Documento de Diseño – Semana 3

Proyecto: *Sistema de Gestión de Procesos*

Curso: Estructuras de Datos
Unidad: Unidad 3 – Estructuras de Datos Dinámicas Lineales
Equipo: Grupo 7

1. Análisis del Problema

Descripción General

El presente proyecto se enmarca en el desarrollo de un Sistema de Gestión de Procesos (SGP), cuya finalidad es simular, de forma simplificada, el comportamiento de un sistema operativo en lo que respecta a la administración de procesos. Esta simulación se realiza exclusivamente a través del uso de estructuras de datos dinámicas lineales, como listas enlazadas, pilas y colas, que permiten gestionar la creación, planificación y asignación de recursos a procesos informáticos.

El sistema permitirá al usuario registrar nuevos procesos, realizar búsquedas por identificador o nombre, eliminar procesos existentes y modificar su prioridad. Para lograrlo, se utilizará una lista enlazada simple, donde cada nodo representará un proceso con sus respectivos atributos. Adicionalmente, se implementará una cola de prioridad, que actuará como planificador de ejecución, priorizando aquellos procesos con mayor importancia de acuerdo con el valor asignado en el campo "prioridad". Por otro lado, la gestión de memoria será simulada mediante una pila, representando la asignación y liberación de bloques de memoria.

Además, el sistema contará con una interfaz de usuario basada en consola, desde la cual se podrá acceder a las funcionalidades anteriormente descritas. Esta interfaz tiene como propósito facilitar la interacción con el sistema, permitiendo ingresar datos, visualizar resultados y ejecutar operaciones de forma secuencial y clara. Asimismo, se contemplará un mecanismo de persistencia de datos, con el fin de guardar y recuperar el estado del sistema entre ejecuciones, asegurando así la continuidad de la información y una experiencia de uso más robusta.

En resumen, el proyecto busca proporcionar un entorno didáctico para la aplicación de estructuras de datos dinámicas en un caso práctico relacionado con los fundamentos de los sistemas operativos, contribuyendo así al desarrollo de habilidades de programación estructurada, lógica algorítmica y abstracción de procesos computacionales (Silberschatz, Galvin, & Gagne, 2021).

Requerimientos Funcionales

- Registrar, buscar, eliminar y modificar procesos.
- Planificar la ejecución de procesos por prioridad.
- Asignar y liberar bloques de memoria a los procesos.
- Interacción mediante interfaz de usuario.
- Persistencia de datos (guardar/cargar estado).

2. Diseño de la Solución

Estructuras de Datos a Implementar

1. Lista Enlazada: Gestor de Procesos

- Cada nodo representa un proceso con atributos:
    - ID de proceso (int)
    - Nombre (string)
    - Prioridad (int)
    - Estado (activo/inactivo)
- Operaciones:
    - Inserción de procesos
    - Eliminación por ID

- Búsqueda por ID o nombre
- Modificación de prioridad

2. Cola de Prioridad: Planificador de CPU

- Simula la ejecución ordenada de procesos.
- Los procesos con mayor prioridad se ejecutan antes.
- Operaciones:
  - Encolamiento por prioridad
  - De Encolamiento (ejecución)
  - Visualización de la cola

3. Pila: Gestor de Memoria

- Simula bloques de memoria disponibles/asignados.
- Operaciones:
  - Asignación (push)
  - Liberación (pop)
  - Visualización de la pila

3. Diagramas de las Estructuras

Diagrama: Lista Enlazada (Gestor de Procesos)

DESCRIPCIÓN: Esta estructura es una lista enlazada donde cada nodo va a tener información de un proceso y un puntero al siguiente nodo.

Diagrama: Cola de Prioridad

DESCRIPCIÓN: Esta es una cola de prioridad y nos ayuda a organizar los procesos por orden, osea los primos serán los de mayor valor o menor valor según le indiquemos.

Diagrama: Pila de Memoria

DESCRIPCIÓN: Esta estructura la usamos para gestionar los bloques de memoria y también puede simular el uso de la pila.

4. Pseudocódigo de Operaciones Principales

4.1 Insertar Proceso en Lista

```
VARIABLE listaProcesos ← NULL

FUNCIÓN insertarProceso(id, nombre, prioridad, estado):
    nuevo ← crear_nodo()
    nuevo.id ← id
    nuevo.nombre ← nombre
    nuevo.prioridad ← prioridad
    nuevo.estado ← estado
    nuevo.siguiente ← NULL

    SI listaProcesos = NULL ENTONCES
        listaProcesos ← nuevo
    SINO
        actual ← listaProcesos
        MIENTRAS actual.siguiente ≠ NULL HACER
            actual ← actual.siguiente
        FIN MIENTRAS
```

```
      actual.siguiente ← nuevo
   FIN SI
FIN FUNCIÓN

FUNCIÓN mostrarProcesos():
   actual ← listaProcesos
   MIENTRAS actual ≠ NULL HACER
      MOSTRAR "ID:", actual.id, "Nombre:", actual.nombre, "Prioridad:", actual.prioridad, "Estado:",
actual.estado
      actual ← actual.siguiente
```

