

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 | 3 |
| 1.3. Ejercicio 3 | 4 |
| 2. Cuestionario | 5 |
| 2.1. ¿Cuál es la diferencia entre compilar con GCC y G++? | 5 |
| 2.2. ¿En que se diferencia el archivo generado .o contra un .exe | 5 |
| 2.3. ¿Explique cuál es la diferencia entre el proceso de compilación y enlazado? Proponga un ejemplo en haga uso de más de un archivo donde se evidencie ambos procesos. | 5 |
| 3. Conclusiones | 6 |
| 4. Recomendaciones | 6 |

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: Contenidos Makefile

```
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)
```

proceso.cpp

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

Source Code 2: Contenidos Makefile

```
#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;  
}
```

ejemplo.cpp

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

Source Code 3: Contenidos Makefile

```
#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;  
}
```

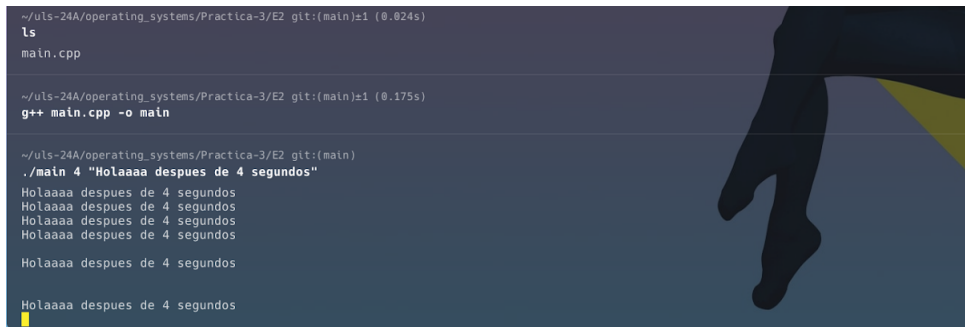
1.2. Ejercicio 2

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 4: 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;
}
```



```
~/uls-24A/operating_systems/Practica-3/E2 glt:(main)±1 (0.024s)
ls
main.cpp

~/uls-24A/operating_systems/Practica-3/E2 glt:(main)±1 (0.175s)
g++ main.cpp -o main

~/uls-24A/operating_systems/Practica-3/E2 glt:(main)
./main 4 "Holaaaaa despues de 4 segundos"
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
```

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:

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

Source Code 5: 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
```

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

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

~/uls-24A/operating_systems/Practica-3/E2 git:(main) ./main 4 "Holaaaaa despues de 4 segundos"
./main 4 "Holaaaaa despues de 4 segundos"
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
Holaaaaa despues de 4 segundos
```

2. Cuestionario

2.1. ¿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.

2.2. ¿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.

2.3. ¿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

- Utilizando la salida de Valgrind, identificamos problemas críticos de manejo de memoria, como lecturas y liberaciones inválidas, y fugas de memoria relacionadas con las operaciones en tus clases ListNode y LinkedList.
- 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

- Se sugirió evitar métodos recursivos de eliminación en estructuras de datos que potencialmente podrían ser muy largas para prevenir desbordamientos de pila. Se enfatizó la importancia de asegurar que cada operación de new tenga su correspondiente delete para evitar fugas de memoria.
- Es recomendable utilizar herramientas como CMake 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.
- Realizar pruebas exhaustivas durante y después del proceso de desarrollo para asegurarse de que el código funciona correctamente en diferentes plataformas y configuraciones de compilación.

Indice Source Code

| | | |
|----|-------------------------------|---|
| 1. | Contenidos Makefile | 2 |
| 2. | Contenidos Makefile | 2 |
| 3. | Contenidos Makefile | 3 |
| 4. | E2/main.cpp | 3 |
| 5. | E3/Makefile | 4 |