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 : 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`).

```
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ ls -ld
drwxrwxr-x 4 usuarioso usuarioso 4096 nov  8 11:28 .
```

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 (+ es para añdir permisos, - es por defecto)
```

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

```
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ mkdir Prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ ls
'PRACTICA 1'  'PRACTICA 2'   Prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ chmod 666 Prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ ls
'PRACTICA 1'  'PRACTICA 2'   Prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR$ cd Prueba
bash: cd: Prueba: Permiso denegado
```

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

int main(){
   int fd = open("./fichero.txt", O_CREAT, 0645);
   if (fd == -1){
      printf(strerror(errno));
   }
```

```
   printf("Descriptor = %i\n", fd);
   return 1;
}

----
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 24
-rw-r--r-x 1 usuarioso usuarioso     0 Nov  8 12:07 fichero.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)`.

```
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ umask 0027
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ mkdir prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 28
drwxr-x--- 2 usuarioso usuarioso  4096 Nov  8 12:12 prueba

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ umask 0027
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ touch pruebaaa
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 28
-rw-r----- 1 usuarioso usuarioso     0 Nov  8 12:44 pruebaaa
```

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



int main(){
   mode_t aux = umask(0027);
   int fd = open("./fichero2.txt", O_CREAT, 0777);
   if (fd == -1){
      printf(strerror(errno));
   }

   umask(aux);

   return 1;
}
```

```
------
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ./ej7
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ umask
0022
-rwxr-x--- 1 usuarioso usuarioso     0 Nov  8 12:47 fichero2.txt
```

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.

```
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -i
543130 Practica_2.2.cpp  543162 ej5  543197 ej7  543166 fichero.txt  543179 prueba

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat .
  File: .
  Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 801h/2049d     Inode: 543033      Links: 3
Access: (0775/drwxrwxr-x)  Uid: ( 1000/usuarioso)   Gid: ( 1001/usuarioso)
Access: 2021-11-08 12:19:19.831963849 +0100
Modify: 2021-11-08 12:19:19.827963849 +0100
Change: 2021-11-08 12:19:19.827963849 +0100
 Birth: -

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero.txt
  File: ./fichero.txt
  Size: 0             Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d     Inode: 543166      Links: 1
Access: (0645/-rw-r--r-x)  Uid: ( 1000/usuarioso)   Gid: ( 1001/usuarioso)
Access: 2021-11-08 12:07:13.891971849 +0100
Modify: 2021-11-08 12:07:13.891971849 +0100
Change: 2021-11-08 12:07:13.891971849 +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 <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/sysmacros.h>
#include <time.h>

int main(int argc, char **argv) {
	if (argc < 2) {
		printf("ERROR: Hay poner la ruta del archivo como parámetro.\n");
		return -1;
	}

	struct stat buff;
	int statint = stat(argv[1], &buff);
	if (statint == -1) {
		printf("ERROR: No existe el directorio.\n");
		return -1;
	}

	//Major y Minor
	printf("MAJOR: %li\n", (long) minor(buff.st_dev));
	printf("MINOR: %li\n", (long) major(buff.st_dev));

	//I-Node
	printf("I-Node: %li\n",buff.st_ino);

	//Tipo de archivo
	printf("MODE: %i - ",buff.st_mode);

	mode_t mode = buff.st_mode;

	if (S_ISLNK(mode)){
		printf("%s es un enlace simbólico\n", argv[1]);
	} else if (S_ISREG(mode)) {
		printf("%s es un archivo normal\n", argv[1]);
	} else if (S_ISDIR(mode)) {
		printf("%s es un directorio\n", argv[1]);
	}

	//Hora ficheros A
	time_t t1 = buff.st_atime;
	struct tm *time1 = localtime(&t1);
	printf("Último acceso: %d-%d-%d %d:%d:%d\n", time1->tm_year,
time1->tm_yday, time1->tm_mday, time1->tm_hour, time1->tm_min, time1->tm_sec);

	//Hora ficheros M
	time_t t2 = buff.st_mtime;
```

```
        struct tm *time2 = localtime(&t2);
        printf("Última modificación: %d-%d-%d %d:%d:%d\n", time2->tm_year,
time2->tm_yday, time2->tm_mday, time2->tm_hour, time2->tm_min, time2->tm_sec);

        //Hora ficheros C
        time_t t3 = buff.st_ctime;
        struct tm *time3 = localtime(&t3);
        printf("Último cambio: %d-%d-%d %d:%d:%d\n", time3->tm_year,
time3->tm_yday, time3->tm_mday, time3->tm_hour, time3->tm_min, time3->tm_sec);

        return 1;
}

-----

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ./ej9 ./fichero.txt
MAJOR: 1
MINOR: 8
I-Node: 543166
MODE: 33189 - ./fichero.txt es un archivo normal
Último acceso: 121-311-8 12:7:13
Última modificación: 121-311-8 12:7:13
Último cambio: 121-311-8 12:7:13
```

