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](#_14a3ftqman5y)

[Creación y atributos de ficheros](#_rk750rh0zbua)

[Redirecciones y duplicación de descriptores](#_6kob7pg9jkqo)

[Cerrojos de ficheros](#_l4wunudw25f)

[Directorios](#_7ry6brgd9914)

# 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, incluidos los ocultos que comienzan con un punto (.) en el nombre.**

**-l: Muestra una lista detallada que incluye información sobre permisos, propietario, grupo, tamaño, fecha de modificación y nombre del archivo.**

**-d: Muestra solo el nombre del directorio, no su contenido.**

**-h: Muestra tamaños de archivo legibles por humanos, con unidades como KB, MB, GB, etc.**

**-h: Muestra tamaños de archivo legibles por humanos, con unidades como KB, MB, GB, etc.**

**-R: Muestra el contenido de los directorios de forma recursiva, es decir, también muestra el contenido de los subdirectorios.**

**-1: Muestra la lista de archivos o directorios en una sola columna.**

**-F: Agrega un indicador al final de cada entrada para indicar su tipo (por ejemplo, "/" para directorios, "\*" para archivos ejecutables).**

**--color: Colorea la salida para mejorar la legibilidad. Los diferentes tipos de archivos pueden tener diferentes colores.**

***Ejercicio 2.***El *modo* de un fichero es <tipo><rwx\_usuario><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,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 -).

**chmod 644 fichero**

**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.

**Sudo mkdir <nombre\_directorio>**

**Chmod a-x <nombre\_directorio>**

**Cd <nombre\_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).

#include <fcntl.h>

#include <stdlib.h>

int main(){

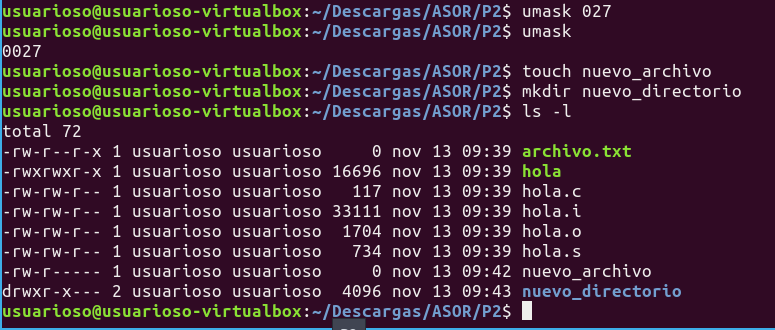
int arch = open("archivo.txt",O\_CREAT,0645);

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).

**Con umask los numeros significan los permisos a quitar**



***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 <fcntl.h>

#include <stdlib.h>

#include <sys/stat.h>

int main(){

umask(027);

int arch = open("archivo2.txt",O\_CREAT,0645);

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.

***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 <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <time.h>

void mostrar\_estadisticas(char \*filename) {

struct stat stat\_info;

// Obtener información del archivo

if (stat(filename, &stat\_info) == -1) {

perror("Error al obtener información del archivo");

exit(EXIT\_FAILURE);

}

// Mostrar información

printf("Número major y minor del dispositivo: %ld, %ld\n", (long)major(stat\_info.st\_dev), (long)minor(stat\_info.st\_dev));

printf("Número de inodo del fichero: %ld\n", (long)stat\_info.st\_ino);

// Determinar el tipo de archivo

if (S\_ISDIR(stat\_info.st\_mode)) {

printf("Tipo de fichero: Directorio\n");

} else if (S\_ISLNK(stat\_info.st\_mode)) {

printf("Tipo de fichero: Enlace simbólico\n");

} else {

printf("Tipo de fichero: Fichero ordinario\n");

}

// Obtener y mostrar la hora de acceso del archivo

printf("Hora de acceso: %s", ctime(&stat\_info.st\_atime));

// La diferencia entre st\_mtime y st\_ctime:

// st\_mtime: Hora de la última modificación del contenido del archivo.

// st\_ctime: Hora de la última modificación de los metadatos del archivo (como permisos o propietario).

printf("Hora de la última modificación del contenido: %s", ctime(&stat\_info.st\_mtime));

printf("Hora de la última modificación de los metadatos: %s", ctime(&stat\_info.st\_ctime));

}

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

if (argc != 2) {

fprintf(stderr, "Uso: %s <nombre\_del\_archivo>\n", argv[0]);

exit(EXIT\_FAILURE);

}

mostrar\_estadisticas(argv[1]);

return 0;

}

***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.

**Ln -s <loquequierascopiar>**

* 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?

