

Practica 03

DOCENTE	CARRERA	CURSO
MSc. Maribel Molina Barriga	Escuela Profesional de Ingeniería de Software	Sistemas Operativos

GRUPO	TEMA	DURACIÓN
6	Compilacion en C y C++ en Linux	5 horas

Integrantes

- José Carlos Machaca Vera
- Jhosep Alonso Mollapaza Morocco
- Patrick Andres Ramirez Santos

Índice

1. Ejercicios propuestos	2
1.1. Ejercicio 1	2
1.2. Ejercicio 2	7
1.3. Ejercicio 3	8
2. Cuestionario	10
3. Conclusiones	10
4. Recomendaciones	10
Indice Source Code	11
Indice de Capturas de Pantalla	11

1. Ejercicios propuestos

Se deberá de probar, compilar y ejecutar los siguientes códigos:

1.1. Ejercicio 1

Se crea un archivo Cmake para facilitar la compilacion y ejecucion del codigo, este se presenta a continuacion, y que se puede utilizar con los siguientes comandos desde el directorio con los archivos:

```
$ mkdir cmake-build-debug/  
$ cd cmake-build-debug/  
$ cmake .. # Buscar el archivo CMakeLists.txt en el directorio superior  
$ make # Compilar el proyecto  
$ ./E1 # Ejecutar el proyecto
```

Source Code 1: E1/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.27)  
project(E1)  
  
set(CMAKE_CXX_STANDARD 14)  
  
include_directories(.)  
  
add_executable(E1  
    main.cpp  
    LinkedList.cpp  
    LinkedList.h  
    ListNode.cpp  
    ListNode.h)
```

A continuación los archivos que se usaron y los resultados de la compilación y ejecución:

Source Code 2: E1/LinkedList.h

```
#pragma once  
  
#include "ListNode.h"  
#include <iostream>  
  
class LinkedList {  
private:  
    ListNode *_phead;  
  
public:  
    LinkedList();  
    void insert(int n);  
    void print(void);  
  
    void deleteAll(void);  
    void deleteNodes(ListNode* pn);  
};
```

Source Code 3: E1/LinkedList.cpp

```
#include "LinkedList.h"

LinkedList::LinkedList() { _phead = nullptr; }

void LinkedList::insert(int n) {
    if (_phead == nullptr) {
        _phead = new ListNode();
        _phead->_value = n;
        return;
    } else {
        ListNode *pn = new ListNode();
        pn->_value = n;
        ListNode *pnode = _phead;
        while (pnode->_pNext != nullptr && pnode->_pNext->_value < n) {
            pnode = pnode->_pNext;
        }
        if (pnode->_pNext != nullptr && pnode->_value > n) {
            pn->_pNext = _phead;
            _phead = pn;
        }
        if (pnode->_pNext == nullptr) {
            pnode->_pNext = pn;
        } else {
            pn->_pNext = pnode->_pNext;
            pnode->_pNext = pn;
        }
    }
}

void LinkedList::print(void) {
    ListNode *pnodes = _phead;
    while (pnodes->_pNext != nullptr) {
        std::cout << pnodes->_value << ' ';
        pnodes = pnodes->_pNext;
    }
}

void LinkedList::deleteAll(void) {
    if (_phead != nullptr) {
        deleteNodes(_phead);
        _phead = nullptr;
    }
}

void LinkedList::deleteNodes(ListNode *pn) {
    if (pn != nullptr) {
        deleteNodes(pn->_pNext);
        delete pn;
        pn = nullptr;
    }
}
```

```
}  
}
```

Source Code 4: E1/ListNode.cpp

```
#include "ListNode.h"  
  
ListNode::ListNode() {  
    _value = -1;  
    _pNext = nullptr;  
}  
  
ListNode::~~ListNode() {  
}
```

Source Code 5: E1/ListNode.h

```
#pragma once  
  
class ListNode {  
public:  
    int _value;  
    ListNode* _pNext;  
  
    ListNode();  
    ~ListNode();  
};
```

Archivo ejecutable

Este código utiliza los anteriores archivos para generar una Linked List e imprimir sus elementos para finalmente borrarla.

Source Code 6: E1/main.cpp

```
#include <iostream>  
#include <stdlib.h>  
#include "LinkedList.h"  
  
int main() {  
    int max = 10;  
    auto* plist = new LinkedList();  
  
    for (int i = 0; i < max; i++) {  
        int num = rand()%max;  
        plist->insert(num);  
    }  
  
    plist->print();  
    plist->deleteAll();  
    delete(plist);  
  
    return 0;  
}
```

Figura 1: Compilación del ejercicio 1 con CMake

```
~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main) (0.306s)
make
[ 25%] Building CXX object CMakeFiles/E1.dir/main.cpp.o
[ 50%] Building CXX object CMakeFiles/E1.dir/LinkedList.cpp.o
[ 75%] Building CXX object CMakeFiles/E1.dir/ListNode.cpp.o
[100%] Linking CXX executable E1
[100%] Built target E1