4.2 Encolar Proceso por Prioridad

```
VARIABLE frente ← NULL

FUNCIÓN encolarProceso(id, nombre, prioridad):
   nuevo ← crear_nodo()
   nuevo.id ← id
   nuevo.nombre ← nombre
   nuevo.prioridad ← prioridad
   nuevo.siguiente ← NULL

   SI frente = NULL O prioridad < frente.prioridad ENTONCES
      nuevo.siguiente ← frente
      frente ← nuevo
   SINO
      actual ← frente
      MIENTRAS actual.siguiente ≠ NULL Y actual.siguiente.prioridad ≤ prioridad HACER
         actual ← actual.siguiente
      FIN MIENTRAS
      nuevo.siguiente ← actual.siguiente
      actual.siguiente ← nuevo
   FIN SI
FIN FUNCIÓN

FUNCIÓN mostrarCola():
   actual ← frente
   MIENTRAS actual ≠ NULL HACER
      MOSTRAR "ID:", actual.id, "Nombre:", actual.nombre, "Prioridad:", actual.prioridad
      actual ← actual.siguiente
   FIN MIENTRAS
FIN FUNCIÓN

FUNCIÓN ejecutarProceso():
   SI frente = NULL ENTONCES
      MOSTRAR "No hay procesos en cola"
      RETORNAR
   FIN SI
   MOSTRAR "Ejecutando proceso:", frente.nombre
```

```
    temp ← frente
    frente ← frente.siguiente
    liberar(temp)
```

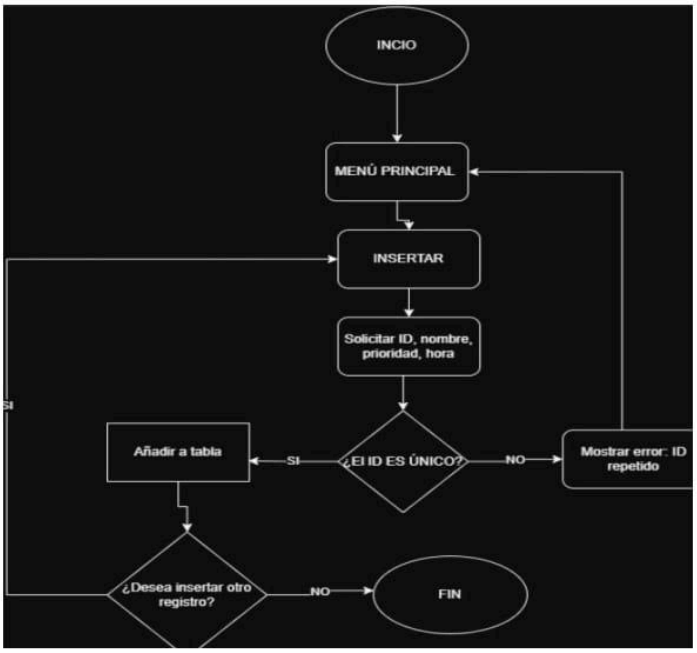
4.3 Asignar Memoria (Pila)

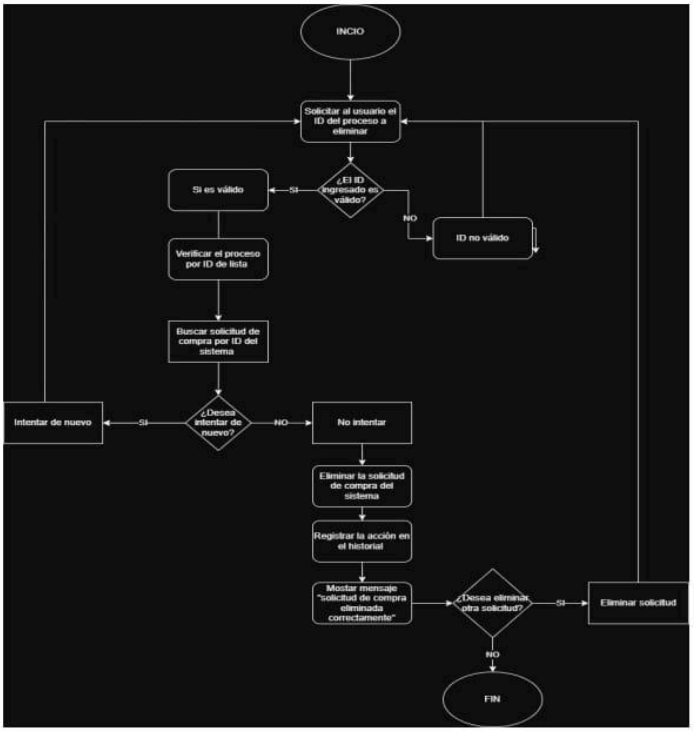
```
VARIABLE pilaMemoria ← NULL

FUNCIÓN asignarMemoria(id, tamanio):
    nuevo ← crear_nodo()
    nuevo.id ← id
    nuevo.tamanio ← tamanio
    nuevo.siguiente ← pilaMemoria
    pilaMemoria ← nuevo
    MOSTRAR "Memoria asignada de", tamanio, "KB al proceso", id
FIN FUNCIÓN

FUNCIÓN liberarMemoria():
    SI pilaMemoria = NULL ENTONCES
        MOSTRAR "No hay memoria asignada para liberar"
        RETORNAR
    FIN SI
    temp ← pilaMemoria
    MOSTRAR "Liberando memoria del proceso:", temp.id
    pilaMemoria ← pilaMemoria.siguiente
    liberar(temp)
FIN FUNCIÓN
```

