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.

```
usuario_vms@pto0820:~$ ls
 Descargas    Música       thinclient_drives      workspace_ibmrsa9p6
 Documentos   Plantillas   Vídeos
 Escritorio   Público      'VirtualBox VMs'
 Imágenes     snap          workspace-4.14-x86_64
```

-a : muestra todos los archivos
-l: usa formato de lista larga
-d: muestra los directorios
-h: imprime tamaños
-i: imprime número de inodo del fichero
-R: lista recursiva de subdirectorios
-1: lista un archivo por línea
*-F: clasifica la salida(con */=>@|)*
-- color: imprime según color.

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`).

```
usuario_vms@pto0820:~/Plantillas$ ls -ld
drwxr-xr-x 2 usuario_vms users 4096 jun 27  2018 .
usuario_vms@pto0820:~/Plantillas$
```

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-wx,o-wxr fichero`

¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas?

chmod 645 fichero
chmod u+rw-x, g+r-wx, o +rx-w fichero

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

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)`.

Fichero:

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


int main() {
        int f = open("prueba.txt", O_CREAT, 0645);
        if(f == -1){
                printf(strerror(errno));
        }
        return 1;
}
```

Terminal:
-rw-r--r-x 1 usuario_vms users 0 nov 8 12:20 prueba.txt

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)`.

```
usuario_vms@pto0820:~/prueba$ umask 027
usuario_vms@pto0820:~/prueba$ mkdir d6
usuario_vms@pto0820:~/prueba$ ls -l
total 12
drwxr-x--- 2 usuario_vms users 4096 nov  8 12:46 d6
drwxr-x--- 2 usuario_vms users 4096 nov  8 12:25 p2
-rw-r--r-- 1 usuario_vms users  243 nov  8 12:36 practica2.cpp
usuario_vms@pto0820:~/prueba$ rmdir p2
usuario_vms@pto0820:~/prueba$ ls -l
total 8
drwxr-x--- 2 usuario_vms users 4096 nov  8 12:46 d6
-rw-r--r-- 1 usuario_vms users  243 nov  8 12:36 practica2.cpp
usuario_vms@pto0820:~/prueba$ touch fich
usuario_vms@pto0820:~/prueba$ ls -l
total 8
drwxr-x--- 2 usuario_vms users 4096 nov  8 12:46 d6
-rw-r----- 1 usuario_vms users    0 nov  8 12:47 fich
-rw-r--r-- 1 usuario_vms users  243 nov  8 12:36 practica2.cpp
```

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.

```
usuario_vms@pto0820:~/prueba$ g++ ./practica2.cpp -o ej7
usuario_vms@pto0820:~/prueba$ ./ej7
usuario_vms@pto0820:~/prueba$ umask
0022
usuario_vms@pto0820:~/prueba$ ls -l
total 48
-rwxr-xr-x 1 usuario_vms users 16848 nov  8 12:20 ej5
-rwxr-xr-x 1 usuario_vms users 16744 nov  8 12:36 ej7
-rw-r----- 1 usuario_vms users     0 nov  8 12:24 fich2
drwxr-x--- 2 usuario_vms users  4096 nov  8 12:25 p2
-rw-r--r-- 1 usuario_vms users   243 nov  8 12:36 practica2.cpp
-rwxr-x--- 1 usuario_vms users     0 nov  8 12:33 prueba2.txt
-rw-r--r-x 1 usuario_vms users     0 nov  8 12:20 prueba.txt
```

Fichero:

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

int main() {
        mode_t oldMask = umask(027);
        int f = open("prueba2.txt", O_CREAT, 0777);
        mode_t last = umask(oldMask);
        return 0;
}
```

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
  193043 Desktop    17280582 Music        51068183 Public
  193044 Documents  33729703 Pictures     33729702 Templates
17280581 Downloads  53120806 practica2.cpp  51068185 Videos

[cursoredes@localhost ~]$ stat practica2.cpp
  File: 'practica2.cpp'
  Size: 103         Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 53120806    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-10 13:35:34.170191713 +0100
Modify: 2021-11-10 13:36:41.659031063 +0100
Change: 2021-11-10 13:36:41.659031063 +0100
```

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`?

Fichero:

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

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del fichero");
      return -1;
   }
   struct stat buf;
   int s = stat(argv[1],&buf);
   if(s == -1){
      printf("Error. Chequea el nombre introducido\n");
      return -1;
   }
   printf("Los numeros mayor y minor son: %i, %i \n", major(buf.st_dev),minor(buf.st_dev));
   printf("El i-nodo es: %i\n", buf.st_ino);
   mode_t m = buf.st_mode;
   if(S_ISLNK(m))
      printf("Enlace simbolico\n");
   else{
      if(S_ISREG(m))
         printf("Fichero normal\n");
      if(S_ISDIR(m))
         printf("Directorio\n");
   }

   time_t t = buf.st_atime;
   struct tm * tOk = localtime(&t);
   printf("Ultimo acceso; %d-%d-%d
%d:%d:%d\n",tOk->tm_year,tOk->tm_yday,tOk->tm_mday,tOk->tm_hour,tOk->tm_min,tOk->tm_
sec);

   return 1;
}
```