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?

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ln -s fichero2.txt fichero3.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
-rwxr-x--- 1 usuarioso usuarioso 0 Nov 8 **12:47 fichero2.txt**
lrwxrwxrwx 1 usuarioso usuarioso 12 Nov 8 **20:10 fichero3.txt** -> fichero2.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -i
543130 Practica_2.2.cpp 529490 ej7 543166 fichero.txt **543101 fichero3.txt** 543345 pruebaaa
543162 ej5 543226 ej9 **529517 fichero2.txt** 543179 prueba

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ln -s prueba prueba2

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
drwxr-x--- 2 usuarioso usuarioso 4096 Nov 8 12:44 prueba
lrwxrwxrwx 1 usuarioso usuarioso 6 Nov 8 20:12 prueba2 -> prueba
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -i

543130 Practica_2.2.cpp 529490 ej7 543166 fichero.txt 543101 fichero3.txt 543155 prueba2
543162 ej5 543226 ej9 529517 fichero2.txt 543179 prueba 543345 pruebaaa

---

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ln fichero2.txt fichero4.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
-rwxr-x--- 2 usuarioso usuarioso 0 Nov 8 **12:47 fichero2.txt**
lrwxrwxrwx 1 usuarioso usuarioso 12 Nov 8 20:10 fichero3.txt -> fichero2.txt
-rwxr-x--- 2 usuarioso usuarioso 0 Nov 8 **12:47 fichero4.txt**

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -i
543130 Practica_2.2.cpp 529490 ej7 543166 fichero.txt 543101 fichero3.txt 543179 prueba 543345 pruebaaa
543162 ej5 543226 ej9 **529517 fichero2.txt 529517 fichero4.txt** 543155 prueba2

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero2.txt
  File: ./fichero2.txt
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 801h/2049d **Inode: 529517** Links: 2
Access: (0750/-rwxr-x---) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 12:47:05.039945496 +0100
Modify: 2021-11-08 12:47:05.039945496 +0100
Change: 2021-11-08 20:19:31.535970655 +0100
 Birth: -

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero4.txt
  File: ./fichero4.txt
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 801h/2049d **Inode: 529517** Links: 2
Access: (0750/-rwxr-x---) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 12:47:05.039945496 +0100
Modify: 2021-11-08 12:47:05.039945496 +0100
Change: 2021-11-08 20:19:31.535970655 +0100
 Birth: -

BORRO ENLACE RÍGIDO:

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ rm fichero4.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
-rwxr-x--- 1 usuarioso usuarioso 0 Nov 8 12:47 fichero2.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -i
543130 Practica_2.2.cpp 529490 ej7 543166 fichero.txt 543101 fichero3.txt 543155 prueba2
543162 ej5 543226 ej9 **529517 fichero2.txt** 543179 prueba 543345 pruebaaa

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero2.txt
  File: ./fichero2.txt
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 801h/2049d Inode: 529517 **Links: 1**
Access: (0750/-rwxr-x---) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 12:47:05.039945496 +0100
Modify: 2021-11-08 12:47:05.039945496 +0100
Change: 2021-11-08 20:22:49.507968473 +0100
 Birth: -

BORRO ENLACE SIMBÓLICO: No cambia nada. Se elimina el fichero 3 y ya

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ rm fichero4.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero2.txt
  File: ./fichero2.txt
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 801h/2049d Inode: 529517 **Links: 1**
Access: (0750/-rwxr-x---) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 12:47:05.039945496 +0100
Modify: 2021-11-08 12:47:05.039945496 +0100
Change: 2021-11-08 20:22:49.507968473 +0100
 Birth: -

