Práctica 2: Programación en C y acceso a ficheros mediante la biblioteca estándar

Índice

1 Objetivos

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

Se aconseja al alumno que cree un directorio para la práctica con un subdirectorio por ejercicio. 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 usarse 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 Ejercicios

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;
}
```

Ejercicio 2: Escritura y lectura de cadenas de caracteres en ficheros

Desarrollar dos programas sencillos `write_strings.c` y `read_strings.c` que permitan respectivamente escribir y leer de un fichero un conjunto de cadenas de caracteres de longitud variable terminadas por `'\0'`. Dicho carácter terminador deberá almacenarse en el fichero con el resto de caracteres de cada cadena. Para el desarrollo de los dos programas se utilizarán las siguientes funciones de la biblioteca estándar: `fopen()`,`fclose()`, `fread()`, `fwrite()`, `fseek()` y `malloc()`

El programa `write-strings.c` aceptará como primer parámetro el nombre de un fichero de texto donde se escribirán los strings pasados a continuación a la línea de comandos (argumento 2, argumento 3, etc.). Si el fichero destino existe, el programa reescribirá su contenido.

El programa `read-strings.c` aceptará como parámetro el nombre del fichero de texto donde se almacenen las cadenas de caracteres terminadas en `'\0'`. Este programa leerá las cadenas y las imprimirá por pantalla separadas por un salto de línea, como se muestra en el siguiente ejemplo de ejecución:

```
## Write strings to file
usuarioso@debian:~/exercise2$ ./write_strings out London Paris Madrid Barcelona Berlin Lisbon

## Check whether file structure is correct (null-terminated strings)
usuarioso@debian:~/exercise2$ $ xxd out
00000000: 4c6f 6e64 6f6e 0050 6172 6973 004d 6164  London.Paris.Mad
00000010: 7269 6400 4261 7263 656c 6f6e 6100 4265  rid.Barcelona.Be
00000020: 726c 696e 004c 6973 626f 6e00            rlin.Lisbon.

## Read strings from file
usuarioso@debian:~/exercise2$ ./read_strings out
London
Paris
Madrid
Barcelona
Berlin
Lisbon
```

Por simplicidad para la implementación del programa `read-strings.c`, se ha de desarrollar una función auxiliar `char* loadstr(FILE* input)`. Esta función lee una cadena de caracteres terminada en `'\0'` del fichero cuyo descriptor se pasa como parámetro, reservando dinámicamente la cantidad de memoria adecuada para la cadena leída y retornando dicha cadena. La función tendrá que averiguar primero el número de caracteres de la cadena que comienza a partir de la ubicación actual del puntero de posición del fichero, leyendo carácter a carácter. Una vez detectado el

caracter terminador, restaurará el indicador de posición del fichero (moviéndolo hacia atrás) y, finalmente realizará una lectura de la cadena completa.

Ejercicio 3: Gestión de ficheros de texto y binarios con la biblioteca estándar de C

Objetivo: Comprender las diferencias fundamentales entre almacenar y recuperar datos estructurados en ficheros de texto y binarios usando funciones de la biblioteca estándar de C (`stdio.h`).

Estructura de Datos:

Para este ejercicio, utilizaremos una estructura simple:

```c
#define LABEL_MAX_LEN 15 // Max chars sin contar el '\0'

typedef struct {
    int id;                 // Identificador entero
    double value;           // Un valor de punto flotante
    char label[LABEL_MAX_LEN + 1]; // Etiqueta de texto (tamaño fijo)
} SimpleRecord;
```

Parte A: Ficheros de Texto (Human-Readable)

(A.1) Escritura en Fichero de Texto:

- **Tarea:** Crea un programa `write_records_text.c`.
- **Funcionalidad:**
  1. Define estáticamente un array con 3 instancias de `SimpleRecord` e inicializalas con datos de ejemplo (asegúrate de que `label` no exceda `LABEL_MAX_LEN`).
  2. Pide al usuario un nombre de fichero de salida (se ha de pasar como argumento `argv[1]`).
  3. Abre el fichero especificado para escritura (`"w"`). Realiza una comprobación básica de errores por si `fopen` falla.
  4. Recorre el array de registros y escribe cada uno en el fichero usando `fprintf`. Formatea cada línea como: `id valor etiqueta\n` (separados por espacios).
  5. Cierra el fichero con `fclose`.
  6. Imprime un mensaje indicando que la escritura ha finalizado.
- **Ejecución de prueba:**

