-rw-r----- 1 cursoredes cursoredes    0 Nov 21 11:51 fichprueba2
-rw-rw-r-- 2 cursoredes cursoredes    0 Nov 21 16:58 prueba
-rw-rw-r-- 1 cursoredes cursoredes    0 Nov 21 12:29 prueba2rig
drw-rw-r-- 2 cursoredes cursoredes    6 Nov 21 11:11 PruebaDir
-rw-rw-r-- 2 cursoredes cursoredes    0 Nov 21 16:58 prueba.hard
lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 12:36 pruebasim -> prueba
lrwxrwxrwx 1 cursoredes cursoredes    6 Nov 21 16:58 prueba.sym -> prueba
[cursoredes@localhost Pr2]$ ls -i
  309709 ej11      161297 ej7      309701 fichprueba   17302769 PruebaDir
  309708 ej11.c    309703 ej7.c    309702 fichprueba2    309710 prueba.hard
  309700 ej5       309705 ej9      309710 prueba         309706 pruebasim
  309699 ej5.c     309704 ej9.c    309707 prueba2rig     309711 prueba.sym
```

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/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }

   int fd = open(argv[1], O_CREAT|O_RDWR, 0666);
   if (fd == -1) {
       printf("Error al abrir fichero.\n");
       exit(1);
   }

   int fd2 = dup2(fd, 1);

   printf("Prueba de redireccion.\n");
   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/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }

   int fd = open(argv[1], O_CREAT|O_RDWR, 0666);
```

```
    if (fd == -1) {
        printf("Error al abrir fichero.\n");
        exit(1);
    }

    int fd2 = dup2(fd, 1);
    int fd3 = dup2(fd, 2);

    printf("Prueba de redireccion.\n");
    fprintf(stderr, "Prueba de redireccion de error\n");
    return 0;
}
```

`ls > dirlist 2>&1`

Redirecciona la salida estándar a dirlist. Luego la salida de errores la redirecciona a la 1, la cual corresponde a dirlist. Así ambas van a dirlist.

`ls 2>&1 > dirlist`

Redirecciona la salida de error a la salida estándar y la salida estándar a dirlist. Así los errores se van a ver por la salida estándar por pantalla, y la salida normal en el fichero dirlist.

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.

```
1: POSIX  ADVISORY  WRITE 1584 fd:00:53013121 0 EOF
2: POSIX  ADVISORY  WRITE 1575 fd:00:53013120 0 EOF
3: POSIX  ADVISORY  WRITE 1568 fd:00:53012991 0 EOF
4: POSIX  ADVISORY  WRITE 1558 fd:00:53012987 0 EOF
5: POSIX  ADVISORY  WRITE 1351 00:13:21251 0 EOF
6: FLOCK  ADVISORY  WRITE 1304 fd:00:17458434 0 EOF
7: FLOCK  ADVISORY  WRITE 1304 fd:00:30831 0 EOF
8: FLOCK  ADVISORY  WRITE 1069 00:13:19853 0 EOF
9: POSIX  ADVISORY  WRITE 1067 00:13:19836 0 EOF
10: POSIX  ADVISORY  WRITE 809 00:13:17305 0 EOF
12: POSIX  ADVISORY  WRITE 476 00:13:12580 0 EOF
```

Primero aparece un número único para cada cerrojo. Luego el tipo de cerrojo (POSIX si se hizo con lockf y FLOCK si se hizo con flock). Luego aparece ADVISORY si solo bloquea otros intentos de bloqueo y permite acceso a datos. Luego WRITE porque bloquea lectura y escritura puesto que se está escribiendo. Luego el id del proceso que está bloqueando, seguido del id del fichero bloqueado, así como el comienzo y final de la región bloqueada del 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 <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }

   int fd = open(argv[1], O_CREAT|O_RDWR, 0666);
   if (fd == -1) {
       printf("Error al abrir fichero.\n");
       exit(1);
   }

   off_t f_ini = lseek(fd, 0, SEEK_SET);
   off_t f_end = lseek(fd, 0, SEEK_END);

   if(lockf(fd,F_TEST, f_end)==-1){
      perror("El fichero ya esta bloqueado.");
   }
   else{
      printf("El fichero no está bloqueado.\nLo bloqueamos...\n");
      int lck = lockf(fd, F_LOCK, f_end);
      time_t tim = time(NULL);
      struct tm *hora = localtime(&tim);
      printf ("Son las %i:%i\n", hora->tm_hour, hora->tm_min);
      sleep(10);
      printf("Desbloqueando...\n");
      lockf(fd, F_ULOCK, f_end);
      sleep(10);
   }

   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.

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

int main(int argc, char **argv){
    if(argc==1){
      printf("Error, faltan argumentos.\n");
      exit(1);
   }

   DIR *dr = opendir(argv[1]);
   if (dr == NULL) {
       printf("Error al abrir directorio.\n");
       exit(1);
   }
   struct dirent *rddir = readdir(dr);
   long int tam = 0;
   while(rddir != NULL){
      char *copiaDir = malloc(strlen(argv[1])+strlen(rddir->d_name)+3);
      strcpy(copiaDir, argv[1]);
      char *path = strcat(copiaDir, "/");
      path = strcat(copiaDir,rddir->d_name);

      if(rddir->d_type == DT_REG){
         struct stat sb;
         if(lstat(path,&sb)==-1){
            perror("Error en lstat.");
            exit(1);
         }
         if(sb.st_mode & (S_IXUSR | S_IXGRP | S_IXOTH)){
            printf("%s*\n",rddir->d_name);
         }
         else{
```

```
            printf("%s\n",rddir->d_name);
          }
          tam += sb.st_size;
       }
       else if(rddir->d_type == DT_DIR){
          printf("%s/\n", rddir->d_name);
       }
      else if(rddir->d_type == DT_LNK){
          struct stat sb;
          char *buf;
          ssize_t nbts, sizebuf;
          if(lstat(path,&sb)==-1){
             perror("Error en lstat.");
             exit(1);
          }
          tam += sb.st_size;
          sizebuf = sb.st_size + 1;
          if(sb.st_size == 0){
             sizebuf = PATH_MAX;
          }
          buf = malloc(sizebuf);
          if(buf == NULL){
             perror("Error en malloc.");
             exit(1);
          }
          nbts = readlink(path, buf, sizebuf);
          if(nbts == -1){
             perror("Error en readlink.");
             exit(1);
          }
          printf("%s->%s\n", rddir->d_name, buf);
          free(buf);
       }
            rddir = readdir(dr);
    }
    closedir(dr);
    printf("\nTamano total que ocupan los ficheros: %li\n", tam);

    return 0;
}
```

.