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
-rwxr-x--- 1 usuarioso usuarioso 0 Nov 8 12:47 fichero2.txt

BORRO EL ORIGINAL CON EL ENLACE RÍGIDO Y SIMBOLICO: "Se elimina" el simbolico, pero al rigido no le pasa nada

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ rm fichero2.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 68
lrwxrwxrwx 1 usuarioso usuarioso 12 Nov 8 20:30 fichero3.txt -> fichero2.txt
-rwxr-x--- 1 usuarioso usuarioso 0 Nov 8 12:47 fichero4.txt

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero3.txt
  File: ./fichero3.txt -> fichero2.txt
  Size: 12 Blocks: 0 IO Block: 4096 symbolic link
Device: 801h/2049d Inode: 543101 **Links: 1**
Access: (0777/lrwxrwxrwx) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 20:30:28.035963419 +0100
Modify: 2021-11-08 20:30:24.371963460 +0100
Change: 2021-11-08 20:30:24.371963460 +0100
 Birth: -

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ stat ./fichero4.txt
  File: ./fichero4.txt
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 801h/2049d Inode: 529517 **Links: 1**
Access: (0750/-rwxr-x---) Uid: (1000/usuarioso) Gid: (1001/usuarioso)
Access: 2021-11-08 12:47:05.039945496 +0100
Modify: 2021-11-08 12:47:05.039945496 +0100
Change: 2021-11-08 20:30:35.179963341 +0100
 Birth: -

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

int main(int argc, char **argv) {
	if (argc < 2) {
		printf("ERROR: Hay poner la ruta del archivo como parámetro.\n");
		return -1;
	}

	struct stat buff;
	int statint = stat(argv[1], &buff);
	if (statint == -1) {
		printf("ERROR: No existe el directorio.\n");
		return -1;
	}

	char* hard = (char*)malloc(sizeof(char)*(5 + strlen(argv[1]))); //5 para el .hard
	char* sym = (char*)malloc(sizeof(char)*(4 + strlen(argv[1]))); //4 para el .sym
	strcpy(hard, argv[1]);
	strcpy(sym, argv[1]);
	hard = strcat(hard, ".hard");
	sym = strcat(sym, ".sym");

	//Tipo de archivo
	mode_t mode = buff.st_mode;

	if (S_ISREG(mode)) {
		if(link(argv[1], hard) == -1){
			printf("ERROR: No se ha podido crear el enlace rigido\n", argv[1]);
		}
		else{
			printf("Enlace rigido creado!!\n", argv[1]);
		}
		if(symlink(argv[1], sym) == -1){
			printf("ERROR: No se ha podido crear el enlace simbolico\n", argv[1]);
		}
		else{
			printf("Enlace simbolico creado!!\n", argv[1]);
		}
	} else if (S_ISDIR(mode)) {
		printf("ERROR: %s no es un fichero regular\n", argv[1]);
	}

	return 1; //free(hard); free(sym);
}
```

```
-------
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ./ej11 ./fichero.txt
Enlace rigido creado!!
Enlace simbolico creado!!

usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ls -l
total 88
-rw-r--r-x 2 usuarioso usuarioso     0 Nov  8 12:07 fichero.txt.hard
lrwxrwxrwx 1 usuarioso usuarioso    13 Nov  9 10:59 fichero.txt.sym -> ./fichero.txt
```

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

int main(int argc, char **argv) {
        if (argc < 2) {
                printf("ERROR: Hay poner la ruta del archivo como parámetro.\n");
                return -1;
        }

        int fd = open(argv[1], O_CREAT | O_RDWR, 0777);
        if (fd == -1) {
                printf("ERROR: No se ha podido crear/abrir el fichero.\n");
                return -1;
        }

        int fd2 = dup2(fd, 1);
        printf("Esto se muestra en %s\n", argv[1]);
        printf("No se ha producido ningun fallo\n");
        dup2(fd2, fd);

        return 1;
}

-------
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ./ej12 salida.txt
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ cat salida.txt
Esto se muestra en salida.txt
No se ha producido ningún fallo
```

