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)

[Proyecto: Comando ls extendido](#_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.

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.

|  |
| --- |
| -a no ignore las entradas que comienzan con…  -i imprime el número de índice de cada archivo  -h con -l y -s , tamaños de impresión como 1K 234M 2G, etc.  -d enumerar los directorios en sí mismos, no su contenido  -1 lista un archivo por línea. Evite '\ n' con -q o -b  -R enumerar subdirectorios de forma recursiva  -F añadir indicador (uno de \* / => @ |) a las entradas  --color colorear la salida; CUÁNDO puede ser 'siempre' (predeterminado si  omitido), 'auto' o 'nunca'; más información a continuación  -l usa un formato de listado extenso/detallado. |

***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).

|  |
| --- |
| [cursoredes@localhost Documents]$ ls  Prac1 Prac2  [cursoredes@localhost Documents]$ ls -ld Prac2  drwxrwxr-x 2 cursoredes cursoredes 6 Dec 18 12:46 Prac2 |

***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  $ chmod u+rw, g+r-wx, o+rx |

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

|  |
| --- |
| [cursoredes@localhost Prac2]$ mkdir Dir  [cursoredes@localhost Prac2]$ ls  Dir  [cursoredes@localhost Prac2]$ ls -l  total 0  drwxrwxr-x 2 cursoredes cursoredes 6 Dec 18 13:14 Dir  [cursoredes@localhost Prac2]$ sudo chmod -x Dir/  [cursoredes@localhost Prac2]$ cd Dir/  bash: cd: Dir/: 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.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  //open libraries  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  int main(){  int fd = open("fichero",O\_CREAT|O\_TRUNC|O\_RDWR,0645);    if( fd == -1){  printf("Error al abrir el fichero \n");  return -1;  }  close(fd);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej5.c -o ej5  [cursoredes@localhost Prac2]$ ./ej5  [cursoredes@localhost Prac2]$ ls -l  total 16  -rwxrwxr-x 1 cursoredes cursoredes 9560 Dec 18 13:28 ej5  -rw-rw-r-- 1 cursoredes cursoredes 298 Dec 18 13:28 ej5.c  **-rw-r--r-x 1 cursoredes cursoredes 0 Dec 18 13:28 fichero** |

***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.

|  |
| --- |
| [cursoredes@localhost Prac2]$ umask 027  [cursoredes@localhost Prac2]$ touch file  [cursoredes@localhost Prac2]$ mkdir dir  [cursoredes@localhost Prac2]$ ls -l  total 4  **drwxr-x--- 2 cursoredes cursoredes 6 Dec 18 13:39 dir**  -rw-rw-r-- 1 cursoredes cursoredes 298 Dec 18 13:28 ej5.c  **-rw-r----- 1 cursoredes cursoredes 0 Dec 18 13:39 file** |

***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 <stdlib.h>  #include <string.h>  //open libraries  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  int main(){  umask(027);  int fd = open("fichero",O\_CREAT|O\_TRUNC|O\_RDWR,0645);    if( fd == -1){  printf("Error al abrir el fichero \n");  return -1;  }  close(fd);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej7.c -o ej7  [cursoredes@localhost Prac2]$ ./ej7  [cursoredes@localhost Prac2]$ ls -l  total 20  -rw-rw-r-- 1 cursoredes cursoredes 298 Dec 18 13:28 ej5.c  -rwxr-x--- 1 cursoredes cursoredes 9616 Dec 18 13:51 ej7  -rw-rw-r-- 1 cursoredes cursoredes 311 Dec 18 13:51 ej7.c  **-rw-r----- 1 cursoredes cursoredes 0 Dec 18 13:51 fichero** |

***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.

***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 <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <time.h>  #include <sys/stat.h>  int main(int argc, char \*\*argv) {  struct stat s;  lstat(argv[1], &s);  printf("Major: %d\n", major(s.st\_dev));  printf("Minor: %d\n", minor(s.st\_dev));  printf("I-node: %ld\n", (long) s.st\_ino);  printf("Tipo de fichero: ");  if (S\_ISDIR(s.st\_mode)) {  printf("directorio\n");  }  else if (S\_ISLNK(s.st\_mode)) {  printf("enlace simbólico\n");  }  else if (S\_ISREG(s.st\_mode)) {  printf("fichero ordinario\n");  }  printf("La hora en la que se accedió el fichero por última vez: %s\n", ctime(&s.st\_ctime));  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej9.c -o ej9  [cursoredes@localhost Prac2]$ ./ej9 fichero  Major: 253  Minor: 0  I-node: 33807593  Tipo de fichero: fichero ordinario  La hora en la que se accedió el fichero por última vez: Fri Dec 18 15:51:33 2020  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?