```
>  ./write_records_text mis_datos.txt
La escritura ha finalizado correctamente

> cat mis_datos.txt
1 3.10 Barcelona
0 19.77 Madrid
2 7.42 Valencia
```

(A.2) Lectura de Fichero de Texto:

- **Tarea:** Crea un programa `read_records_text.c`.
- **Funcionalidad:**
  1. Recibe como argumento de línea de comandos (`argv[1]`) el nombre del fichero a leer (el creado en A.1).
  2. Abre el fichero para lectura (`"r"`). Comprueba errores.
  3. Lee los datos del fichero registro a registro usando `fscanf` dentro de un bucle (el bucle debe terminar cuando `fscanf` no pueda leer 3 elementos o llegue a EOF). *Nota: Asegúrate de que `fscanf` no provoque desbordamiento en `record.label`.*

4. Por cada registro leído, imprime sus campos en la consola de forma legible (p.ej., `ID: %d, Valor: %.2f, Etiqueta: '%s'\n`).
5. Cierra el fichero.

- **Ejecución de prueba:**

```
> ./read_records_text mis_datos.txt
ID:1, Valor:3.10, Etiqueta: 'Barcelona'
ID:0, Valor:19.77, Etiqueta: 'Madrid'
ID:2, Valor:7.42, Etiqueta: 'Valencia'
```

Parte B: Ficheros Binarios (Machine-Readable)

(B.1) Escritura en Fichero Binario:

- **Tarea:** Crea un programa `write_records_bin.c`.
- **Funcionalidad:**

1. Define el mismo array estático con 3 instancias de `SimpleRecord` que en A.1.
2. Pide al usuario un nombre de fichero de salida binario (pasado como `argv[1]`)
3. Abre el fichero especificado para escritura (`"w"`). Comprueba errores.
4. Recorre el array de registros y escribe cada estructura *directamente* en el fichero usando `fwrite`. Asegúrate de usar el tamaño correcto (`sizeof(SimpleRecord)`) y de escribir 1 elemento cada vez. Comprueba el valor de retorno de `fwrite` por si hay errores.
5. Cierra el fichero.
6. Imprime un mensaje indicando que la escritura ha finalizado.

- **Ejecución de prueba:**

```
> ./write_records_bin.c mis_datos.bin
La escritura ha finalizado correctamente
```

(B.2) Lectura de Fichero Binario:

- **Tarea:** Crea un programa `read_records_bin.c`.
- **Funcionalidad:**

1. Recibe como argumento de línea de comandos (`argv[1]`) el nombre del fichero binario a leer.
2. Abre el fichero para lectura (`"r"`). Comprueba errores.
3. Crea una variable local de tipo `SimpleRecord`.
4. Lee los datos del fichero registro a registro usando `fread` dentro de un bucle. Lee `sizeof(SimpleRecord)` bytes cada vez, directamente en la variable `SimpleRecord`. El bucle debe terminar cuando `fread` indique que no ha podido leer 1 elemento completo.
5. Por cada registro leído, imprime sus campos en la consola de forma legible (igual que en A.2).
6. Cierra el fichero.

- **Ejecución de prueba:**