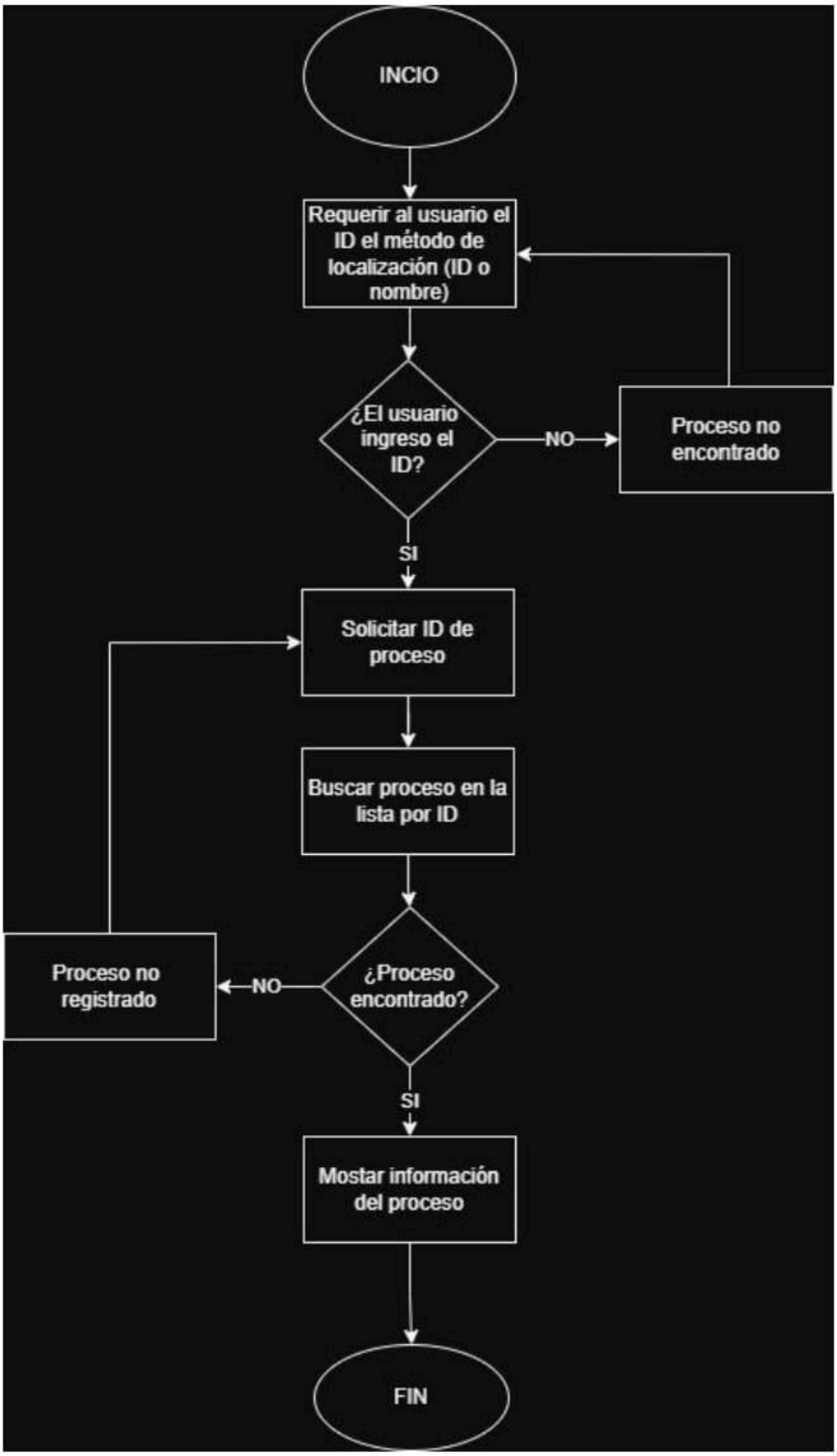
4.4 Diagrama de Flujo

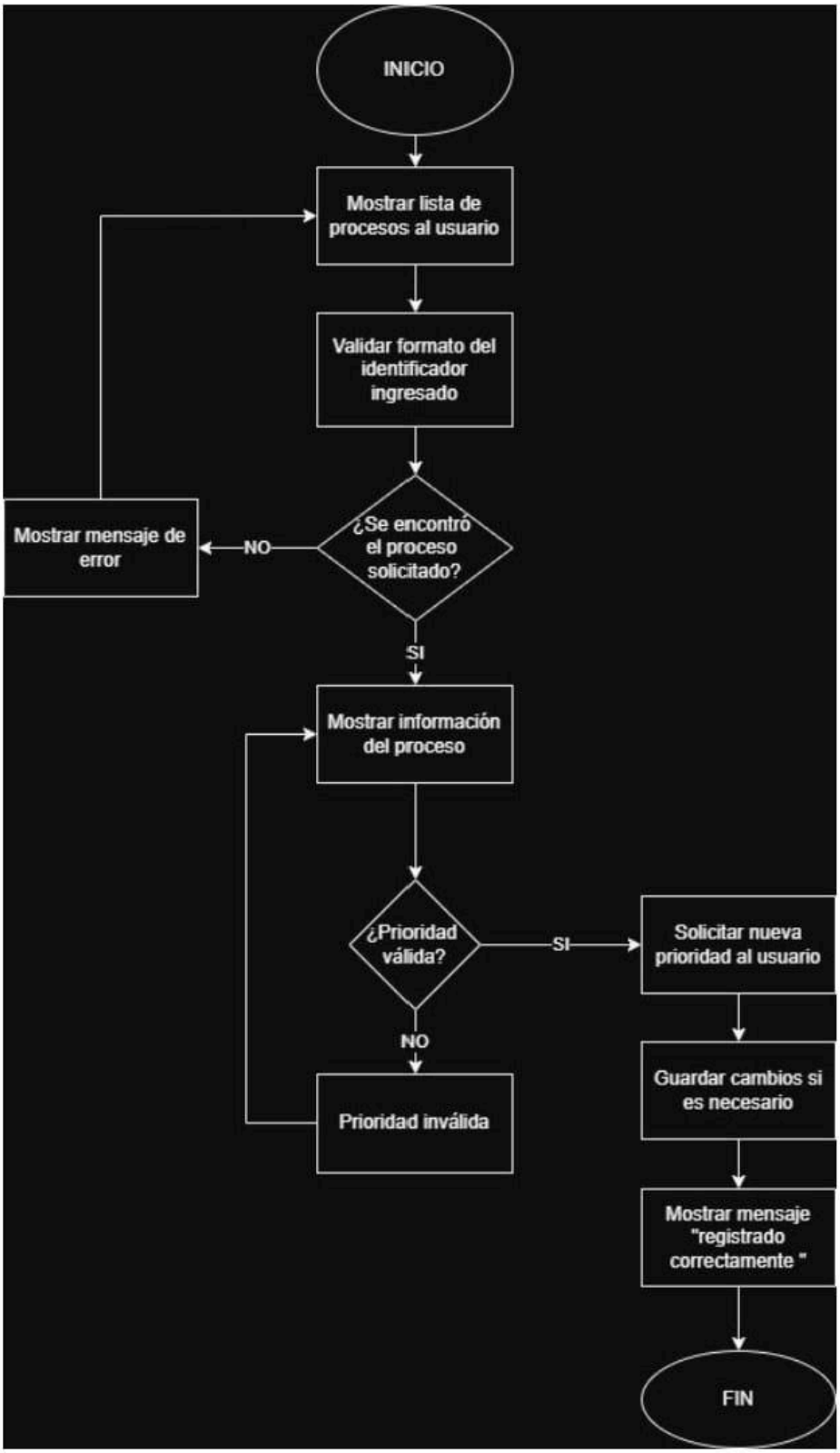
Diagrama de las estructuras de datos

1. Gestor de Procesos

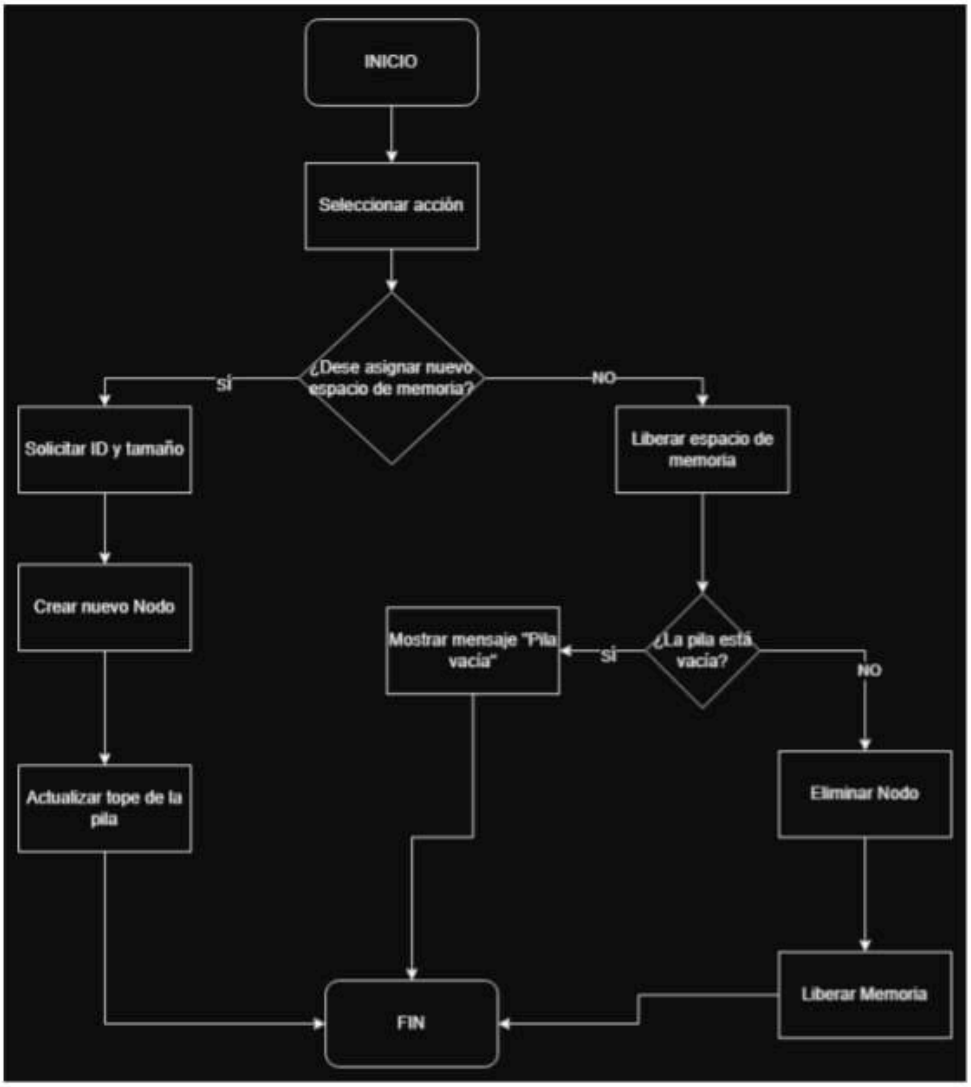




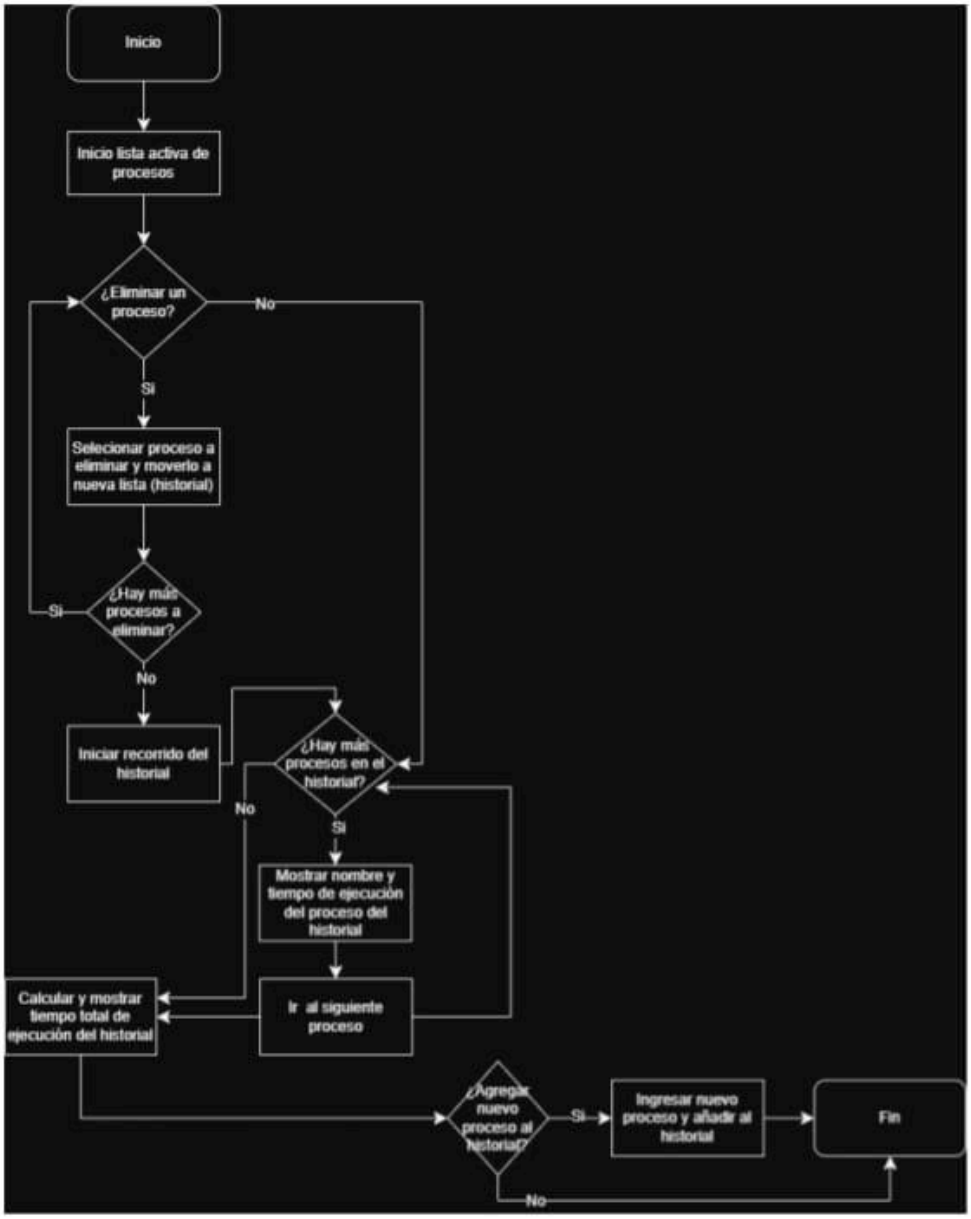

INCIO
Requerir al usuario el ID el método de localización (ID o nombre)
¿El usuario ingreso el ID?
NO
Proceso no encontrado
SI
Solicitar ID de proceso
Buscar proceso en la lista por ID
¿Proceso encontrado?
NO
Proceso no registrado
SI
Mostar información del proceso
FIN


INICIO
Mostrar lista de procesos al usuario
Validar formato del identificador ingresado
¿Se encontró el proceso solicitado?
NO
Mostrar mensaje de error
SI
Mostrar información del proceso
¿Prioridad válida?
SI
Solicitar nueva prioridad al usuario
NO
Prioridad inválida
Guardar cambios si es necesario
Mostrar mensaje "registrado correctamente "
FIN

1.2. Gestor de Memoria



1.3. Historial de proceso



5. Planificación de Tareas y Responsabilidades

Integrante	Rol inicial	Tareas asignadas
Anthony Castro	Programador	Implementar lista enlazada y los procesos (Descripción)
Anderson Solis Sedano	Documentación	Redacción del documento
Jair Limas Chagua	Encargado del código	Interfaz y cola de prioridad
Antoni Quispe Linares	Programador	Implementación de pila y pruebas unitarias

6. Manual de Usuario

Manual de Usuario
Sistema de Gestión de Procesos
Descripción General
Este programa permite gestionar procesos simulando la administración de memoria y CPU. El usuario puede registrar procesos, asignarlos a memoria, eliminarlos, modificarlos y simular su ejecución en la CPU.

