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.

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 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 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?

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

Ejercicio 5. 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`.

Ejercicio 6. Cuando se crea un fichero, los permisos por defecto se derivan de la máscara de usuario (*umask*). El comando `umask` (que es un comando interno de la *shell*) 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`.

Ejercicio 7. Modificar el ejercicio 5 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 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 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`?

Ejercicio 10. 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 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? En el caso de los enlaces simbólicos, ¿qué sucede si se borra el enlace? ¿y si se borra el fichero original?

Ejercicio 11. 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 (con el mismo nombre terminado en `.sym` y `.hard`). Comprobar el resultado con la orden `ls`.

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 haciendo que el programa escriba 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`"?

***Ejercicio 3* (Opcional).** La llamada `fcntl(2)` 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(2)`.

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 2. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero. El programa 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 programa.

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

***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 `'*'`
- Al final de la lista el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.