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 archivos 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 archivos 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.

- -a: lista todos los elementos del directorio actual
- -l: lista todos los directorios con un formato largo
- -d: lista todos los directorios dentro del directorio actual
- -h: lista todos los elementos del directorio actual con formato legible por humanos
- -i: lista todos los elementos del directorio actual junto con su i-nodo.
- -R: lista los directorios recursivamente.
- -1: lista una fichero por línea.
- -F : añadir el indicador (uno de * / => @ |) a las entradas
- --color: colorea la salida.

Ejercicio 2. Los permisos de un fichero son <tipo><rwx_propietario><rwx_grupo><rwx_resto>:

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

Comprobar los permisos del directorio `$HOME` del usuario y de `/etc/sudoers.d` (`ls -ld`) e intentar cambiar a ese directorio.

`ls -la`: Lista todos los elementos del directorio actual junto con sus permisos

`ls -ld <nombreFicheroODirectorio>`

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 mi_echo.sh`
- `chmod u+rx,g+r-wx,o-wxr mi_echo.sh`

¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas? Crear un directorio y quitar los permisos de ejecución para usuario, grupo y otros. Intentar cambiar al directorio.

- **`chmod 645 <fichero>`**
- **`chmod u+rw,g+r,o+r-w <fichero>`**
- **`mkdir midirectorio`**
- **`chmod 666 midirectorio`**

Ejercicio 4. Escribir un programa que, usando la llamada `open`, cree un fichero con los permisos rw-r—r-x. Comprobar el resultado y las características del fichero con la orden `ls`.

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

int main(int argc, char *argv[]) {
        if(open("miarchivo.txt", O_CREAT) == -1) {
                printf("ERROR: %s\n", strerror(errno));
        } else {
                if(chmod("miarchivo.txt", 0666) == -1) {
                        printf("ERROR: %s\n", strerror(errno));
                }
        }
}
```

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

int main(){
        int filedesc = open("testfile.txt",O_WRONLY | O_CREAT | O_TRUNC, 0645);
    if(filedesc < 0)
        return EXIT_FAILURE;

        return EXIT_SUCCESS;
}
```

`gcc miprograma.c -o miprograma →./miprograma`

Ejercicio 5. Cuando se crea un fichero, los permisos por defecto se derivan de la máscara de usuario (*umask*). El comando `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` y `ls` -r

"no tengan permiso de escritura para el grupo y ningún permiso para otros" → umask 0747

```
root@frontend:~/Descargas# umask 0727
root@frontend:~/Descargas# umask
0727
root@frontend:~/Descargas# touch myfirstfile
root@frontend:~/Descargas# ls -ld myfirstfile
----r----- 1 root root 0 dic  4 17:24 myfirstfile
root@frontend:~/Descargas#
```

Ejercicio 6. Modificar el ejercicio 4 para que, antes de crear el fichero, se fije la máscara igual que en el ejercicio anterior. Una vez creado el fichero, debe restaurarse la máscara original del proceso. Comprobar el resultado con el comando `ls`.

Ejercicio 7. El comando `ls` puede mostrar el inodo con la opción `-i`. El resto de información del inodo puede obtenerse usando el comando `stat`. Consultar las opciones del comando y comprobar su funcionamiento.

Ejercicio 8. Escribir un programa que emule el comportamiento del comando `stat` y muestre:

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

Se ha creado un programa ./stat que acepta como argumento el nombre del fichero e indica sus datos.

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

int main(int argc, char *argv[]) {
        if(argc != 2)
                exit(EXIT_FAILURE);

        const char* filename = argv[1];
        struct stat buf;
```

```
        int res;
        char buffer [80];
        struct tm* timeinfo;
        char string [24] = "Fichero";
        res = lstat(filename, &buf);

        if(res == -1) {
                printf("Error:", strerror(errno));
        }

printf("Major number: %i Minor number: %i\n", major(buf.st_rdev),
minor(buf.st_rdev));
        printf("Numero de i-nodo: %i\n", buf.st_ino);
        if(S_ISBLK(buf.st_mode) != 0) strcpy(string, "Fichero de bloques");
        if(S_ISCHR(buf.st_mode) != 0) strcpy(string, "Fichero de caracteres");
        if(S_ISDIR(buf.st_mode) != 0) strcpy(string, "Directorio");
        if(S_ISFIFO(buf.st_mode) != 0) strcpy(string, "Fichero FIFO");
        if(S_ISLNK(buf.st_mode) != 0) strcpy(string, "Enlace simbolico");
        if(S_ISREG(buf.st_mode) != 0) strcpy(string, "Fichero regular");
        printf("Tipo archivo: %s\n", string);