```
> ./read_records_bin mis_datos.bin
ID:1, Valor:3.10, Etiqueta: 'Barcelona'
ID:0, Valor:19.77, Etiqueta: 'Madrid'
ID:2, Valor:7.42, Etiqueta: 'Valencia'
```

Parte C: Comparación y Análisis

- **Acciones:** Ejecuta todos los programas creados. Luego, utiliza comandos del shell para comparar los ficheros resultantes:

- `ls -l mis_datos.txt mis_datos.bin` (Compara tamaños)
- `cat mis_datos.txt` (Visualiza el texto)
- `cat mis_datos.bin` (Intenta visualizar el binario como texto - ¿qué ves?)
- `xxd mis_datos.bin` o `hexdump -C mis_datos.bin` (Visualiza el contenido binario real)

- **Preguntas:**

1. Compara los tamaños de `mis_datos.txt` y `mis_datos.bin`. ¿Cuál es más grande o más pequeño? ¿A qué se debe la diferencia? (Considera cómo se almacenan los números y las cadenas en cada formato).
2. Describe lo que ocurre al intentar ver el fichero binario con `cat`. ¿Por qué no es legible?
3. Observando la salida de `xxd` o `hexdump` para `mis_datos.bin`, ¿puedes identificar aproximadamente dónde empieza y termina cada `SimpleRecord`? ¿Se parece la representación binaria a cómo se alinearían los campos de la `struct` en memoria?
4. Menciona brevemente una ventaja y una desventaja de usar el formato de texto y una ventaja y desventaja de usar el formato binario para almacenar estos datos estructurados.
5. Considera que se reaizara una modificación en la estructura `SimpleRecord` de tal forma que la etiqueta se representase como `char*` y su memoria se reservase dinámicamente con `malloc`, quedando la estructura como sigue:

```c
typedef struct {
    int id;                 // Identificador entero
    double value;       // Un valor de punto flotante
    char* label;        // Etiqueta de texto
} SimpleRecord;
```

Responde a las siguientes preguntas asumiendo que se dispone de un array de estructuras de este tipo (en memoria) correctamente inicializado:

- ¿Se podría escribir en un fichero binario el array de estructuras con una única llamada a `fwrite()` como en B.1? ¿Por qué? En caso negativo, propón un mecanismo para hacerlo correctamente, prestando especial atención a la representación de la cadena en disco.
- ¿Asumiendo que hemos logrado escribir el citado array en binario en un fichero, podría leerse el array de golpe con una única llamada a `fread()` como en B.2? ¿Por qué? En caso negativo, propón un método para hacerlo correctamente.

Parte D: Conversión de formatos binario y texto

- **Tarea:** Crea un programa conversion.c, con el siguiente prototipo de llamada en la línea de comandos:

```
$ ./conversion [-i t|b] [-o t|b] fichero_entrada fichero_salida
```

- **Funcionalidad:**

1. El programa recibe por la linea de comandos las siguientes opciones y argumentos (se parsearán con getopt):
   - *-i*: formato de fichero de entrada. Espera un argumento, una cadena de caracteres, que indica si el formato de entrada es texto (“t”) o binario (“b”). Cualquier otra cadena se tomará como un error del usuario.
   - *-o*: formato de fichero de salida. Espera un argumento, una cadena de caracteres, que indica si el formato de entrada es texto (“t”) o binario (“b”). Cualquier otra cadena se tomará como un error del usuario. Las opciones *-i* y *-o* son opcionales, si no se pasan el programa asumirá que los formatos de entrada y salida son de texto.
   - Los nombres de los ficheros de entrada y salida respectivamente (consultar el uso de optind en *man 3 getopt*).
2. El programa leerá registros del tipo SimpleRecord del fichero de entrada y los escribrá en el fichero de salida, respetando los formatos indicados por las opciones *i* y *o*.

- **Recomendaciones:**

1. Reutilizar el código de las partes A y B.
2. Separar cláramente el parsing de opciones de la funcionalidad del programa.
3. Tratar de factorizar bien el código, creando funciones pequeñas para simplificar el diseño y habilitar el reuso de código.
4. Observar que en el computador los datos se almacenan en formato binario. Cuando hacemos lecturas de un fichero en en formto texto algunos datos hay que convertirlos al formato usado por el computador, y para ello usamos fscanf. Asimismo, si queremos escribir en formato texto tendremos que usar fprintf para hacer las conversiones adecuadas.

- **Bonus:**
- Adapta el programa para que use la entrada estándar como fichero de entrada y la salida estándar como fichero de salida, si se pasa "-" como nombre de los ficheros correspondientes.
- Indica algunos comandos de shell que te permitan comprobar el funcionamiento correcto de este cambio.