Menú Principal
Al iniciar el programa, se muestra el siguiente menú:

Seleccione una opción ingresando el número correspondiente y presione Enter.

1. Introducción

Este manual te guiará detalladamente en el uso del programa de Gestión de Procesos, explicando cada función y opción disponible.

2. Primeros Pasos

2.1 Compilación y Ejecución

1. Copia el código completo y guárdalo en un archivo llamado `gestor_procesos.cpp`
2. Abre una terminal o símbolo del sistema
3. Navega hasta la ubicación del archivo
4. Compila con: `g++ gestor_procesos.cpp -o gestor_procesos`
5. Ejecuta con: `./gestor_procesos` (Linux/Mac) o `gestor_procesos.exe` (Windows)

2.2 Pantalla Inicial

Al iniciar verás el menú principal:

```
      ***Menu de Gestion***
      [1]. Gestion de Procesos
      [2]. Gestion de Memoria
      [3]. Gestion de CPU
[4]. Salir
```

3. Uso Detallado

3.1 Gestión de Procesos (Opción 1)

Registrar nuevos procesos

1. Selecciona opción 1 en menú principal
2. Elige opción 1 (Registrar proceso)
3. Ingresa cuántos procesos deseas registrar (ej. 2)
4. Para cada proceso:
    - Ingresa ID (número único): ej. 101
    - Nombre: ej. "Chrome"
    - Prioridad (número menor = más prioritario): ej. 1

Ejemplo completo:

```
      Cuantos procesos desea registrar? 2

      Registro del proceso 1
```

```
      Ingrese el ID del proceso: 101
      Ingrese el nombre del proceso: Chrome
      Ingrese la prioridad del proceso: 1

      Registro del proceso 2
      Ingrese el ID del proceso: 102
      Ingrese el nombre del proceso: Word
Ingrese la prioridad del proceso: 3
```

Ver procesos registrados

1. Menú principal → Opción 1 → Opción 2
2. Verás lista completa con formato:

   ```
   ID: 101, Nombre: Chrome, Prioridad: 1
   ```

3. `ID: 102, Nombre: Word, Prioridad: 3`

Buscar proceso

1. Menú principal → Opción 1 → Opción 3
2. Ingresa el ID a buscar (ej. 101)
3. Verás los detalles si existe

Eliminar proceso

1. Menú principal → Opción 1 → Opción 4
2. Ingresa el ID a eliminar (ej. 102)
3. Confirmación: "Proceso eliminado"

Modificar proceso

1. Menú principal → Opción 1 → Opción 5
2. Ingresa el ID a modificar (ej. 101)
3. Ingresa nuevos valores para nombre y prioridad

3.2 Gestión de Memoria (Opción 2)

Asignar procesos a memoria

1. Menú principal → Opción 2 → Opción 1
2. Verás lista de procesos disponibles
3. Ingresa el ID del proceso a cargar en memoria (ej. 101)
4. Confirmación: "Proceso agregado a la memoria"

Nota: Máximo 5 procesos en memoria

Quitar proceso específico de memoria

1. Menú principal → Opción 2 → Opción 2
2. Verás lista de procesos en memoria
3. Ingresa ID a remover (ej. 101)
4. Confirmación: "Proceso eliminado de la memoria"

Liberar toda la memoria

1. Menú principal → Opción 2 → Opción 3
2. Confirmación: "Todos los procesos quitados de la memoria"

Ver procesos en memoria

1. Menú principal → Opción 2 → Opción 4
2. Verás lista con formato:
3. `ID: 101, Nombre: Chrome, Prioridad: 1`

3.3 Gestión de CPU (Opción 3)

Ver orden de ejecución

1. Menú principal → Opción 3 → Opción 1

   ```
   Orden de ejecucion
   ID: 101, Nombre: Chrome, Prioridad: 1 (se ejecutará primero)
   ```

2. `ID: 102, Nombre: Word, Prioridad: 3`

Ejecutar procesos

1. Menú principal → Opción 3 → Opción 2

2. Verás ejecución simulada:

```
Ejecutando -> ID: 101, Nombre: Chrome, Prioridad: 1
[Pausa de 1 segundo]
Ejecutando -> ID: 102, Nombre: Word, Prioridad: 3
```

3. `Todos los procesos han sido ejecutados`

4. Consejos y Buenas Prácticas

1. Asigna prioridades bajas (1-3) a procesos importantes
2. Verifica siempre el orden de ejecución antes de procesar
3. Libera memoria cuando termines de ejecutar procesos
4. Usa IDs numéricos únicos para evitar confusiones

5. Solución de Problemas

- Error "Memoria llena": Libera espacio con Opción 2 → 3
- Proceso no encontrado: Verifica IDs con Opción 1 → 2
- Ejecución vacía: Asegúrate de asignar procesos a memoria primero

6. Ejemplo Completo de Flujo

1. Registrar 2 procesos (Chrome prio 1, Word prio 3)
2. Asignar ambos a memoria
3. Ver orden ejecución (Chrome primero)
4. Ejecutar procesos
5. Liberar memoria
6. Salir del programa

7. Explicación de la estructuras de datos implementadas

- **Lista Enlazada Simple de Procesos**
  ¿Qué es?
  Una lista enlazada simple es una estructura dinámica donde cada elemento (nodo) contiene datos y un puntero al siguiente nodo. En este programa, cada nodo representa un proceso.