~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main) (0.027s)
ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  E1  Makefile

~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main) (0.029s)
./E1
1 3 5 5 6 6 7

~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main)
```

Figura 2: Salida del ejercicio 1 con CMake

```
~/uls-24A/operating_systems/Practica-3/E1 git:(main) (0.029s)
mkdir cmake-build-debug/

~/uls-24A/operating_systems/Practica-3/E1 git:(main) (0.025s)
cd cmake-build-debug/

~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main) (0.723s)
cmake ..
-- The C compiler identification is GNU 13.2.1
-- The CXX compiler identification is GNU 13.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/lib64/ccache/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/lib64/ccache/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.7s)
-- Generating done (0.0s)
-- Build files have been written to: /home/pat/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug

~/uls-24A/operating_systems/Practica-3/E1/cmake-build-debug git:(main)
```

proceso.c

Este código crea un proceso hijo que imprime un mensaje en pantalla.

Source Code 7: E1/proceso.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void) {
    pid_t pid;
    // fork a child process
    pid = fork();
    if (pid < 0) { /* error occurred */
```

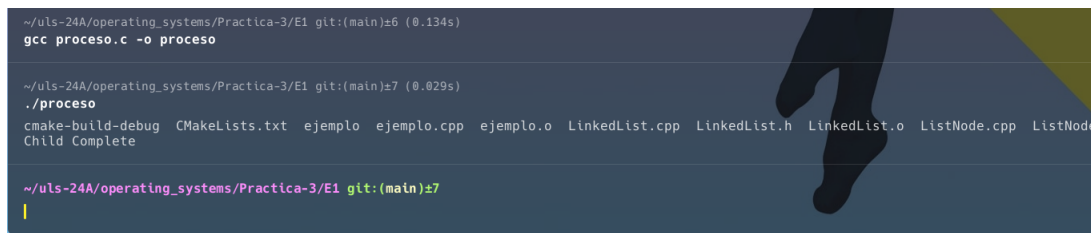
```

    fprintf(stderr, "Fork Failed");
    return 1;
} else if (pid == 0) { /* child process */
    execlp("/bin/ls", "ls", NULL);
} else { /* parent process */
    // parent will wait for the child to complete
    wait(NULL);
    printf("Child Complete");
}

return 0;
}

```

Figura 3: Compilación y ejecución de proceso.c con GCC



```

~/uls-24A/operating_systems/Practica-3/E1 git:(main)±6 (0.134s)
gcc proceso.c -o proceso

~/uls-24A/operating_systems/Practica-3/E1 git:(main)±7 (0.029s)
./proceso
cmake-build-debug CMakeLists.txt ejemplo ejemplo.cpp ejemplo.o LinkedList.cpp LinkedList.h LinkedList.o ListNode.cpp ListNode.h
Child Complete

~/uls-24A/operating_systems/Practica-3/E1 git:(main)±7

```

ejemplo.cpp

Este código utiliza polimorfismo para detectar el tipo de un objeto e imprimir una función específica en base a ello:

Source Code 8: E1/ejemplo.cpp

```

#include <functional>
#include <iostream>

class Laboratorio {
    int num;
};

class Practica {
    int a;
    Laboratorio lab;

public:
    operator Laboratorio() { return lab; }

    operator int() { return a; }
};

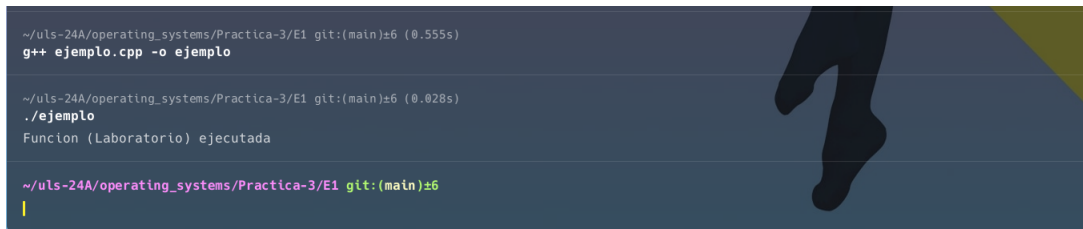
void function(int a) { std::cout << "funcion (int) ejecutada"; }

void function(Laboratorio la) {
    std::cout << "Funcion (Laboratorio) ejecutada";
}

```

```
}  
  
int main() {  
    Practica p;  
    function((Laboratorio)p);  
    return 0;  
}
```