        timeinfo = localtime(&buf.st_atime);
        strftime(buffer, sizeof(buffer), "Ultima hora acceso: %T", timeinfo);
        puts(buffer);
        return 0;
}
```

Ejercicio 9. Los enlaces se crean con la orden `ln`:

- La opción `-s` crea un enlace simbólico. Hacer 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? En el caso de los enlaces simbólicos ¿qué sucede si se borra el enlace? ¿y si se borra el fichero original?

Ejercicio 10. Las llamadas `link` y `symlink` 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 crear un enlace simbólico y rígido (mismo nombre terminado en `.sym` y `.hard`). Comprobar el resultado con la orden `ls`.

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

```c
#define MAX_SIZE 100

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

        char *path = argv[1];
        char *tmppath = strdup(path);
        char *tmppath2 = strdup(path);

        const char *newpath = strcat(tmppath, ".hard");
        const char *newpath2 = strcat(tmppath2, ".sym");

        struct stat buf;
        int res;
        res = lstat(path, &buf);

        if(res == -1) {
                printf("Error:\n", strerror(errno));
        } else {
           if(S_ISREG(buf.st_mode) != 0) { // Si es un fichero regular
              int res2 = link(path, newpath);
                if(res2 == -1)  {
                        printf("error:%s\n", strerror(errno));
                } else {
                        printf("Enlace rígido creado\n");
                }

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

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 y dup2.

Ejercicio 1. Escribir un programa que redirija la salida estándar a un fichero, cuya ruta se pasa como primer argumento. Probar escribiendo varias cadenas en la salida estándar.

Ejercicio 2. 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 <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>

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

        int filed = open(argv[1], O_RDWR | O_CREAT | O_TRUNC, 0645);

        dup2(filed, 1); // Salida estandar (1)
        dup2(filed, 2); // Salida de error (2)

        printf("Mensajes salida estandar: prueba1 prueba2\n");
        printf("otralinea\n");

        perror("Mensajes salida error: perror");

        close(filed);

    return 0;
}
```

Si.

`ls > mydirlist 2>&1` will work because it directs both stdout and stderr to the file mydirlist. "send stdout to the file mydirlist and duplicate stderr onto that stdout I've set up"

`ls 2>&1 > mydirlist` directs only stdout, and not stderr, to file mydirlist, because stderr was made a copy of stdout before stdout was redirected to mydirlist. "copy stderr onto stdout and send stdout to mydirlist"

***Ejercicio 3* (Opcional).** La llamada `fcntl()` también permite duplicar descriptores de fichero. Estudiar qué opciones hay que usar para que `fcntl` duplique los descriptores.

Cerrojos de ficheros

El sistema de ficheros ofrece un sistema de bloqueos consultivo. Estas funciones se pueden acceder mediante `flock` o `fcntl`. En esta sección usaremos únicamente `fcntl()`.

Ejercicio 1. El estado y cerrojos de fichero en uso en el sistema se pueden consultar en el archivo `/proc/locks`. Estudiar el contenido de este archivo.

Ejercicio 1.5 Escribir un programa que bloquee un fichero (indicado por su segundo argumento) creando un cerrojo sobre el mismo:

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main (int argc, char* argv[]) {
        char* file = argv[1];
        int fd;
        struct flock lock;

        printf("Trying to open and lock %s...\n", file);
        fd = open(file, O_WRONLY);
        memset(&lock, 0, sizeof(lock));
        lock.l_type = F_WRLCK; // Write lock type
        fcntl(fd, F_SETLKW, &lock); // Place the lock on the file

        printf("File opened and locked! hit enter to unlock... "); // Until
enter is pressed the file is locked
        getchar();

        printf("Unlocking...\n");
        lock.l_type = F_UNLCK; // Unlock type
        fcntl(fd, F_SETLKW, &lock); // Unlock the file

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

Para probar su ejecución usar ./programa archivobloqueado y sin pulsar enter intentar ejecutarlo sobre el mismo fichero en otro terminal, observar que no es posible hacerlo hasta que se desbloquee el primer cerrojo.

Ejercicio 2. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero. El proceso mostrará el estado del cerrojo (bloqueado, desbloqueado). Además:

- Si está desbloqueado, fijará un cerrojo de escritura y escribirá la hora actual. Después suspenderá su ejecución durante 30 segundos (función `sleep`) y a continuación liberará el cerrojo.

- Si el cerrojo está bloqueado terminará el proceso.

El programa no deberá modificar el contenido del fichero si no tiene el cerrojo.