  Implementación:

```
struct Proceso {
    int id;
    string nombre;
    int prioridad;
    Proceso* siguiente;
    // Constructor...
};
```

  ¿Para qué se usa?
  Registrar procesos: Cada vez que el usuario registra un proceso, se crea un nodo y se agrega al final de la lista.
  Buscar, modificar y eliminar: Permite recorrer la lista para buscar procesos por ID, modificarlos o eliminarlos.
  Visualizar: Se recorre la lista para mostrar todos los procesos registrados.
  Ejemplo visual:

  [Proceso1] -> [Proceso2] -> [Proceso3] -> NULL

- **Pila (Stack) para Memoria RAM**
  ¿Qué es?
  Una pila es una estructura de tipo LIFO (Last In, First Out), donde el último elemento en entrar es el primero en salir. Aquí se simula la memoria RAM.

  Implementación:

```
Proceso* pila[MAX];
int tope = -1;
```

  ➔ **pila** es un arreglo de punteros a procesos.

➔ **tope** indica la posición del último proceso en la pila.

¿Para qué se usa?

Asignar procesos a memoria: Cuando el usuario asigna un proceso a memoria, se agrega al tope de la pila.

Eliminar de memoria: Se puede eliminar un proceso específico de la pila (no necesariamente el último).

Liberar memoria: Vacía toda la pila.

Visualizar memoria: Muestra los procesos actualmente en la pila.

Ejemplo visual:

Tope
↓
[Proc3]
[Proc2]
[Proc1]

3. Cola (Queue) para la CPU

¿Qué es?

Una cola es una estructura de tipo FIFO (First In, First Out), pero en este caso la inserción es ordenada por prioridad (mayor prioridad, más adelante en la cola).

Implementación:

```
Proceso* cola[MAX];
int frente = -1, final = -1;
```

➔ **cola** es un arreglo de punteros a procesos.

➔ **frente y final** indican el inicio y fin de la cola.

¿Para qué se usa?

Encolar procesos: Cuando un proceso se asigna a memoria, también se encola para la CPU, insertándolo según su prioridad.

Ejecutar procesos: Se ejecutan en orden de prioridad, eliminando del frente de la cola.

Visualizar cola: Muestra el orden de ejecución de los procesos.

Ejemplo visual:

frente → [PAlta] [PMedia] [PBaja] ← final

Donde PAlta es el de mayor prioridad.

Relación entre las estructuras
Lista enlazada: Es el registro maestro de todos los procesos.
Pila: Solo los procesos que el usuario asigna a memoria pasan aquí (máximo 5).
Cola: Cuando un proceso entra a la pila, también se encola para la CPU, ordenado por prioridad.

Flujo típico:

Registrar procesos → Asignar a memoria (pila) → Encolar para CPU → Ejecutar procesos (cola)

