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

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.

En la realización de las prácticas se puede usar cualquier editor gráfico o de terminal. Además se puede usar tanto el lenguaje C (compilador `gcc`) como C++ (compilador `g++`). Si fuera necesario compilar varios ficheros se recomienda el uso de alguna herramienta para la compilación de proyectos como `make`. Finalmente, el depurador recomendado en las prácticas es `gdb`. **No está permitido** el uso de IDEs como Eclipse.

Creación y atributos de ficheros

El i-nodo 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. La herramienta principal para consultar el contenido y atributos básicos de un fichero es `ls`. 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.

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`).

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
```

```
$ ls fichero -ld
-rw-r--r-x 1 luis luis 1 dic  8 12:28 fichero


$chmod u+rw-x,g+r-wx,o+rx-w fichero1
$ ls fichero1 -ld
-rw-r--r-x 1 luis luis 1 dic  8 12:28 fichero1
```

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

```
$ mkdir prueba
$ chmod 666 prueba

$ cd prueba
bash: cd: prueba: Permission denied
```

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

```
$ ./ej5
$ ls ej5.txt -ld
-rw-r--r-x 1 luis luis 0 dic  8 12:48 ej5.txt
```

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

int main(int argc, char *argv[]){

    if(argc!=2) {
       printf("Ha olvidado el nombre del fichero.\n");
       exit(1);
       }

      const char *path = argv[1];

      int df = open(path, O_CREAT, 0645);

      printf("Descriptor de fichero %s = %i\n", argv[1], df);

    return 0;
}
```

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 ningún permiso para otros. Comprobar el funcionamiento con los comandos `touch`, `mkdir` y `ls`.

```
$ umask 027           (777 - 750 = 027) -> (777 = rwxrwxrwx, 750 = rwxr-x---)
$ mkdir directorio1   (Permisos de directorio 0777, de archivos 0666)

$ ls -l
drwxr-x--- 2 luis luis  4096 dic  9 11:32 directorio1
```

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 el comando `ls`. 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 <stdlib.h>

int main(int argc, char *argv[]){

    if(argc!=2) {
       printf("Ha olvidado el nombre del fichero.\n");
       exit(1);
    }

    mode_t prev = umask(027); //027

    const char *path = argv[1];
    int df = open(path, O_CREAT, 0777);

    mode_t last = umask(prev);

    printf("Mask = %i\n", last);
    return 0;
}
```

```
$ gcc ej7.c -o ej7
$ ./ej7 prueba
Mask = 23
$ ls -l
-rwxr-x--- 1 luis luis    0 dic  9 11:29 prueba

$ umask
0022
```

Ejercicio 8. El comando `ls` puede mostrar el i-nodo con la opción `-i`. El resto de información del

i-nodo puede obtenerse usando el comando `stat`. Consultar las opciones del comando y comprobar su funcionamiento.

```
$l s -i

14818725 ej5  14818817 ej5.c  14811184 ej7  14811183 ej7.c

$ stat ej7.c

  File: ej7.c
  Size: 462          Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d    Inode: 14811183  Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ luis)   Gid: ( 1000/      luis)
Access: 2020-12-09 08:33:54.623925459 +0100
Modify: 2020-12-09 11:28:23.960984191 +0100
Change: 2020-12-09 11:28:23.960984191 +0100
 Birth: 2020-12-09 08:33:54.623925459 +0100
```

Ejercicio 9. Escribir un programa que emule el comportamiento del comando `stat` y muestre:

- El número *major* y *minor* asociado al dispositivo.
- El número de i-nodo 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 <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sys/sysmacros.h>


int main(int argc, char *argv[]){

    if(argc != 2) {
       printf("ERROR: Ha olvidado la ruta del fichero.\n");
       return -1;
    }

    char *path = argv[1];
    struct stat buff;

    if(stat(path, &buff) == -1){
       printf("ERROR: stat()\n");
       return -1;
    }
```

```
    printf("Tipo fichero:  ");

    if (S_ISREG(buff.st_mode)) {
       printf("fichero ordinario.\n");
    }
    else if (S_ISLNK(buff.st_mode)) {
       printf("enlace simbolico.\n");
    }
    else if (S_ISDIR(buff.st_mode)) {
       printf("directorio.\n");
    }
    else {
       printf("otros...\n");
    }


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

    printf("MAJOR: %li\n", major(buff.st_dev));
    printf("MINOR: %li\n", minor(buff.st_dev));

    printf("Ultimo cambio de estado: %s", ctime(&buff.st_ctime));
    printf("Ultimo acceso:           %s", ctime(&buff.st_atime));
    printf("Ultima modificación:     %s", ctime(&buff.st_mtime));
    return 0;
}
```

```
$ gcc ej9.c -o ej9
$ ./ej9 fichero

Tipo fichero:  fichero ordinario.
I-node: 14811201
MAJOR: 8
MINOR: 2
Ultimo cambio de estado:  Wed Dec  9 12:58:43 2020
Ultimo acceso:            Wed Dec  9 11:50:45 2020
Ultima modificacion:      Wed Dec  9 12:58:43 2020
```

st_atime: es el último acceso (cuando se hace open por ejemplo).