Figura 4: Compilación y ejecución de ejemplo.cpp con G++



```
~/uls-24A/operating_systems/Practica-3/E1 git:(main)±6 (0.555s)  
g++ ejemplo.cpp -o ejemplo  
  
~/uls-24A/operating_systems/Practica-3/E1 git:(main)±6 (0.028s)  
./ejemplo  
Funcion (Laboratorio) ejecutada  
  
~/uls-24A/operating_systems/Practica-3/E1 git:(main)±6  
|
```

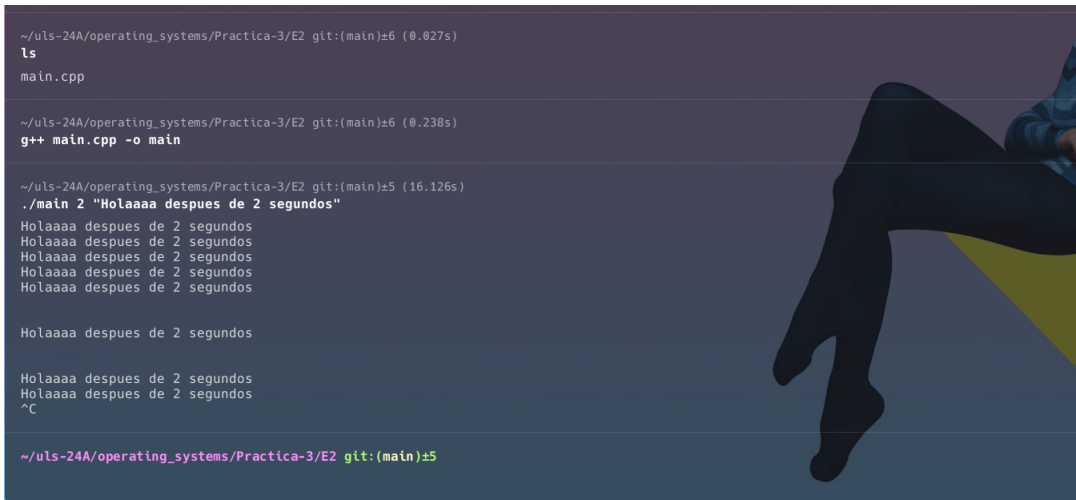
1.2. Ejercicio 2

Este código recibe 2 argumentos vía línea de comandos, el primero es un número de segundos y el segundo es un mensaje, el código espera el tiempo definido por el primer argumento y luego imprime el mensaje en pantalla de forma indefinida.

Source Code 9: E2/main.cpp

```
#include <stdio.h>  
#include <string.h>  
#include <unistd.h>  
  
int main(int argc, char *argv[])  
{  
    int segundos;  
    if (argc != 3) {  
        fprintf(stderr, "Uso: %s <segundos> <mensaje>\n", argv[0]);  
        return 1;  
    }  
    sscanf(argv[1], "%d", &segundos);  
    while (1) {  
        sleep(segundos);  
        printf("%s\n", argv[2]);  
    }  
    return 0;  
}
```

Figura 5: Compilación y ejecución del ejercicio 2 con G++



```
~/uls-24A/operating_systems/Practica-3/E2 git:(main)±6 (0.027s)
ls
main.cpp

~/uls-24A/operating_systems/Practica-3/E2 git:(main)±6 (0.238s)
g++ main.cpp -o main

~/uls-24A/operating_systems/Practica-3/E2 git:(main)±5 (16.126s)
./main 2 "Holaaaaa despues de 2 segundos"
Holaaaaa despues de 2 segundos
Holaaaaa despues de 2 segundos
Holaaaaa despues de 2 segundos
Holaaaaa despues de 2 segundos
Holaaaaa despues de 2 segundos

Holaaaaa despues de 2 segundos

Holaaaaa despues de 2 segundos
Holaaaaa despues de 2 segundos
^C

~/uls-24A/operating_systems/Practica-3/E2 git:(main)±5
```

1.3. Ejercicio 3

En este ejercicio se utiliza un Makefile para compilar el archivo mensaje.c y los archivos que este requiere para ejecutarse, para ejecutarlo se deben seguir los siguientes comandos, luego se muestra el contenido del Makefile:

Source Code 10: E3/salida_alt.h

```
#ifndef __SALIDA_ALT__
#define __SALIDA_ALT__

void muestra(char *);

#endif
```

Source Code 11: E3/salida_alt.c

```
#include <stdio.h>

void muestra(char *msg) { printf("Mensaje: %s\n", msg); }
```

Source Code 12: E3/mensaje.c

```
#include "salida_alt.h"

int main() {
    muestra("Muestra este bonito mensaje por la salida estándar");
    return 0;
}
```

```
$ make # Compilar el proyecto con el makefile
$ ./mensaje # Ejecutar el proyecto
```


