Sistemas Operativos

Universidad Complutense de Madrid

2022-2023

Práctica 1

Introducción a la programación de sistemas en Linux

Juan Carlos Sáez

Introducción

Objetivos

- Familiarizarse con el entorno de desarrollo de aplicaciones C en LINUX
  - Uso de la biblioteca estándar de C
- Familiarizarse con el manejo básico del shell y aprender a desarrollar *scripts* sencillos

Introducción

Objetivos

- Familiarizarse con el entorno de desarrollo de aplicaciones C en LINUX
  - Uso de la biblioteca estándar de C
- Familiarizarse con el manejo básico del shell y aprender a desarrollar *scripts* sencillos

Requisitos

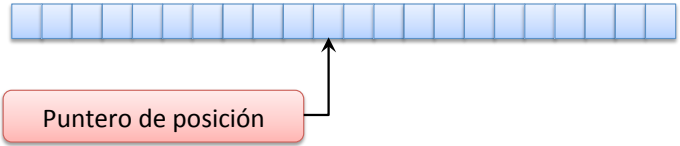
- Leer los siguientes documentos:
  - Manual descriptivo "Entorno de desarrollo C para GNU/Linux"
  - Introducción al entorno de desarrollo
  - Revisión: Programación en C
  - Introducción al shell Bash

Guión Práctica 1: ejercicios

- **Ejercicio 1**: Ejercicio ficheros con biblioteca estándar de C
- **Ejercicio 2**: Depuración
  - GDB (`gdb -tui ./badsort-ptr`)
  - Depurador de VSCode
- **Ejercicio 3**: Ejemplo `show-passwd`
  - `Parsing` de ficheros de texto con funciones biblioteca estándar de C
  - Funciones clave: `getopt()`, `sscanf()` y `strsep()`
  - Reusar ideas de diseño para desarrollo de Ejercicio 4
- **Ejercicio 4**: Desarrollo programa `student-records`
- **Ejercicio 5**: Script BASH de prueba para `student-records`

Visión lógica ficheros

- Un espacio lógico de direcciones contiguas usado para almacenar datos + puntero de posición



- Abrir/crear fichero para acceder a su representación lógica
  - Descriptor del fichero (abstracción programador)
    - Objeto, manejador (`FILE*`) o número
- Operaciones (en base a modo de apertura)
  - lectura, escritura
  - movimiento explícito puntero posición
  - cierre

Operaciones ficheros binarios y de texto

Apertura

```
FILE* fopen(const char *path, const char *mode);
```

Funciones para ficheros de texto

```
int fscanf(FILE * stream, const char * format, ...);
int fprintf(FILE*  stream, const char *  format, ...);
```

Funciones para ficheros binarios

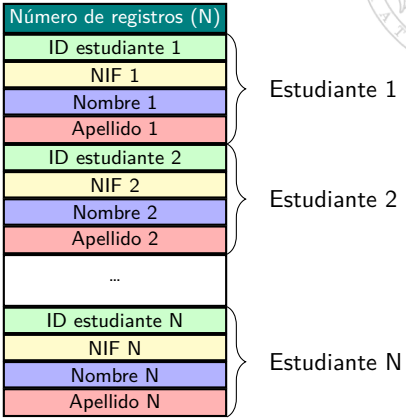
```
size_t fread(void * ptr, size_t size, size_t nitems, FILE * stream);
size_t fwrite(const void * ptr, size_t size, size_t nitems, FILE * stream);
```

Lectura escritura byte a byte

```
int getc(FILE *stream);
int putc(int c, FILE *stream);
```

Ejercicio 4

- Desarrollar programa de gestión de ficheros de registros de estudiantes
  - Fichero binario
- Cada estudiante (estructura) posee:
  1. ID numérico
  2. NIF
  3. Nombre
  4. Apellidos



Programa Ejercicio 4: modo de uso

Modo de uso

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

- `-c` : Crear fichero de estudiantes
  - Ejemplo: `./student-records -f data -c 27:67659034X:Chris:Rock \ 34:78675903J:Antonio:Banderas`
- `-l` : Mostrar todas las entradas fichero de estudiantes (ASCII)
  - Ejemplo: `./student-records -f data -l`
- `-a` : Añadir nuevos registros al final de un fichero de estudiantes existente
  - Ejemplo: `./student-records -f data -a 3:58943056J:Santiago:Segura \ 4:6345239G:Penelope:Cruz`
- `-x` : Mostrar entradas seleccionadas del fichero
  - Filtrado por ID (`-i`) o NIF (`-n`)
  - Ejemplo: `./student-records -f data -q -i 34`

Ejercicio 4: Ejemplo de ejecución

```
## 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

## 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
```

Ejercicio 4: Proyecto C con Makefile

- El proyecto debe construirse de cero, pero reusando ideas de diseño del ejemplo `show-passwd` (Ejercicio 3)
  - Procesamiento de opciones de línea de comandos
  - Representación en memoria de los registros del fichero (array de estructuras)

Estructura recomendada (ficheros)

- `defs.h`: Declaraciones de tipos de datos
- `student-records.c`: implementación de `main()` y de funciones auxiliares
  - Procesamiento de opciones de línea de comandos
- `Makefile`
  - Adaptar Makefile del ejemplo `show-passwd` (Ejercicio 3)