st_mtime: es la última modificación dentro del archivo.

st_ctime: es la última modificación ya sea de permisos, del archivo, de nombre…

Ejercicio 10. Los enlaces se crean con la orden `ln`:

- La opción `-s` 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 i-nodo de cada fichero.
- Repetir el apartado anterior con enlaces rígidos. Determinar los i-nodos 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?

```
$ touch fichero
$ touch directorio

$ ln -s fichero fichero1
$ ln -s directorio directorio1

$ ls -li
14818802 drwxr-xr-x 2 luis luis  4096 dic  9 13:07 directorio
14818804 lrwxrwxrwx 1 luis luis  10 dic  9 13:07 directorio1 -> directorio
14818800 -rw-r--r-- 1 luis luis  0 dic  9 13:06 fichero
14818801 lrwxrwxrwx 1 luis luis  7 dic  9 13:06 fichero1 -> fichero
```

```
$ touch fichero2
$ touch directorio2

$ ln directorio2 directorio3
$ ln fichero2 fichero3

$ ls -li
14818801 -rw-r--r-- 2 luis luis  0 dic  9 13:22 directorio2
14818801 -rw-r--r-- 2 luis luis  0 dic  9 13:22 directorio3
14818800 -rw-r--r-- 2 luis luis  0 dic  9 13:22 fichero2
14818800 -rw-r--r-- 2 luis luis  0 dic  9 13:22 fichero3
```

Enlace Duro: Es un nuevo nombre para un fichero, apunta al mismo nodo-i y no ocupan espacio en el disco. (No se pueden crear para directorios).

Enlace Simbólico: El archivo con enlace simbólico contiene la ruta del archivo al que se le ha hecho el enlace.

Al eliminar uno de los enlaces rígidos solo se decrementa el número de enlaces, tanto si borras el fichero original como el enlazado.

Al eliminar uno de los enlaces simbólicos se decrementa el número de enlaces, y si elimino el fichero original se rompe el enlace simbólico.

Ejercicio 11. Las llamadas `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 la orden `ls`.

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



int main(int argc, char *argv[]){

   if(argc != 2) {
       printf("ERROR: Ha olvidado la ruta del fichero.\n");
       return -1;
   }

   char *path = argv[1];
   struct stat buff;

   if(stat(path, &buff) == -1){
       printf("ERROR: funcion stat.\n");
       return -1;
   }

   char slink[255];
   char hlink[255];

   strcpy(slink, argv[1]);
   strcat(slink, ".sym");
   strcpy(hlink, argv[1]);
   strcat(hlink, ".hard");


   if (S_ISREG(buff.st_mode)) {
       if(symlink(argv[1], slink) == -1){ return -1; }
        printf("Enlace simbólico creado.\n");

       if(link(argv[1], hlink) == -1){ return -1;}
       printf("Enlace rígido creado.\n");
   }
   else{
       printf("ERROR. El archivo especificado no es regular.\n");
       return -1;
   }

   return 0;
}
```

```
$ gcc ej11.c -o ej11
$ ./ej11 f1
Enlace simbólico creado.
Enlace rígido creado.
```

```
$ ls -li
14818800 -rw-r--r-- 2 luis luis  0 dic  9 13:48 f1
14818800 -rw-r--r-- 2 luis luis  0 dic  9 13:48 f1.hard
14818806 lrwxrwxrwx 1 luis luis  2 dic  9 13:49 f1.sym -> f1
```

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 las llamadas `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>


int main(int argc, char *argv[]) {

    if (argc < 2) {
        printf("ERROR: Se debe especificar la ruta.\n");
        return -1;
    }

       int fd = open(argv[1], O_CREAT | O_RDWR, 0777);

       if (fd == -1) {
       printf("ERROR: No se ha podido abrir/crear el fichero.\n");
       return -1;
       }

       if(dup2(fd, 1) == -1){
       printf("ERROR: Fallo en dup2.");
       }

       printf("Esto se ha redirigido de la salida estandar del fichero.\n");

       return 1;
}
```

Ejercicio 13. Modificar el programa anterior para que además de escribir en el fichero la salida estándar también se escriba la salida estándar de error. Comprobar el funcionamiento incluyendo varias sentencias que impriman en ambos flujos. ¿Hay alguna diferencia si las redirecciones se hacen en diferente orden? ¿Por qué no es lo mismo "`ls > dirlist 2>&1`" que "`ls 2>&1 > dirlist`"?

