# 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: todos los directorios (- - all)*

*-l: usar un formato de lista larga*

*-d: lista de los directorios, pero no su contenido*

*-h: con -l y -s, te muestra los tamaños como 1K 234M 2G etc.*

*-i: indica el número de índice de cada archivo*

*-R: lista los subdirectorios de forma recursiva*

*-1: muestra un archivo por línea*

*-F: añade un indicador a cada entrada (*

*- -color: colorear la salida*

***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`).
***Comando:*** *ls -ld*
***Salida:*** *drwxr-xr-x 37 usuario_vms users 4096 nov 21 10:58 .*

***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 -).

***Comando:***
*vim fichero*
*chmod 540 fichero*
*ls -ld fichero*
***Salida:***
*-r-xr----- 1 usuario_vms users 24 nov 21 11:09 fichero*

***Comando:***
*chmod u=rx,g=r,o= fichero*
*s -ld fichero*
***Salida:***
*-r-xr----- 1 usuario_vms users 24 nov 21 11:09 fichero*

**¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas?**
*El permiso* `rw-r--r-x` *se escribe como 110100101 como número binario, asignando '1' a cada permiso concedido y '0' en caso contrario. Si convertimos este número a octal, tenemos el 645. Por tanto, de la primera forma nos quedaría el comando* `chmod 645 fichero`. *Por otro lado, la otra forma quedaría como* `chmod u=rw,g=r,o=rx fichero`.

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

***Comando:***
*mkdir Directorio*
*chmod -x Directorio*
*ls -ld Directorio*
***Salida:***
*drw-r--r-- 2 usuario_vms users 4096 nov 21 11:20 Directorio*

***Comando:***
*cd Directorio*
***Salida:***
*bash cd: Directorio: 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)`.

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

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

int main() {

   if (open("ejercicio5", O_CREAT, 0645) != -1) {

       printf("Fichero creado con éxito \n");

   }

   else {

       perror("Error al crear el fichero ");

   }

   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 no tengan ningún permiso para otros. Comprobar el funcionamiento con `touch(1)`, `mkdir(1)` y `ls(1)`.

***Comando:***

*Umask 0027*

*mkdir ejercicio6*

*ls -ld ejercicio6*

***Salida:***

*drwxr-x--- 1 usuario_vms users 12 nov 21 11:46 ejercicio6*

***Comando:***

*touch ejercicio6B*

*ls -ld ejercicio6B*

***Salida:***

*-rw-r----- 1 usuario_vms users 0 nov 21 11:49 ejercicio6B*

***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.

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

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>


int main() {

   umask(0027);

   if (open("ejercicio7", O_CREAT, 0645) != -1) {

       printf("Fichero creado con exito \n");

   }

   else {

       perror("Error al crear el fichero ");

   }




   return 0;

}
```

***Comando :*** *ls -ld ejercicio7*

***Salida :*** *-rw-r----- 1 usuario_vms users 0 nov 21 11:54 ejercicio7*

***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.

***Comando:*** *ls -i*

***Salida:*** *5768589 a.out 5772311 ejercicio6B 5768572 ejerc-pract2.c 5768578 ejercicio6 5772699 ejercicio7 5772713 fichero*

***Comando:*** *stat ejercicio7*

***Salida:***

*Fichero: ejercicio7*

*Tamaño: 0 Bloques: 0 Bloque E/S: 4096 fichero regular vacío*

*Dispositivo: 809h/2057d Nodo-i: 5772699  Enlaces: 1*

*Acceso: (0640/-rw-r-----)  Uid: ( 1565/usuario_vms)  Gid: (  100/  users)*

*Acceso: 2022-11-21 11:54:14.826501986 +0100*

*Modificación: 2022-11-21 11:54:14.826501986 +0100*

*Cambio: 2022-11-21 11:54:14.826501986 +0100*

*Creación: 2022-11-21 11:54:14.826501986 +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`?

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


int main() {
   struct stat s;
   if(stat("ejercicio7", &s) != -1){
       printf("Major: %i \n", major(s.st_dev));
       printf("Minor: %i \n", minor(s.st_dev));
       printf("Nodo-i: %li\n", s.st_ino);
       if (S_ISREG(s.st_mode))
           printf("Es un fichero regular \n");
       else if (S_ISDIR(s.st_mode))
           printf("Es un directorio \n");
       else if (S_ISLNK(s.st_mode))
           printf("Es un enlace simbólico \n");
       printf("Ultimo acceso: %li\n", s.st_atime);
   }
   else{ perror("Error en stat"); }
   return 0;
}
```

