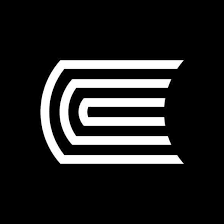
“AÑO DE LA RECUPERACIÓN Y CONSOLIDACIÓN DE LA ECONOMÍA PERUANA”



Evaluación 01 - Consolidado 02

**"Sistema de Gestión de Procesos"**

**INFORME DEL PROYECTO: SISTEMA DE GESTIÓN DE PROCESOS**

**Autores:**

**Fuentes Echavigurin Marco Antonio**

**Rodriguez Alania Leonel Lorenzo**

**Tornero Bermudez Diego Alí Gadiel**

**Veliz Cochachi Christopher Jesús**

**NRC: 29902**

# **CAPÍTULO 1: ANÁLISIS DEL PROBLEMA**

## **1.1 Descripción del Problema**

En la vida diaria de una computadora, constantemente se están ejecutando tareas o "procesos". Algunos de estos procesos requieren mucha atención del procesador, otros menos; algunos necesitan reservar más memoria, otros apenas unos pocos megabytes. En un sistema operativo real, existen módulos complejos que se encargan de decidir qué proceso se ejecuta primero, cuánta memoria se le asigna, y cómo se gestionan todos estos recursos de forma eficiente.

Como estudiantes de ingeniería, se nos plantea el desafío de replicar una versión simplificada de este sistema: un "Sistema de Gestión de Procesos" que simule estas operaciones fundamentales utilizando las herramientas vistas en clase, en particular estructuras de datos dinámicas lineales como listas enlazadas, colas y pilas.

Este sistema debe permitirnos:

* Registrar nuevos procesos con información básica como su ID, nombre, prioridad y memoria requerida.
* Buscar y modificar procesos cuando sea necesario.
* Simular la ejecución de procesos según su prioridad, como lo haría una CPU.
* Administrar la memoria que usan los procesos, agregándolos o quitándolos de una pila.

El objetivo principal no es solo lograr que el sistema funcione, sino entender profundamente cómo estas estructuras nos ayudan a resolver un problema del mundo real. Así, este proyecto representa una oportunidad concreta para aplicar conocimientos teóricos en una situación práctica, desarrollar habilidades de programación estructurada y colaborar en equipo como futuros profesionales de la ingeniería.

## **1.2 Requerimientos del Sistema**

### **Requerimientos Funcionales**

**Registro de Procesos:**

* El sistema debe permitir ingresar nuevos procesos. Cada proceso se registrará con la siguiente información:
* **ID:** Un identificador único.
* **Nombre:** La denominación del proceso.
* **Prioridad:** Un valor numérico que indica la importancia del proceso (por ejemplo, de 1 a 10).
* **Memoria Usada:** La cantidad de memoria que el proceso necesita, en megabytes.

**Búsqueda y Modificación:**

* **Búsqueda:** Se debe poder buscar y localizar un proceso por su ID de forma rápida y sencilla.
* **Modificación:** Se debe permitir actualizar la prioridad de un proceso, lo que ayudará a reflejar cambios de estado o importancia en tiempo real.

**Eliminación de Procesos:**

* El sistema debe poder eliminar un proceso ya registrado, removiéndolo de la estructura de datos y liberando sus recursos asociados.

**Gestión de CPU:**

* **Encolamiento:** Se debe encolar procesos para la CPU según su prioridad, de modo que los procesos de mayor prioridad sean atendidos primero.
* **Ejecución:** El sistema debe "ejecutar" el proceso ubicado al frente de la cola, retirándolo una vez que comienza su ejecución.
* **Visualización:** Siempre se debe poder visualizar el estado actual de la cola, mostrando los procesos pendientes de ejecución.

**Gestión de Memoria:**

* **Asignación de Memoria:** Se debe simular la asignación de memoria a un proceso mediante una pila. Al asignar, el proceso se inserta en la parte superior de la pila.
* **Liberación de Memoria:** Se debe permitir liberar memoria retirando el último proceso que se agregó, siguiendo el comportamiento típico de una pila (Last In, First Out).
* **Visualización y Cálculo:** El sistema debe mostrar el contenido actual de la pila (estado de la memoria) y calcular la cantidad total de memoria actualmente utilizada.

**Persistencia de Datos:**

* El sistema debe guardar la información de los procesos en un archivo externo (por ejemplo, un archivo de texto) al finalizar la ejecución.
* Al iniciar el sistema, se deben cargar los procesos guardados previamente, permitiendo la continuidad de la información entre sesiones.

### **Requerimientos No Funcionales**

**Lenguaje de Programación:**

* El sistema debe desarrollarse en C++.

**Implementación de Estructuras de Datos:**

* Las estructuras dinámicas (lista enlazada, pila y cola) deben ser implementadas desde cero, sin recurrir a bibliotecas predefinidas. Esto asegurará que se comprenden a fondo su funcionamiento y se demuestre la capacidad de construirlas de manera manual.

**Interfaz de Usuario:**

* La interacción con el sistema se realizará a través de una interfaz de consola.
* Debe incluir un menú intuitivo y bien organizado, que permita al usuario acceder a todas las funcionalidades (registro, búsqueda, modificación, encolamiento, ejecución, asignación y liberación de memoria, etc.) de manera clara y sencilla.

## **1.3 Estructuras de Datos Propuestas**

Las estructuras de datos seleccionadas para este proyecto no son aleatorias: fueron escogidas porque se alinean perfectamente con la lógica que un sistema operativo real utilizaría, y porque nos permiten poner en práctica los conceptos aprendidos en clase de una manera concreta y funcional.

* **Listas**: Se utilizan para almacenar todos los procesos registrados en el sistema. Permiten agregar, buscar, modificar o eliminar procesos de forma sencilla. Son ideales para mantener un registro completo y flexible de los procesos disponibles.
* **Colas**: Se emplean para simular la planificación de la CPU. Los procesos son encolados según su prioridad y se ejecutan en ese orden. Esta estructura permite reflejar el comportamiento real de la administración de procesos en sistemas operativos.
* **Pilas**: Se utilizan para la gestión de memoria. Cuando un proceso necesita memoria, se "empuja" a la pila, y cuando libera memoria, se "saca". Esto permite representar el uso temporal de recursos, siguiendo una lógica de último en entrar, primero en salir (LIFO).

Estas tres estructuras trabajan en conjunto para ofrecernos un panorama completo de la gestión de procesos: desde su registro, pasando por su ejecución, hasta el uso eficiente de la memoria. El resultado es un sistema simple pero muy representativo del funcionamiento real de un sistema operativo.

## **1.4 Justificación de la Elección**

La selección de estas estructuras de datos se basa en sus propiedades intrínsecas y en cómo se adaptan a las necesidades específicas del sistema:

* **Listas Enlazadas:** Permiten una inserción y eliminación flexible, ya que no requieren el desplazamiento de grandes bloques de datos. Esto es esencial en aplicaciones donde la frecuencia de creación y borrado de registros es alta, facilitando la administración continua de los procesos.
* **Colas con Prioridad:** Reflejan con precisión el proceso de planificación de la CPU. Al organizar los procesos según su prioridad, se garantiza que los procesos más críticos sean atendidos en primer lugar, lo cual es fundamental para optimizar el rendimiento y la eficiencia del sistema.
* **Pilas:** Son la estructura ideal para simular la gestión temporal de la memoria. La naturaleza LIFO de las pilas se ajusta perfectamente a escenarios como la gestión de llamadas a funciones o la asignación de memoria que se libera en un orden inverso al de su asignación, asegurando un uso coherente y predecible de los recursos.

En resumen, la combinación de listas, colas y pilas no solo nos permite desarrollar un sistema funcional, sino que también nos ofrece una experiencia práctica de cómo cada estructura contribuye a la solución de problemas complejos dentro de un entorno operativo. Esto nos ayuda a consolidar los conceptos teóricos y a comprender la importancia de elegir la estructura adecuada para cada necesidad específica

# **CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN**

## **2.1 Descripción de Estructuras de Datos y Operaciones**

Para desarrollar un sistema funcional y educativo como el "Sistema de Gestión de Procesos", es fundamental elegir las estructuras de datos adecuadas. A continuación, se explica detalladamente cómo se utilizaron las **listas**, **colas** y **pilas**, y qué operaciones se implementaron con cada una.

### **Lista enlazada (gestión de procesos)**

La lista enlazada es una estructura donde cada elemento (llamado "nodo") contiene los datos de un proceso y un puntero al siguiente nodo. Se usó para mantener un registro de todos los procesos creados en el sistema.

**Operaciones principales:**

* **Agregar proceso:** Inserta un nuevo nodo al final de la lista. Esto permite mantener el orden de creación.
* **Buscar por ID:** Recorre la lista nodo por nodo hasta encontrar el ID solicitado.
* **Modificar prioridad:** Una vez encontrado un proceso, se actualiza su nivel de prioridad.
* **Eliminar proceso:** Se remueve un nodo de la lista, ajustando los punteros para no romper la cadena.

Esta estructura permite que los procesos se almacenen de forma flexible, sin un tamaño fijo, y que se pueda trabajar con ellos de manera dinámica.

### **Cola con prioridad (planificación de CPU)**

La cola con prioridad simula el comportamiento de una CPU que atiende primero los procesos más importantes. En lugar de funcionar sólo como una cola FIFO (First In, First Out), aquí se da preferencia a procesos con mayor prioridad.

**Operaciones principales:**

* **Encolar proceso:** El proceso se inserta en la posición correspondiente según su prioridad, de mayor a menor.
* **Ejecutar proceso:** Se toma el nodo del frente de la cola (el de mayor prioridad) y se elimina, simulando su ejecución.
* **Mostrar cola:** Muestra todos los procesos pendientes, en orden de ejecución.

Esta estructura refleja fielmente un algoritmo de planificación por prioridades, común en sistemas operativos reales.

### **Pila (gestión de memoria)**

La pila es una estructura que sigue la lógica LIFO (Last In, First Out), es decir, el último elemento que se agrega es el primero en salir. Se utilizó para simular la asignación y liberación de bloques de memoria.

**Operaciones principales:**

* **Asignar memoria (push):** Se agrega el proceso al tope de la pila, simulando que se le ha asignado memoria.
* **Liberar memoria (pop):** Se elimina el nodo del tope, liberando la memoria asignada más recientemente.
* **Mostrar estado:** Recorre la pila para mostrar qué procesos están ocupando memoria actualmente.
* **Calcular memoria total:** Suma la memoria usada por todos los procesos en la pila.

La pila permite visualizar de forma clara cómo se gestiona la memoria temporal de los procesos, como en llamadas a funciones o tareas breves.

## **2.2 Algoritmos Principales (Pseudocódigo)**

### **Módulo: Gestión de Procesos (Lista)**

#### **Agregar Proceso**

Inicio

Solicitar ID, nombre, prioridad y memoria

Si ID ya existe en la lista:

Mostrar mensaje de error

Sino:

Crear nodo nuevo con los datos

Insertar nodo al final de la lista

Fin

#### **Buscar Proceso por ID**

Inicio

Solicitar ID

Recorrer lista desde el inicio

Si se encuentra el ID:

Mostrar los datos del proceso

Sino:

Mostrar mensaje de no encontrado

Fin

#### **Modificar Prioridad**

Inicio

Solicitar ID del proceso

Buscar nodo en la lista

Si se encuentra:

Solicitar nueva prioridad

Asignar nueva prioridad al nodo

Sino:

Mostrar mensaje de no encontrado

Fin

#### **Eliminar Proceso**

Inicio

Solicitar ID del proceso

Buscar el nodo en la lista

Si se encuentra:

Eliminar el nodo de la lista

Liberar memoria

Sino:

Mostrar mensaje de no encontrado

Fin

### **Módulo: CPU (Cola con prioridad)**

#### **Encolar Proceso por Prioridad**

Inicio

Solicitar ID del proceso

Buscar el proceso en la lista

Si se encuentra:

Crear nodo nuevo

Insertar en la cola según prioridad descendente

Sino:

Mostrar mensaje de error

Fin

#### **Ejecutar Proceso**

Inicio

Si la cola está vacía:

Mostrar mensaje de cola vacía

Sino:

Mostrar datos del proceso al frente

Eliminar el nodo del frente de la cola

Fin

#### **Mostrar Cola**

Inicio

Si la cola está vacía:

Mostrar mensaje de cola vacía

Sino:

Recorrer la cola y mostrar los procesos en orden

Fin

### **Módulo: Memoria (Pila)**

#### **Asignar Memoria (Push)**

Inicio

Solicitar ID del proceso

Buscar el proceso en la lista

Si se encuentra:

Crear nodo nuevo con los datos

Insertar nodo al tope de la pila

Sino:

Mostrar mensaje de error

Fin

#### **Liberar Memoria (Pop)**

Inicio

Si la pila está vacía:

Mostrar mensaje de pila vacía

Sino:

Mostrar proceso del tope

Eliminar nodo del tope

Fin

#### **Mostrar Pila**

Inicio

Si la pila está vacía:

Mostrar mensaje de pila vacía

Sino:

Recorrer pila desde el tope

Mostrar ID, nombre y memoria de cada proceso

Fin

#### **Mostrar Memoria Total**

Inicio

Inicializar total en 0

Recorrer pila sumando memoria de cada nodo

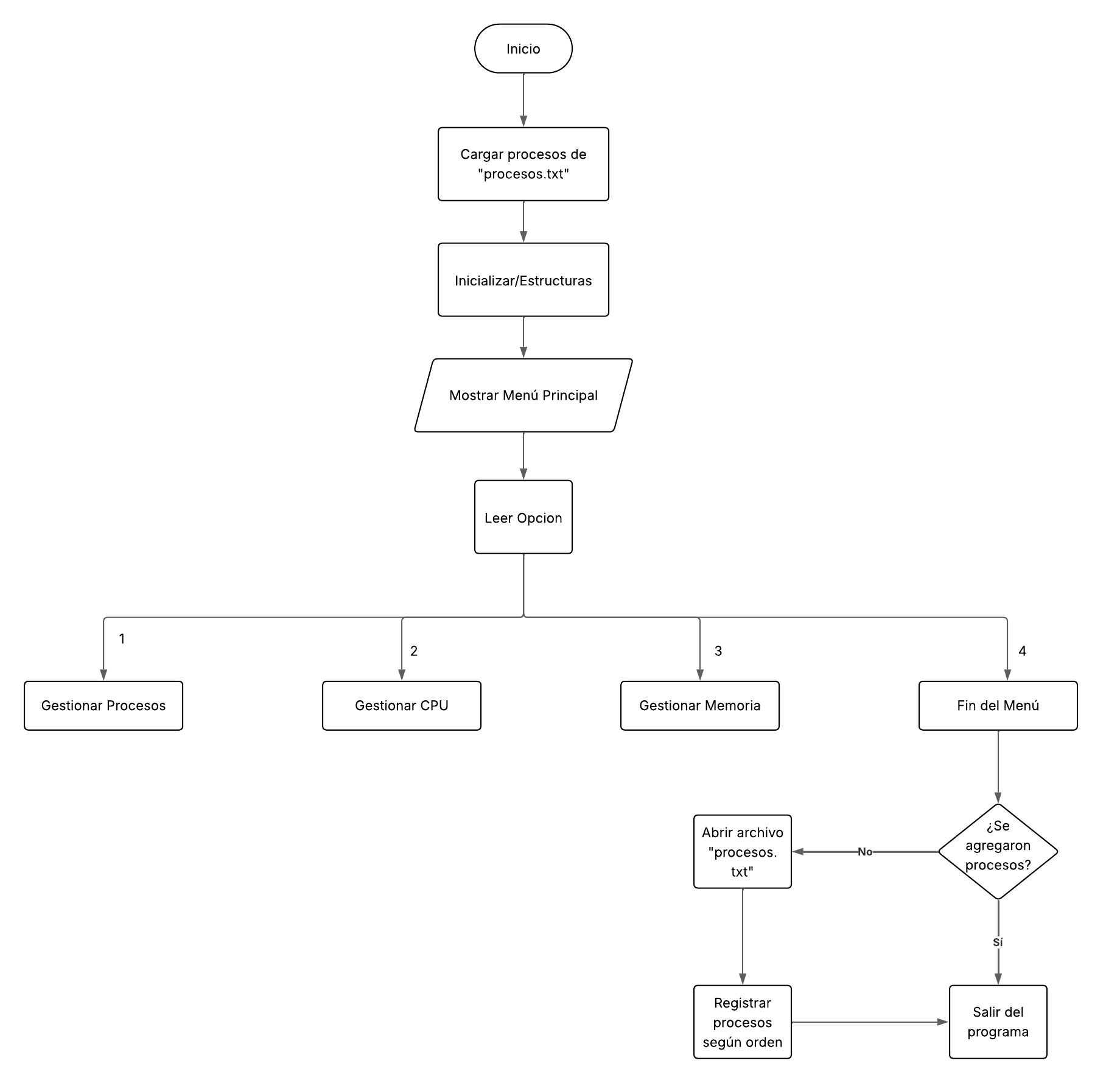
Mostrar total acumulado

Fin

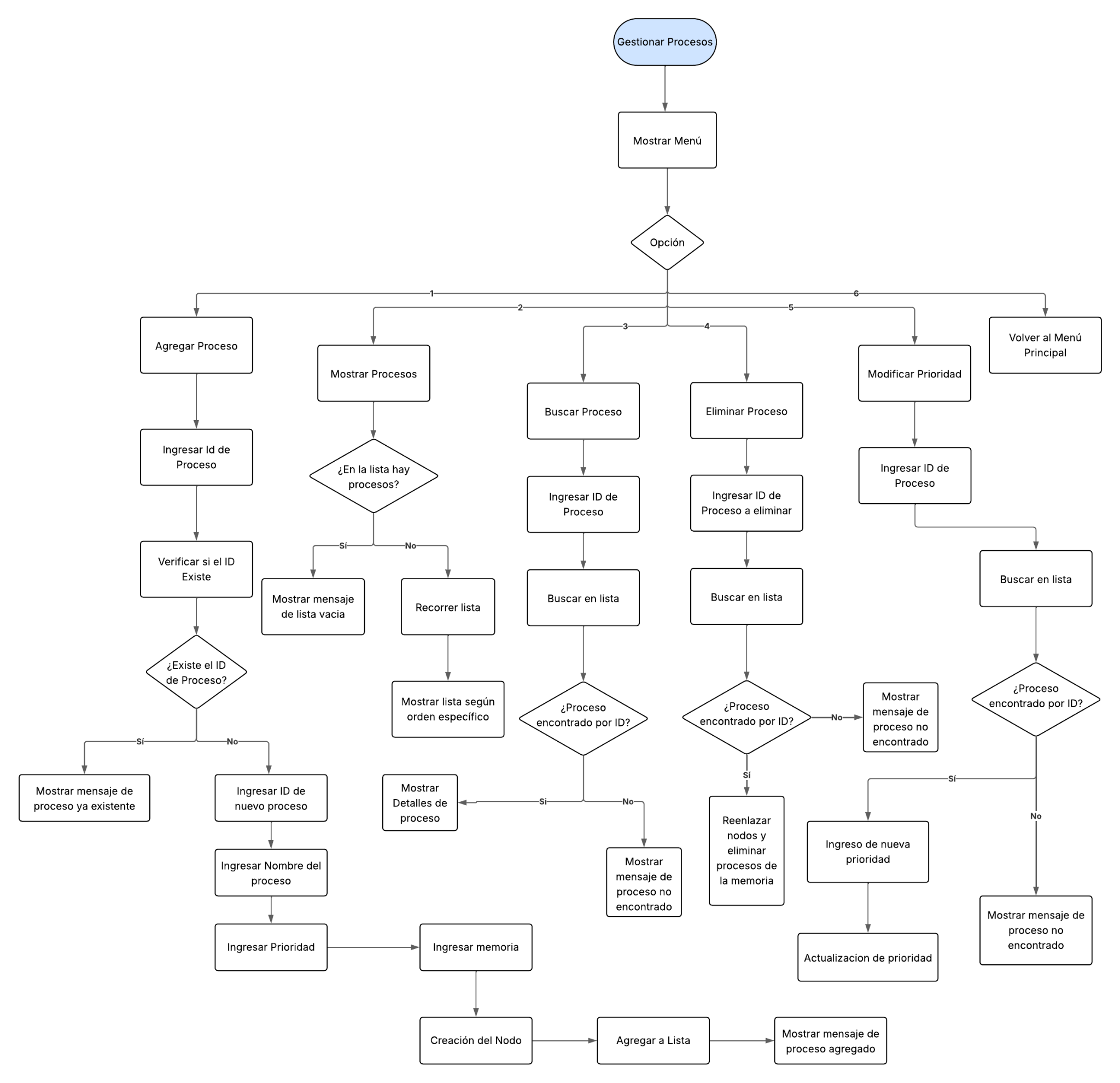
## **2.3 Diagramas de Flujo**

<https://lucid.app/lucidchart/6264498a-9ad1-428d-9df6-661d94e3cacd/edit?viewport_loc=962%2C-2137%2C13862%2C6187%2C0_0&invitationId=inv_45e3b14a-b9ba-4bfd-8a38-678956cd0a93>

### **Menú Principal**



### **Gestión de Procesos (Lista)**



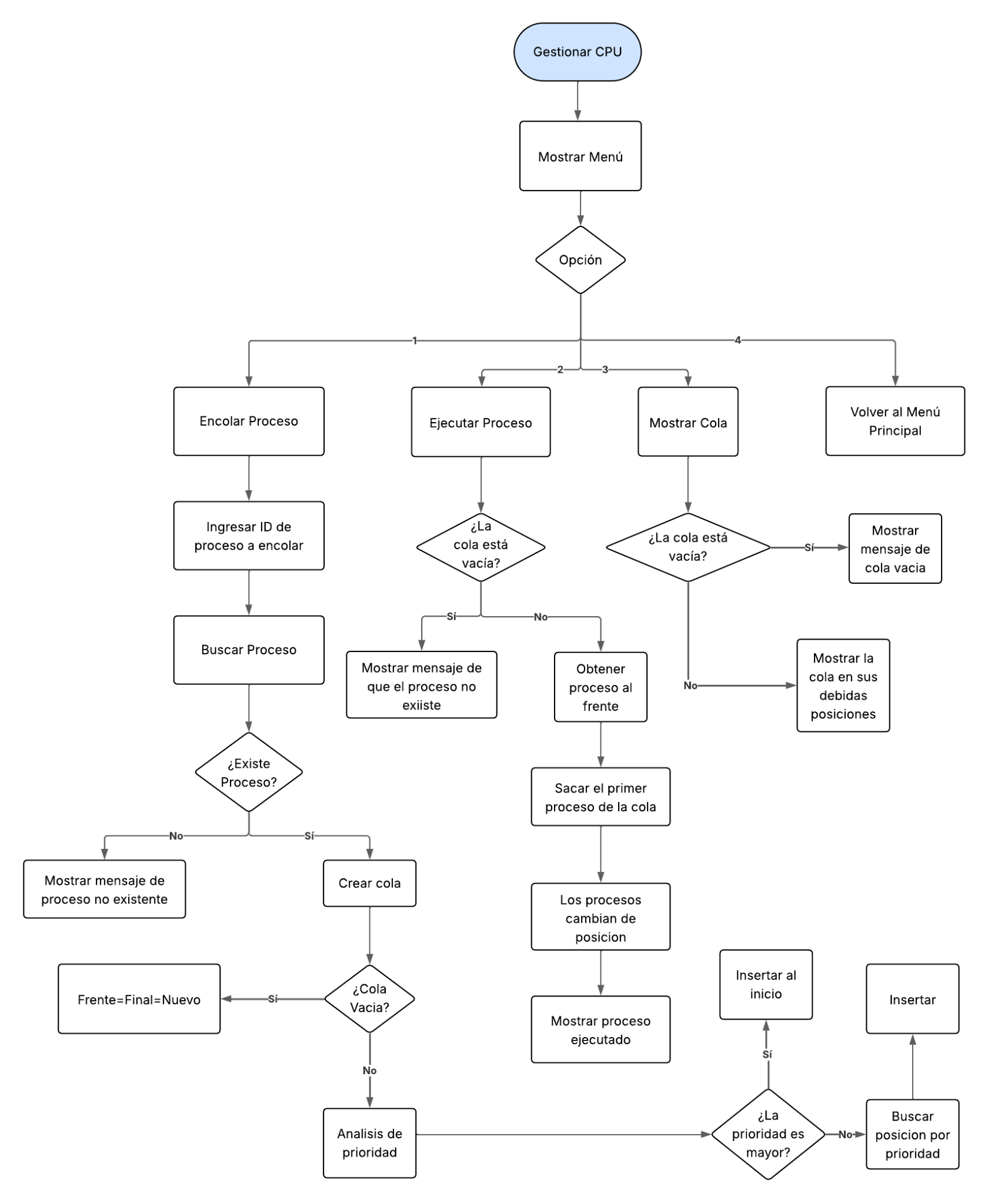
### 

### 

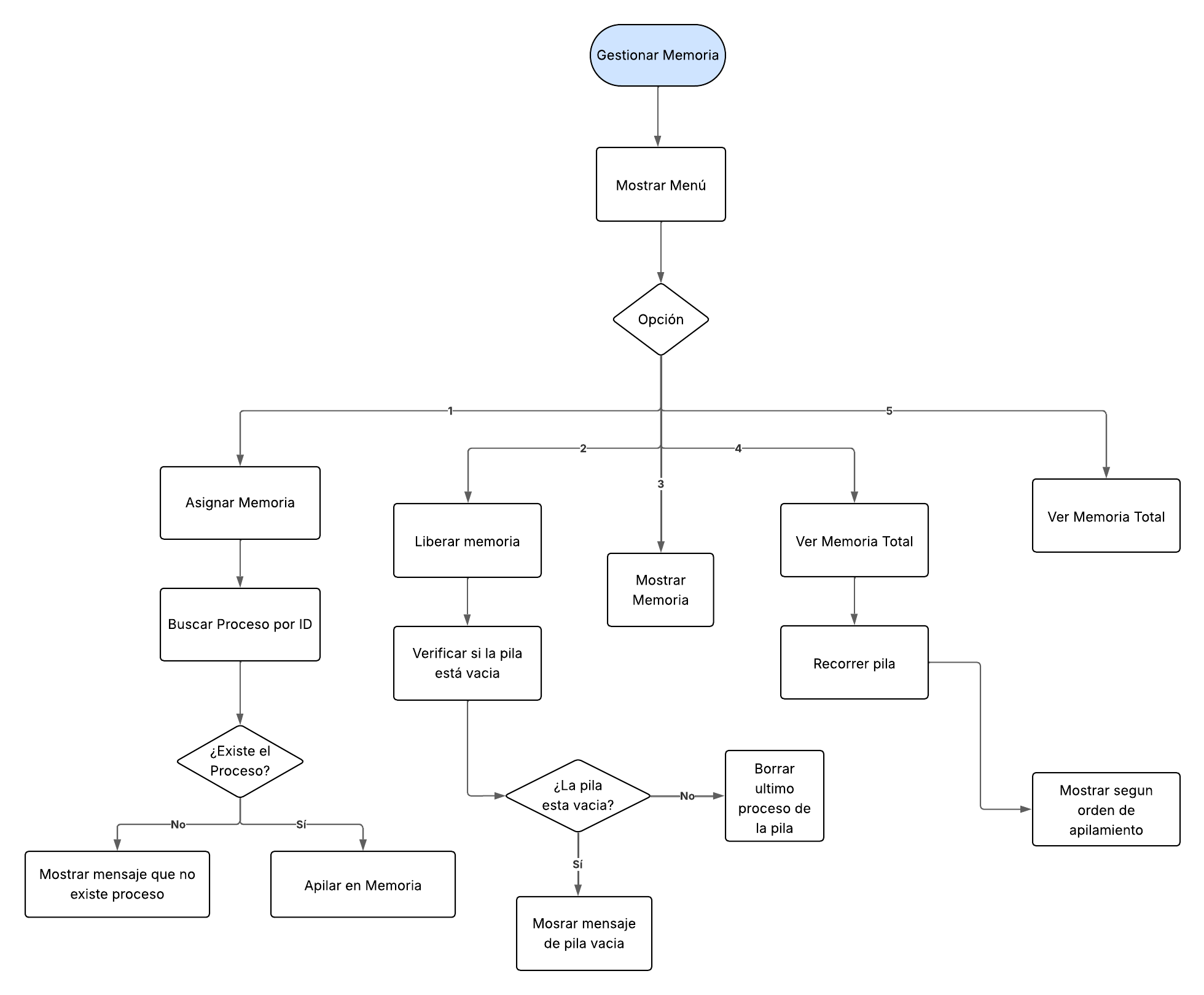
### 

### 

### **🔹 Gestión de CPU (Cola con prioridad)**



**Gestión de Memoria (Pila)**

**

## **2.4 Justificación del Diseño**

* El diseño del Sistema de Gestión de Procesos se fundamenta en principios clave de modularidad, eficiencia, simplicidad y claridad en la interacción. La finalidad de este diseño no solo es cumplir con los requerimientos funcionales establecidos, sino también proporcionar una estructura pedagógica que permita a los estudiantes de ingeniería comprender y aplicar el uso de estructuras de datos dinámicas.
* La modularidad se refleja en la separación clara de las funcionalidades: cada operación (registro, búsqueda, modificación, encolamiento, ejecución, asignación de memoria, entre otras) está encapsulada en funciones independientes. Esto permite mantener un código organizado, facilitar el trabajo colaborativo, y realizar mantenimiento o ampliaciones sin alterar el funcionamiento global del sistema.
* En cuanto a eficiencia y simplicidad, se optó por utilizar estructuras lineales como listas enlazadas, pilas y colas, implementadas manualmente, sin el uso de bibliotecas predefinidas. Esta decisión tiene un doble propósito: primero, asegurar el cumplimiento de los objetivos académicos de comprensión y programación de estructuras desde cero; segundo, permitir un control total sobre el comportamiento del sistema para garantizar su correcto funcionamiento.
* El diseño también contempla la persistencia de datos, mediante la carga y guardado automático de procesos en archivos de texto. Esto asegura la continuidad de la información entre sesiones, simulando una gestión de datos permanente sin la necesidad de una base de datos compleja.
* Finalmente, se diseñó una interfaz por consola clara y estructurada por menús, con validación de datos de entrada que previene errores comunes del usuario. Se consideran escenarios de error como entradas inválidas, duplicación de IDs o valores fuera de rango, lo que refuerza la robustez y facilidad de uso del sistema.

# **CAPÍTULO 3: SOLUCIÓN FINAL**

## **3.1 Código Fuente**

#include <iostream>

#include <string>

#include <cstdio> //Esta libreria es para que funcionen las funciones de guardado y cargado de procesos en un archivo txt

using namespace std;

struct Proceso

{

int id;

string nombre;

int prioridad;

float memoriaMB;

};

struct NodoProceso

{

Proceso dato;

NodoProceso\* siguiente;

};

NodoProceso\* cabeza = NULL;

struct NodoCola

{

Proceso dato;

NodoCola\* siguiente;

};

NodoCola\* frenteCPU = NULL;

NodoCola\* finalCPU = NULL;

struct NodoPila

{

Proceso dato;

NodoPila\* siguiente;

};

NodoPila\* topeMemoria = NULL;

bool existeProceso(int id)

{

NodoProceso\* actual = cabeza;

while (actual != NULL)

{

if (actual->dato.id == id) return true;

actual = actual->siguiente;

}

return false;

}

void agregarProceso()

{

Proceso nuevo;

do {

cout << "Ingrese ID del proceso: ";

cin >> nuevo.id;

if (nuevo.id <= 0) {

cout << "Error: El ID debe ser un número mayor a 0.\n";

}

} while (nuevo.id <= 0);

if (existeProceso(nuevo.id))

{

cout << "Error: Ya existe un proceso con ese ID.\n";

return;

}

cin.ignore();

cout << "Ingrese nombre del proceso: "; getline(cin, nuevo.nombre);

do {

cout << "Ingrese prioridad (1-10): ";

cin >> nuevo.prioridad;

if (nuevo.prioridad <= 1 || nuevo.prioridad >= 10) {

cout << "Error: La prioridad debe estar entre 1 y 10.\n";

}

} while (nuevo.prioridad <= 1 || nuevo.prioridad >= 10);

do {

cout << "Ingrese memoria que usara el proceso (MB): ";

cin >> nuevo.memoriaMB;

if (nuevo.memoriaMB < 0) {

cout << "Error: La memoria debe ser un número positivo.\n";

}

} while (nuevo.memoriaMB < 0);

NodoProceso\* nodo = new NodoProceso;

nodo->dato = nuevo;

nodo->siguiente = NULL;

if (cabeza == NULL) cabeza = nodo;

else

{

NodoProceso\* actual = cabeza;

while (actual->siguiente != NULL) actual = actual->siguiente;

actual->siguiente = nodo;

}

cout << "\nProceso agregado correctamente.\n";

}

void mostrarProcesos()

{

if (cabeza == NULL)

{

cout << "No hay procesos registrados.\n"; return;

}

cout << "\n=== LISTA DE PROCESOS ===\n";

NodoProceso\* actual = cabeza;

while (actual != NULL)

{

cout << "ID: " << actual->dato.id

<< " | Nombre: " << actual->dato.nombre

<< " | Prioridad: " << actual->dato.prioridad

<< " | Memoria: " << actual->dato.memoriaMB << " MB\n";

actual = actual->siguiente;

}

}

void buscarProceso()

{

int id;

cout << "Ingrese ID del proceso a buscar: "; cin >> id;

NodoProceso\* actual = cabeza;

while (actual != NULL)

{

if (actual->dato.id == id)

{

cout << "\nProceso encontrado:\n";

cout << "ID: " << actual->dato.id << "\nNombre: " << actual->dato.nombre

<< "\nPrioridad: " << actual->dato.prioridad

<< "\nMemoria: " << actual->dato.memoriaMB << " MB\n";

return;

}

actual = actual->siguiente;

}

cout << "Proceso no encontrado.\n";

}

void eliminarProceso()

{

int id;

cout << "Ingrese ID del proceso a eliminar: "; cin >> id;

NodoProceso\* actual = cabeza;

NodoProceso\* anterior = NULL;

while (actual != NULL && actual->dato.id != id)

{

anterior = actual;

actual = actual->siguiente;

}

if (actual == NULL)

{

cout << "Proceso no encontrado.\n";

return;

}

if (anterior == NULL) cabeza = actual->siguiente;

else anterior->siguiente = actual->siguiente;

delete actual;

cout << "Proceso eliminado correctamente.\n";

}

void modificarPrioridad()

{

int id, nuevaPrioridad;

cout << "Ingrese ID del proceso: "; cin >> id;

cout << "Ingrese nueva prioridad: "; cin >> nuevaPrioridad;

