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.

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
chmod u+rw-x,g+r-wx,o+rx-w fichero
```

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

```
chmod -x fichero
```

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

int main(){

    //int fd = open("fichero", O_CREAT, 00645);    mode en este caso se almacena en octal

   int fd = open("fichero", O_CREAT,  S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH | S_IXOTH);
   if(fd != -1){

            printf("Fichero creado. \n");
            close(fd);
   }

   else printf("Error al crear el fichero.\n");


   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)`.

```
umask 727 (la máscara se niega, por lo que quedan como permisos 050)
```

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

int main(){


            umask(0727);

            int fd = open("fichero", O_CREAT,  0777);
            if(fd != -1){
```

```
        printf("Fichero creado. \n");
        close(fd);
        }

        else printf("Error al crear el fichero.\n");


        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 <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/sysmacros.h>

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

   struct stat sb;

   if (argc != 2) {
        fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
        exit(EXIT_FAILURE);
        }

        if (stat(argv[1], &sb) == -1) {
        perror("stat");
        exit(EXIT_FAILURE);
        }

   printf("Major: %ld\n", (long) major(sb.st_dev));
   printf("Minor: %ld\n", (long) minor(sb.st_dev));
   printf("I-node number: %ld\n", (long) sb.st_ino);
   printf("File type: ");

   switch (sb.st_mode & S_IFMT) {
        case S_IFBLK:  printf("block device\n");        break;
        case S_IFCHR:  printf("character device\n");    break;
        case S_IFDIR:  printf("directory\n");           break;
        case S_IFIFO:  printf("FIFO/pipe\n");           break;
        case S_IFLNK:  printf("symlink\n");             break;
        case S_IFREG:  printf("regular file\n");        break;
        case S_IFSOCK: printf("socket\n");              break;
```

```
        default:        printf("unknown?\n");          break;
        }

  /* OTRA FORMA: IFs anidados
  if(S_ISREG(sb.st_mode)){
        printf("regular file\n");
  }*/

  printf("Last status change: %s", ctime(&sb.st_ctime));
        printf("Last file access: %s", ctime(&sb.st_atime));
        printf("Last file modification: %s", ctime(&sb.st_mtime));

        return 0;
}
```

st_mtime se actualiza cuando se modifica el contenido del fichero, mientras que st_ctime se actualiza cuando se modifica el inodo.

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.
- 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?

ln -s fichero fichero2

ls -l:

```
----r-x--- 1 cursoredes cursoredes        0 Nov  8 12:22 fichero
lrwxrwxrwx 1 cursoredes cursoredes        7 Nov 14 18:46 fichero2 -> fichero
```

ls -i:

```
18833056 fichero
17153624 fichero2
```

ln fichero fichero3

ls -l:

```
lrwxrwxrwx 1 cursoredes cursoredes        7 Nov 14 18:46 fichero2 -> fichero
----r-x--- 2 cursoredes cursoredes        0 Nov  8 12:22 fichero3
```

ls -i:

```
18833056 fichero
18833056 fichero3
```

stat fichero:

```
File: ‘fichero’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 18833056    Links: 2
Access: (0050/----r-x---)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-08 12:22:02.217594166 +0100
Modify: 2021-11-08 12:22:02.217594166 +0100
```

```
Change: 2021-11-14 18:49:23.144511774 +0100
 Birth: -
```

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

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

   char slink[255];
   char hlink[255];
   strcpy(slink, argv[1]);
   strcat(slink,".sym");
   strcpy(hlink, argv[1]);
   strcat(hlink,".hard");

   struct stat sb;

   if (argc != 2) {
           fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
           exit(EXIT_FAILURE);
   }

   if (stat(argv[1], &sb) == -1) {
           perror("stat");
           exit(EXIT_FAILURE);
   }

   if(S_ISREG(sb.st_mode)){

           if(link(argv[1], hlink) == -1){
                   perror("Enlace rígido");
                   exit(EXIT_FAILURE);;
           }

           if(symlink(argv[1], slink) == -1){
                   perror("Enlace simbólico");
                   exit(EXIT_FAILURE);
           }
   }

   else printf(" %s no es un fichero regular.\n", argv[1]);
   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 <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/sysmacros.h>
#include <unistd.h>

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

   int fd;

   if (argc != 2) {
          fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
               exit(EXIT_FAILURE);
          }

          if ((fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644)) == -1) {
               perror("Error abriendo fichero");
               exit(EXIT_FAILURE);
          }

   if (dup2(fd, 1) == -1)  {
           perror("Error dup2");
           exit(EXIT_FAILURE);
   }

   printf("Escribiendo en el fichero...\n");
   close(fd);

   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 <stdlib.h>
#include <sys/sysmacros.h>
#include <unistd.h>
```

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

   int fd;

   if (argc != 2) {
          fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
                exit(EXIT_FAILURE);
          }

          if ((fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644)) == -1) {
                perror("Error abriendo fichero");
                exit(EXIT_FAILURE);
          }

   if (dup2(fd, 1) == -1)  {
           perror("Error dup2");
           exit(EXIT_FAILURE);
   }

   if (dup2(fd, 2) == -1)  {
           perror("Error dup2");
           exit(EXIT_FAILURE);
   }

   printf("Escribiendo en el fichero...\n");
   fprintf(stderr, "Escribiendo desde la salida de error...\n");
   close(fd);

   return 0;
}
```

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 usando `lockf(3)`. 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 <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

```
  int fd;

  if (argc != 2) {
          fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
          exit(EXIT_FAILURE);
  }

  if ((fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644)) == -1) {
          perror("Error abriendo fichero");
          exit(EXIT_FAILURE);
 }

  if ( lockf(fd,F_TLOCK, 0) == -1) {
          printf("Cerrojo bloqueado\n");
  }
  else {
          printf("Hora: %ld\n", time(NULL));
          sleep(30);
          lockf(fd, F_ULOCK, 0);
          printf("Cerrojo liberado\n");
  }


  close(fd);
  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 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 <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/sysmacros.h>
#include <dirent.h>
```

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

   struct stat sb;
   int tam = 0;

   if (argc != 2) {
           fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
           exit(EXIT_FAILURE);
           }

           if (stat(argv[1], &sb) == -1) {
           perror("stat");
           exit(EXIT_FAILURE);
           }

   if(!S_ISDIR(sb.st_mode)){
            fprintf(stderr, "%s debe ser un directorio", argv[0]);
            exit(EXIT_FAILURE);
   }

   else {
           DIR *dir;
           dir = opendir(argv[1]);
           struct dirent *pdir = readdir(dir);

           while(pdir != NULL){
                   if (pdir->d_type == DT_REG) {
                           printf("%s\n", pdir->d_name);
                           tam = tam + pdir->d_reclen;
                   }
                   else if(pdir->d_type == DT_DIR){
                           printf("%s/\n", pdir->d_name);
                           tam = tam + pdir->d_reclen;
                   }
                   else if (pdir->d_type == DT_LNK) {
                           char path[PATH_MAX];
                           int bytes = readlink(pdir->d_name, path, PATH_MAX);

                           if (bytes== -1) {
                                   perror("Error nombre fichero enlazado");
                                   exit(EXIT_FAILURE);
                           }

                           path[bytes] = '\0';
                           printf("%s->%s\n", pdir->d_name, path);
                   }

                   pdir = readdir(dir);
           }

   }

   printf("Tamaño: %d\n", tam);
           return 0;
}
```