Práctica 2: Ficheros y directorios

Índice

1 Objetivos

En esta práctica vamos a hacer varios ejercicos orientados a afianzar nuestro conocimiento del manejo del API POSIX de ficheros y directorios, así como del uso de la biblioteca estándar de C para el manejo de ficheros.

La práctica está organizada en dos partes:

- API POSIX de ficheros y Directorios
- Proyecto C de manejo de ficheros con la bibliteca estándar de C

En la primera parte revisaremos el API de ficheros y directorios y las correspondientes funciones de la biblioteca estándar de C haciendo pequeños ejercicios de programación para conocer su uso.

En la segunda parte tendremos que diseñar una utilidad de línea de comandos más compleja en la que pondremos en práctica el manejo de ficheros desde un programa C.

La práctica tiene asignada dos sesiones de laboratorio. Se recomienda completar la primera parte antes de la segunda sesión de laboratorio, con el fin de reservar dicha sesión a la segunda parte de la práctica.

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

El archivo ficheros_p2.tar.gz contiene una serie de ficheros que pueden ser usados como punto de partida para el desarrollo de los ejercicios de esta práctica, así como unos makefiles que pueden ser usados para la compilación de los distintos proyectos.

2 API POSIX de ficheros y Directorios

En estos ejercicios 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 como strlen, malloc, free, snprintf y printf.

Ejercicio 1: uso de la librería estándar

Analiza el código del programa `show_file.c`, que lee byte a byte el contenido de un fichero, cuyo nombre se pasa como parámetro, y lo muestra por pantalla usando funciones de la biblioteca estándar de "C". Compila y comprueba el funcionamiento correcto del programa. Después modifica el código reemplazando el uso de `getc()` por el de la función `fread()` y el uso de `putc()` por el de la función `fwrite()`. Consulta las páginas de manual correspondientes.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    FILE* file=NULL;
    int c,ret;

    if (argc!=2) {
        fprintf(stderr,"Usage: %s <file_name>\n",argv[0]);
        exit(1);
    }

    /* Open file */
    if ((file = fopen(argv[1], "r")) == NULL)
        err(2,"The input file %s could not be opened",argv[1]);

    /* Read file byte by byte */
    while ((c = getc(file)) != EOF) {
        /* Print byte to stdout */
        ret=putc((unsigned char) c, stdout);

        if (ret==EOF){
            fclose(file);
            err(3,"putc() failed!!");
        }
    }

    fclose(file);
    return 0;
}
```

Ejercicio 2: Copia de ficheros regulares

Diseña un programa copy.c que permita hacer la copia de un fichero regular usando las llamadas al sistema del estándar POSIX: open, read, write y close. Se deben consultar sus páginas de manual, prestando especial atención a los flags de apertura: O_RDONLY, O_WRONLY, O_CREAT, O_TRUNC.

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, usando un array local como almacenamiento intermedio entre la lectura y la escritura. El programa debe ir leyendo bloques de 512 bytes del fichero origen y escribiendo los bytes leídos en el fichero destino. Debe tenerse 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 (conultar el apartado RETURN VALUE en la página de manual de read). Por ello siempre se deben escribir en el fichero destino tantos bytes como se hayan leído del fichero origen.

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).

Ejercicio 3: 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 cambio, 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.

Ejercicio 4: 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).

Ejercicio 5: Recorrido de directorios.

En este ejercicio vamos a crear un programa *espacio.c* 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
```

3 Proyecto C de manejo de ficheros con la bibliteca estándar de C

En esta parte de la práctica vamos a diseñar un programa más elaborado que tenga que leer y escribir de un fichero regular en formato binario. Para que el programa sea lo más portable posible se recomienda a los estudiantes utilizar las funciones de la biblioteca estándar de C: fopen, fread, fwrite, fclose, fseek y feof. Se debe consultar las páginas de manual de estas funciones en caso de duda sobre su comportamiento.

Ejercicio 1

Desarrollar un programa `student-record` que permita crear ficheros binarios que almacenen un conjunto de registros con información de distintos estudiantes, y también permita consultar/imprimir información almacenada en estos ficheros. Cada estudiante estará representado mediante 4 campos: identificador numérico único, NIF, nombre, y apellidos. El fichero binario ha de contener una cabecera (entero de 32 bits) que indique cuál es el número de registros almacenados, y a continuación incluir los registros de estudiantes en formato binario, uno detrás del otro.