***Comando:***

*gcc ejerc-pract2.c*

*./a.out*

***Salida:***

*Major: 8*

*Minor: 9*

*Nodo-i: 5772699*

*Es un fichero regular*

*Ultimo acceso: 1669028054*

**¿Qué diferencia hay entre `st_mtime` y `st_ctime`?**

St_mtime *indica la fecha de la última modificación de los datos del fichero, no del inodo. Por otro lado,* st_ctime *muestra la última modificación del inodo (por ejemplo, el cambio de permisos).*

***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.

  ***Comandos:*** *ln -s ejercicio7 ejercicio10*
  *ln -s ejercicio6 ejercicio10d*
  *ls -l*
  *ls -i*

  ***Salida:***
  *total 28*
  *-rwxr-x--- 1 usuario_vms users 16240 nov 21 12:16 a.out*
  *lrwxrwxrwx 1 usuario_vms users 10 nov 21 12:24 ejercicio10 -> ejercicio7*
  *lrwxrwxrwx 1 usuario_vms users 10 nov 21 12:25 ejercicio10d -> ejercicio6*
  *drwxr-x--- 2 usuario_vms users 4096 nov 21 11:48 ejercicio6*
  *-rw-r----- 1 usuario_vms users 0 nov 21 11:49 ejercicio6B*
  *-rw-r----- 1 usuario_vms users 0 nov 21 11:54 ejercicio7*
  *-rw-r--r-- 1 usuario_vms users 700 nov 21 12:15 ejerc-pract2.c*
  *-rw-r----- 1 usuario_vms users 12 nov 21 11:46 fichero*

  *5768580 a.out 5768578 ejercicio6 5768572 ejerc-pract2.c*
  *5772738 ejercicio10 5772311 ejercicio6B 5772713 fichero*
  *5772758 ejercicio10d 5772699 ejercicio7*

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

  ***Comandos:***

  *ln ejercicio7 ejercicio10h*

*ls -i*

***Salida:***

*5768580 a.out  5772699 ejercicio10h  5772699 ejercicio7*

*5772738 ejercicio10  5768578 ejercicio6  5768572 ejerc-pract2.c*

*5772758 ejercicio10d  5772311 ejercicio6B  5772713 fichero*

***Comando:***

*stat ejercicio7*

***Salida:***

*Fichero: ejercicio7*

*Tamaño: 0  Bloques: 0  Bloque E/S: 4096  fichero regular vacío*

*Dispositivo: 809h/2057d Nodo-i: 5772699* ***Enlaces: 2***

*Acceso: (0640/-rw-r-----)  Uid: ( 1565/usuario_vms)  Gid: ( 100/ users)*

*Acceso: 2022-11-21 11:54:14.826501986 +0100*

*Modificación: 2022-11-21 11:54:14.826501986 +0100*

*Cambio: 2022-11-21 12:28:40.233972982 +0100*

*Creación: 2022-11-21 11:54:14.826501986 +0100*

- ¿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?

*Si se borra un enlace rígido el contador de enlaces decrementa. Si se borra el simbólico, desaparece su i-nodo pero no ocurre nada sobre el original. Si borramos el original, no ocurre nada si hay algún enlace rígido y si no desaparece 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 <stdio.h>
#include <sys/sysmacros.h>
#include <unistd.h>
```

```c
int main(int argc, char** argv) {
   struct stat s;

   if(argc == 1){
       return 0;
   }

   if(stat(argv[1], &s) == -1){
       perror("Error");
   }
   else {
       if (S_ISREG(s.st_mode)) {
           if(link(argv[1], "fichero.hard") != -1){
               printf("Enlace rígido creado");
           }
           else{
               perror("Error");
           }

           if(symlink(argv[1], "fichero.sym") != -1){
               printf("Enlace simbólico creado");
           }
           else{
               perror("Error");
           }
       }
   }