**Si se borra uno de los enlaces rígidos, el archivo permanece en el sistema de archivos siempre que haya al menos un enlace apuntando a él. El espacio en disco no se libera hasta que el último enlace es eliminado.**

**Si se borra uno de los enlaces simbólicos, no afecta al archivo original. El archivo original y otros enlaces siguen existiendo.**

**Si se borra el fichero original, los enlaces rígidos aún mantienen el contenido del archivo en el sistema de archivos, mientras que los enlaces simbólicos apuntarán a un destino inexistente.**

***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 <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

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

if (argc != 2) {

fprintf(stderr, "Uso: %s <ruta\_al\_fichero>\n", argv[0]);

exit(EXIT\_FAILURE);

}

const char \*archivo\_original = argv[1];

struct stat sb;

char enlace\_simbolico[256];

char enlace\_rigido[256];

stat(archivo\_original, &sb);

if(S\_ISREG(sb.st\_mode)){

// Crear nombres para los enlaces simbólico y rígido

snprintf(enlace\_simbolico, sizeof(enlace\_simbolico), "%s.sym", archivo\_original);

snprintf(enlace\_rigido, sizeof(enlace\_rigido), "%s.hard", archivo\_original);

// Crear enlace simbólico

if (symlink(archivo\_original, enlace\_simbolico) == -1) {

perror("Error al crear enlace simbólico");

exit(EXIT\_FAILURE);

}

// Crear enlace rígido

if (link(archivo\_original, enlace\_rigido) == -1) {

perror("Error al crear enlace rígido");

exit(EXIT\_FAILURE);

}

}

printf("Enlaces creados: %s y %s\n", enlace\_simbolico, enlace\_rigido);

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 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 <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

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

// Verificar que se proporciona la ruta del archivo

if (argc != 2) {

fprintf(stderr, "Uso: %s <ruta\_del\_fichero>\n", argv[0]);

exit(EXIT\_FAILURE);

}

// Abrir el archivo para escribir, creándolo si no existe

int fd = open(argv[1], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd == -1) {

perror("Error al abrir el archivo");

exit(EXIT\_FAILURE);

}

// Redirigir la salida estándar al archivo

if (dup2(fd, STDOUT\_FILENO) == -1) {

perror("Error al redirigir la salida estándar");

exit(EXIT\_FAILURE);

}

// Cerrar el descriptor de archivo duplicado

close(fd);

// Ahora, la salida estándar está redirigida al archivo

// Escribir varias cadenas en la salida estándar (ahora redirigida al archivo)

printf("Esta cadena se redirige al archivo.\n");

printf("Otra cadena en el archivo.\n");

// Puedes continuar escribiendo más contenido aquí

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?

**Hay que modificar este if**

if (dup2(fd, STDOUT\_FILENO) == -1 || dup2(fd, STDERR\_FILENO) == -1)

# 

# 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 WRITE 100 00:12:345 1234 0123456789abcdef123456789abcdef 1**

**Donde los campos significan:**

* **1: Identificador del cerrojo.**
* **POSIX: Tipo de cerrojo.**
* **ADVISORY: Modo del cerrojo.**
* **WRITE: Tipo de acceso.**
* **100: PID del proceso que posee el cerrojo.**
* **00:12:345: Números major y minor del dispositivo del archivo.**
* **1234: Número de inodo del archivo.**
* **0123456789abcdef123456789abcdef: ID de bloqueo único.**
* **1: Contador de referencias al cerrojo.**

***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.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <time.h>

int main() {

const char \*nombre\_fichero = "archivo\_lockf.txt";

int fd;

// Abrir o crear el archivo

fd = open(nombre\_fichero, O\_RDWR | O\_CREAT, 0644);

if (fd == -1) {

perror("Error al abrir o crear el archivo");

exit(EXIT\_FAILURE);

}

// Intentar bloquear el archivo

if (lockf(fd, F\_TLOCK, 0) == 0) {

// Bloqueo exitoso

time\_t ahora = time(NULL);

printf("Bloqueo exitoso a las %s", ctime(&ahora));

// Suspender la ejecución durante 10 segundos

sleep(10);

// Desbloquear el archivo

if (lockf(fd, F\_ULOCK, 0) == 0) {

printf("Desbloqueo exitoso.\n");

} else {

perror("Error al desbloquear el archivo");

}

// Suspender la ejecución durante otros 10 segundos

sleep(10);

} else {

// Error al bloquear el archivo

perror("Error al bloquear el archivo");

}

// Cerrar el archivo

if (close(fd) == -1) {

perror("Error al cerrar el archivo");

exit(EXIT\_FAILURE);

}

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.