Cada registro de estudiantes estará representado en memoria mediante la siguiente estructura:

```c
#define MAX_CHARS_NIF  9

typedef struct {
    int student_id;
    char NIF[MAX_CHARS_NIF+1];
    char* first_name;
    char* last_name;
} student_t;
```

Por cada registro debe escribirse en el fichero (representación en disco) el identificador numérico único (4 bytes), seguido de las cadenas de caracteres asociadas a los tres campos restantes. Almacenar cada una de las cadenas en el fichero conlleva escribir todos sus caracteres, incluyendo el terminador (`\0`). Esto es esencial para permitir posteriormente la lectura correcta de los campos del fichero.

El modo de uso del programa debe poder consultarse con la opción -h, del siguiente modo:

```
$ ./student-records -h
Usage: ./student-records -f file [ -h | -l | -c | -a | -q [ -i|-n ID] ] ] [ list of records ]
```

Además de `-h`, el programa implementará opciones para crear (`-c`) y listar (`-l`) ficheros de registros de estudiantes, así como para poder añadir nuevos registros al final de un fichero existente `-a`, o realizar búsquedas (*queries*) de registros específicos (`-q`) por identificador de estudiante (`-i`) o NIF `-n`. En el caso de las opciones `-c` y `-a`, será preciso indicar en la línea de comando una lista de registros de estudiantes a almacenar en el fichero. Cada registro de la lista especificará los campos de cada estudiante mediante una cadena de caracteres con elementos separados por **":"**. Por ejemplo, considérese el siguiente comando para crear un nuevo fichero de estudiantes llamado `database` que almacenará 2 registros:

```
$ ./student-records -f database -c 27:67659034X:Chris:Rock 34:78675903J:Antonio:Banderas
2 records written succesfully
```

El programa deberá constuirse de cero pero se recomienda reutilizar código y algunas ideas de diseño de los programas vistos en la práctica anterior de introducción al entorno. Se aconseja también implementar las siguientes funciones auxiliares para simplificar el desarrollo del programa:

- `student_t* parse_records(char* records[], int* nr_records);`

  Esta función acepta como parámetro el listado de registros en formato ASCII pasados como argumentro al programa en la línea de comando (`records`), así como el número de registros (`nr_records`), y devuelve la representación binaria en memoria de los mismos. Esta representación será un array de estructuras cuya memoria ha de reservarse con `malloc()` dentro de la propia función.

- `int dump_entries(student_t* entries, int nr_entries, FILE* students)`

La función vuelca al fichero binario ya abierto (`students`) los registros de estudiantes pasados como parámetro (`entries`). Para maximizar la reutilización de código, esta función NO escribirá en el fichero la cabecera numérica que indica el número el número de registros.

- `student_t* read_student_file(FILE* students, int* nr_entries)`

Esta función lee toda la información de un fichero binario de registros de estudiantes ya abierto, y devuelve tanto la información de la cabecera (parámetro de retorno `nr_entries`), como el array de registros de estudiantes (valor de retorno de la función). La memoria del array que se retorna debe reservarse con `malloc()` dentro de la propia función.

- `char* loadstr(FILE* students)`

La función lee una cadena de caracteres terminada en `'\0'` del fichero cuyo descriptor se pasa como parámetro, reservando la cantidad de memoria adecuada para la cadena leída.