NodoProceso\* actual = cabeza;

while (actual != NULL)

{

if (actual->dato.id == id)

{

actual->dato.prioridad = nuevaPrioridad;

cout << "Prioridad actualizada.\n";

return;

}

actual = actual->siguiente;

}

cout << "Proceso no encontrado.\n";

}

void encolarProcesoCPU()

{

int id;

cout << "Ingrese ID del proceso a encolar: "; cin >> id;

NodoProceso\* actual = cabeza;

while (actual != NULL && actual->dato.id != id) actual = actual->siguiente;

if (actual == NULL)

{

cout << "Error: No existe un proceso con ese ID.\n"; return;

}

NodoCola\* nuevo = new NodoCola;

nuevo->dato = actual->dato;

nuevo->siguiente = NULL;

if (frenteCPU == NULL || nuevo->dato.prioridad > frenteCPU->dato.prioridad)

{

nuevo->siguiente = frenteCPU;

frenteCPU = nuevo;

if (finalCPU == NULL) finalCPU = nuevo;

}

else

{

NodoCola\* temp = frenteCPU;

while (temp->siguiente != NULL && temp->siguiente->dato.prioridad >= nuevo->dato.prioridad)

temp = temp->siguiente;

nuevo->siguiente = temp->siguiente;

temp->siguiente = nuevo;

if (nuevo->siguiente == NULL) finalCPU = nuevo;

}

cout << "Proceso encolado correctamente.\n";

}

void ejecutarProcesoCPU()

{

if (frenteCPU == NULL)

{

cout << "Error: No hay procesos en la cola.\n"; return;

}

NodoCola\* temp = frenteCPU;

cout << "Ejecutando proceso: " << temp->dato.nombre << " (ID: " << temp->dato.id << ")\n";

frenteCPU = frenteCPU->siguiente;

if (frenteCPU == NULL) finalCPU = NULL;

delete temp;

}

void mostrarColaCPU()

{

if (frenteCPU == NULL)

{

cout << "Cola vacía.\n"; return;

}

cout << "\n=== COLA DE CPU ===\n";

NodoCola\* actual = frenteCPU;

while (actual != NULL)

{

cout << "ID: " << actual->dato.id << " | Nombre: " << actual->dato.nombre

<< " | Prioridad: " << actual->dato.prioridad << "\n";

actual = actual->siguiente;

}

}

void asignarMemoria()

{

int id;

cout << "Ingrese ID del proceso a asignar memoria: "; cin >> id;

NodoProceso\* actual = cabeza;

while (actual != NULL && actual->dato.id != id) actual = actual->siguiente;

if (actual == NULL)

{

cout << "Proceso no encontrado.\n"; return;

}

NodoPila\* nuevo = new NodoPila;

nuevo->dato = actual->dato;

nuevo->siguiente = topeMemoria;

topeMemoria = nuevo;

cout << "Memoria asignada al proceso correctamente.\n";

}

void liberarMemoria()

{

if (topeMemoria == NULL)

{

cout << "No hay memoria para liberar.\n"; return;

}

NodoPila\* temp = topeMemoria;

cout << "Liberando memoria de: " << temp->dato.nombre << " (" << temp->dato.memoriaMB << " MB)\n";

topeMemoria = topeMemoria->siguiente;

delete temp;

}

void mostrarMemoria()

{

if (topeMemoria == NULL)

{

cout << "Memoria vacía.\n"; return;

}

cout << "\n=== PILA DE MEMORIA ===\n";

NodoPila\* actual = topeMemoria;

while (actual != NULL)

{

cout << "ID: " << actual->dato.id

<< " | Nombre: " << actual->dato.nombre

<< " | Memoria: " << actual->dato.memoriaMB << " MB\n";

actual = actual->siguiente;

}

}

void mostrarMemoriaTotal()

{

float total = 0.0;

NodoPila\* actual = topeMemoria;

while (actual != NULL)

{

total += actual->dato.memoriaMB;

actual = actual->siguiente;

}

cout << "\n -----> Memoria total usada: " << total << " MB\n";

}

void guardarProcesosEnArchivo()

{

FILE\* archivo = fopen("procesos.txt", "w");

if (archivo == NULL)

{

cout << "Error al abrir el archivo para guardar.\n";

return;

}

NodoProceso\* actual = cabeza;

while (actual != NULL)

{

fprintf(archivo, "%d;%s;%d;%.2f\n",

actual->dato.id,

actual->dato.nombre.c\_str(),

actual->dato.prioridad,

actual->dato.memoriaMB);

actual = actual->siguiente;

}

fclose(archivo);

cout << "Procesos guardados en archivo correctamente.\n";

}

void cargarProcesosDesdeArchivo()

{

FILE\* archivo = fopen("procesos.txt", "r");

if (archivo == NULL)

{

cout << "No se encontró el archivo de procesos.\n";

return;

}

while (true)

{

Proceso nuevo;

char nombreTemp[100];

int leidos = fscanf(archivo, "%d;%99[^;];%d;%f\n",

&nuevo.id,

nombreTemp,

&nuevo.prioridad,

&nuevo.memoriaMB);

if (leidos != 4) break;

nuevo.nombre = nombreTemp;

NodoProceso\* nodo = new NodoProceso;

nodo->dato = nuevo;

nodo->siguiente = NULL;

if (cabeza == NULL) cabeza = nodo;

else

{

NodoProceso\* actual = cabeza;

while (actual->siguiente != NULL) actual = actual->siguiente;

actual->siguiente = nodo;

}

}

fclose(archivo);

cout << "Procesos cargados desde archivo correctamente.\n";

}

void menuProcesos()

{

int op;

do {

cout << "\n+--------------------------------+\n";

cout << "| GESTION DE PROCESOS |\n";

cout << "|--------------------------------|\n";

cout << "| 1. Agregar Proceso |\n";

cout << "| 2. Mostrar Procesos |\n";

cout << "| 3. Buscar Proceso por Id |\n";

cout << "| 4. Eliminar Proceso |\n";

cout << "| 5. Modificar Prioridad |\n";

cout << "| 6. Volver |\n";

cout << "+--------------------------------+\n";

do {

cout << "Opcion (1-6): ";

cin >> op;

if (op < 1 || op > 6) {

cout << "Error: Ingrese una opcion valida entre 1 y 6.\n";

}

} while (op < 1 || op > 6);

switch (op) {

case 1: agregarProceso(); break;

case 2: mostrarProcesos(); break;

case 3: buscarProceso(); break;

case 4: eliminarProceso(); break;

case 5: modificarPrioridad(); break;

}

} while (op != 6);

}

void menuCPU()

{

int op;

do {

cout << "\n+------------------------------+\n";

cout << "| GESTION DE CPU |\n";

cout << "|--------------------------------|\n";

cout << "| 1. Encolar Proceso |\n";

cout << "| 2. Ejecutar Proceso |\n";

cout << "| 3. Mostrar Cola |\n";

cout << "| 4. Volver |\n";

cout << "+--------------------------------+\n";

do {

cout << "Opcion (1-4): ";

cin >> op;

if (op < 1 || op > 4) {

cout << "Error: Ingrese una opcion valida entre 1 y 4.\n";

}

} while (op < 1 || op > 4);

switch (op) {

case 1: encolarProcesoCPU(); break;

case 2: ejecutarProcesoCPU(); break;

case 3: mostrarColaCPU(); break;

}

} while (op != 4);

}

void menuMemoria()

{

int op;

do {

cout << "\n+--------------------------------+\n";

cout << "| GESTION DE MEMORIA |\n";

cout << "|--------------------------------|\n";

cout << "| 1. Asignar Memoria |\n";

cout << "| 2. Liberar Memoria |\n";

cout << "| 3. Mostrar Memoria |\n";

cout << "| 4. Ver Memoria Total |\n";

cout << "| 5. Volver |\n";

cout << "+--------------------------------+\n";

do {

cout << "Opcion (1-5): ";

cin >> op;

if (op < 1 || op > 5) {

cout << "Error: Ingrese una opcion valida entre 1 y 5.\n";

}

} while (op < 1 || op > 5);

switch (op) {

case 1: asignarMemoria(); break;

case 2: liberarMemoria(); break;

case 3: mostrarMemoria(); break;

case 4: mostrarMemoriaTotal(); break;

}

} while (op != 5);

}

int main()

{

cargarProcesosDesdeArchivo();

int opcion;

do {

cout << "\n-------------------------------------\n";

cout << " SISTEMA DE GESTION DE PROCESOS \n";

cout << "-------------------------------------\n";

cout << "1. Gestion de Procesos\n";

cout << "2. Gestion de CPU\n";

cout << "3. Gestion de Memoria\n";

cout << "4. Salir\n";

do {

cout << "Seleccione una opcion (1-4): ";

cin >> opcion;

if (opcion < 1 || opcion > 4) {

cout << "Error: Ingrese una opcion valida entre 1 y 4.\n";

}

} while (opcion < 1 || opcion > 4);

switch (opcion) {

case 1: menuProcesos(); break;

case 2: menuCPU(); break;

case 3: menuMemoria(); break;

case 4: guardarProcesosEnArchivo();

cout << "Saliendo del programa...\n";

break;

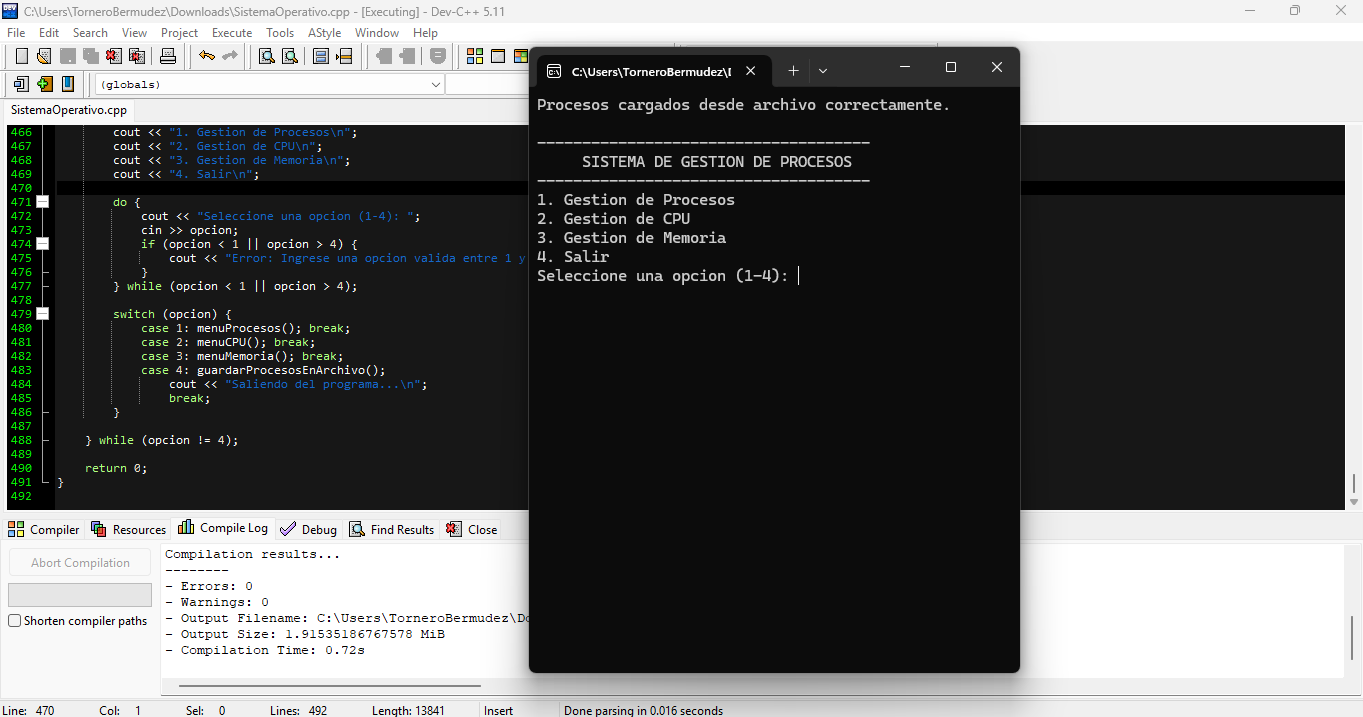
}

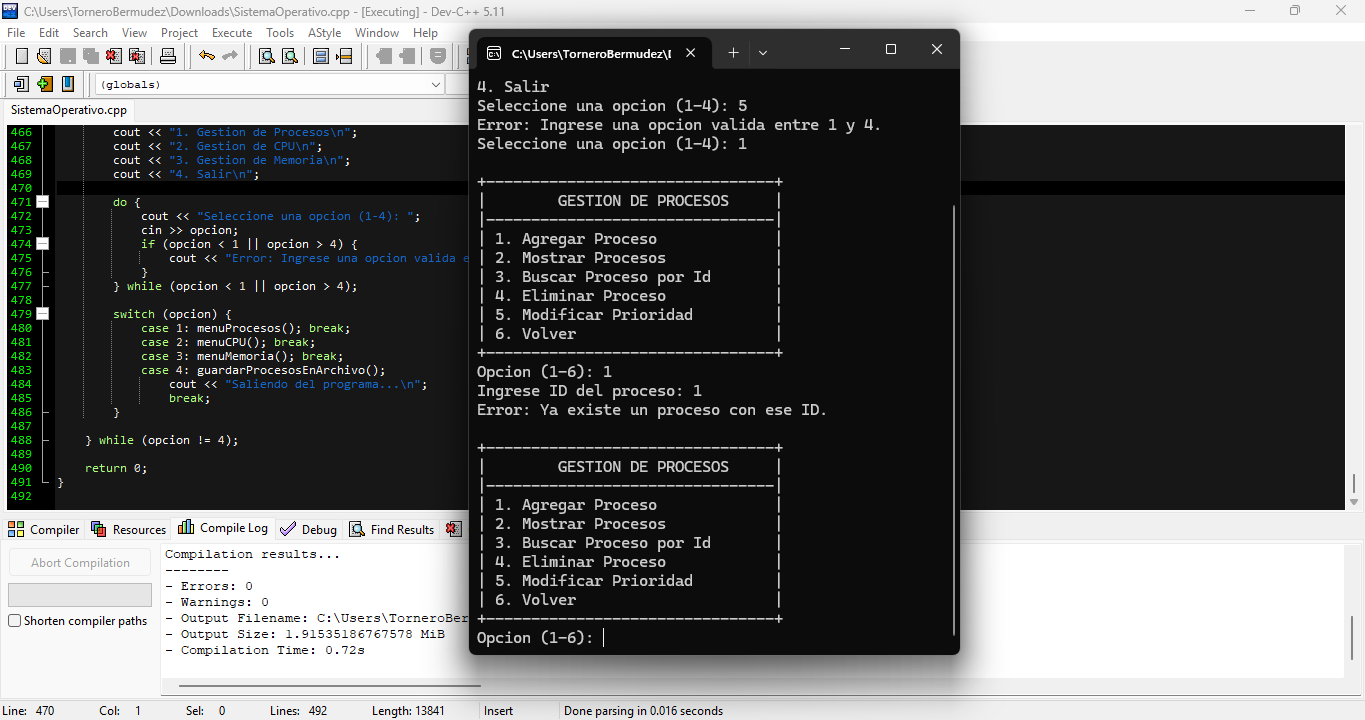
} while (opcion != 4);

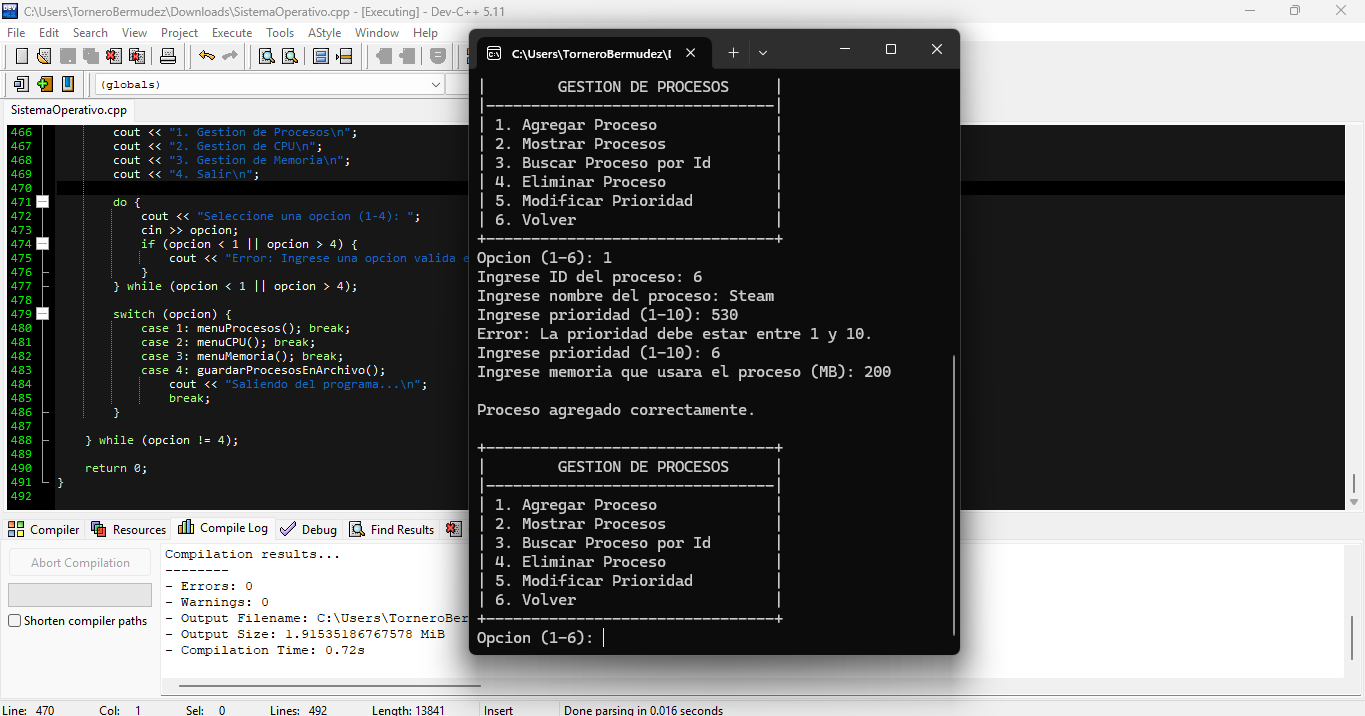
return 0;