Nombre	Decriptores	Salida
entrada estándar (*stdin*)	0	teclado
salida estándar (*stdout*)	1	pantalla
error estándar (*stderr*)	2	pantalla

Hemos duplicado el descriptor del fichero (fd) en el 1 que es el de la salida estándar y devuelve el descriptor 1 que se almacena en fd2. Entonces la salida estándar ahora es el fichero. Al final dup2(fd2, fd) lo que hace es restaurar la salida estándar pues duplica el fd2 (que tiene el descriptor de escribir por pantalla) en fd (la salida estándar de ahora)

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

int main(int argc, char **argv) {
	if (argc < 2) {
		printf("ERROR: Hay poner la ruta del archivo como parámetro.\n");
		return -1;
	}

	int fd = open(argv[1], O_CREAT | O_RDWR, 0777);
	if (fd == -1) {
		printf("ERROR: No se ha podido crear/abrir el fichero.\n");
		return -1;
	}

	int fd2 = dup2(fd, 1);
	int fd3 = dup2(fd, 2);
	printf("Esto se muestra en %s\n", argv[1]);
	printf("No se ha producido ningún fallo\n");
	printf("Descriptor duplicado: %d\n", fd);
	printf("Descriptor estándar: %d\n", fd2);
	printf("Descriptor error: %d\n", fd3);
	char *s;
	if (setuid(0) == -1){
		perror(s);
	}
	dup2(fd2, fd);
	dup2(fd3, fd);

	return 1;
}
```

```
-------
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ ./ej13 salida.txt
usuarioso@usuarioso-virtualbox:~/Desktop/ASOR/PRACTICA 2$ cat salida.txt
Operation not permitted
Esto se muestra en salida.txt
No se ha producido ningún fallo
Descriptor duplicado: 3
Descriptor estándar: 1
Descriptor error: 2
```

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  READ 1620 08:01:461834 128 128
2: POSIX  ADVISORY  READ 1620 08:01:460655 1073741826 1073742335
3: POSIX  ADVISORY  WRITE 977 08:01:529513 0 EOF
4: POSIX  ADVISORY  WRITE 969 08:01:529512 0 EOF
5: POSIX  ADVISORY  WRITE 962 08:01:529511 0 EOF
6: POSIX  ADVISORY  WRITE 950 08:01:529495 0 EOF
7: FLOCK  ADVISORY  WRITE 556 00:1b:6 0 EOF
8: FLOCK  ADVISORY  WRITE 377 00:18:379 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.

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

int main(int argc, char **argv) {
        if (argc < 2) {
                printf("ERROR: Hay poner la ruta del archivo como parámetro.\n");
                return -1;
        }

        int fd = open(argv[1], O_CREAT | O_RDWR, 0777);
        if (fd == -1) {
                printf("ERROR: No se ha podido crear/abrir el fichero.\n");
                return -1;
        }

        struct flock lock;
  lock.l_type = F_UNLCK;
  lock.l_whence = SEEK_SET;
  lock.l_start = 0; //Desde el inicio
  lock.l_len = 0; //hasta EOF
  lock.l_pid = getpid();

  fcntl(fd, F_GETLK, &lock);

  if(lock.l_type == F_UNLCK){
    printf("Cerrojo desbloqueado!\n");
    lock.l_type = F_WRLCK;
    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 activar el cerrojo.\n");
      close(fd);
      return 1;
    }
    else{
      printf("Cerrojo de escritura creado!\n");

      //Escribimos hora actual en el fichero
      char buffer[1024];
```

```
        time_t t = time(NULL);
        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));

        //Suspendemos ejec. 30 sec
        sleep(30);

        //Liberamos cerrojo
        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");
           close(fd);
           return 1;
        } else
           printf("Cerrojo liberado!\n");
           close(fd);
        }
   }
   else{
      printf("Cerrojo bloqueado!\n");
      close(fd);
      return 1;
   }

          return 1;
}

-------

Cerrojo desbloqueado!
Cerrojo de escritura creado!
Cerrojo liberado!
```

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

```
NAME
     flock - manage locks from shell scripts

SYNOPSIS
     flock [options] file|directory command [arguments]
     flock [options] file|directory -c command
     flock [options] number

DESCRIPTION
     This utility manages flock(2) locks from within shell scripts or from the command line.

     The  first  and  second  of the above forms wrap the lock around the execution of a command,
```

in a manner similar to su(1) or newgrp(1). They lock a specified file or directory, which is created (assuming appropriate permissions) if it does not already exist. By default, if the lock cannot be immediately acquired, flock waits until the lock is available.