   return 0;
}
```

***Comando :***

*ls -l*

***Salida:***

*total 28*

*-rwxr-x--- 1 usuario_vms users 16160 nov 21 12:42 a.out*

*lrwxrwxrwx 1 usuario_vms users  10 nov 21 12:24 ejercicio10 -> ejercicio7*

*lrwxrwxrwx 1 usuario_vms users  10 nov 21 12:25 ejercicio10d -> ejercicio6*

*drwxr-x--- 2 usuario_vms users 4096 nov 21 11:48 ejercicio6*

*-rw-r----- 1 usuario_vms users  0 nov 21 11:49 ejercicio6B*

*-rw-r----- 3 usuario_vms users  0 nov 21 11:54 ejercicio7*

*-rw-r--r-- 1 usuario_vms users 1070 nov 21 12:42 ejerc-pract2.c*

*-rw-r----- 1 usuario_vms users  12 nov 21 11:46 fichero*

*-rw-r----- 3 usuario_vms users  0 nov 21 11:54 fichero.hard*

*lrwxrwxrwx 1 usuario_vms users  10 nov 21 12:43 fichero.sym -> ejercicio7*

## 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.

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/sysmacros.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char**argv) {
  if(argc < 2){
       printf("Argumentos insuficientes");
       exit(1);
  }
   int file = open(argv[1], O_CREAT | O_RDWR, 0666);
  if (file == -1) {
       perror("open argv[1]");
       exit(1);
  }
```

```
   dup2(file,1);
   printf("Ejercicio 12, probar dup2");

  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 <stdio.h>

#include <sys/sysmacros.h>

#include <stdlib.h>

#include <unistd.h>

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

   if(argc < 2){

       printf("Argumentos insuficientes");

       exit(1);

   }

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

   if (file == -1) {

       perror("open argv[1]");

       exit(1);

   }

   dup2(file,1); dup2(file, 2);

   printf("Ejercicio 13, probar dup2\n");

   fprintf(stderr, "Redireccionamos el error\n");

  return 0;

}
```

**¿Por qué `ls > dirlist 2>&1` es diferente a `ls 2>&1 > dirlist`?**

*Para `ls > dirlist 2>&1` iindica que el resultado va al archivo dirlist y `2>&1` indica que los errores se redirigen a la salida estándar, a 1. Como 1 está redirigido a su vez a dirlist, 2 también.*

*En `ls 2>&1>dirlist` se redirecciona la salida de error a la estándar luego al fichero.*

## 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 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.

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

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <sys/sysmacros.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>



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

   int file = open("ejercicio15", O_CREAT | O_RDWR, 0666);

   int bloq = lockf(file, F_TEST,0);

   if(bloq == 0){ //region desbloqueada

       lockf(file,F_LOCK,0);

       time_t t = time(NULL);

       struct tm* hora = localtime(&t);

       printf("Son las %i:%i \n", hora->tm_hour, hora->tm_min);

       sleep(10);
```

```
        lockf(file, F_ULOCK, 0);

        t = time(NULL);

        hora = localtime(&t);

        printf("Diez segundos más tarde);

    }

    else { perror("Fichero bloqueado \n"); }

    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 <stdio.h>
#include <sys/sysmacros.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <dirent.h>
#include <string.h>

int main(int argc, char**argv) {
    if (argc<2){
        printf("Argumentos insuficientes \n");
        exit(1);
    }

    DIR* path = opendir(argv[1]);
```

```
if (path == NULL){
    perror("opendir");
    exit(1);
}

struct dirent* dir = readdir(path);
long int tam = 0;
while (dir != NULL) {
    char* copy_path;
    strcpy(copy_path, argv[1]);
    char* r = strcat(copy_path, "/");
    r = strcat(copy_path, dir->d_name);

    if (dir->d_type == DT_REG) {
        struct stat sb;
        if (lstat(r, &sb) == -1) {
            perror("lstat");
            exit(1);
        }
        else printf("%s*\n", dir->d_name);
        tam += sb.st_size;
    }

    else if (dir->d_type == DT_DIR) {
        printf("%s/ \n", dir->d_name);
    }

    else if (dir->d_type == DT_LNK) {
        struct stat sb;
        char *buf;
        ssize_t nbytes, bufsize;
        if (lstat(r, &sb) == -1) {
            perror("lstat");
            exit(1);
        }
        tam += sb.st_size;
        bufsize = sb.st_size+1;

        if (sb.st_size == 0) {
            bufsize = PATH_MAX;
        }

        buf = malloc(bufsize);
        if (buf == NULL) {
            perror("malloc");
            exit(1);
        }

        nbytes=readlink(r, buf, bufsize);
        if (nbytes == -1) {
            perror("readlink");
```

```
                exit(1);
            }

            printf("%s -> %s \n", dir->d_name, buf);
            free(buf);
        }

        dir = readdir(path);
    }

    closedir(path);
    printf("Tamaño total en bytes: %li \n", tam);


    return 0;
}
```