Terminal:

```
[cursoredes@localhost ~]$ ./ej9 pratica2.cpp
Error. Chequea el nombre introducido
[cursoredes@localhost ~]$ ./ej9 practica2.cpp
Los numeros major y minor son: 253, 0
El i-nodo es: 53120806
Fichero normal
Ultimo acceso; 121-313-10 13:35:34
```

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.

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

Enlace Simbólico:

[cursoredes@localhost ~]$ ln -s pratica2.cpp fichSimbolico.txt
[cursoredes@localhost ~]$ ls -l
total 16
drwxr-xr-x 2 cursoredes cursoredes 116 Sep 9 2018 Desktop
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Documents
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 10 19:25 ej9
lrwxrwxrwx 1 cursoredes cursoredes 12 Nov 10 19:30 fichSimbolico.txt -> pratica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Music
drwxr-xr-x 2 cursoredes cursoredes 147 Sep 22 2018 Pictures
-rw-rw-r-- 1 cursoredes cursoredes 1030 Nov 10 19:26 practica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Public
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
193043 Desktop **53120809 fichSimbolico.txt** 51068183 Public
193044 Documents 17280582 Music 33729702 Templates
17280581 Downloads 33729703 Pictures 51068185 Videos
53013170 ej9 **53120806 practica2.cpp**

[cursoredes@localhost ~]$ ln -s directorioP aaaa
[cursoredes@localhost ~]$ ls -l
total 16
lrwxrwxrwx 1 cursoredes cursoredes 11 Nov 10 19:32 aaaa -> directorioP
drwxr-xr-x 2 cursoredes cursoredes 116 Sep 9 2018 Desktop
drwxrwxr-x 2 cursoredes cursoredes 6 Nov 10 19:32 directorioP
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Documents
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 10 19:25 ej9
lrwxrwxrwx 1 cursoredes cursoredes 12 Nov 10 19:30 fichSimbolico.txt -> pratica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Music
drwxr-xr-x 2 cursoredes cursoredes 147 Sep 22 2018 Pictures
-rw-rw-r-- 1 cursoredes cursoredes 1030 Nov 10 19:26 practica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Public
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
53126726 aaaa 53013170 ej9 51068183 Public
193043 Desktop 53120809 fichSimbolico.txt 33729702 Templates
30901 directorioP 17280582 Music 51068185 Videos
193044 Documents 33729703 Pictures
17280581 Downloads 53120806 practica2.cpp

Enlace Rígido:

[cursoredes@localhost ~]$ ln practica2.cpp fichRigido
[cursoredes@localhost ~]$ ls -l
total 20
lrwxrwxrwx 1 cursoredes cursoredes 11 Nov 10 19:32 aaaa -> directorioP
drwxr-xr-x 2 cursoredes cursoredes 116 Sep 9 2018 Desktop
drwxrwxr-x 2 cursoredes cursoredes 6 Nov 10 19:32 directorioP
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Documents
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 10 19:25 ej9
-rw-rw-r-- 2 cursoredes cursoredes 1030 Nov 10 19:26 fichRigido
lrwxrwxrwx 1 cursoredes cursoredes 12 Nov 10 19:30 fichSimbolico.txt -> pratica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Music
drwxr-xr-x 2 cursoredes cursoredes 147 Sep 22 2018 Pictures
-rw-rw-r-- 2 cursoredes cursoredes 1030 Nov 10 19:26 practica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Public
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
53126726 aaaa 53013170 ej9 **53120806 practica2.cpp**
193043 Desktop **53120806 fichRigido** 51068183 Public
30901 directorioP 53120809 fichSimbolico.txt 33729702 Templates
193044 Documents 17280582 Music 51068185 Videos
17280581 Downloads 33729703 Pictures