Source Code 13: E3/Makefile

```
mensaje: mensaje.o salida_alt.o
    gcc -o mensaje mensaje.o salida_alt.o

mensaje.o: mensaje.c salida_alt.h
    gcc -c -g mensaje.c

salida_alt.o: salida_alt.c salida_alt.h
    gcc -c -g salida_alt.c
```

Figura 6: Compilación y ejecución del ejercicio 3 con MakeFile



```
~/uls-24A/operating_systems/Practica-3/E3 git:(main)±7 (15.118s)
nvim Makefile

~/uls-24A/operating_systems/Practica-3/E3 git:(main)±7 (0.031s)
ls
Makefile  mensaje.c  mensaje.o  salida_alt.c  salida_alt.h

~/uls-24A/operating_systems/Practica-3/E3 git:(main)±7 (0.141s)
make
gcc -c -g salida_alt.c
gcc -o mensaje mensaje.o salida_alt.o

~/uls-24A/operating_systems/Practica-3/E3 git:(main)±8 (0.027s)
./mensaje
Mensaje: Muestra este bonito mensaje por la salida estándar

~/uls-24A/operating_systems/Practica-3/E3 git:(main)±8
```

2. Cuestionario

1. ¿Cuál es la diferencia entre compilar con GCC y G++?
GCC es el compilador de GNU para C y otros lenguajes, mientras que G++ es específicamente el compilador de GNU para C++. La principal diferencia es en el lenguaje de programación para el cual están optimizados, además que G++ automáticamente vincula las bibliotecas estándar de C++ que no están vinculadas por GCC.
2. ¿En qué se diferencia el archivo generado .o contra un .exe?
 - Un archivo .o es un archivo de objeto, resultado de compilar un archivo de código fuente pero sin enlazarlo. Los archivos de objeto contienen código máquina (binario), y no son ejecutables por sí mismos, ya que dependen de otros archivos.
 - Un archivo .exe es un archivo ejecutable completo, resultado del proceso de enlazado de uno o varios archivos de objeto, junto con todas las bibliotecas necesarias para formar un programa que puede ser ejecutado por el sistema operativo.
3. ¿Explique cuál es la diferencia entre el proceso de compilación y enlazado? Proponga un ejemplo en el que haga uso de más de un archivo donde se evidencie ambos procesos.
 - La compilación es la conversión del código fuente a código objeto, mientras que el enlazado es la unión de varios archivos de código objeto en un ejecutable.
 - En el Ejemplo N°2 se puede ver cómo el Makefile enlaza varios archivos para compilar correctamente el programa final.

3. Conclusiones

- A través de los ejercicios realizados, se observó la importancia de la correcta utilización de las herramientas de compilación como GCC y G++ para manejar adecuadamente las dependencias y particularidades de cada lenguaje.
- La implementación de archivos CMakeLists.txt y Makefile en los ejercicios demostró cómo la automatización del proceso de compilación y enlazado mejora la eficiencia del desarrollo y permite una gestión más clara de los proyectos grandes.
- El manejo correcto de múltiples archivos en el proceso de compilación y enlazado fue crucial para la construcción exitosa de aplicaciones, destacando la importancia de entender ambos procesos para resolver dependencias y errores de enlazado.

4. Recomendaciones

- Asegurar que cada operación de new tenga su correspondiente delete para evitar fugas de memoria y que esta operación no se realice varias veces sobre el mismo objeto.
- Es recomendable utilizar herramientas como CMake y Makefile para gestionar proyectos más complejos, ya que simplifican y estandarizan el proceso de compilación y enlazado en diferentes entornos de desarrollo.
- Mantenerse actualizado sobre las mejores prácticas y nuevas características de las herramientas de compilación puede ayudar a optimizar el rendimiento y la eficiencia del código.
- Implementar pruebas durante y después del proceso de desarrollo para asegurarse de que el código funciona correctamente.

Indice Source Code

1.	E1/CMakeLists.txt	2
2.	E1/LinkedList.h	2
3.	E1/LinkedList.cpp	3
4.	E1/ListNode.cpp	4
5.	E1/ListNode.h	4
6.	E1/main.cpp	4
7.	E1/proceso.c	5
8.	E1/ejemplo.cpp	6
9.	E2/main.cpp	7
10.	E3/salida_alt.h	8
11.	E3/salida_alt.c	8
12.	E3/mensaje.c	8
13.	E3/Makefile	9

Indice de Capturas de Pantalla

1.	Compilación del ejercicio 1 con CMake	5
2.	Salida del ejercicio 1 con CMake	5
3.	Compilación y ejecución de proceso.c con GCC	6
4.	Compilación y ejecución de ejemplo.cpp con G++	7
5.	Compilación y ejecución del ejercicio 2 con G++	8
6.	Compilación y ejecución del ejercicio 3 con MakeFile	9