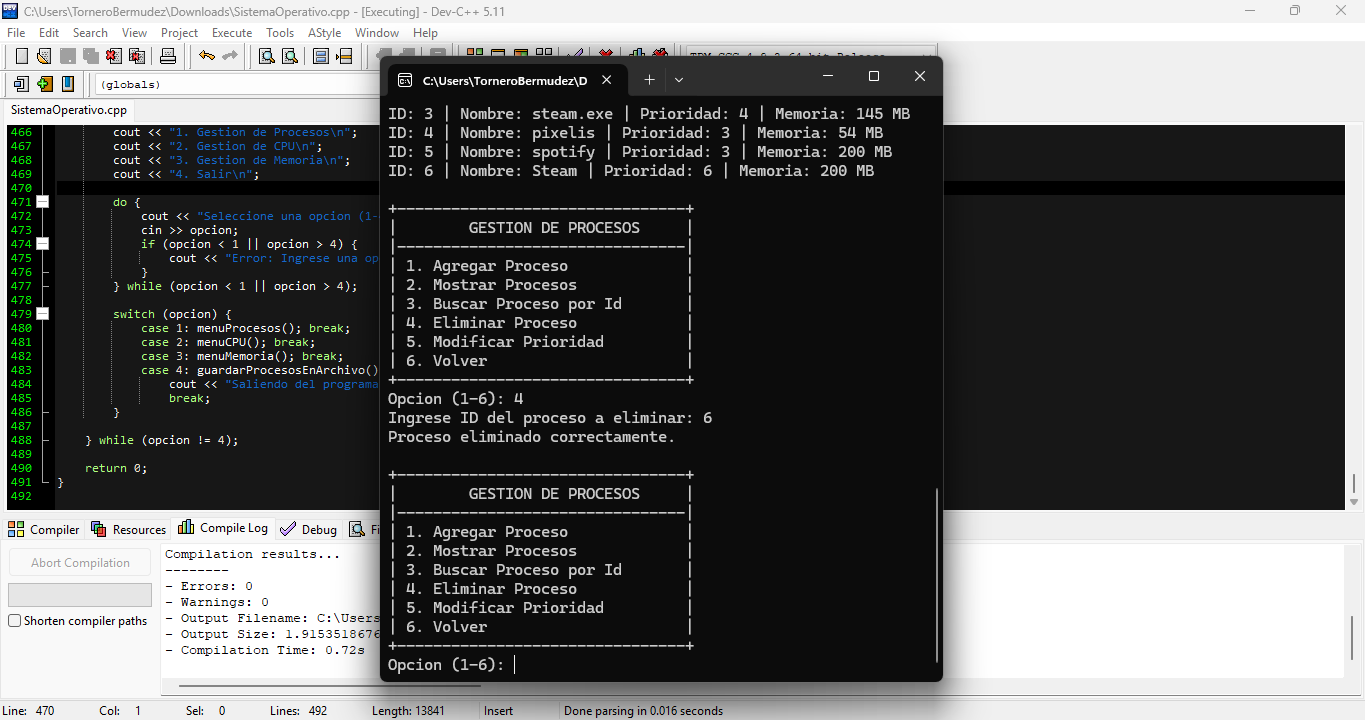
}

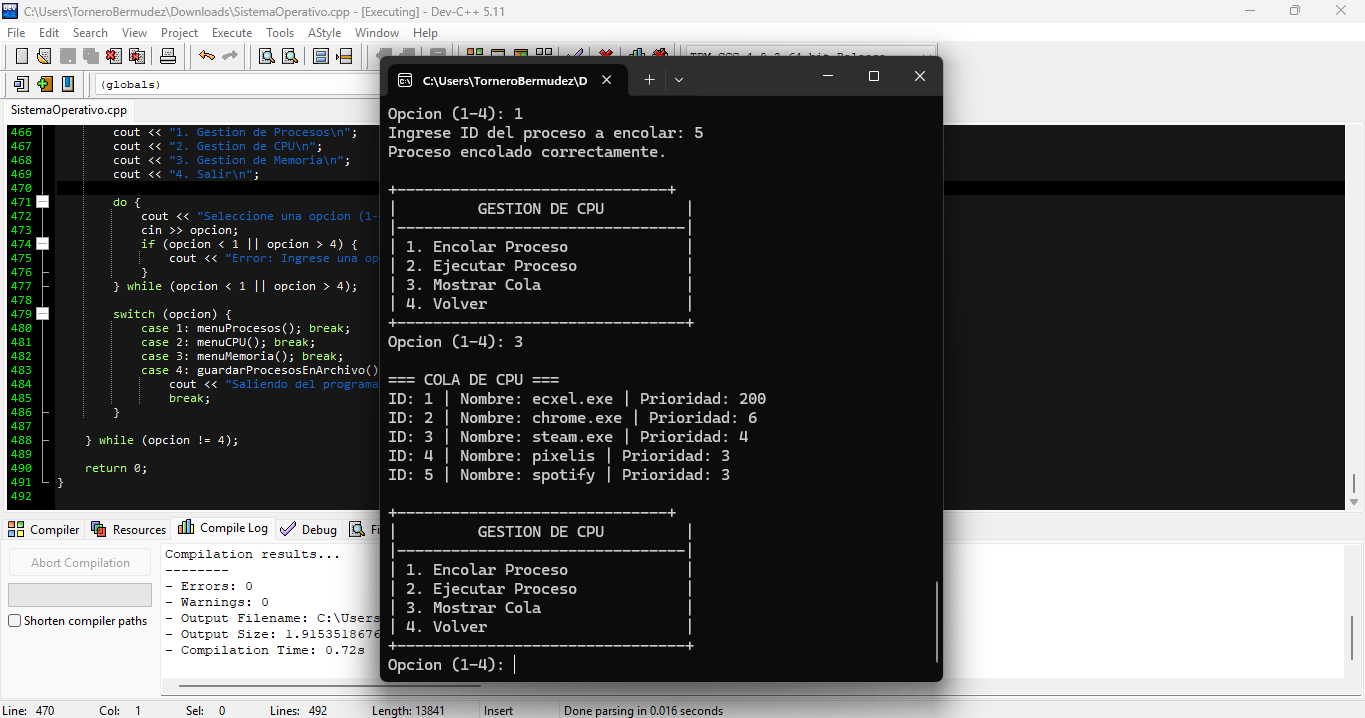
## **3.2 Capturas de Pantalla**

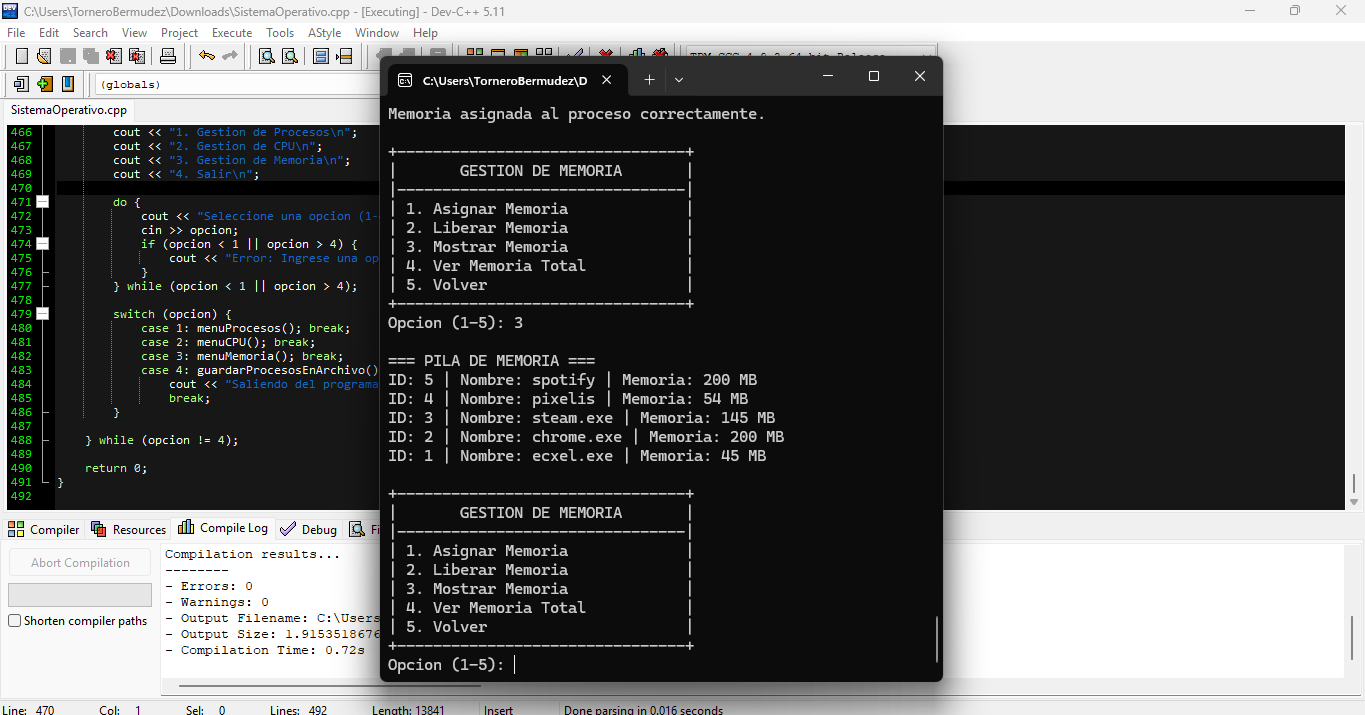


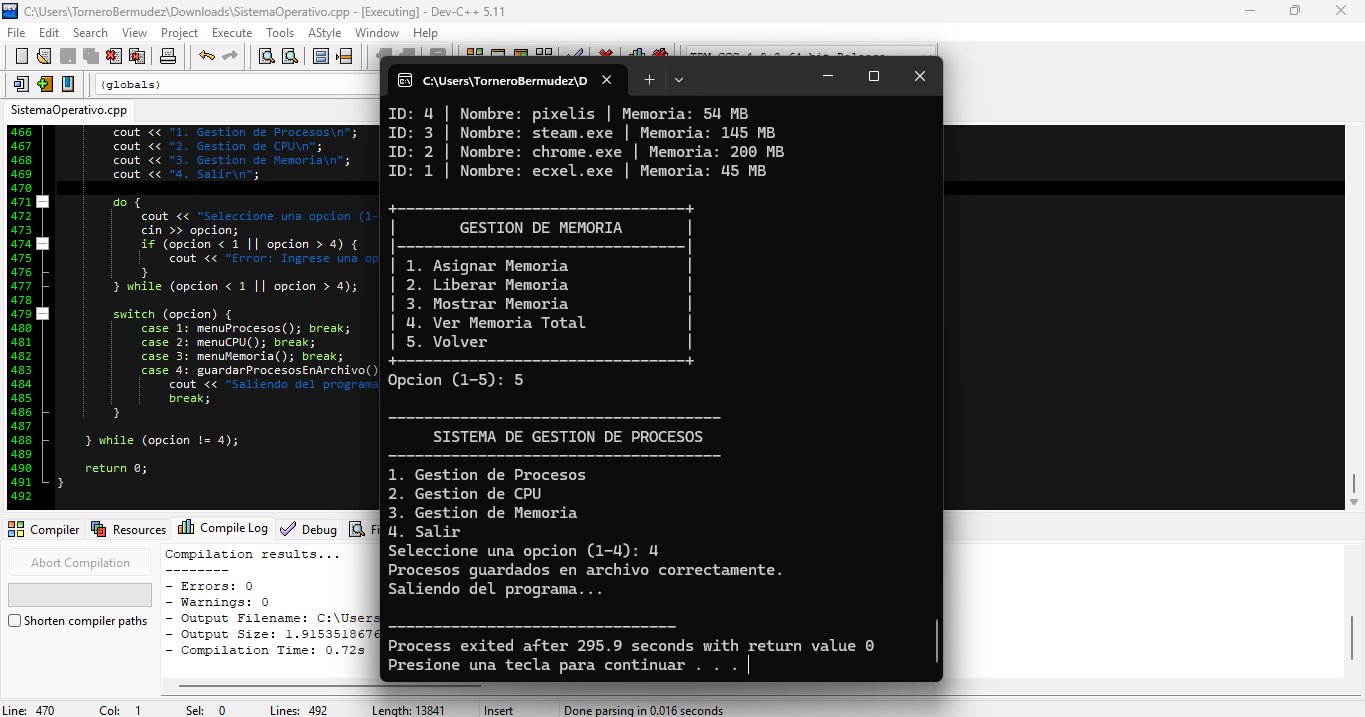


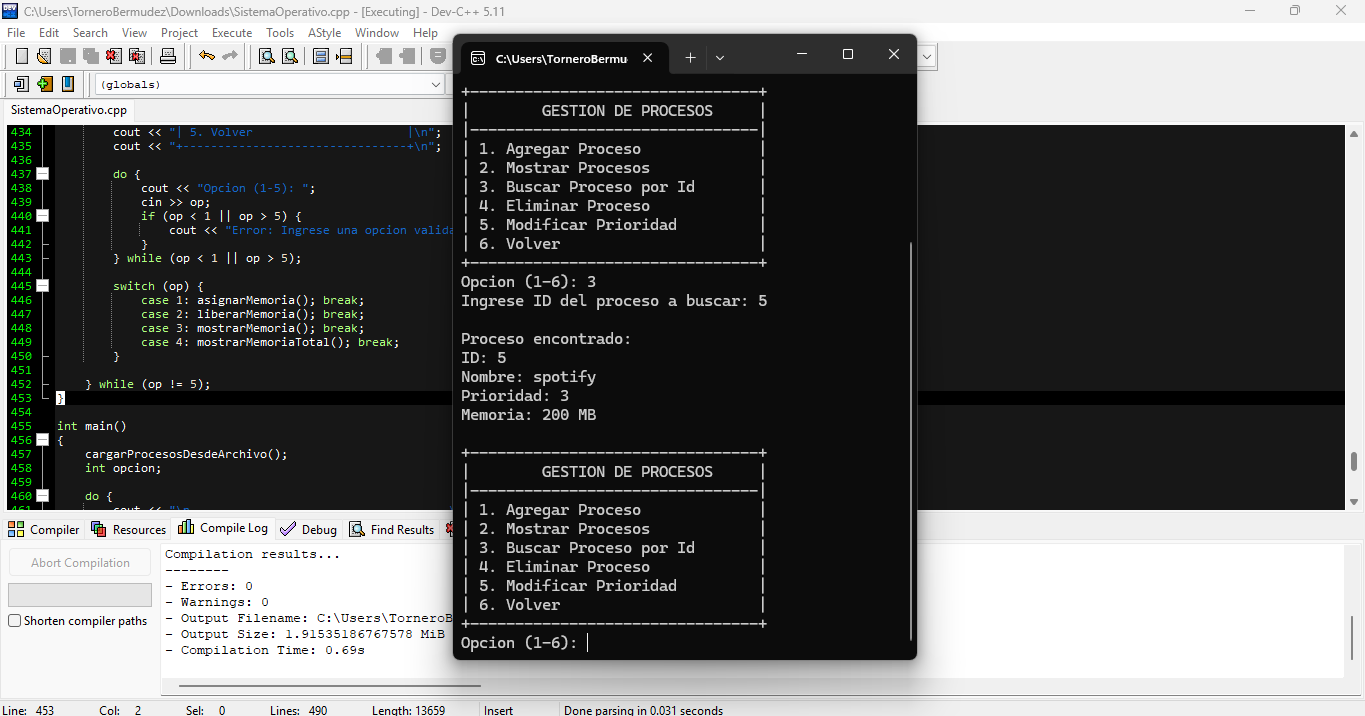


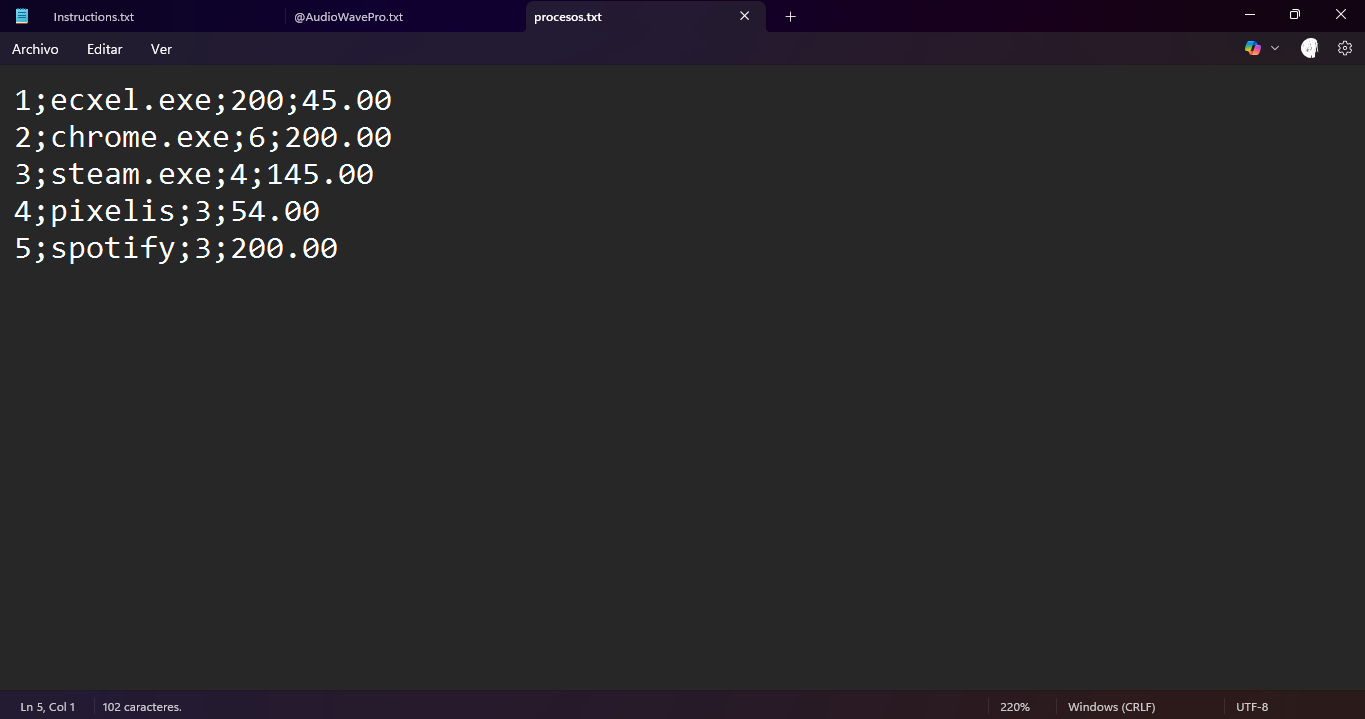






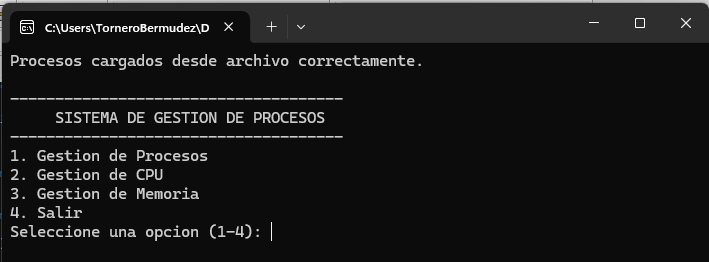




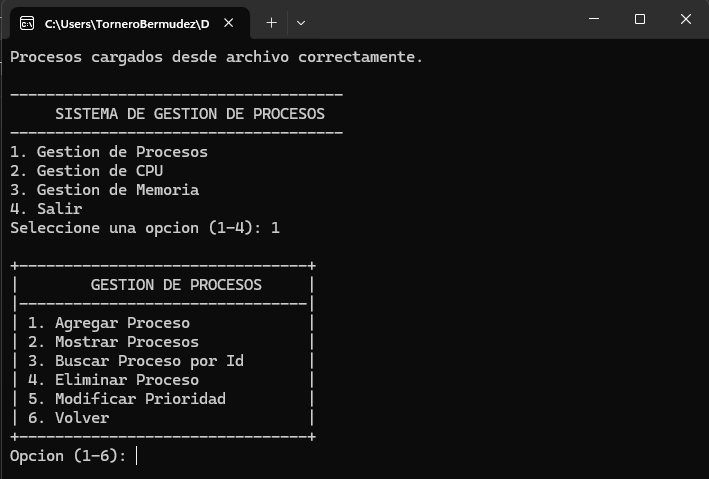


## **3.3 Manual de Usuario**

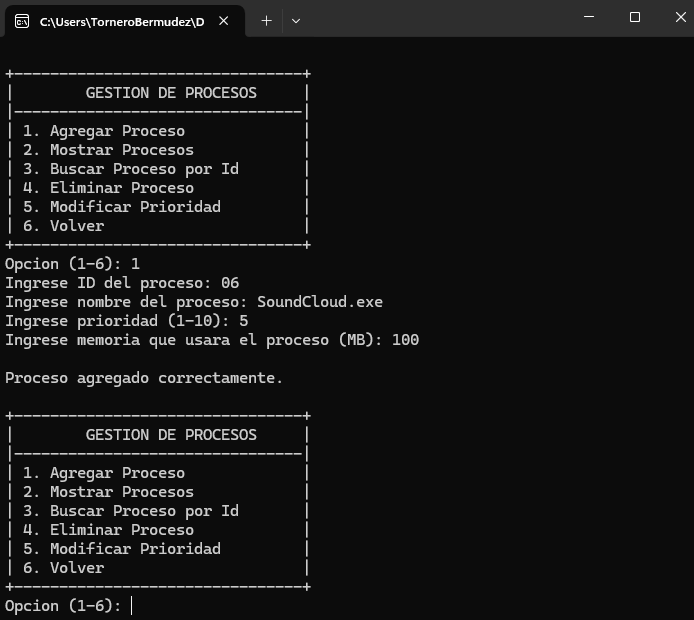
**Bienvenido al programa “Sistema de Gestión de Procesos”** Este sistema ha sido desarrollado para simular el comportamiento básico de un sistema operativo, utilizando estructuras de datos como listas enlazadas, pilas y colas. A continuación, se presenta una guía que explica el funcionamiento general del sistema, sus módulos principales y cómo interactuar con cada uno de ellos desde la interfaz por consola.



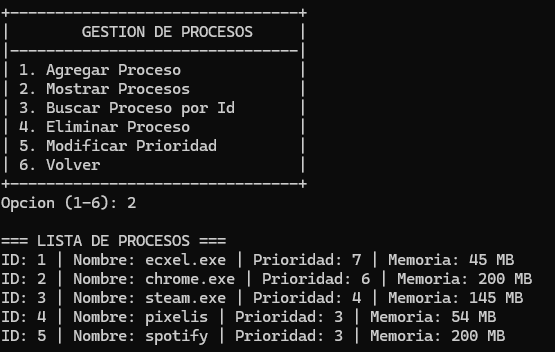
Iniciemos seleccionando la opción 01: Gestión de Procesos, donde podrás agregar, buscar, modificar, eliminar y listar procesos usando una lista enlazada dinámica.



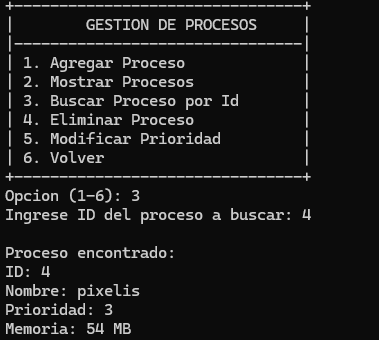
Primero, agreguemos el proceso que deseamos registrar usando la opción 1 del menú de Gestión de Procesos. Aquí ingresaremos su ID, nombre, prioridad y memoria requerida.



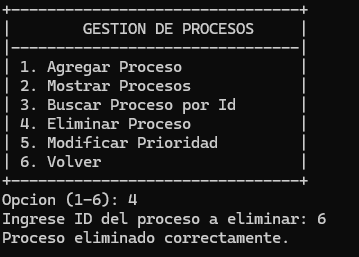
Ingresemos a la opción 2 para la muestra de nuestros procesos.



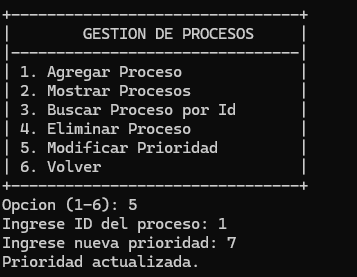
La opción 03 está destinada a la búsqueda de un proceso específico a través de su ID.



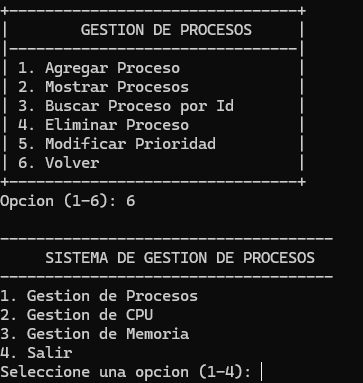
La opción 04 está destinada a la eliminación de un proceso específico a través de su ID.



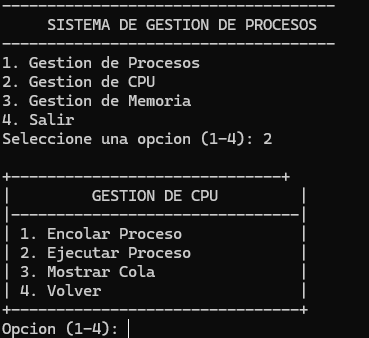
La opción 05 permite modificar un proceso específico mediante su ID y luego editar la prioridad.



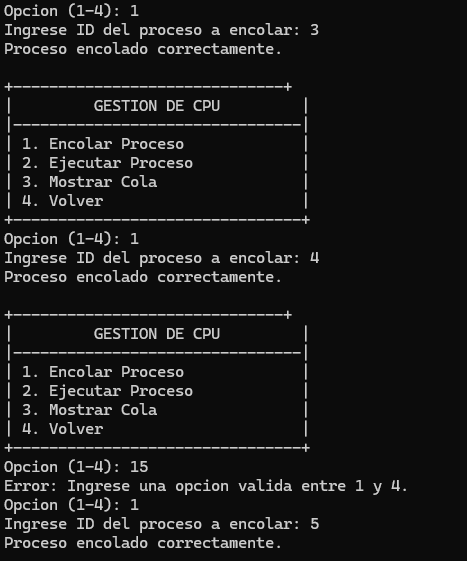
La opción 06 está destinada a retornar al menú principal.



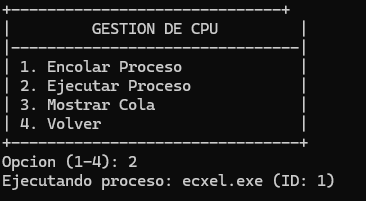