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

int main(int argc, char *argv[]) {

    if (argc < 2) {
        printf("ERROR: Se debe especificar la ruta.\n");
        return -1;
    }

       int fd = open(argv[1], O_CREAT | O_RDWR, 0777);

       if (fd == -1) {
       printf("ERROR: No se ha podido abrir/crear el fichero.\n");
       return -1;
       }

       if(dup2(fd, 1) == -1 && dup2(fd, 2) == -1){
       printf("ERROR: Fallo en dup2.");
       }

       printf("Esto se ha redirigido de la salida estandar.\n");

       fprintf(stderr, "Esto se ha redirigido de la salida estandar de error.\n");

       return 1;
}
```

`ls > dirlist 2>&1`: Redirecciona la salida estándar a dirlist y luego la salida de errores la redirecciona a la 1 que corresponde a dirlist. Por lo tanto, ambas salidas se van a ver en dirlist.

`ls 2>&1 > dirlist`: Redirecciona la salida de error a la salida estándar y la salida estándar a dirlist, por lo tanto los errores se van a ver por pantalla y la salina 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.

Ejercicio 15. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero. 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.

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

Proyecto: Comando `ls` extendido

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

int main(int argc, char **argv) {

    if(argc < 2){
        printf("ERROR: Debes introducir la ruta del directorio como parametro\n");
    }

    DIR *directorio = opendir(argv[1]);

    if(directorio == NULL){
        printf("ERROR: No existe el directorio\n");
    }

    struct dirent *sig = readdir(directorio);

    int totalsize = 0;
    struct stat sb;

    while(sig != NULL){

        char path[250];                 /*Preparo la siguiente entrada...*/
        strcpy(path, argv[1]);
        strcat(path, "/");
        strcat(path, sig->d_name);
```

```
        if (stat(path, &sb) == -1) {
               closedir(directorio);
              printf("ERROR: (stat)\n");
       }


       /*Distingo entre el tipo de ficheros [OJO -> Acordarse de stat y S_IS--(sb.st_mode) ]*/
       if(S_ISREG(sb.st_mode)){
               printf("%s (OCUPA: %i Bytes)\n", sig->d_name, sb.st_size);
               totalsize = totalsize + sb.st_size;
       }
       else if(S_ISDIR(sb.st_mode)){
               printf("/ %s \n", sig->d_name);
       }
       else if(S_ISLNK(sb.st_mode)){   /* Leo el link con "readLink"... */
               char buff_name[1024];
               ssize_t len = readlink(path, buff_name, sizeof(buff_name)-1);

               printf("%s -> %s (OCUPA: %i Bytes)\n", sig->d_name, buff_name, sb.st_size);
               totalsize = totalsize + sb.st_size;
       }

        sig = readdir(directorio);
    }

    printf("Tamaño total que ocupan los ficheros: %i KB.\n", totalsize/1024);

    closedir(directorio);
    return 0;
}
```

```
[luis@luis practica2_ASOR]$ gcc ej16.c -o ej16
[luis@luis practica2_ASOR]$ ./ej16 .
ej9     (OCUPA: 16992 Bytes)
ej12    (OCUPA: 16752 Bytes)
ej16.c  (OCUPA: 1596 Bytes)
f1      (OCUPA: 0 Bytes)
/ ..
/ .
ej5     (OCUPA: 16752 Bytes)
Ej16    (OCUPA: 17192 Bytes)
ej13    (OCUPA: 16848 Bytes)
ej13.c  (OCUPA: 666 Bytes)
f1.sym  (OCUPA: 0 Bytes)
ej11    (OCUPA: 16960 Bytes)
ej9.c   (OCUPA: 1346 Bytes)
ej11.c  (OCUPA: 972 Bytes)
f1.hard (OCUPA: 0 Bytes)
ej7     (OCUPA: 16800 Bytes)
ej7.c   (OCUPA: 462 Bytes)
ej5.c   (OCUPA: 421 Bytes)
ej12.c  (OCUPA: 569 Bytes)
Tamaño total que ocupan los ficheros: 121 KB.
```