Adaptación Makefile Ejercicio 3

- Establecer valor de la variable `PROG` al nombre del fichero C del programa pero sin la extensión ⇒ `PROG=student-records`

Makefile Ejercicio 3

```
CC = gcc
CFLAGS = -Wall -g

PROG = show-passwd
HEADERS = defs.h
OBJECTS = $(PROG).o

all : $(PROG)

$(PROG) : $(OBJECTS)
    gcc -g -o $(PROG) $(OBJECTS)

%.o : %.c $(HEADERS)
    gcc -c $(CFLAGS) $< -o $@

clean :
    -rm -f $(OBJECTS) $(PROG)
```

Estructura estudiante (student_t)

defs.h

```
#ifndef DEFS_H
#define DEFS_H

#define MAX_CHARS_NIF  9

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

...

#endif
```

Estructura función `main()`

- Se aconseja definir una estructura `options` a medida como en el Ejercicio 3

```
int main(int argc, char *argv[]) {
    struct options options;
    int opt, nr_records;
    /* Initialize default values for options */
    ...

    /* Parse command-line options */
    while((opt = getopt(argc, argv, "hcalqf:i:n:")) != -1) {
        switch(opt) {
            ...
        }
    }

    nr_records=argc-optind;

    /* Do whatever the user asked for */
    switch (options.action){
        ...
    }
    ...
}
```

Funciones auxiliares (I)

Aconsejable definir las siguientes funciones auxiliares ...

- `student_t* parse_records(char* records[], int nr_records);`
  - Convierte la lista de registros (ASCII) pasada como argumento en la línea de comandos (`records`) en un array de estructuras (valor de retorno)
- `int dump_entries(student_t* entries, int nr_entries, FILE* students)`
  - Vuelca al fichero binario ya abierto (students) los registros de estudiantes pasados como parámetro (entries).

Funciones auxiliares (II)

Aconsejable definir las siguientes funciones auxiliares ...

- `student_t* read_student_file(FILE* students, int* nr_entries)`
  - Lee la información de un fichero abierto de registros de estudiantes `students`, y devuelve el número de registros (vía `nr_entries`) así como el array de estructuras correspondiente (valor de retorno)
- `char* loadstr(FILE* students)`
  - Lee una cadena de caracteres del fichero cuyo descriptor se pasa como parámetro
  - La función reserva memoria para la cadena leída. La dirección de memoria donde comienza la cadena se devuelve como valor de retorno.

Orden de implementacion

- No es aconsejable comenzar implementando TODAS las funciones auxiliares

Pasos sugeridos

1. Implementación procesamiento línea de comando en `main()`
2. Implementación opción `-c` (creación)
   - Crear funciones `parse_records()` y `dump_entries()`
   - Verificar que fichero creado correctamente con `xxd`
3. Implementación opcion `-l` (listar)
   - Crear funciones `read_student_file()` y `loadstr()`
4. Implementación opciones `-a` y `-q`

Visualizando el contenido del fichero binario

- Es posible usar un editor hexadecimal, como `xxd` o `ghex2` para visualizar el contenido de un fichero de registros
  - Esto permite detectar problemas en el fichero a simple vista
- Cada línea en la salida de `xxd` muestra 16 bytes tanto en formato hexadecimal como en ASCII
  - Los primeros 4 bytes codifican el número de ficheros en el archivo
  - Nótese que x86 es una arquitectura *little-endian*

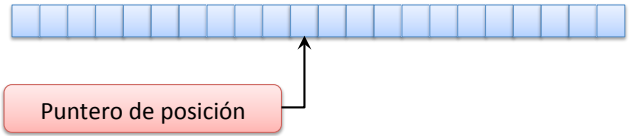
```
usuarioso@debian:~/student-records$ xxd database
00000000: 0200 0000 1b00 0000 3637 3635 3930 3334  ........67659034
00000010: 5800 4368 7269 7300 526f 636b 0022 0000  X.Chris.Rock."..
00000020: 0037 3836 3735 3930 334a 0041 6e74 6f6e  .78675903J.Anton
00000030: 696f 0042 616e 6465 7261 7300            io.Banderas.
```

Pseudocódigo read_student_file()

```
student_t* read_student_file(FILE* students, int* nr_entries)
{
    student_t* entries;
    int entry_count;
    int i=0;

    ... Read the number of registers (N) from student file and
        store it in entry_count ...

    entries=malloc(sizeof(student_t)*entry_count);

    ... Read student records from file and populate entries (array) ...


    /* Store the number of records in the output parameter */
    (*nr_entries)=entry_count;

    return entries;
}
```

Problemas lectura de string desde fichero

- `char* loadstr(FILE* students)`: ha de leer una cadena almacenada en el fichero y retornar un string de C bien formado



- No sabemos longitud del string antes de leerlo
- Procedimento:
  1. Buscar '\0' leyendo caracter a caracter (p.ej., `getc()`) pero sin almacenar cadena en memoria
     - Llevamos la cuenta de número de caracteres hasta llegar a '\0': `k`
  2. Mover puntero de posición fichero al comienzo del string (`fseek()` con `offset=-k`)
  3. Reservar memoria con `malloc()`
  4. Leer *k* caracteres del fichero (incluyendo '\0') usando como destino el nuevo buffer reservado