|  |
| --- |
| * [cursoredes@localhost Prac2]$ touch fichero   [cursoredes@localhost Prac2]$ mkdir directorio  [cursoredes@localhost Prac2]$ ln -s fichero ficheroSym  [cursoredes@localhost Prac2]$ ln -s directorio/ dirSym  [cursoredes@localhost Prac2]$ ls -l  total 16  drwxrwxr-x 2 cursoredes cursoredes 6 Dec 18 16:14 directorio  lrwxrwxrwx 1 cursoredes cursoredes 11 Dec 18 16:15 dirSym -> directorio/  -rw-rw-r-- 1 cursoredes cursoredes 10 Dec 18 16:14 fichero  lrwxrwxrwx 1 cursoredes cursoredes 7 Dec 18 16:15 ficheroSym -> fichero  [cursoredes@localhost Prac2]$ ls -i  17153618 directorio 33807443 ficheroSym  34213108 dirSym 33807593 fichero   * [cursoredes@localhost Prac2]$ touch fichero2   [cursoredes@localhost Prac2]$ mkdir directorio2  [cursoredes@localhost Prac2]$ ln fichero2 ficheroSym2  [cursoredes@localhost Prac2]$ ln directorio2/ dirSym2  ln: ‘directorio2/’: hard link not allowed for directory  [cursoredes@localhost Prac2]$ stat fichero2  File: ‘fichero2’  Size: 0 Blocks: 0 IO Block: 4096 regular empty file  Device: fd00h/64768d Inode: 36224451 Links: 3  Access: (0664/-rw-rw-r--) Uid: ( 1000/cursoredes) Gid: ( 1000/cursoredes)  Access: 2020-12-18 16:19:29.999485874 +0100  Modify: 2020-12-18 16:19:29.999485874 +0100  Change: 2020-12-18 16:20:55.964532316 +0100  Birth: -  [cursoredes@localhost Prac2]$ stat ficheroSym2  File: ‘ficheroSym2’  Size: 0 Blocks: 0 IO Block: 4096 regular empty file  Device: fd00h/64768d Inode: 36224451 Links: 3  Access: (0664/-rw-rw-r--) Uid: ( 1000/cursoredes) Gid: ( 1000/cursoredes)  Access: 2020-12-18 16:19:29.999485874 +0100  Modify: 2020-12-18 16:19:29.999485874 +0100  Change: 2020-12-18 16:20:55.964532316 +0100  Birth: -  (No se puede crear hard link para directorios)   * ¿Qué sucede cuando se borra uno de los enlaces rígidos? → No pasa nada se sigue pudiendo acceder al archivo original. * ¿Qué sucede si se borra uno de los enlaces simbólicos? → No pasa nada se sigue pudiendo acceder al archivo original. * ¿Y si se borra el fichero original? → Se corrompe 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.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/stat.h>  #include <unistd.h>  int main(int argc, char \*\*argv) {  struct stat s;  stat(argv[1], &s);  char\* hard = malloc(sizeof(char)\*(strlen(argv[1])));  char\* sym = malloc(sizeof(char)\*(strlen(argv[1])));  strcpy(hard, argv[1]);  strcpy(sym, argv[1]);  hard = strcat(hard, ".hard");  sym = strcat(sym, ".sym");  if((s.st\_mode & S\_IFMT) == S\_IFREG) {  link(argv[1],hard);  symlink(argv[1],sym);  }  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej11.c -o ej11  [cursoredes@localhost Prac2]$ ./ej11 fichero  [cursoredes@localhost Prac2]$ ls -li  total 28  33974754 -rw-rw-r-- 3 cursoredes cursoredes 0 Dec 18 16:40 fichero  33974754 -rw-rw-r-- 3 cursoredes cursoredes 0 Dec 18 16:40 fichero.hard  33974762 lrwxrwxrwx 1 cursoredes cursoredes 7 Dec 18 16:46 fichero.sym -> fichero |

# 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 <stdlib.h>  #include <string.h>  //open libraries  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  //dup2  #include <unistd.h>  int main(int argc, char \*\*argv){  int fd = open(argv[1],O\_CREAT|O\_TRUNC|O\_RDWR,0645);    if( fd == -1){  printf("Error al abrir el fichero \n");  return -1;  }  dup2(fd, 1);  printf("Se ha redirigido a la salida estándar a un fichero llamado %s\n", argv[1]);  close(fd);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej12.c -o ej12  [cursoredes@localhost Prac2]$ ./ej12 file  [cursoredes@localhost Prac2]$ cat file  Se ha redirigido a la salida estándar a un fichero cuya ruta es file |