```
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>

int main(int argc, char *argv[]) {
        if(argc != 2) {
                printf("Vuelve a ejecutar el programa especificando un
fichero\n");
                exit(EXIT_FAILURE);
        }
        struct flock lock;
        time_t rawtime;
        struct tm* timeinfo;
        char buffer [9];
        int fd;

        printf("Comprobando el estado del cerrojo...\n");
        fd = open(argv[1], O_WRONLY);
        memset(&lock, 0, sizeof(lock));
        fcntl(fd, F_GETLK, &lock); // Trying to get the lock on the file

        if(lock.l_type != F_UNLCK) {
                printf("Cerrojo bloqueado\n");
                exit(EXIT_SUCCESS);
        } else {
                printf("Cerrojo desbloqueado, tratando adquirir el
cerrojo...\n");
                lock.l_type = F_WRLCK; // Write lock type
                memset(&lock, 0, sizeof(lock));
                fcntl(fd, F_SETLKW, &lock); // Locked
                // Writing the actual hour
                time(&rawtime);
                timeinfo = localtime(&rawtime);
                strftime(buffer, sizeof(buffer), "%T", timeinfo);
                write(fd, buffer, 8);

       printf("Cerrojo adquirido, escribiendo la hora actual en el
fichero...\n");
                //getchar();
                sleep(5);

                printf("Liberando cerrojo...\n");
```

```
                lock.l_type = F_UNLCK;
                fcntl(fd, F_SETLKW, &lock);
        }
        close(fd);
        return 0;
}
```

Usar el programa anterior (***ejercicio 1.5***) para su comprobar su correcto funcionamiento.

Ejercicio 3 **(Opcional).** El comando `flock` proporciona funcionalidad de cerrojos 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 el nombre seguido de '->`<fichero al que apunta>`'. Usar la función `readlink(2)` y dimensionar adecuadamente el buffer de la función.
    - Si el fichero es ejecutable escribirá el nombre seguido del carácter '*' PENDIENTE
- Al final de la lista el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.

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

int main(int argc, char *argv[]) {
        if(argc != 2) {
                printf("Vuelva a ejecutar el programa especificando la ruta a
un directorio\n");
                exit(EXIT_FAILURE);
        }

        DIR *dir;
        struct dirent *dirp;
        char * dirname = argv[1];
        char * path;
```

```
        struct stat buf;
        char buffer[1024];
        int res;
        int tam = 0;

        res = lstat(dirname, &buf);
        if(res == -1) {
                perror("Error");
                exit(EXIT_FAILURE);
        } else if(!S_ISDIR(buf.st_mode)) { // Si la ruta no acaba en un
directorio...
                printf("La ruta especificada debe apuntar a un directorio\n");
                exit(EXIT_FAILURE);
        }

        dir = opendir(argv[1]);
        // Recorremos las entradas del directorio:
        while((dirp = readdir(dir)) != NULL) {
                if(dirp->d_type == DT_REG) {
                        printf("%s\n", dirp->d_name);
                        tam = tam + buf.st_size;
                }
                if(dirp->d_type == DT_DIR) {
                        printf("%s/\n", dirp->d_name);
                        tam = tam + buf.st_size;
                }
                if(dirp->d_type == DT_LNK) {
                        path = strcat(dirname, "/");
                        path = strcat(path, dirp->d_name);
                        ssize_t tambuf = readlink(path, buffer, sizeof(buffer)
- 1);
                        if(tambuf != -1) {
                                buffer[tambuf] = '\0';
                                printf("%s -> %s\n", dirp->d_name, buffer);
                                tam = tam + buf.st_size;
                        } else perror("Error");
                }
                if(dirp->d_type == DT_BLK) {
                        printf("%s*\n", dirp->d_name);
                        tam = tam + buf.st_size;
                }
        }
        closedir(dir);
        tam = tam / 1024; // Bytes a KiloBytes
        printf("Tamaño total de los archivos del directorio (KB): %i\n", tam);

        return 0;
}
```