En la opción 02 del menú principal, Gestión de CPU, el usuario puede simular la ejecución de procesos en la CPU, encolándolos según su prioridad y ejecutándolos en orden descendente.



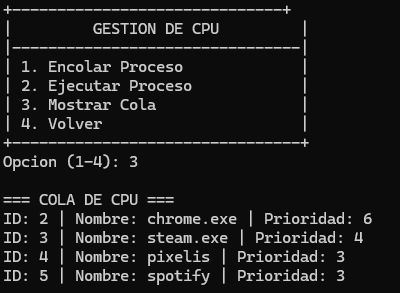
En la opción 01 de Gestión de CPU, se permite encolar un proceso previamente registrado, colocándolo en la cola según su nivel de prioridad para ser ejecutado más adelante. Encolamos 5 procesos.



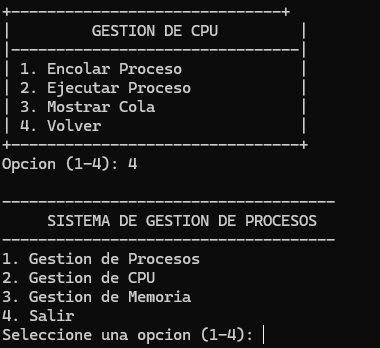
En la opción 02 de Gestión de CPU, se ejecuta el proceso con mayor prioridad, eliminándolo de la cola y simulando su atención por la CPU.



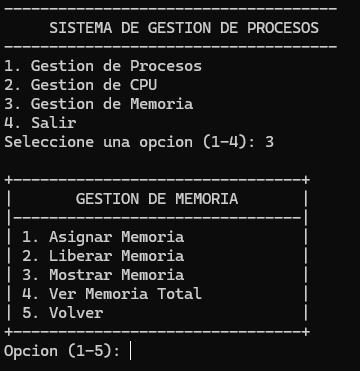
En la opción 03 de Gestión de CPU, se muestra el estado actual de la cola, listando todos los procesos en espera de ejecución ordenados por su prioridad.



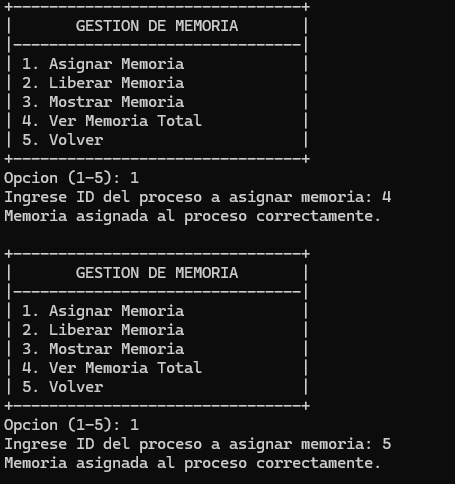
En la opción 04 de Gestión de CPU, se regresa al menú principal del sistema.



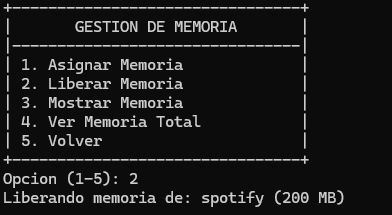
En la opción 03 del menú principal, llamada **Gestión de Memoria**, se simula la asignación y liberación de memoria a procesos mediante una pila, permitiendo administrar el uso de memoria de forma dinámica.



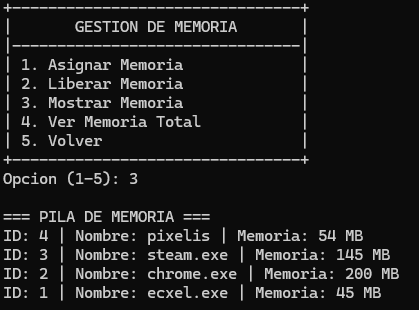
La opción "Asignar Memoria" permite registrar un nuevo proceso y reservarle espacio en la memoria del sistema.



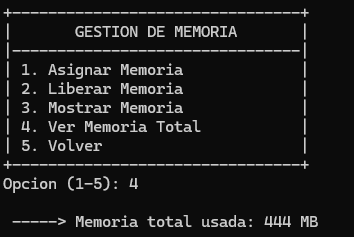
La opción "Liberar Memoria" permite eliminar un proceso y liberar el espacio que ocupaba en la memoria.



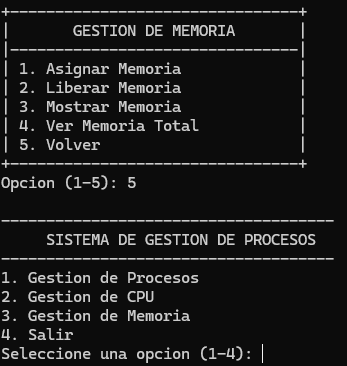
La opción "Mostrar Memoria" permite ver los procesos de acuerdo al orden de apilamiento.



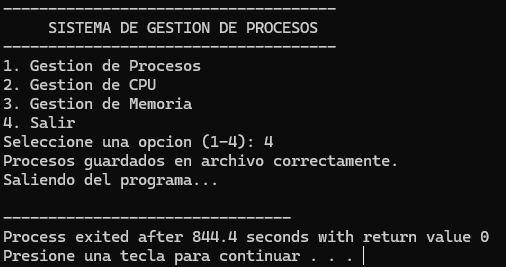
La opción "Ver Memoria Total" permite ver la suma total de memoria de los procesos asignados.

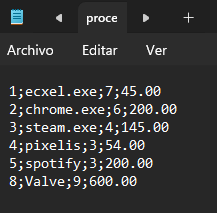


En la opción 05 de Gestión de memoria, se regresa al menú principal del sistema.



La opción 04 del menú principal permite salir del programa, guardando automáticamente todos los procesos agregados en un archivo txt. Muchas gracias por usar nuestro programa.