***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”?

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  //open libraries  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  //dup2  #include <unistd.h>  int main(int argc, char \*\*argv){  int fd = open(argv[1],O\_CREAT|O\_TRUNC|O\_RDWR,0645);    if( fd == -1){  printf("Error al abrir el fichero \n");  return -1;  }  dup2(fd, 2);  printf("Se ha redirigido a la salida estándar de nuevo\n");  dup2(fd, 1);  printf("Se ha redirigido a la salida estándar a un fichero llamado %s\n", argv[1]);  close(fd);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej13.c -o ej13  [cursoredes@localhost Prac2]$ ./ej13 file  Se ha redirigido a la salida estándar de nuevo  [cursoredes@localhost Prac2]$ cat file  Se ha redirigido a la salida estándar a un fichero llamado file  **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.

|  |
| --- |
| [cursoredes@localhost Prac2]$ cat /proc/locks  1: POSIX ADVISORY WRITE 5442 fd:00:684648 0 EOF  2: POSIX ADVISORY WRITE 1578 fd:00:53013121 0 EOF  3: POSIX ADVISORY WRITE 1575 fd:00:53013120 0 EOF  4: POSIX ADVISORY WRITE 1570 fd:00:53012991 0 EOF  5: POSIX ADVISORY WRITE 1560 fd:00:53012987 0 EOF  6: POSIX ADVISORY WRITE 1348 00:13:21318 0 EOF  7: FLOCK ADVISORY WRITE 1158 fd:00:17458434 0 EOF  8: FLOCK ADVISORY WRITE 1158 fd:00:30831 0 EOF  9: POSIX ADVISORY WRITE 1078 00:13:20013 0 EOF  10: FLOCK ADVISORY WRITE 1077 00:13:19907 0 EOF  11: POSIX ADVISORY WRITE 790 00:13:18719 0 EOF  13: POSIX ADVISORY WRITE 483 00:13:12805 0 EOF |

***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.

|  |
| --- |
| #include<stdlib.h>  #include<stdio.h>  #include<string.h>  #include<time.h>  #include <unistd.h>  #include <fcntl.h>  int main(int argc, char \*\*argv){  char buf[100];  time\_t t = time(NULL);  struct tm \*tm = localtime(&t);  int fd = open(argv[1], O\_CREAT|O\_TRUNC|O\_RDWR, 0666);  int rc = lockf(fd, F\_TLOCK, 0);  printf("Bloqueado\n");  if(rc == -1 ){  //Si está bloqueado, terminará el programa  return 0;  }  printf("Fijando un cerrojo y que escriba la hora actual\n");  sprintf (buf, "Hora: %d:%d\n", tm->tm\_hour, tm->tm\_min);  write(fd, &buf, strlen(buf));  sleep(3);    lockf(fd, F\_ULOCK, 0);  printf("Desbloqueado\n");  close(fd);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej15.c -o ej15  [cursoredes@localhost Prac2]$ ./ej15 file  Bloqueado  Fijando un cerrojo y que escriba la hora actual  Desbloqueado  [cursoredes@localhost Prac2]$  [cursoredes@localhost Prac2]$ cat file  Hora: 18:3 |

***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>  //open  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  int main(int argc, char \*\*argv) {  DIR \*directorio = opendir(argv[1]);  struct dirent \*current;  struct stat s;  current = readdir(directorio);  double totalsize = 0;    while (current != NULL) {  char \*path = malloc(sizeof(char)\*(strlen(argv[1]) + strlen(current->d\_name)));  strcpy(path, argv[1]);  strcat(path, "/");  strcat(path, current->d\_name);  lstat(path, &s);  if (S\_ISDIR(s.st\_mode)) {  printf("%s/\n", current->d\_name);  }  else if (S\_ISLNK(s.st\_mode)) {  char \*linkname = malloc(s.st\_size + 1);  readlink(path, linkname, s.st\_size + 1);  printf("%s->%s\n", current->d\_name, linkname);  free(linkname);  }  else if (S\_ISREG(s.st\_mode)) {  totalsize = totalsize + (double) s.st\_size/1024;  printf("%s\*\n", current->d\_name);  }  free(path);  current = readdir(directorio);  }  printf("Tamaño total que ocupan los ficheros: %f kilobytes.\n", totalsize);  closedir(directorio);  return 0;  }  [cursoredes@localhost Prac2]$ gcc -Wall -g ej16.c -o ej16  [cursoredes@localhost Prac2]$ ./ej16 Dir/  ./  ../  ej16\*  dir2/  hardlink\*  file.c\*  sym2->file.c  Tamaño total que ocupan los ficheros: 15.103516 kilobytes. |