API de ficheros y directorios

Índice

En esta práctica vamos a hacer varios ejercicos orientados a afianzar nuestro conocimiento del manejo del API POSIX de ficheros y directorios. Concretamente trabajaremos las llamadas al sistema: open, read, write, close, lstat, readlink, symlink, lseek, opendir y readdir, además de algunas funciones de la librería estándar de C que serán necesarias, como strlen, snprintf, malloc y printf.

Se aconseja al alumno que cree un directorio por ejercico. En las instrucciones se asume que el ejercicio N se hace en un subdirectorio llamado ejercicio*N* dentro de un directorio común para esta práctica.

1 Copia de ficheros regulares

Diseña un programa copy.c que permita hacer la copia de un fichero regular. El programa recibira dos parámetros por la línea de llamadas. El primero será el nombre del fichero a copiar (fichero origen) y el segundo será el nombre que queremos darle a la copia (fichero destino). El programa debe realizar la copia en bloques de 512B. Para ello se reservará un buffer de 512 bytes como almacenamiento intermedio. El programa debe ir leyendo bloques de 512 bytes del fichero origen y escribiendo los bytes leídos en el fichero destino. Debéis tener en cuenta que si el tamaño del fichero no es múltiplo de 512 bytes la última vez no se leerán 512 bytes, sino lo que quede hasta el final del fichero (es decir, read devolverá menos de 512). Por ello siempre se debe escribir en el fichero destino tantos bytes como se hayan leído del fichero origen (read devolverá el número de bytes leídos).

El alumno debe consultar las páginas de manual de las siguientes llamadas al sistema: open, read, write y close. En la página de manual de open prestad especial atención a los flags de apertura, teniendo que usar en este caso O_RDONLY, O_WRONLY, O_CREAT, O_TRUNC.

Para comprobar el efecto de O_TRUNC, se sugiere al alumno que antes de ejecutar su programa de copia, cree un fichero con cualquier contenido que se llame como el fichero destino. Después puede copiar otro fichero usando el nombre elegido para el fichero destino y comprobar que el contenido anterior desaparece al usarse el flag O_TRUNC.

Para comprobar el funcionamiento correcto de nuestro programa podemos usar los comandos de shell diff y hexdump (este último para ficheros binarios).

2 Enlaces simbólicos.

Lo primero que vamos a hacer en este ejercicio es crear un enlace simbólico a un fichero cualquiera usando el comando ln. Por ejemplo, si queremos crear un enlace que se llame mylink y que apunte al fichero ../ejercicio1/Makefile usaremos el siguiente comando del shell:

```
$ ln -s ../ejercicio1/Makefile mylink
```

Invocando *ls -l* podremos comprobar que el fichero creado es realmente un enlace simbólico y veremos el fichero apuntado:

```
$ ls -l
...
lrwxrwxrwx 1 christian christian   22 Jul 14 13:23 mylink -> ../ejercicio1/Makefile
...
```

Ahora usaremos nuestro programa de copia para copiar el enlace simbólico. Asumiendo que dicho programa es *../ejercicio1/copy*, ejecutamos:

```
$ ../ejercicio1/copy mylink mylinkcopy
```

¿Qué tipo de fichero es mylinkcopy? ¿Cuál es el contenido del fichero mylinkcopy? Se pueden usar los comandos ls, stat, cat y diff para obtener las respuestas a estas preguntas.

Es posible que este sea el comportamiento que deseemos, pero también es posible que no. ¿Y si queremos que la copia de un enlace simbólico sea otro enlace simbólico que apunte al mismo fichero que apuntaba el enlace simbólico original?

Vamos a hacer una modificación de nuestro programa de copia del ejercicio anterior, que llamaremos copy2.c. Podemos empezar copiando el programa anterior para luego modificarlo. Haremos entonces una copia usando el comando cp:

```
$ cp ../ejercicio1/copy.c copy2.c
```

Después editaremos el fichero copy2.c de modo que: 1. Antes de hacer la copia identifique si el fichero origen es un fichero regular, un enlace simbólico u otro tipo de fichero, haciendo uso de la llamada al sistema lstat (consultar su página de manual).

2. Si el fichero origen es un fichero regular, haremos la copia como en el ejercicio anterior.
3. En camnio, si el fichero origen es un enlace simbólico no tenemos que hacer la copia del fichero apuntado sino crear un enlace simbólico que apunte al mismo fichero al que apunta el fichero origen. Para ello tenemos que seguir los siguientes pasos:

   a Reservar memoria para hacer una copia de la ruta apuntada. Una llamada a lstat sobre el fichero origen nos permitirá conocer el número de bytes que ocupa el enlace simbólico, que se corresponde con el tamaño de esta ruta sin el carácter null ('\0') de final de cadena (consultar la página de manual de lstat). Por tanto sumaremos uno al tamaño obtenido de lstat.

   b Copiar en este buffer la ruta del fichero apuntado haciendo uso de la llamada al sistema readlink. Deberemos añadir manualmente el caracter null de final de cadena.

   c Usar la llamada al sistema symlink para crear un nuevo enlace simbólico que apunte a esta ruta.

   Debéis consultar las páginas de manual de lstat, readlink y symlink.
4. Si el fichero origen es de cualquier otro tipo (por ejemplo un directorio) mostrarán un mensaje de error y el programa terminará.