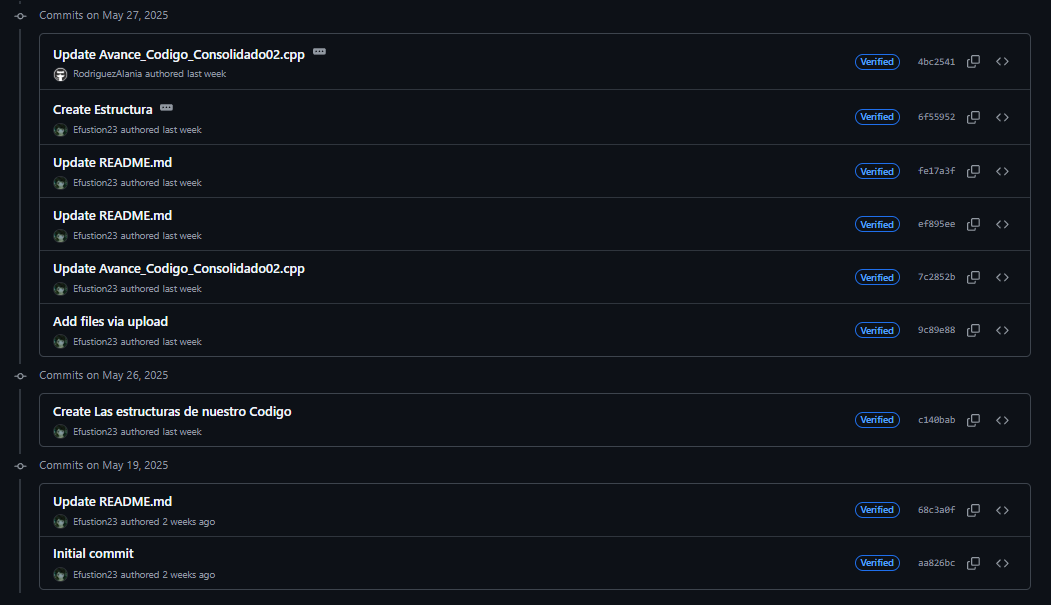


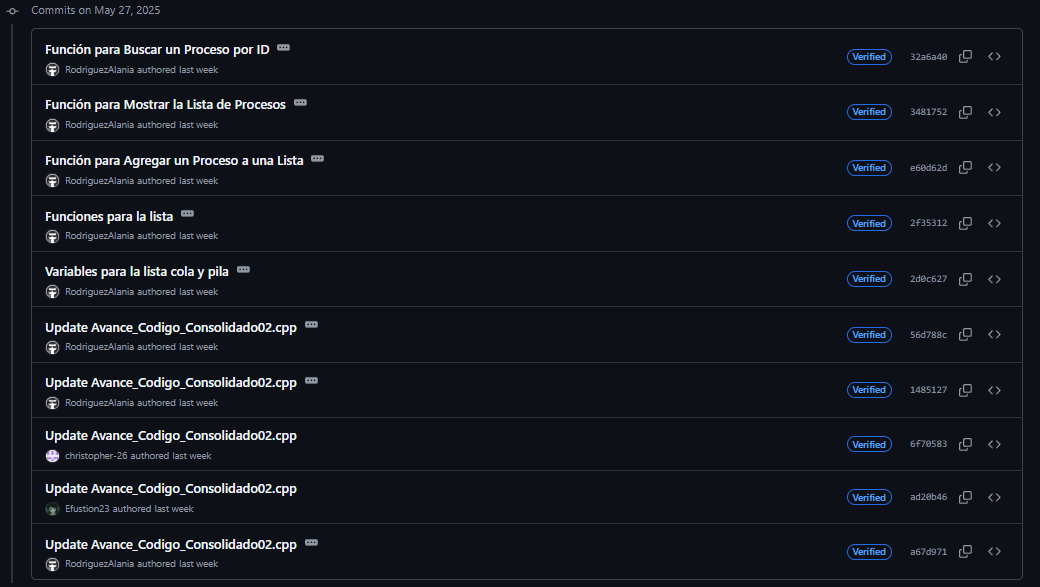


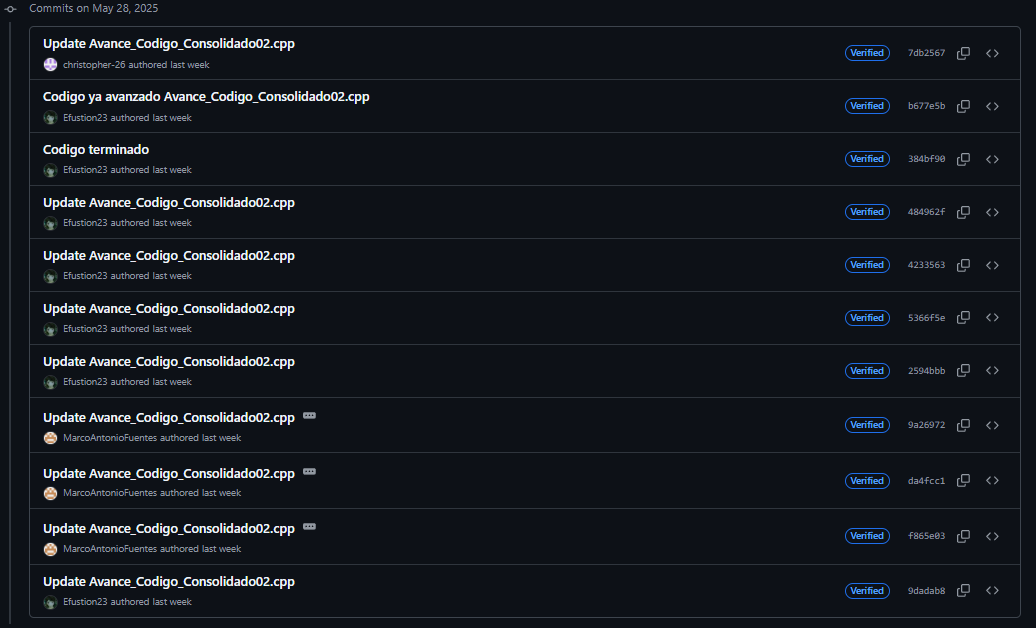
# **CAPÍTULO 4: EVIDENCIAS DE TRABAJO EN EQUIPO**

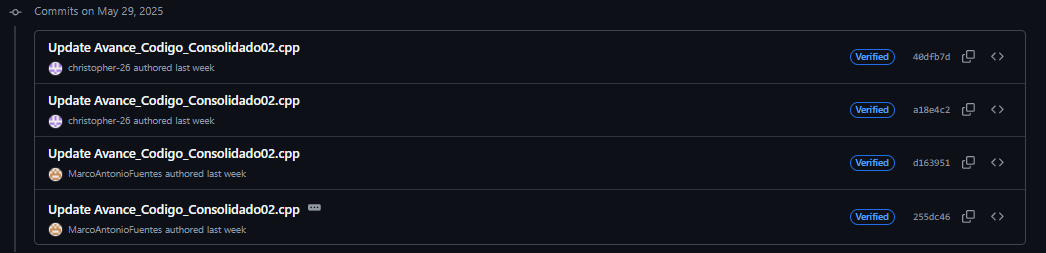
## **4.1 Repositorio con Control de Versiones**

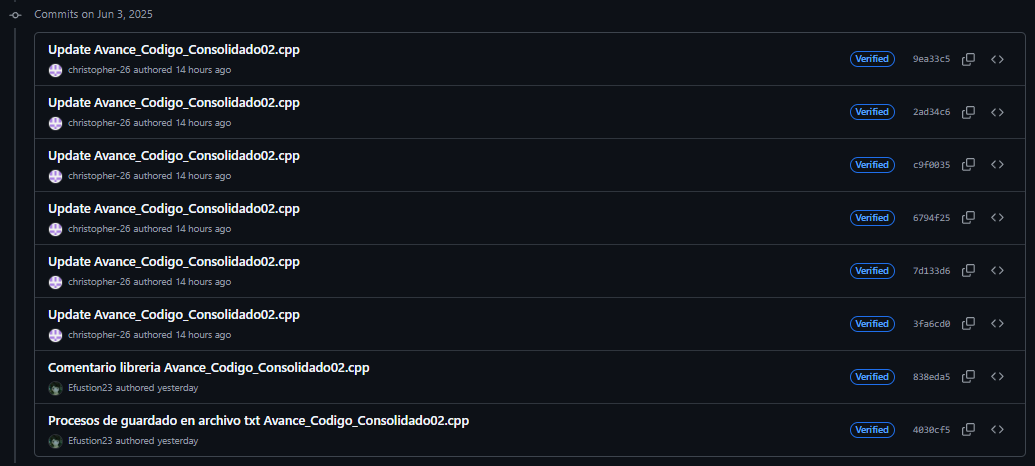
* Capturas del historial de commits.

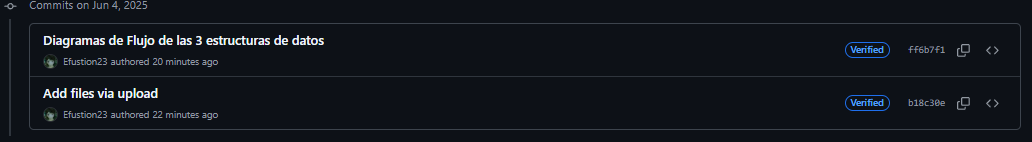




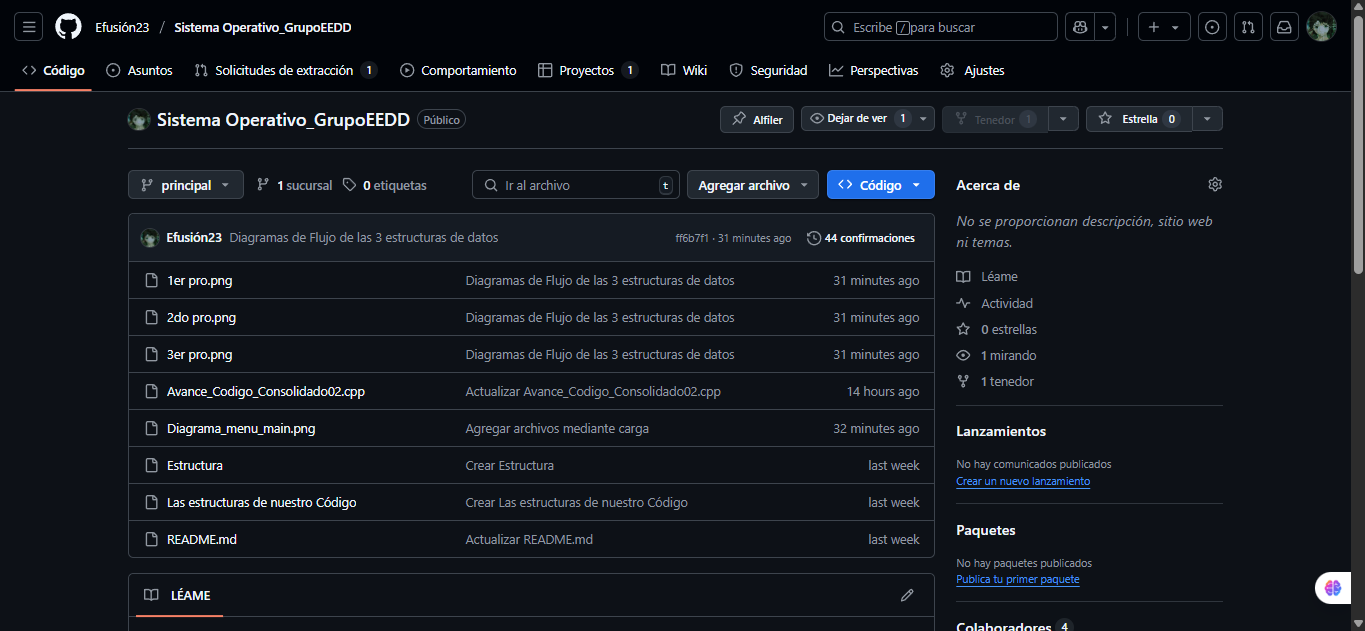








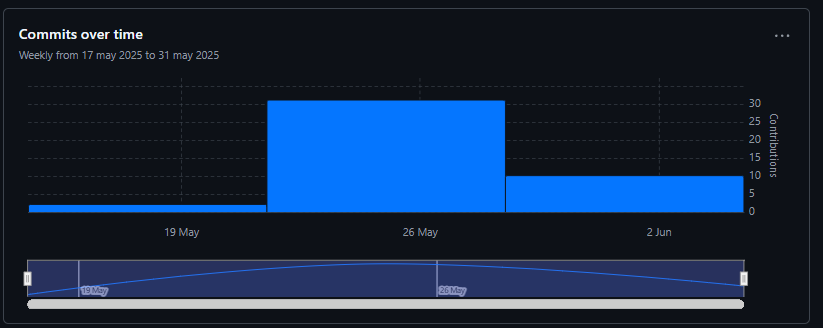
* Evidencia de ramas y fusiones.



* Enlace al repositorio GitHub: <https://github.com/Efustion23/SistemaOperativo_GrupoEEDD>

