Práctica: Introducción a la programación de sistemas en Linux

Índice

1 Introducción al entorno de desarrollo

1.1 Objetivos

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

1.2 Requisitos

Para poder realizar con exito 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 configurar Eclipse IDE para 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 alumnos con poca experiencia con el lenguaje C.
- Manual "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.

1.3 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 estandar 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`:

- 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()`
- 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 algun/os error/es. Utilizando un depurador de C (p.ej.,`gdb` o

ddd) el alumno debe encontrar y corregir el/los error/es.

```
#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;
}
```

1.4 Desarrollo de la práctica

En esta práctica el alumno debe crear una herramienta de linea de órdenes llamada *mytar*, capaz de crear un archivo *tarball* que contenga varios ficheros, o de extraer los ficheros de un *tarball* previamente creado. Por ejemplo, la orden:

```
./mytar -cf project1.mtar mytar.c mytar.h mytar_routines.c
```

deberá crear un archivo `project1.mtar` que recopile el contenido de los ficheros `mytar.c`, `mytar.h` y `mytar_routines.c`. Así mismo, la orden:

```
./mytar -xf project1.mtar
```

extraerá el contenido del *tarball* `project1.mtar`, es decir, los ficheros `mytar.c`, `mytar.h` y `mytar_routines.c`.

Nuestra versión del *tarball* consistirá en una cabecera que describirá el contenido del archivo, seguida del contenido binario de cada uno de los ficheros incluidos en él. La cabecera estará formada por un entero de 4 bytes que indicará el número de ficheros almacenados (N), seguido de N descriptores de fichero. Cada uno de estos descriptores se compondrá de una cadena de caracteres (terminada en un byte 0) con el nombre del fichero (su ruta), y un un entero sin signo de cuatro bytes con el tamaño en bytes del fichero. Es decir, la estructura descriptor en memoria sería:

```
typedef struct {
  char* name;
  unsigned int size;
} stHeaderEntry;
```

Para almacenar el nombre del fichero habrá que reservar memoria del heap almacenando en el campo name la dirección de comienzo devuelta por *malloc*.

Volviendo sobre nuestro ejemplo, el fichero `project1.mtar` comenzará con cuatro bytes que representan un entero con valor 3 (el número de ficheros almacenados), seguido de 3 descriptores de fichero, cada uno con el nombre de uno de los ficheros y su tamaño en bytes.

El proyecto *mytar* estará constituido por los siguientes ficheros que proporcionará el profesor:

- *Makefile*: fichero de entrada a la herramienta make para la compilación del proyecto.
- *mytar.h*: contiene las declaraciones de tipos de datos que se usarán en el resto del código.
- *mytar.c*: es el fichero principal, que contiene la función **main**.
- *mytar_routines.c*: fichero en el que se implementan las funciones de tratamiento del *tarball*. Este fichero se da incompleto, y los alumnos deben completarlo siguiendo las indicaciones de los comentarios.

1.5 Script de comprobación

Con el fin de practicar el uso del shell, se pide al alumno que desarrolle su propio script bash para la comprobación de la práctica. El script creará un tarball con el contenido de unos ficheros utilizando nuestro programa mytar, y luego extraerá el contenido y comprobará que los ficheros extraídos son idénticos a los originales. Si todo es correcto terminará devolviendo 0. Si hay errores terminará devolviendo 1. El script deberá seguir el siguiente esquema:

1. Comprobará que el programa `mytar` está en el directorio actual y que es ejecutable. En caso contrario mostrará un mensaje informativo por pantalla y terminará.
2. Comprobará si existe un directorio `tmp` dentro del directorio actual. Si existe lo borrará, incluyendo todo lo que haya dentro de él (mirar la opción `-r` del comando `rm`).
3. Creará un nuevo directorio temporal `tmp` dentro del directorio actual y cambiará a este directorio.
4. Creará tres ficheros (dentro del directorio):
    - `file1.txt`: con el contenido "Hello world!", utilizando la orden `echo` y redirigiendo la salida al fichero.
    - `file2.txt`: con una copia de las 10 primeras líneas del fichero `/etc/passwd`. Se hace fácil utilizando el programa `head` y redirigiendo la salida al fichero.

- `file3.dat`: con un contenido binario aleatorio de 1024 bytes, tomado del dispositivo `/dev/urandom`. De nuevo conviene utilizar `head` con la opción `-c`.

5. Invocará el programa `mytar` que hemos desarrollado, para crear un fichero `filetar.mtar` con el contenido de los tres ficheros anteriores.
6. Creará un directorio `out` (dentro del directorio actual, que debe ser `tmp`) y copiará el fichero `filetar.mtar` al nuevo directorio.
7. Cambiará al directorio `out` y ejecutará el programa `mytar` para extraer el contenido del tarball.
8. Usará el programa `diff` para comparar los fichero extraídos con los originales, que estarán en el directorio anterior (`..`).
9. Si los tres ficheros extraídos son iguales que los originales, volverá al directorio original (`../..`), mostrará el mensaje "Correct" por pantalla y devolverá 0. Si hay algún error, volverá al directorio original, mostrará un mensaje descriptivo por pantalla y devolverá 1.