3 Desplazamiento del marcador de posición en ficheros.

En este ejercicio vamos a crear un programa mostrar.c similar al comando cat, que reciba como parámetro el nombre de un fichero y lo muestre por la salida estándar. En este caso asumiremos que es un fichero estándar. Además, nuestro programa recibirá dos argumentos que parsearemos con getopt (consultar su página de manual): - -n N: indica que queremos saltarnos N bytes desde el comienzo del fichero o mostrar únicamente los N últimos bytes del fichero. Que se haga una cosa o la otra depende de la presencia o no de un segundo flag -e. Si el flag -n no aparece N tomará el valor 0. --e: si aparece, se leerán los últimos N bytes del fichero. Si no aparece, se saltarán los primeros N bytes del fichero.

El programa debe abrir el fichero indicado en la línea de comandos (consultar optind en la página de manual de getopt) y después situar el marcador de posición en la posición correcta antes de leer. Para ello haremos uso de la llamada al sistema lseek (consultar la página de manual). Si el usuario ha usado el flag -e debemos situar el marcador N bytes antes del final del fichero. Si el usuario ha usado el flag -e debemos avanzar el marcador N bytes desde el comienzo del fichero.

Una vez situado el marcador de posición, debemos leer leer byte a byte hasta el final de fichero, escribiendo cada byte leído por la salida estándar (descriptor 1).

4 Apertura para añadir.

En este ejercicio vamos a experimentar el efecto del flag O_APPEND en la apertura de un fichero. Lo primero que vamos a hacer es crear un fichero de texto con un contenido aleatorio llamado *fichero.txt*. Lo crearemos exáctamente con 134144 bytes usando el siguiente filtro de la shell:

```
$ base64 /dev/urandom | head -c 134144 > fichero.txt
```

A continuación crearemos un programa progA.c en el mismo directorio, lo compilaremos y lo ejecutaremos:

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

int main(void)
{
    int i;
    int fd = open("fichero.txt", O_RDONLY);

    if (fd == -1) {
        return -1;
    }

    i = lseek( fd, 0, SEEK_END );
    printf("%d\n", i);

    return 0;
}
```

¿Qué muestra este programa por la salida estándar? Consulta la página de manual de lseek para comprobar el valor que devuelve.

A continuación crearemos otro programa progB.c en el mismo directorio, lo compilaremos y lo ejecutaremos:

```c
//progB.c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void)
{
    char c = 'a';
    int fd = open("fichero.txt", O_WRONLY | O_CREAT | O_APPEND);
    if (fd == -1) {
        return -1;
    }

    lseek(fd, 1024, SEEK_CUR);
    write(fd, &c, 1);

    return 0;
}
```

¿Qué hace este programa? Puedes usar ls y diff para comprobar los cambios sobre *fichero.txt*. ¿Tiene algún efecto la llamada a lseek del programa progB.c?

Modifica ahora el programa progB.c eliminando el flag de apertura O_APPEND. ¿Cuál es la diferencia? ¿Tiene ahora algún efecto el flag O_APPEND?

El flag O_APPEND garantiza que la operación de escritura y el desplazamiento del marcador al final del fichero son atómicas de forma global en el sistema, es decir, que si dos procesos están escribiendo al mismo tiempo en un fichero que han abierto con O_APPEND siempre añadirán información al final del fichero, independientemente del orden en el que se hagan las escrituras.

5 Recorrido de directorios.

En este ejercicio vamos a crear un programa que reciba una lista de nombres de fichero como parámtros de la llamada, y calculará para cada uno el número total de kilobytes reservados por el sistema para almacenar dicho fichero. En caso de que alguno de los ficheros procesados sean de tipo directorio, se sumarán también los kilobytes ocupados por los ficheros contenidos el directorio (notar que esto es recursivo, porque un directorio puede contener otros directorios).

Para conocer el número de kilobytes reservados por el sistema para almacenar un fichero podemos hacer uso de la llamada a lstat, que nos permite saber el número de bloques de 512 bytes reservados por el sistema.

Para identificar si un fichero es un directorio deberemos hacer una llamada a lstat y consultar el campo st_mode (consultar la página de manual de lstat).

Para recorrer un directorio, primero deberemos abrirlo usando la función de biblioteca *opendir* y luego leer sus entradas usando la función de biblioteca *readdir*. Consultar las páginas de manual de estas dos funciones. Notar que las entradas de un directorio no tienen un orden establecido y que todo directorio tiene dos entradas "." y "..", que deberemos ignorar si no queremos tener una recursión infinita.

El programa debe mostrar por la salida estándar una línea por fichero de la línea de comandos, con el tamaño total en kilobytes del fichero y el nombre de dicho fichero. Para comprobar si nuestro programa funciona correctamente podemos

comparar su salida con la del comando *du -ks*, pasando a este comando la misma lista de ficheros que al nuestro. Notar que se pueden usar los comodines del shell.

Ejemplo de uso:

```
$ ls -l .
total 40
-rwxr-xr-x 1 christian christian 20416 Jul 15 12:41 espacio
-rw-r--r-- 1 christian christian  1639 Jul 15 12:41 espacio.c
-rw-r--r-- 1 christian christian  9056 Jul 15 12:41 espacio.o
-rw-r--r-- 1 christian christian   273 Jul 15 09:54 Makefile
$ ./espacio .
44K .
$ ./espacio *
20K espacio
4K  espacio.c
12K espacio.o
4K  Makefile
```