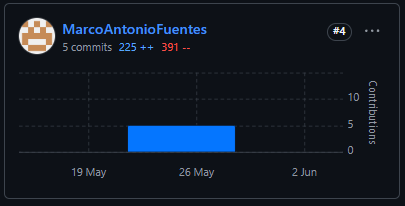
## **4.2 Evidencia por Integrante**

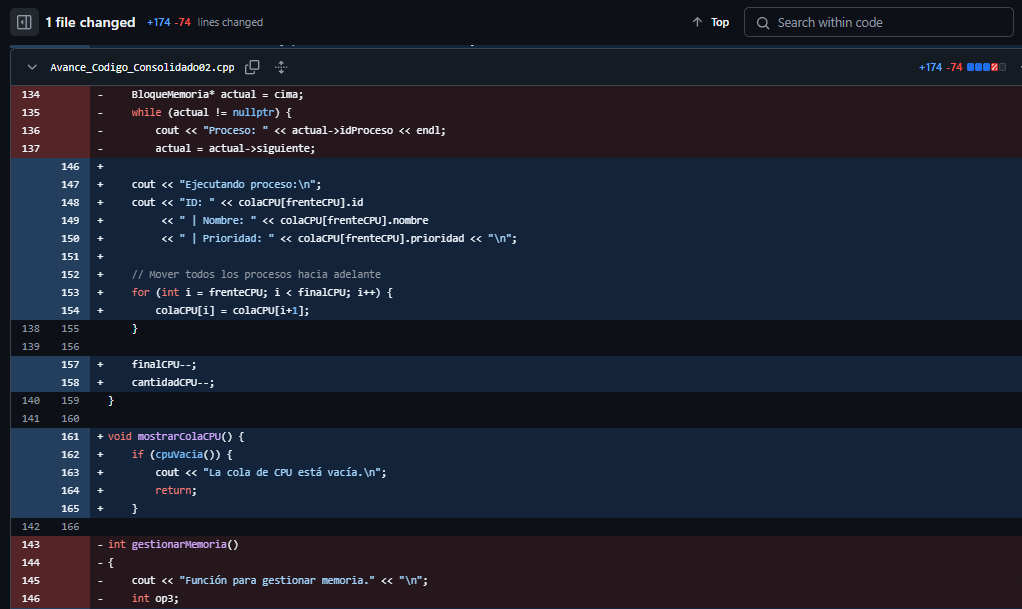


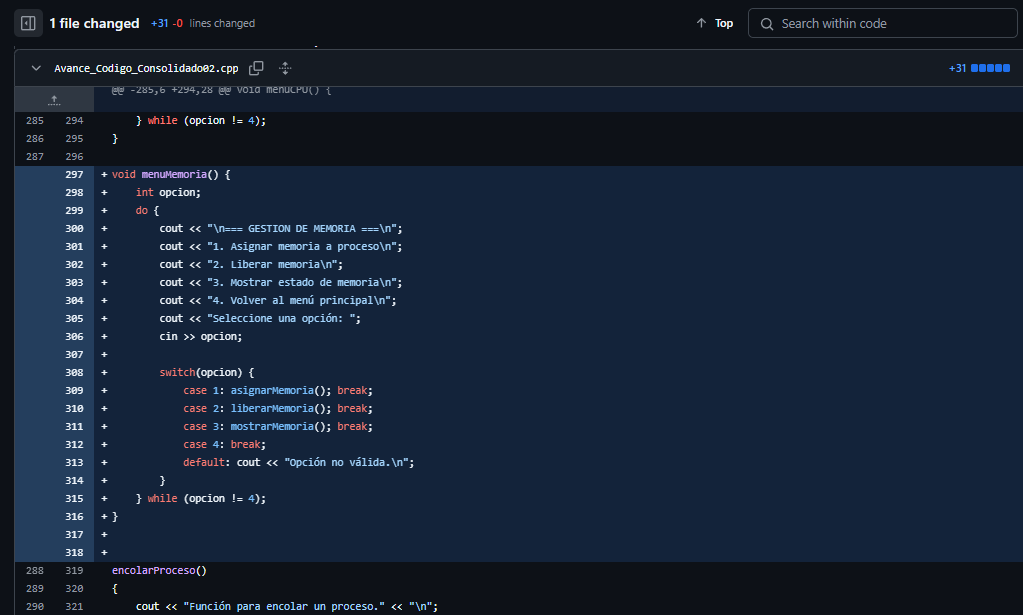


* Nombre: **Fuentes Echavigurin Marco Antonio**

Contribuciones: 5

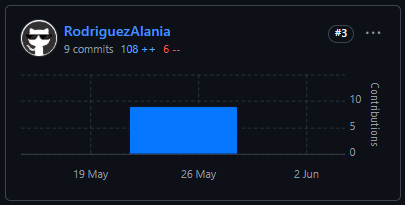


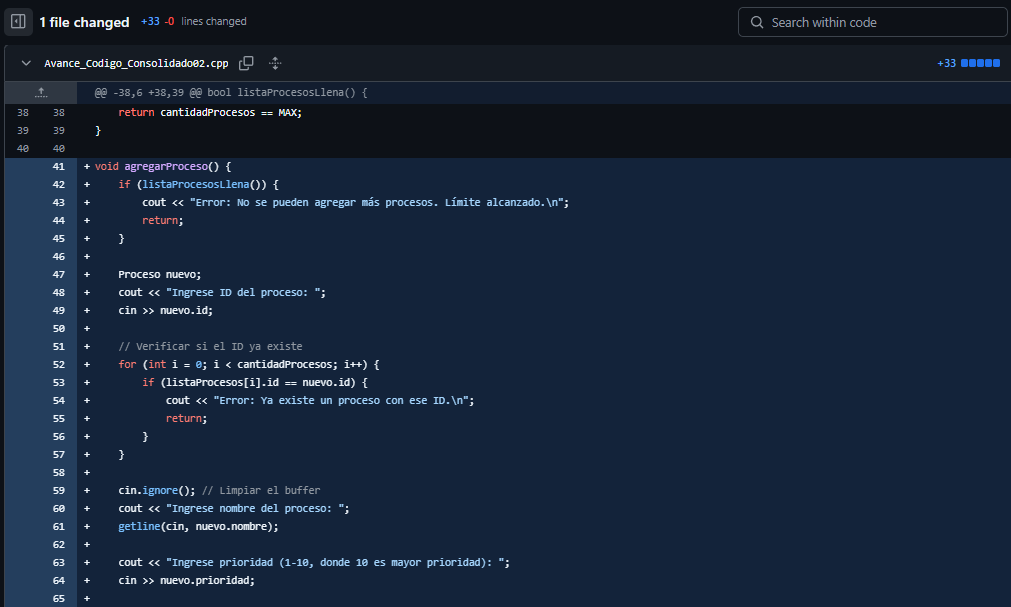


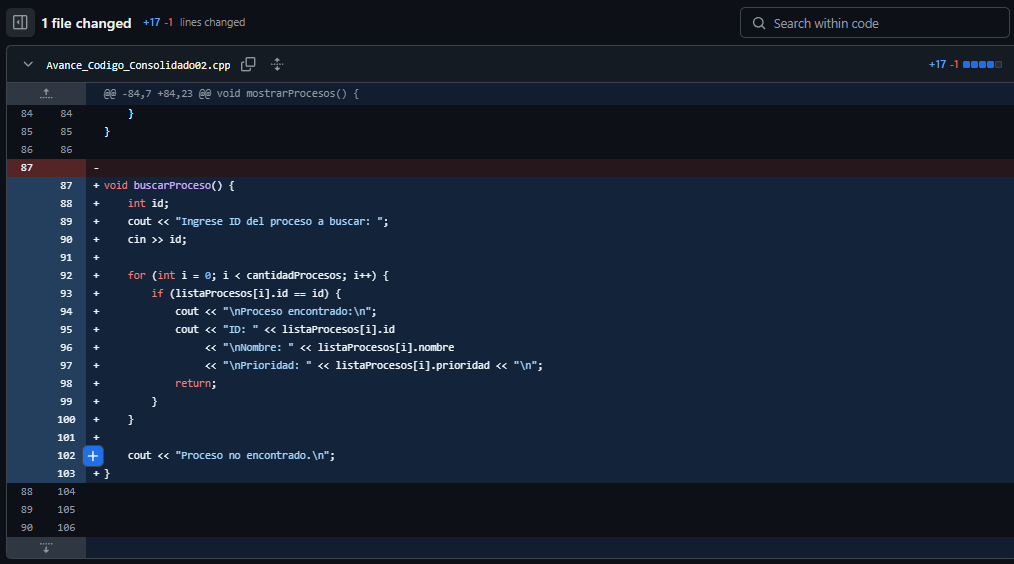


* Nombre: **Rodriguez Alania Leonel Lorenzo**

Contribuciones: 9

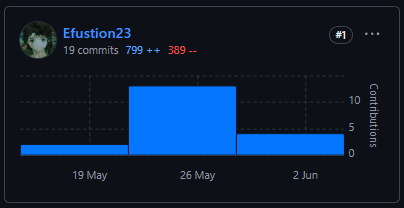


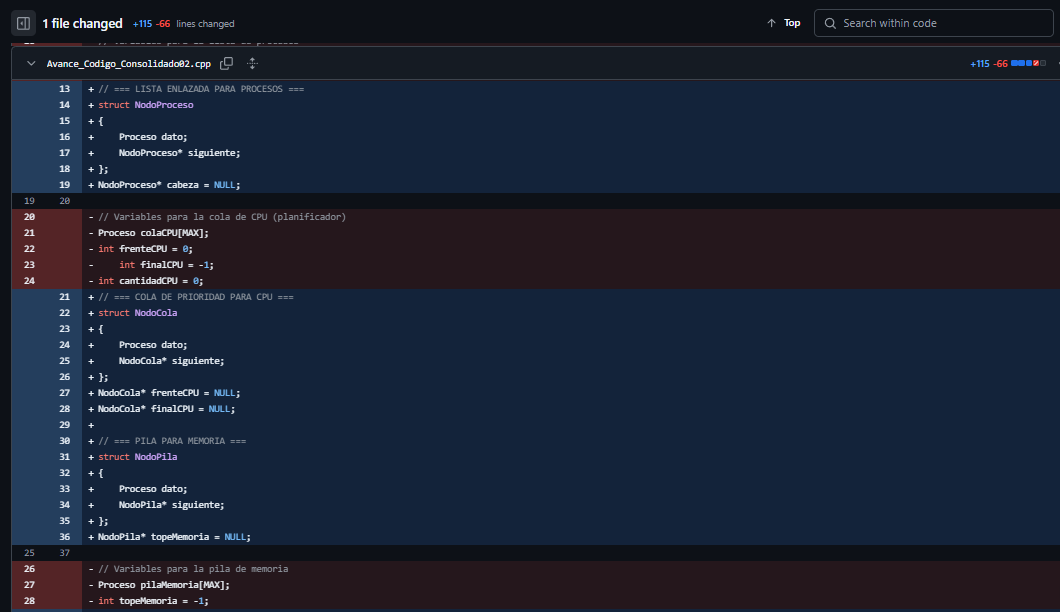


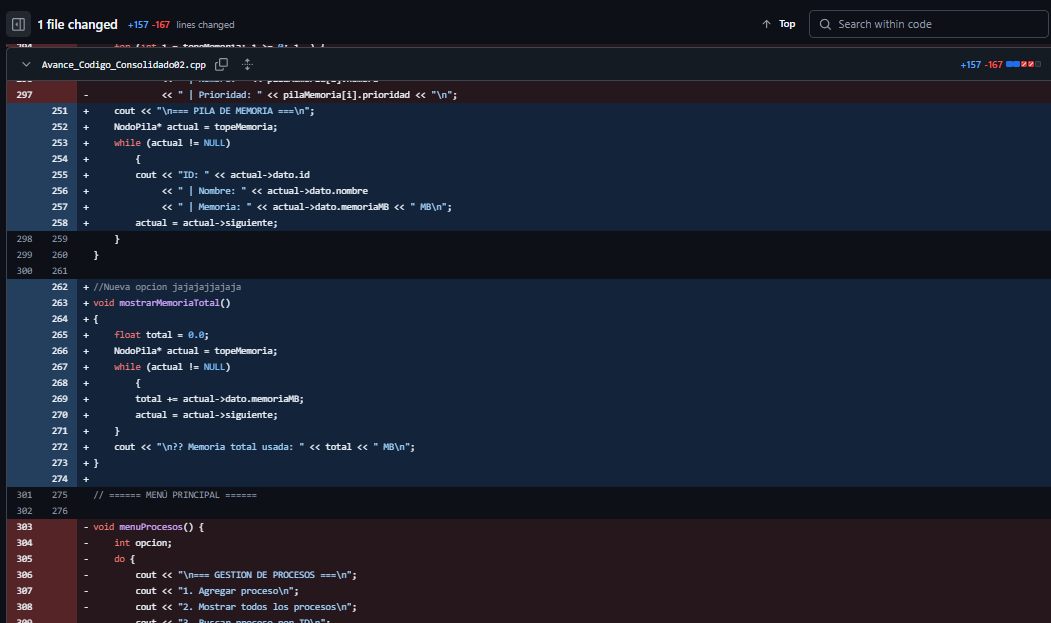


* Nombre: **Tornero Bermudez Diego Alí Gadiel**

Contribuciones: 19

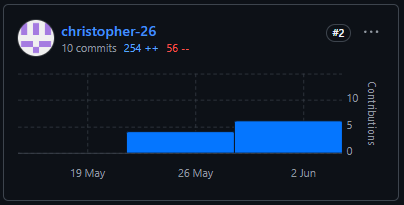


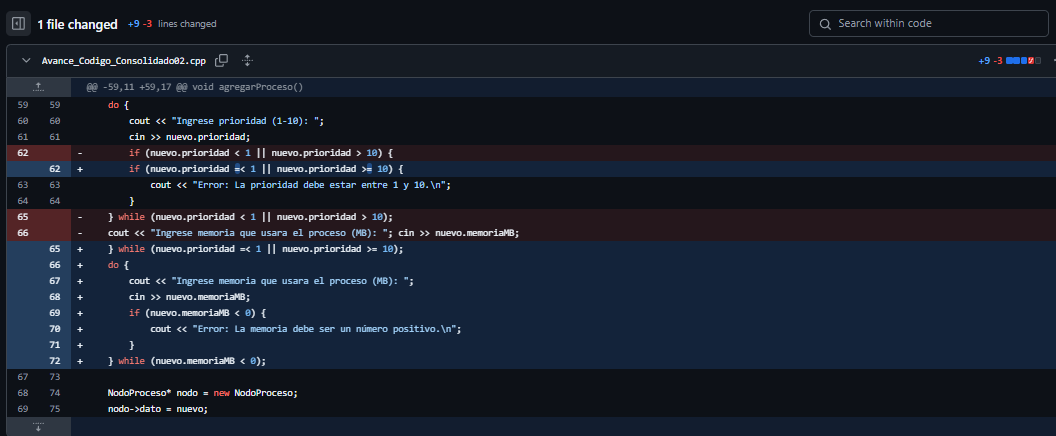


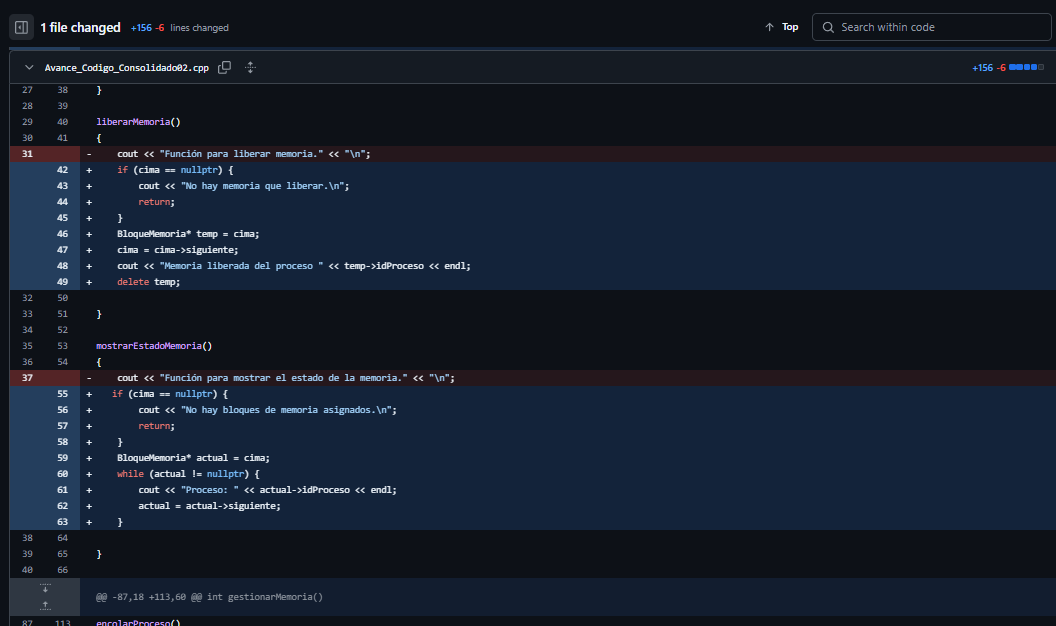


* Nombre: **Veliz Cochachi Christopher Jesús**

Contribuciones: 10

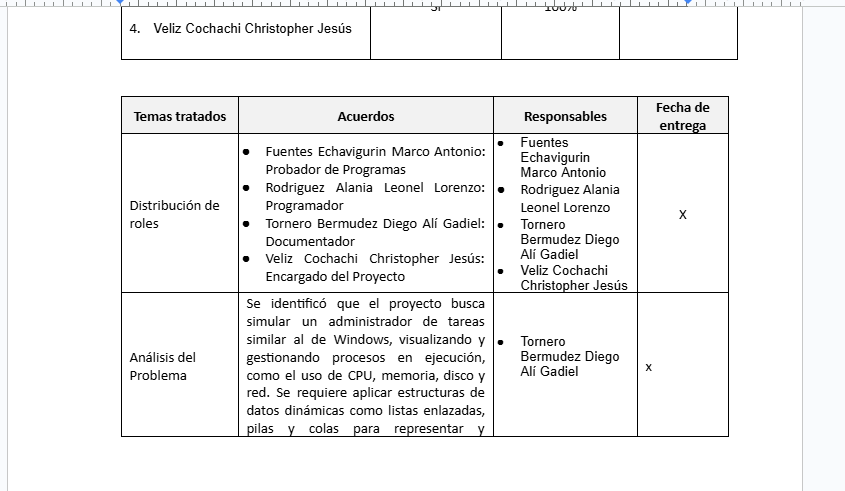






## **4.3 Plan de Trabajo y Roles Asignados**

* Documento inicial:



* Cronograma:

| **Actividad** | **Responsable(s)** | **Fecha Inicio** | **Fecha Fin** | **Duración (días)** | **Procesos** |
| --- | --- | --- | --- | --- | --- |
| **Análisis de Requerimientos** | Todo el equipo | 17/05/2025 | 17/05/2025 | 1 | Análisis |
| **Diseño de Estructuras** | Leonel, Christopher | 18/05/2025 | 19/05/2025 | 2 | Análisis |
| **Implementación Lista (Procesos)** | Marco Antonio y Leonel Lorenzo | 20/05/2025 | 22/05/2025 | 3 | Diseño |
| **Implementación Cola (CPU)** | Christopher Jesús | 20/05/2025 | 22/05/2025 | 3 | Diseño |
| **Implementación Pila (Memoria)** | Diego Alí Gadiel | 20/05/2025 | 22/05/2025 | 3 | Diseño |
| **Creación de Lista (Procesos)** | Marco Antonio y Leonel Lorenzo | 23/05/2025 | 26/05/2025 | 3 | Creación script |
| **Creación de Cola (CPU)** | Christopher Jesús | 23/05/2025 | 26/05/2025 | 3 | Creación script |
| **Creación de Pila (Memoria)** | Diego Alí Gadiel | 23/05/2025 | 26/05/2025 | 3 | Creación script |
| **Integración de Módulos** | Diego Alí Gadiel | 23/05/2025 | 24/05/2025 | 2 | Lista/Cola/Pila implementadas |
| **Menú General y Validaciones** | Christopher Jesús | 25/05/2025 | 26/05/2025 | 2 | Integración |
| **Pruebas** | Marco Antonio | 27/05/2025 | 27/05/2025 | 1 | Integración |
| **Correcciones de problemas** | Todo el equipo | 01/06/2025 | 02/06/2025 | 1 | Corrección |
| **Mejoras del programa** | Todo el equipo | 03/06/2025 | 04/06/2025 | 1 | Sacar el programa final |
| **Documentación Final** | Diego Alí Gadiel | 01/06/2025 | 03/06/2025 | 2 | Implementación completa |
| **Entrega Final** | Todo el equipo | 03/06/2025 | 04/06/2025 | 1 | Pruebas y documentación |

* Roles:

| **Rol** | **Responsable** | **Tareas Principales** |
| --- | --- | --- |
| Diseñador | Leonel, Christopher | Diseñar estructuras (lista, cola, pila) |
| Programador | Marco, Leonel | Implementar lista de procesos |
| Programador | Christopher | Implementar cola de CPU |
| Programador | Diego | Implementar pila de memoria |
| Integrador | Diego | Unir módulos y resolver conflictos |
| Tester | Marco | Probar funcionalidades y reportar errores |
| Documentador | Diego | Redactar documentación final |
| Coordinador | Christopher | Gestionar menú principal y validaciones |

## **4.4 Registro de Reuniones**

**ACTA DE REUNIÓN N° 01**

| Asignatura | **ESTRUCTURA DE DATOS** | Fecha: | 17/05/2025 |
| --- | --- | --- | --- |
| Responsable de grupo | Veliz Cochachi Christopher Jesús | Hora de inicio: | 21:00 |
| Modalidad de Reunión | Virtual | Hora de fin | 23:00 |

**Integrantes:**

| **Apellidos y nombres** | **Asistió (Si/No)** | **% Participación** | **Firma** |
| --- | --- | --- | --- |
| 1. Fuentes Echavigurin Marco Antonio | Si | 100% |  |
| 1. Rodriguez Alania Leonel Lorenzo | Si | 100% |  |
| 1. Tornero Bermudez Diego Alí Gadiel | Si | 100% |  |
| 1. Veliz Cochachi Christopher Jesús | Si | 100% |  |

| **Temas tratados** | **Acuerdos** | **Responsables** | **Fecha de entrega** |
| --- | --- | --- | --- |
| Distribución de roles | * Fuentes Echavigurin Marco Antonio: Probador de Programas * Rodriguez Alania Leonel Lorenzo: Programador * Tornero Bermudez Diego Alí Gadiel: Documentador * Veliz Cochachi Christopher Jesús: Encargado del Proyecto | * Fuentes Echavigurin Marco Antonio * Rodriguez Alania Leonel Lorenzo * Tornero Bermudez Diego Alí Gadiel * Veliz Cochachi Christopher Jesús | X |
| Análisis del Problema | Se identificó que el proyecto busca simular un administrador de tareas similar al de Windows, visualizando y gestionando procesos en ejecución, como el uso de CPU, memoria, disco y red. Se requiere aplicar estructuras de datos dinámicas como listas enlazadas, pilas y colas para representar y manipular estos procesos en un entorno educativo. | * Tornero Bermudez Diego Alí Gadiel | x |
| Diseño de la Estructura de Datos | Se propuso una arquitectura con tres módulos:   * Gestor de procesos: usa listas enlazadas. * Planificador de CPU: usa colas de prioridad. * Gestor de memoria: usa pilas. | * Rodriguez Alania Leonel Lorenzo * Veliz Cochachi Christopher Jesús | x |
| Redacción del documento de diseño | El equipo acordó redactar el documento incluyendo:   * El análisis del problema. * La distribución de roles. * El diseño del sistema con pseudocódigo. * Diagramas de las estructuras de datos. * Ejemplos de uso del sistema desde la consola. | * Fuentes Echavigurin Marco Antonio * Veliz Cochachi Christopher Jesús | x |

Observaciones:

* Todos los miembros participaron activamente en la discusión.
* Se acordó trabajar en colaboración mediante Google Docs y GitHub.
* Se dejó clara la importancia de cumplir los roles asignados para asegurar el éxito del proyecto.