[cursoredes@localhost ~]$ stat ./practica2.cpp
File: ‘./practica2.cpp’
Size: 1030 Blocks: 8 IO Block: 4096 regular file
Device: fd00h/64768d Inode: 53120806 **Links: 2**
Access: (0664/-rw-rw-r--) Uid: (1000/cursoredes) Gid: (1000/cursoredes)
Access: 2021-11-10 13:35:34.170191713 +0100
Modify: 2021-11-10 19:26:00.075610794 +0100
Change: 2021-11-10 19:35:03.577042732 +0100
Birth: -
[cursoredes@localhost ~]$ stat ./fichRigido
File: ‘./fichRigido’
Size: 1030 Blocks: 8 IO Block: 4096 regular file
Device: fd00h/64768d Inode: 53120806 **Links: 2**
Access: (0664/-rw-rw-r--) Uid: (1000/cursoredes) Gid: (1000/cursoredes)
Access: 2021-11-10 13:35:34.170191713 +0100
Modify: 2021-11-10 19:26:00.075610794 +0100
Change: 2021-11-10 19:35:03.577042732 +0100
Birth: -

Borrar enlace rígido: Tan solo decrece el nº de links asociados al fichero original.
[cursoredes@localhost ~]$ rm fichRigido
[cursoredes@localhost ~]$ stat ./practica2.cpp
File: ‘./practica2.cpp’
Size: 1030 Blocks: 8 IO Block: 4096 regular file
Device: fd00h/64768d Inode: 53120806 **Links: 1**
Access: (0664/-rw-rw-r--) Uid: (1000/cursoredes) Gid: (1000/cursoredes)
Access: 2021-11-10 13:35:34.170191713 +0100
Modify: 2021-11-10 19:26:00.075610794 +0100
Change: 2021-11-10 19:39:34.341448658 +0100
Birth: -

Borrar enlace simbólico: No cambia nada del original
[cursoredes@localhost ~]$ rm fichSimbolico.txt
[cursoredes@localhost ~]$ ls -l
total 16
lrwxrwxrwx 1 cursoredes cursoredes 11 Nov 10 19:32 aaaa -> directorioP
drwxr-xr-x 2 cursoredes cursoredes 116 Sep 9 2018 Desktop
drwxrwxr-x 2 cursoredes cursoredes 6 Nov 10 19:32 directorioP
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Documents
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 10 19:25 ej9
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Music
drwxr-xr-x 2 cursoredes cursoredes 147 Sep 22 2018 Pictures
-rw-rw-r-- 1 cursoredes cursoredes 1030 Nov 10 19:26 practica2.cpp
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Public
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Templates
drwxr-xr-x 2 cursoredes cursoredes 6 Sep 9 2018 Videos
[cursoredes@localhost ~]$ stat ./practica2.cpp
File: ‘./practica2.cpp’
Size: 1030 Blocks: 8 IO Block: 4096 regular file
Device: fd00h/64768d Inode: 53120806 Links: 1
Access: (0664/-rw-rw-r--) Uid: (1000/cursoredes) Gid: (1000/cursoredes)
Access: 2021-11-10 13:35:34.170191713 +0100
Modify: 2021-11-10 19:26:00.075610794 +0100
Change: 2021-11-10 19:39:34.341448658 +0100
Birth: -

Si borramos el original el primero, también se elimina el enlace simbólico que le apuntaba. Sin embargo, el enlace rígido permanece intacto, apuntando al mismo fichero. El enlace rígido se visualiza como una nueva copia independiente.

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)`.

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del fichero");
      return -1;
   }
```

```
    struct stat buf;
    int s = stat(argv[1],&buf);
    if(s == -1 || S_ISDIR(buf.st_mode)){
        printf("Error. Chequea el fichero introducido\n");
        return -1;
    }
    char* hard = (char *)malloc(sizeof(char)*(strlen(argv[1])+5)); //.hard
    char* sym = (char *)malloc(sizeof(char)*(strlen(argv[1])+4));  //.sym
    strcpy(hard,argv[1]);
    strcpy(sym,argv[1]);
    hard = strcat(hard,".hard");
    sym = strcat(sym,".sym");

    //Enlace rigido
    if(link(argv[1], hard) == -1){
        printf("Error al crear el enlace rigido");
        return -1;
    }
    else printf("Enlace rigido creado\n");
    //Enlace simbolico
    if(symlink(argv[1], sym) == -1){
        printf("Error al crear el enlace simbolico");
        return -1;
    }
    else printf("Enlace simbolico creado\n");
    return 1;
}
```

Terminal:

```
[cursoredes@localhost ~]$ g++ practica2.cpp -o ej11
[cursoredes@localhost ~]$ ./ej11 practica2.cpp
Enlace rigido creado
Enlace simbolico creado
[cursoredes@localhost ~]$ ls -l
total 32
lrwxrwxrwx 1 cursoredes cursoredes   11 Nov 10 19:32 aaaa -> directorioP
drwxr-xr-x 2 cursoredes cursoredes  116 Sep  9  2018 Desktop
drwxrwxr-x 2 cursoredes cursoredes    6 Nov 10 19:32 directorioP
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Documents
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8848 Nov 10 19:55 ej11
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 10 19:25 ej9
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Music
drwxr-xr-x 2 cursoredes cursoredes  147 Sep 22  2018 Pictures
-rw-rw-r-- 2 cursoredes cursoredes 1162 Nov 10 19:55 practica2.cpp
-rw-rw-r-- 2 cursoredes cursoredes 1162 Nov 10 19:55 practica2.cpp.hard
lrwxrwxrwx 1 cursoredes cursoredes   13 Nov 10 19:55 practica2.cpp.sym -> practica2.cpp
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Public
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Templates
drwxr-xr-x 2 cursoredes cursoredes    6 Sep  9  2018 Videos
```

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.

Fichero:

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del fichero");
      return -1;
   }
   int fd = open(argv[1],O_CREAT | O_RDWR,0777);
   if (fd == -1){
      printf("Error al crear o abrir el fichero\n");
      return -1;
   }
   int fd2 = dup2(fd,1);
   printf("Va todo ok jose luis\n");
   dup2(fd2,fd); //Recuperamos salida estandard

   return 1;
}
```

[cursoredes@localhost ~]$ ./ej12 salida.txt
[cursoredes@localhost ~]$ cat salida.txt
Va todo ok jose luis

IMPORTANTE:dup2 devuelve el descriptor de fichero abandonado.

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`?

Fichero:

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

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del fichero");
      return -1;
   }
   int fd = open(argv[1],O_CREAT | O_RDWR,0777);
   if (fd == -1){
      printf("Error al crear o abrir el fichero\n");
      return -1;
   }
   int fd2 = dup2(fd,1);
   int fd3 = dup2(fd,2);
   printf("Va todo ok jose luis\n");
   if(setuid(0) == -1){
      perror("Error");
   }
   dup2(fd2,fd);
   dup2(fd3,fd);
   return 1;
}
```

Terminal:
[cursoredes@localhost ~]$ ./ej13 salida2.txt
[cursoredes@localhost ~]$ cat salida2.txt
Error: Operation not permitted
Va todo ok jose luis

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.

```
[cursoredes@localhost ~]$ cat /proc/locks
1: POSIX  ADVISORY  WRITE 1583 fd:00:53013121 0 EOF
2: POSIX  ADVISORY  WRITE 1576 fd:00:53013120 0 EOF
3: POSIX  ADVISORY  WRITE 1571 fd:00:53012991 0 EOF
4: POSIX  ADVISORY  WRITE 1561 fd:00:53012987 0 EOF
5: POSIX  ADVISORY  WRITE 1355 00:13:21388 0 EOF
```

```
6: FLOCK  ADVISORY  WRITE 1319 fd:00:17458434 0 EOF
7: FLOCK  ADVISORY  WRITE 1319 fd:00:30831 0 EOF
8: FLOCK  ADVISORY  WRITE 1051 00:13:19964 0 EOF
9: POSIX  ADVISORY  WRITE 1050 00:13:19953 0 EOF
10: POSIX  ADVISORY  WRITE 776 00:13:16820 0 EOF
12: POSIX  ADVISORY  WRITE 482 00:13:12715 0 EOF
```

Ejercicio 15. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero usando `lockf(3)`. El programa mostrará el estado del cerrojo (bloqueado o desbloqueado). Además:

- Si está desbloqueado, fijará un cerrojo y escribirá la hora actual. Después suspenderá su ejecución durante 30 segundos (con `sleep(3)`) y a continuación liberará el cerrojo.
- Si está bloqueado, terminará el programa.

Fichero:

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del fichero");
      return -1;
   }
   int fd = open(argv[1],O_CREAT | O_RDWR,0777);
   if (fd == -1){
      printf("Error al crear o abrir el fichero\n");
      return -1;
   }

   struct flock lock;
   fcntl(fd, F_GETLK, &lock);
   if(lock.l_type == F_UNLCK){ //EL cerrojo esta desbloqueado

      lock.l_type = F_WRLCK;
      lock.l_whence = SEEK_SET;
      lock.l_start = 0;
      lock.l_len = 0;
      lock.l_pid = getpid();
      printf("El cerrojo esta desbloqueado\n");

      if(fcntl(fd,F_SETLK, &lock) == -1){
         printf("Error al crear el cerrojo\n");
         return -1;
      }
```

```
    else{ //Hemos creado con exito el nuevo cerrojo
      printf("Cerrojo de escritura creado\n");

      time_t t = time(NULL);
      char buffer[1024];
      struct tm *tm = localtime(&t);
      sprintf (buffer, "Hora: %d:%d:%d\n", tm->tm_hour, tm->tm_min, tm->tm_sec);
      write(fd, &buffer, strlen(buffer));

      sleep(30);

      lock.l_type = F_UNLCK;
      lock.l_whence = SEEK_SET;
      lock.l_start = 0;
      lock.l_len = 0;
      lock.l_pid = getpid();
      if (fcntl(fd, F_SETLK, &lock) == -1) {
        printf("ERROR al liberar el cerrojo\n");
        return -1;
      }
      else printf("Cerrojo liberado\n");
    }
  }
  else{
    printf("Cerrojo bloqueado\n");
    close(fd);
  }
  return 1;
}
```