Ejemplo de ejecución

```
## List project's files and compile program
usuarioso@debian:~/student-records$ ls
defs.h  Makefile  student-records.c
usuarioso@debian:~/student-records$ make
gcc -c -Wall -g student-records.c -o student-records.o
gcc -g -o student-records student-records.o

## Create a new 2-record file and dump contents of the associated binary file
usuarioso@debian:~/student-records$ ./student-records -f database -c \
> 27:67659034X:Chris:Rock 34:78675903J:Antonio:Banderas
2 records written succesfully
usuarioso@debian:~/student-records$ xxd database
00000000: 0200 0000 1b00 0000 3637 3635 3930 3334  ........67659034
00000010: 5800 4368 7269 7300 526f 636b 0022 0000  X.Chris.Rock."..
00000020: 0037 3836 3735 3930 334a 0041 6e74 6f6e  .78675903J.Anton
00000030: 696f 0042 616e 6465 7261 7300            io.Banderas.

## Add 2 new registers at the end
usuarioso@debian:~/student-records$ ./student-records -f database -a \
> 3:58943056J:Santiago:Segura 4:6345239G:Penelope:Cruz
2 extra records written succesfully
usuarioso@debian:~/student-records$ xxd database
00000000: 0400 0000 1b00 0000 3637 3635 3930 3334  ........67659034
00000010: 5800 4368 7269 7300 526f 636b 0022 0000  X.Chris.Rock."..
00000020: 0037 3836 3735 3930 334a 0041 6e74 6f6e  .78675903J.Anton
00000030: 696f 0042 616e 6465 7261 7300 0300 0000  io.Banderas.....
00000040: 3538 3934 3330 3536 4a00 5361 6e74 6961  58943056J.Santia
00000050: 676f 0053 6567 7572 6100 0400 0000 3633  go.Segura.....63
00000060: 3435 3233 3947 0050 656e 656c 6f70 6500  45239G.Penelope.
```

```
00000070: 4372 757a 00                             Cruz.

## Try to add an entry that matches an existing student ID
$ ./student-records -f database -a 3:58943056J:Antonio:Segura
Found duplicate student_id 3

## List all the entries in the file
usuarioso@debian:~/student-records$ ./student-records -f database -l
[Entry #0]
        student_id=27
        NIF=67659034X
        first_name=Chris
        last_name=Rock
[Entry #1]
        student_id=34
        NIF=78675903J
        first_name=Antonio
        last_name=Banderas
[Entry #2]
        student_id=3
        NIF=58943056J
        first_name=Santiago
        last_name=Segura
[Entry #3]
        student_id=4
        NIF=6345239G
        first_name=Penelope
        last_name=Cruz

## Search for specific entries
usuarioso@debian:~/student-records$ ./student-records -f database -q -i 7
No entry was found
usuarioso@debian:~/student-records$ ./student-records -f database -q -i 34
[Entry #1]
        student_id=34
        NIF=78675903J
        first_name=Antonio
        last_name=Banderas
usuarioso@debian:~/student-records$ ./student-records -f database -q -n 6345239G
[Entry #3]
        student_id=4
        NIF=6345239G
        first_name=Penelope
        last_name=Cruz
```

Ejercicio 2

Con el fin de practicar el uso del shell, se pide al alumno que desarrolle su propio script bash para la comprobación de la funcionalidad del programa desarrollado en el ejercicio anterior.

Antes de elaborar y ejecutar el script se preparará un fichero de texto llamado *records.txt* en el mismo directorio que el programa, que debe incluir un conjunto de registros de estudiantes en texto plano y separados por un salto de línea, como el el siguiente ejemplo:

```
27:67659034X:Chris:Rock
34:78675903J:Antonio:Banderas
3:58943056J:Santiago:Segura
4:6345239G:Penelope:Cruz
```

El script deberá seguir el siguiente esquema:

1. En primer lugar comprobará que el programa `student-records` está en el directorio actual y que es ejecutable. En caso contrario mostrará un mensaje informativo por pantalla y terminará.
2. A continuación comprobará que el fichero *records.txt* está en el directorio actual y que es regular. En caso contrario mostrará un mensaje de error y terminará.
3. El script leerá ese fichero de texto, y almacenará todos los registros separados por espacios en una sola cadena (variable `records` de tipo string). Para ello se utilizará el comando `cat` en combinación con una expansión de órdenes de BASH, que ya sustituye los saltos de línea del fichero por espacios.
4. Acto seguido, el script recorrerá con un bucle `for` todos los registros en formato ASCII almacenados en la variable `records`. En la primera iteración del bucle se creará un nuevo fichero binario de registros de estudiantes llamado `database`, invocando al programa `student-records` con el primer registro del fichero y la opción `-c`. En las iteraciones restantes se añadirán los demás registros uno a uno al fichero `database` utilizando la opción `-a`.
5. El script mostrará el contenido del fichero `databases` de dos formas: usando la opción `-l` de `student-records` y empleando el programa `xxd`, que simplemente realizará un volcado binario del fichero.
6. Finalmente, se utilizará otro buctyle para recorrer de nuevo todos los registros en formato ASCII almacenados en la variable `records`. En este caso para cada registro se comprobará que el NIF del estudiante se encuentra en el fichero `database`, invocando el programa `student-records` con las opciones `-q -n`. Para extraer el NIF de cada estudiante del registro en formato ASCII correspondiente se usará el comando `cut` y un pipe (consultar página de manual de `cut`).

En los distintos puntos del script se ha de comprobar que cada invocación del programa `student-records` devuelve 0. En caso de que se produzca un error, la ejecución del script debe abortarse en ese momento y devolver 1.