**Evidencias de trabajo Grupal**

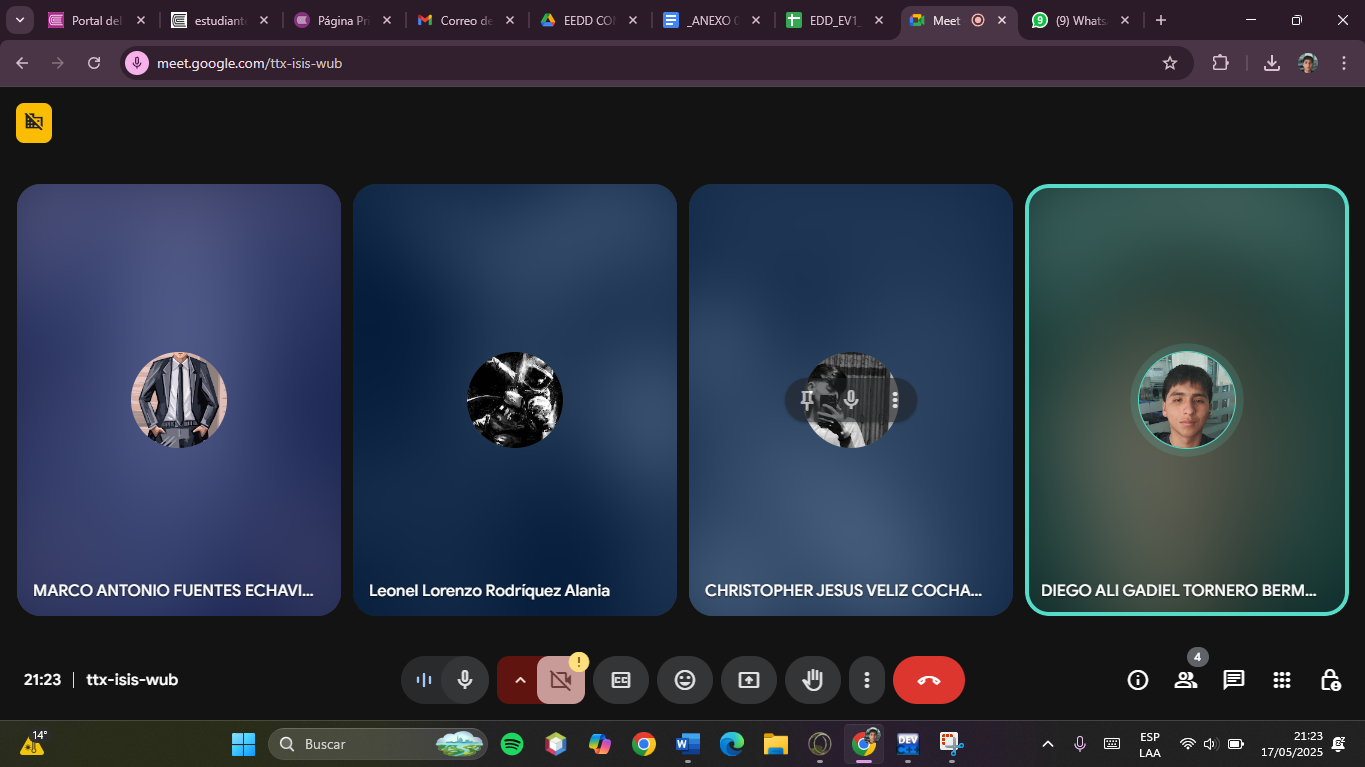
Enlace de Herramienta Colaborativa:

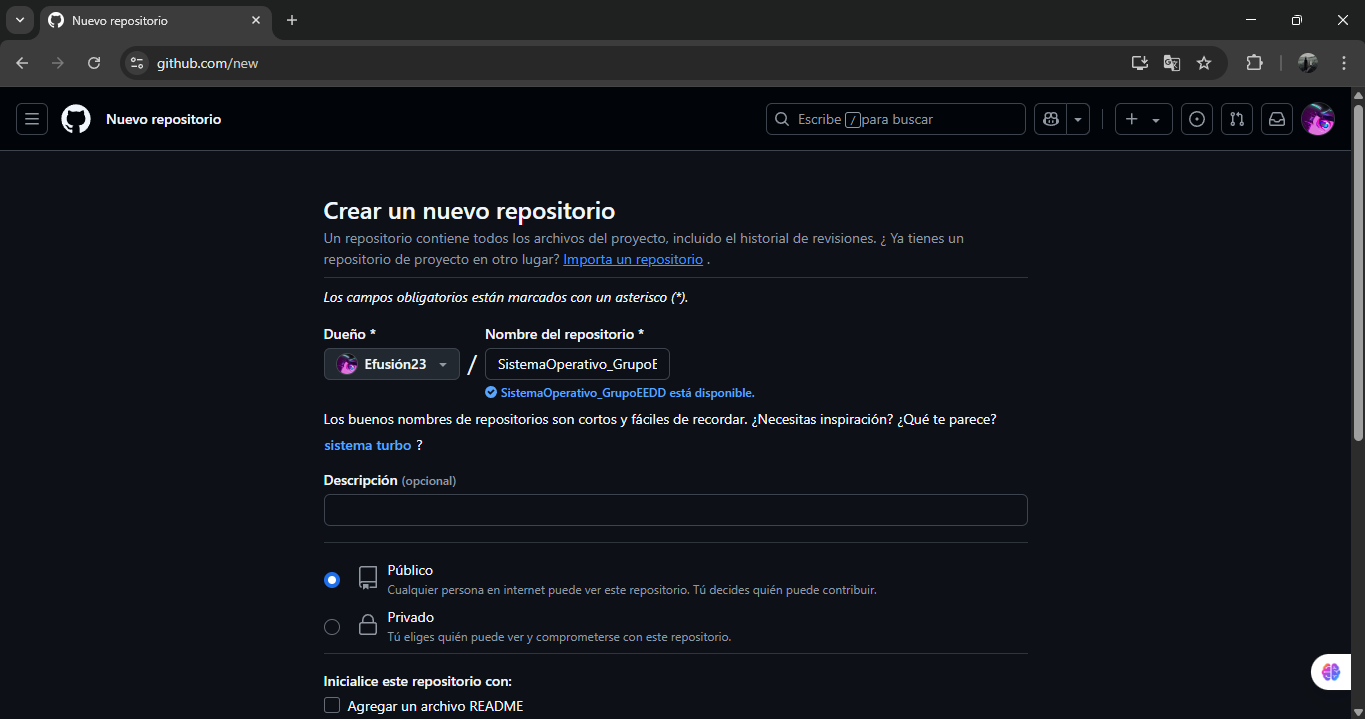
<https://docs.google.com/document/d/1y8Tl73opJkXFT-bAz7Or_66C82elLJbRi4I84SEzGtM/edit?usp=sharing>

<https://meet.google.com/landing>

<https://github.com/Efustion23/SistemaOperativo_GrupoEEDD>

Fotografías





**ACTA DE REUNIÓN N° 02**

| Asignatura | **ESTRUCTURA DE DATOS** | Fecha: | 26/05/2025 |
| --- | --- | --- | --- |
| Responsable de grupo | Veliz Cochachi Christopher Jesús | Hora de inicio: | 14:30 |
| Modalidad de Reunión | Virtual | Hora de fin | 17:00 |

**Integrantes:**

| **Apellidos y nombres** | **Asistió (Si/No)** | **Participación %** |
| --- | --- | --- |
| 1. Fuentes Echavigurin Marco Antonio | Si | 100% |
| 1. Rodriguez Alania Leonel Lorenzo | Si | 100% |
| 1. Tornero Bermudez Diego Alí Gadiel | Si | 100% |
| 1. Veliz Cochachi Christopher Jesús | Si | 100% |

| **Temas tratados** | **Acuerdos** | **Responsables** | **Fecha de entrega** |
| --- | --- | --- | --- |
| Análisis de estructuras dinámicas (lista, pila, cola) | Se definió que se usará lista para procesos, cola para CPU y pila para memoria | Todo el equipo | 28/05/2025 |
| División de tareas e implementación del sistema por módulos | Cada integrante trabajará en una estructura diferente y luego se integrará | Marco: lista, Leonel: cola, Diego: pila, Christopher: menú general | 28/05/2025 |
| Implementación de la estructura de pila con uso de memoria en MB | Se agregó campo memoriaMB como float para mejorar realismo del sistema | Diego Alí Gadiel Tornero Bermúdez | 28/05/2025 |
| Mejora del diseño visual de los menús | Se acordó usar cajas ASCII para hacerlo más presentable en consola | Christopher Jesús Veliz Cochachi | 28/05/2025 |

Observaciones:

* Todos los integrantes del equipo asistieron puntualmente a la reunión y participaron activamente en la planificación y desarrollo del sistema.
* El avance logrado hasta esta semana ha sido significativo. Se han implementado correctamente las estructuras dinámicas (lista, cola y pila) y se han vinculado de manera funcional a través de un menú interactivo.
* La comunicación entre los integrantes ha sido fluida y efectiva, lo que permitió repartir tareas equitativamente y resolver dudas de manera colaborativa.
* Se destaca el compromiso de cada miembro al cumplir con sus tareas asignadas y realizar sus respectivos **commits en GitHub**.

**Evidencias de trabajo Grupal**

Enlace de Herramienta Colaborativa:

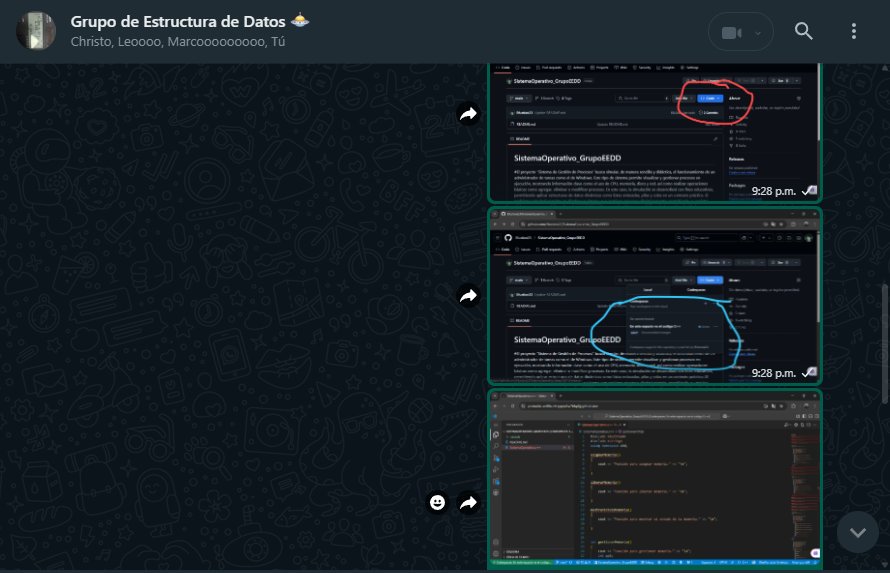
<https://meet.google.com/landing>

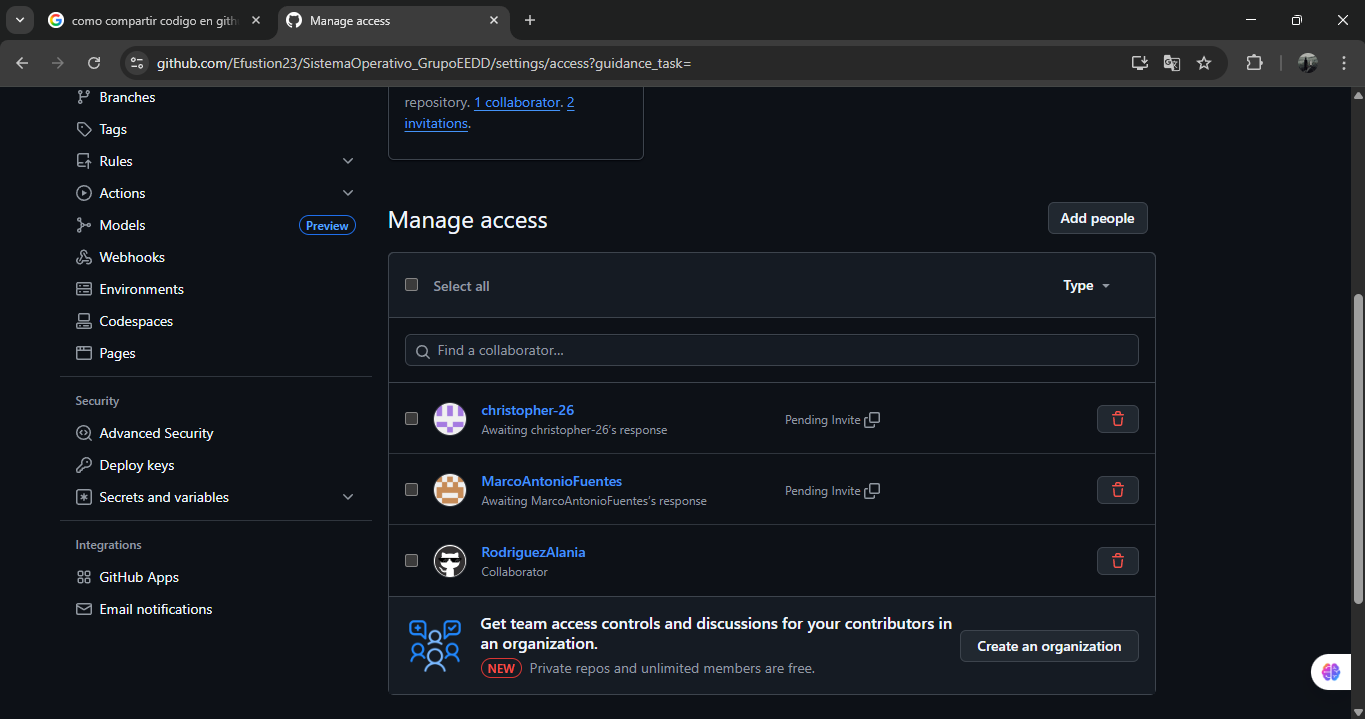
<https://github.com/Efustion23/SistemaOperativo_GrupoEEDD>

Fotografías









**ACTA DE REUNIÓN N° 03**

| Asignatura | **ESTRUCTURA DE DATOS** | Fecha: | 03/06/2025 |
| --- | --- | --- | --- |
| Responsable de grupo | Veliz Cochachi Christopher Jesús | Hora de inicio: | 15:00 |
| Modalidad de Reunión | Virtual | Hora de fin | 18:00 |

**Integrantes:**

| **Apellidos y nombres** | **Asistió (Si/No)** | **Participación %** |
| --- | --- | --- |
| 1. Fuentes Echavigurin Marco Antonio | Si | 100% |
| 1. Rodriguez Alania Leonel Lorenzo | Si | 100% |
| 1. Tornero Bermudez Diego Alí Gadiel | Si | 100% |
| 1. Veliz Cochachi Christopher Jesús | Si | 100% |

| **Temas tratados** | **Acuerdos** | **Responsables** | **Fecha de entrega** |
| --- | --- | --- | --- |
| **Corrección de errores en módulos** | Se revisaron bugs en la lista, cola y pila. Se asignaron correcciones. | Todo el equipo | 03/06/2025 |
| **Optimización del menú principal** | Mejorar navegación y validaciones de entrada. | Christopher Jesús Veliz Cochachi | 03/06/2025 |
| **Pruebas finales del sistema integrado** | Verificar funcionalidad completa (procesos, CPU, memoria). | Marco Antonio Fuentes Echavigurin | 03/06/2025 |
| **Preparación de documentación final** | Incluir manual de usuario y técnico. Subir a GitHub. | Diego Alí Gadiel Tornero Bermúdez | 04/06/2025 |

Observaciones:

* Todos los integrantes participaron activamente en la identificación y solución de errores.
* El sistema está funcional, pero se requiere pulir detalles de usabilidad y documentación.
* Se destacó el trabajo colaborativo en GitHub, con commits frecuentes y merges organizados.
* Se acordó una reunión adicional el 04/06 para revisar la entrega final.

**Evidencias de trabajo Grupal**

Enlace de Herramienta Colaborativa:

<https://meet.google.com/landing>

<https://github.com/Efustion23/SistemaOperativo_GrupoEEDD>

<https://lucid.app/lucidchart/6264498a-9ad1-428d-9df6-661d94e3cacd/edit?viewport_loc=962%2C-2137%2C13862%2C6187%2C0_0&invitationId=inv_45e3b14a-b9ba-4bfd-8a38-678956cd0a93>

Fotografías