Terminal:

```
[cursoredes@localhost ~]$ ./ej15 ficheroPrueba
Cerrojo bloqueado
```

***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 cumpla las siguientes especificaciones:

- 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 de forma que:
  - Si es un fichero normal, escribirá el nombre.
  - Si es un directorio, escribirá el nombre seguido del carácter `'/'`.
  - Si es un enlace simbólico, escribirá su nombre seguido de `'->'` y el nombre del fichero enlazado. Usar `readlink(2)` y dimensionar adecuadamente el *buffer*.
  - Si el fichero es ejecutable, escribirá el nombre seguido del carácter `'*'`.
- Al final de la lista el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.

Fichero:

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

int main(int argc, char ** argv){
   if (argc < 2){
      printf("Error. Debe pasar la ruta del directorio\n");
      return -1;
   }
   DIR* dir = opendir(argv[1]);
   if(dir == NULL){
      printf("Chequee la ruta del directorio\n");
      return -1;
   }

   struct dirent* d = readdir(dir);
   int totalBytes = 0;

   while (d!=NULL){
      if(d->d_type == DT_REG){ //fichero regular
         struct stat s;
         if(lstat(d->d_name,&s) == -1){
            printf("Error al obtener info del fichero %s\n",d->d_name);
            return -1;
         }
         if(s.st_mode && S_IXUSR){ //El fichero es ejecutable
            printf("%s*\n",d->d_name);
         }
         else printf("%s\n",d->d_name);

         totalBytes +=s.st_size;
      }
      if(d->d_type == DT_DIR){ //directorio
         printf("%s/\n", d->d_name);
      }
      if(d->d_type == DT_LNK){ //enlace simbolico
         char* path = (char*)malloc(sizeof(char)*(strlen(argv[1]) + strlen(d->d_name) + 1));
         strcpy(path, argv[1]);
         strcat(path, "/");
         strcat(path, d->d_name);
         struct stat s;
         if(lstat(path,&s) == -1){
            printf("Error al obtener info del fichero %s\n",d->d_name);
            return -1;
         }
         char* buf = (char *)malloc(s.st_size + 1);
         if(readlink(path, buf, s.st_size + 1) == -1){
```

```
            printf("Error al leer el enlace simbolico\n");
            return -1;
          }
          printf("%s -> %s\n",d->d_name,buf);
          free(buf);
          free(path);
          totalBytes += s.st_size;
       }
       d = readdir(dir);

    }
    printf("Bytes de ficheros: %i\n",totalBytes);
    closedir(dir);
    return 1;
}
```

Terminal:

```
[cursoredes@localhost ~]$ ./ej17 '.'
./
../
.bash_logout*
.bash_profile*
.bashrc*
.bash_history*
.cache/
.dbus/
.config/
.ICEauthority*
.local/
Desktop/
Downloads/
Templates/
Public/
Documents/
Music/
Pictures/
Videos/
.esd_auth*
.cinnamon/
.wireshark/
.mozilla/
.pki/
.vscode/
.lesshst*
.dmrc*
.viminfo*
.vboxclient-clipboard.pid*
.vboxclient-display.pid*
.vboxclient-seamless.pid*
.vboxclient-draganddrop.pid*
practica2.cpp*
ej9*
directorioP/
aaaa -> directorioP
```

```
ej11*
practica2.cpp.hard*
practica2.cpp.sym -> practica2.cpp
ej12*
salida.txt*
ej13*
salida2.txt*
ej15*
ficheroPrueba*
ej17*
Bytes de ficheros: 73727
[cursoredes@localhost ~]$
```