Práctica 1: Introducción a la programación de sistemas en Linux

Índice

1 Introducción **1**
1.1 Objetivos 1
1.2 Requisitos 1

2 Ejercicios **2**
Ejercicio 1 2
Ejercicio 2 2
Ejercicio 3 3
Ejercicio 4 5
Ejemplo de ejecución 6
Ejercicio 5 7

1 Introducción

1.1 Objetivos

- Familiarizarse con el entorno de desarrollo de aplicaciones C en GNU/Linux, y comprender los conceptos de proyecto y ejecutable en este contexto.
- Familiarizarse con el manejo básico del shell y aprender a desarrollar *shell scripts* sencillos.

1.2 Requisitos

Para poder realizar con éxito la práctica el alumno debe haber leído y comprendido los siguientes documentos facilitados por el profesor:

- Presentación "Introducción al entorno de desarrollo", que nos introduce al entorno GNU/Linux que utilizaremos en el laboratorio, y describe cómo trabajar con proyectos C con Makefile.
- Presentación "Revisión: Programación en C", que realiza un repaso de los conocimientos de C necesarios para realizar con éxito las prácticas, haciendo especial hincapié en los errores que cometen habitualmente los estudiantes menos experimentados en el lenguaje C.
- Manual del laboratorio titulado "Entorno de desarrollo C para GNU/Linux", que describe las herramientas que componen el entorno de desarrollo que vamos a utilizar, así como las funciones básicas de la biblioteca estándar de C que los alumnos deben conocer.
- Presentación "Introducción a Bash", que presenta una breve introducción al interprete de órdenes (shell) Bash.

2 Ejercicios

Ejercicio 1

Analizar 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". Responda a las siguientes preguntas:

- ¿Qué comando se debe emplear para generar el ejecutable del programa (`show_file`) invocando directamente al compilador `gcc` (sin usar `make`)?
- Indique dos comandos para llevar a cabo respectivamente la compilación del programa (generación de fichero objeto) y el enlazado del mismo de forma independiente.

Realice las siguientes modificaciones en el programa `show_file.c`:

1. Realizar la lectura byte a byte del fichero de entrada empleando la función `fread()` en lugar de `getc()`. Modificar también la invocación a la función `putc()` por una llamada a `fwrite()`
2. Añadir un parámetro al programa modificado para permitir al usuario especificar el tamaño de bloque en bytes a usar en cada lectura realizada por `fread()`.

```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

El programa *badsort-ptr.c*, cuyo código fuente se muestra a continuación, ha sido desarrollado para realizar una ordenación por el método de la burbuja aplicada a un *array* de pares (cadena de caracteres, entero) inicializado dentro del programa. El programa emplea aritmética de punteros para acceder a los distintos elementos del array durante el recorrido. Lamentablemente, el programador ha cometido algunos errores. Utilizando un depurador de C (p.ej.,`gdb`) el alumno debe encontrar y corregir los errores.

```c
#include <stdio.h>

typedef struct {
    char data[4096];
    int key;
} item;

item array[] = {
    {"bill", 3},
    {"neil", 4},
    {"john", 2},
    {"rick", 5},
    {"alex", 1},
};

void sort(item *a, int n) {
    int i = 0, j = 0;
    int s = 1;
    item* p;

    for(; i < n & s != 0; i++) {
        s = 0;
        p = a;
        j = n-1;
        do {
            if( p->key > (p+1)->key) {
                item t = *p;
                *p  = *(p+1);
                *(p+1) = t;
                s++;
            }
        } while ( --j >= 0 );
    }
}

int main() {
    int i;
    sort(array,5);
    for(i = 0; i < 5; i++)
        printf("array[%d] = {%s, %d}\n",
                i, array[i].data, array[i].key);
    return 0;
}
```

Ejercicio 3

Estudiar el código y el funcionamiento del programa `show-passwd.c`, que lee el contenido del fichero del sistema `/etc/passwd` e imprime por pantalla (o en otro fichero dado) las distintas entradas de`/etc/passwd` –una por línea–, así como los distintos campos de cada entrada. El fichero `/etc/passwd` almacena en formato de texto plano información esencial de los usuarios del sistema, como su identificador numérico de usuario o grupo así como el programa configurado como intérprete de órdenes (*shell*) predeterminado para cada usuario. Para obtener más información sobre este fichero se ha de consultar su página de manual: `man 5 passwd`

El modo de uso del programa puede consultarse invocándolo con la opción -h:

```
$ ./show-passwd -h
Usage: ./show-passwd [ -h | -v | -p | -o <output_file> ]
```

Las opciones `-v` y `-p`, permiten configurar el formato en el que el programa imprime la información de `/etc/passwd`.

Las citadas opciones activan respectivamente el modo `verbose`(por defecto) o `pipe`. La opción `-o`, que acepta un argumento obligatorio, permite selecionar un fichero para la salida del programa alternativo a la salida estándar.

Uno de los principales objetivos de este ejercicio es que el estudiante se familiarize con tres funciones muy útiles empleadas por el programa `show-passwd.c` , y cuya página de manual debe consultarse:

- `int sscanf(const char *s, const char *format, ...);`

  Variante de `scanf()` que permite leer con formato a partir de un buffer de caracteres pasado como primer parámetro (`s`) . La función almacena en variables del programa, pasadas como argumento tras la cadena de formato, el resultado de convertir los distintos "tokens" de `s` de ASCII a binario.

- `char *strsep(char **stringp, const char *delim);`

  Permite dividir una cadena de caracteres en *tokens*, proporcionando como segundo parámetro la cadena delimitadora de esos tokens. Como se puede observar en el programa `show-passwd.c`, esta función se utiliza para extraer los distintos campos almacenados en cada línea del fichero `/etc/passwd`, que están separados por `":"`. La función `strsep()` se usa típicamente en un bucle, que para tan pronto como el token devuelto es NULL. El primer argumento de la función es un puntero por referencia. Antes de comenzar el bucle, `*stringp` debe apuntar al comienzo de la cadena que deseamos procesar. Cuando `strsep()` retorna, `*stringp` apunta al resto de la cadena que queda por procesar.

- `int getopt(int argc, char *const argv[], const char *optstring);`

  Esta función es la más sofisticada de las tres, y permite procesar cómodamente las distintas opciones de la línea de comando que acepta un programa C. La función suele invocarse desde `main()`, y sus dos primeros parámetros coinciden con los argumentos `argc` y `argv` pasados a `main()`. El parámetro `optstring` sirve para indicar de forma compacta a `getopt()` cuáles son las opciones que el programa acepta –cada una identificada por una letra–, y si éstas a su vez aceptan parámetros obligatorios u opcionales.

  El estudiante deberá familiarizarse con esta función mediante el estudio del código fuente del programa, y la consulta de la página de manual de `getopt()`: `man 3 getopt`

  Deben tenerse en cuenta las siguientes consideraciones:

  1. La función `getopt()` se usa en combinación con un bucle, que invoca tantas veces la función como opciones ha pasado el usuario en la línea de comandos. Cada vez que la función se invoca y encuentra una opción, `getopt()` retorna el caracter correspondiente a dicha opción. Por lo tanto, dentro del bucle suele emplearse la construcción *switch-case* de C para llevar a cabo el procesamiento de las distintas opciones.
  2. Un aspecto particular de la función `getopt()` es que establece el valor de distintas variables globales tras invocarse, siendo las más relevantes las siguientes:
     - `char* optarg`: almacena el argumento pasado a la opción actual reconocida, si ésta acepta argumentos. Si la opción no incluye un argumento, entonces `optarg` se establece a `NULL`
     - `int optind`: representa el índice del siguiente elemento en el `argv` (elementos que quedan sin procesar). Se usa frecuentemente para procesar argumentos adicionales del programa que no están asociados a ninguna opción. Un ejemplo de ello es la lista de registros de estudiantes que ha de procesarse en el programa a desarrollar en el siguiente ejercicio.

Responda a las siguientes preguntas:

1. Para representar cada una de las entradas del fichero `/etc/passwd` se emplea el tipo de datos `passwd_entry_t` (estructura definida en `defs.h`). Nótese que muchos de los campos almacenan cadenas de caracteres definidas como arrays de caracteres de longitud máxima prefijada, o mediante el tipo de datos `char*`. La función `parse_passwd()`, definida en `show-passwd.c` es la encargada de inicializar los distintos campos de la estructura. ¿Cuál es el propósito de la función `clone_string()` que se usa para inicializar algunos de los citados campos tipo cadena? ¿Por qué no es posible en algunos casos simplemente

copiar la cadena vía `strcpy()` o realizando una asignación `campo=cadena_existente;`? Justifique la respuesta.

2. La función `strsep()`, utilizada en `parse_passwd()`, modifica la cadena que se desea dividir en tokens. ¿Qué tipo de modificaciones sufre la cadena (variable `line`) tras invocaciones sucesivas de `strsep()`? **Pista**: Consúltese el valor y las direcciones de las variables del programa usando un depurador de C como `gdb`.

Realice las siguientes modificaciones en el programa `show-passwd.c`:

- Añada la opción `-i <inputfile>` para especificar una ruta alternativa para el fichero `passwd`. Hacer una copia de `/etc/passwd` en otra ubicación para verificar el correcto funcionamiento de esta nueva opción.
- Implemente una nueva opción `-c` en el programa, que permita mostrar los campos en cada entrada de `passwd` como valores separados por comas (CSV) en lugar de por **`":"`**.

Ejercicio 4

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 construirse de cero, pero se recomienda reutilizar código y algunas ideas de diseño del proyecto del ejercicio anterior. 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 argumento 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 5

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 `database` 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 bucle 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.