The third form uses an open file by its file descriptor number. See the examples below for how that can be used.

OPTIONS

-e, -x, --exclusive
Obtain an exclusive lock, sometimes called a write lock. This is the default.

-n, --nb, --nonblock
Fail rather than wait if the lock cannot be immediately acquired. See the -E option for the exit code used.

-o, --close
Close the file descriptor on which the lock is held before executing command. This is useful if command spawns a child process which should not be holding the lock.

-s, --shared
Obtain a shared lock, sometimes called a read lock.

-u, --unlock
Drop a lock. This is usually not required, since a lock is automatically dropped when the file is closed. However, it may be required in special cases, for example if the enclosed command group may have forked a background process which should not be holding the lock.

-w, --wait, --timeout seconds
Fail if the lock cannot be acquired within seconds. Decimal fractional values are allowed. See the -E option for the exit code used. The zero number of seconds is interpreted as --nonblock.

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.

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

int main(int argc, char **argv) {
        if (argc < 2) {
                printf("ERROR: Hay poner la ruta del directorio como parámetro.\n");
                return -1;
        }

        DIR* directorio = opendir(argv[1]);
  if(directorio == NULL){
    printf("ERROR: el directorio no existe.\n");
    return -1;
  }

  struct dirent *dir = readdir(directorio);
        unsigned long int totalSize = 0;
  struct stat info;

  while(dir != NULL){
    char *path = (char*)malloc(sizeof(char)*(strlen(argv[1]) + strlen(dir->d_name) + 3));
    strcpy(path, argv[1]);
    strcat(path, "/");
    strcat(path, dir->d_name);

    if(lstat(path, &info) == -1){
      printf("ERROR: no se puede leer la entrada del directorio\n");
      free(path);
      closedir(directorio);
      return -1;
    }
    else{  //hacer asi las conds. o con IS_REG(info.st_mode), IS_DIR(st.mode)...
      if(dir->d_type == DT_REG){ //Si es fichero regular (ver si es ejecutable o no)
        if(info.st_mode && S_IXUSR){
          printf("[Archivo] %s*\n", dir->d_name);
        }
        else{
          printf("[Archivo] %s\n", dir->d_name);
        }
        //totalSize += info.st_blocks * (info.st_blksize/8);
        totalSize += info.st_size;
      }
      else if(dir->d_type == DT_DIR){ //Si es directorio
        printf("[Directorio] /%s\n", dir->d_name);
      }
      else if(dir->d_type == DT_LNK){ //Si es enlace simbolico
        char *buff = (char*)malloc(info.st_size + 1);
        readlink(path, buff, info.st_size + 1);
        printf("[Enlace Simbolico] %s->%s\n", dir->d_name, buff);
        free(buff);
```

```
        }
        else { //Si es fichero ejecutable
           printf("[Ejecutable] %s*\n", dir->d_name);
           totalSize += info.st_blocks * (info.st_blksize/8);
        }
      }

      free(path);
      dir = readdir(directorio);
   }

   printf("Tamaño total: %li\n", totalSize);
   closedir(directorio);
           return 1;
}

--------
[Archivo] ej15*
[Archivo] fichero.txt.hard*
[Archivo] Practica_2.2.cpp*
[Enlace Simbolico] fichero.txt.sym->./fichero.txt
[Archivo] cerrojos.txt*
[Archivo] fichero4.txt*
[Directorio] /prueba
[Enlace Simbolico] prueba2->prueba
[Archivo] ej13*
[Directorio] /.
[Archivo] ej7*
[Archivo] ej11*
[Archivo] salida.txt*
[Archivo] pruebaaa*
[Archivo] 1*
[Archivo] fichero.txt*
[Directorio] /..
[Archivo] ej5*
[Archivo] ej9*
[Archivo] ej17*
[Archivo] ej12*
Tamaño total: 138600
```