Práctica 2: Programación en C y librería estándar

Índice

1 Objetivos

En esta práctica vamos a hacer varios ejercicos orientados a afianzar nuestro conocimiento sobre la programación de sistema en C y el uso de su librería estándar para operaciones básicas sobre cadenas, entrada salida y ficheros.

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

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.

Ejercicio 1: Manejo básico de ficheros con 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;
}
```

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

TODO (Juan Carlos): remodelar para descomponer en ejercicios simples para trabajar el uso de cadenas (sprintf, snprintf), E/S (printf, fprintf, scanf, fscanf, sscanf), memoria dinámica (malloc, free), parsing básico (strsep, strtok).

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:

```
#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.