* OPCIONES:
  + -x o --exclusive: Adquiere un cerrojo exclusivo (similar a LOCK\_EX).
  + -n o --nonblock: Intenta adquirir el cerrojo, pero no espera si no está disponible (similar a LOCK\_NB).
  + -u o --unlock: Libera el cerrojo.
* ARCHIVO\_COMANDO: El archivo al que se asocia el cerrojo.
* ARGUMENTOS\_COMANDO: El comando que se ejecutará bajo el cerrojo.

# 

# 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 <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

void mostrar\_contenido\_directorio(const char \*ruta\_directorio);

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

if (argc != 2) {

fprintf(stderr, "Uso: %s <ruta\_al\_directorio>\n", argv[0]);

exit(EXIT\_FAILURE);

}

const char \*ruta\_directorio = argv[1];

// Comprobar la existencia y accesibilidad del directorio

struct stat st;

if (stat(ruta\_directorio, &st) == -1) {

perror("Error al obtener información del directorio");

exit(EXIT\_FAILURE);

}

if (!S\_ISDIR(st.st\_mode)) {

fprintf(stderr, "%s no es un directorio válido.\n", ruta\_directorio);

exit(EXIT\_FAILURE);

}

mostrar\_contenido\_directorio(ruta\_directorio);

return 0;

}

void mostrar\_contenido\_directorio(const char \*ruta\_directorio) {

DIR \*dir = opendir(ruta\_directorio);

if (!dir) {

perror("Error al abrir el directorio");

exit(EXIT\_FAILURE);

}

struct dirent \*entry;

struct stat file\_info;

off\_t total\_size = 0;

printf("Contenido del directorio %s:\n", ruta\_directorio);

while ((entry = readdir(dir)) != NULL) {

// Ignorar las entradas "." y ".."

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) {

continue;

}

// Construir la ruta completa del archivo

char path[PATH\_MAX];

snprintf(path, sizeof(path), "%s/%s", ruta\_directorio, entry->d\_name);

// Obtener información sobre el archivo

if (stat(path, &file\_info) == -1) {

perror("Error al obtener información del archivo");

exit(EXIT\_FAILURE);

}

// Mostrar el nombre del archivo

printf("%s", entry->d\_name);

// Mostrar información adicional según el tipo de archivo

if (S\_ISREG(file\_info.st\_mode) && (file\_info.st\_mode & S\_IXUSR || file\_info.st\_mode & S\_IXGRP || file\_info.st\_mode & S\_IXOTH)) {

printf("\*"); // Tiene permisos de ejecución

} else if (S\_ISDIR(file\_info.st\_mode)) {

printf("/"); // Es un directorio

} else if (S\_ISLNK(file\_info.st\_mode)) {

// Es un enlace simbólico, mostrar el destino

char link\_target[PATH\_MAX];

ssize\_t len = readlink(path, link\_target, sizeof(link\_target) - 1);

if (len != -1) {

link\_target[len] = '\0';

printf(" -> %s", link\_target);

}

}

printf("\n");

// Calcular el tamaño total de los archivos (no directorios)

if (S\_ISREG(file\_info.st\_mode)) {

total\_size += file\_info.st\_size;

}

}

printf("Tamaño total de los archivos: %.2f KB\n", (double)total\_size / 1024.0);

closedir(dir);

}

WUOLAH

#include <sys/types.h>

#include <dirent.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

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

if (argc < 2) {

printf("No se han introducido argumentos suficientes para la función.\n");

return -1;

}

else if (argc > 2) {

printf("Se han introducido demasiados argumentos para la función.\n");

return -1;

}

else {

DIR \* dir = opendir(ruta);

if (dir != NULL) {

struct dirent \*d;

short len = 0;

while ((d = readdir(dir)) != NULL) {

printf("%s ", d->d\_name);

char ruta[100]; strcat(strcat(strcpy(ruta, argv[1]), "/"), d->d\_name);

switch (d->d\_type) {

case DT\_REG:

printf("");

struct stat buf1;

int rs1 = stat(ruta, &buf1);

if (rs1 == 0) {

if (buf1.st\_mode & S\_IXUSR && buf1.st\_mode & S\_IXGRP && buf1.st\_mode & S\_IXOTH) {

printf("\*");

}

}

else

{

perror("stat()");

}

break;

case DT\_LNK:

printf("");

char buf2[200];

int rs2= readlink(ruta, buf2, 200);

if

(rs2 != -1) {

printf("-> %s", buf2);

}

else

{

perror("readlink()");

}

break;

case DT\_DIR:

printf("/");

break;

}

len +=d->d\_reclen;

printf("\n");

}

printf("Total length: %d\n", len);

closedir(dir);

return 1;

}

else

{

perror("dir()");

Return -1;

}

}

}