8. Código implementado

```
#include <iostream>
#include <string>
#include <fstream>
#include <windows.h> // Para usar funciones de Windows como Sleep()
#define MAX 5

using namespace std;



// Lista enlazada para almacenar los procesos

struct Proceso { // Estructura que representa un proceso
    // Atributos del proceso
    int id;
    string nombre;
    int prioridad;
    Proceso* siguiente; // Puntero al siguiente proceso en la lista

    // Constructor para inicializar un nuevo proceso
    Proceso(int ID, string Nombre, int Prioridad) {
        id = ID;
        nombre = Nombre;
        prioridad = Prioridad;
        siguiente = NULL; // Inicializa el puntero siguiente como NULL
    }
};
```

```
// Pila de memoria para los procesos

Proceso* pila[MAX]; // Arreglo que representa la pila de memoria
int tope = -1; // Variable que indica el índice del último elemento en la pila

// Cola para gestionar la memoria

Proceso* cola[MAX]; // Arreglo que representa la cola de memoria
int frente = -1; // Índice del primer elemento en la cola
int final = -1; // Índice del último elemento en la cola

// Funcion para encolar elementos de la memoria

void encolar() {
    if (tope == -1) { // Verifica si la pila está vacía
        cout << "La pila de memoria está vacía. No se puede encolar." << endl;
        return;
    }
    Proceso* nuevo = pila[tope]; // Toma el proceso del tope de la pila

    if (frente == -1) { // Verifica si la cola está vacía
        frente = final = 0; // Si la cola está vacía, inicializa el frente
        cola[0] = nuevo; // Agrega el nuevo proceso al frente de la cola
        return;
    }

    int pos = frente; // Comienza desde el frente de la cola
    while (pos <= final && cola[pos]->prioridad >= nuevo->prioridad) { // Compara las prioridades
de los procesos
        pos++; // Encuentra la posición correcta para insertar el nuevo proceso según su prioridad
    }

    // Desplazar los elementos para hacer espacio
    for (int i = final + 1; i > pos; i--) { // Organiza la cola
        cola[i] = cola[i - 1]; // Mueve los elementos hacia la derecha
    }
    cola[pos] = nuevo; // Inserta el nuevo proceso en la posición encontrada
    final++; // Incrementa el índice del final de la cola
}

// Funcion que elimina elementos de la cola por ID

void eliminarDeColaPorID(int idEliminar) {
```

```
    if (frente == -1) { // Verifica si la cola está vacía
        return;
    }
    int pos = -1; // Variable para almacenar la posición del proceso a eliminar
    for (int i = frente; i <= final; i++) { // Recorre la cola desde el frente hasta el final
        if (cola[i]->id == idEliminar) { // Busca el proceso con el ID especificado
            pos = i; // Guarda la posición del proceso a eliminar
            break;
        }
    }
    if (pos == -1) return; // Si no se encontró el proceso, sale de la función
    for (int i = pos; i < final; i++) { // Desplaza los elementos hacia la izquierda para llenar el
hueco
        cola[i] = cola[i + 1]; // Mueve los elementos hacia la izquierda
    }
    final--; // Decrementa el índice del final de la cola
    if (final < frente) frente = final = -1; // Si la cola queda vacía, reinicia los índices
}

// Funcion para borrar todos los elementos de la cola

void vaciarCola() {
    frente = final = -1; // Reinicia los índices de la cola para indicar que está vacía
}

// Ejecutar Procesos en la cola

void ejecutarProcesosEnCola() {
    if (frente == -1) { // Verifica si la cola está vacía
        cout << "No hay procesos en la cola para ejecutar." << endl;
        return;
    }
    cout << "Ejecutando procesos en orden de prioridad:" << endl;
    while (frente != -1) { // Mientras haya procesos en la cola
        cout << "Ejecutando -> ID: " << cola[frente]->id // Imprime el ID del proceso que se está
ejecutando
             << ", Nombre: " << cola[frente]->nombre // Imprime el nombre del proceso
             << ", Prioridad: " << cola[frente]->prioridad << endl; // Imprime la prioridad del proceso
        Sleep(1000); // Pausa de 1 segundo
        if (frente == final) { // Verifica si es el último proceso en la cola
            frente = final = -1; // Cola vacía
        } else {
            frente++; // Avanza al siguiente proceso en la cola
        }
```

```
    }
    cout << "Todos los procesos han sido ejecutados." << endl;
}

//  Funcion para visualizar la cola de memoria

void visualizarCola() {
    if (frente == -1) { // Verifica si la cola está vacía
        cout << "La cola de CPU está vacía." << endl;
        return;
    } else {
        cout << "Orden de ejecucion" << endl;
        for (int i = frente; i <= final; i++) { // Recorre la cola desde el frente hasta el final
            cout << "ID: " << cola[i]->id << ", Nombre: " << cola[i]->nombre << ", Prioridad: " <<
cola[i]->prioridad << endl; // Imprime los detalles de cada proceso
        }
    }
}

// Funcion para agregar elementos a la pila

void insertar(Proceso*& listaProcesos){
    if (tope == MAX - 1) { // Verifica si la pila está llena
        cout << "Memoria llena. No se puede insertar mas elementos." << endl;
    } else {
        int id2;
        cout << "Ingrese el ID del proceso a asignar a la memoria: "; cin >> id2;
        Proceso* temp = listaProcesos; // Crea un puntero temporal para recorrer la lista de
procesos
        while (temp != NULL) { // Recorre la lista de procesos
            if (temp->id == id2) { // Verifica si el ID del proceso coincide con el ID ingresado
                pila[++tope] = temp; // Agrega el proceso a la pila
                cout << "Proceso con ID " << id2 << " agregado a la memoria." << endl;
                encolar(); // Llama a la función para encolar el proceso en la cola de memoria
                return; // Sale de la función una vez que se ha agregado el proceso
            }
            temp = temp->siguiente; // Avanza al siguiente proceso en la lista
        }
        cout << "No se encontró un proceso con ese ID." << endl;
    }
}

// Funcion para eliminar procesos de la pila
```

```
void eliminarDePila() {
    if (tope == -1) { // Verifica si la pila está vacía
        cout << "La pila de memoria está vacía. No se puede eliminar ningún proceso." << endl;
        return;
    }
    cout << "Procesos en la pila de memoria:" << endl;
    for (int i = 0; i <= tope; i++) { // Recorre la pila e imprime los IDs de los procesos
        cout << "ID: " << pila[i]->id << ", Nombre: " << pila[i]->nombre << ", Prioridad: " <<
pila[i]->prioridad << endl;
    }
    int idEliminar;
    cout << "Ingrese el ID del proceso que desea quitar de la pila: ";
    cin >> idEliminar;

    int pos = -1; // Variable para almacenar la posición del proceso a eliminar
    for (int i = 0; i <= tope; i++) { // Recorre la pila desde el inicio hasta el tope
        if (pila[i]->id == idEliminar) { // Busca el proceso con el ID especificado
            pos = i; // Guarda la posición del proceso a eliminar
            break;
        }
    }
    if (pos == -1) {
        cout << "No se encontró un proceso con ese ID en la pila." << endl; // Mensaje si no se
encuentra el proceso
        return;
    }
    Proceso* procesoEliminado = pila[pos]; // Guarda el proceso que se va a eliminar
    // Desplazar los elementos para llenar el hueco
    for (int i = pos; i < tope; i++) { // Desplaza los elementos hacia la izquierda
        pila[i] = pila[i + 1]; // Mueve los elementos hacia la izquierda
    }
    tope--; // Decrementa el índice del tope de la pila
    cout << "Proceso con ID " << procesoEliminado->id << " eliminado de la memoria." << endl;
    eliminarDeColaPorID(idEliminar); // <-- sincroniza la cola
}

// Funcion para liberar la memoria de los procesos en la pila

void liberarMemoria() {
    if (tope == -1) { // Verifica si la pila está vacía
        cout << "La pila de memoria está vacía. No hay procesos para liberar." << endl;
        return;
    }
    // Solo vacía la pila, no elimina los procesos de la lista enlazada
```

```
    tope = -1; // Reinicia el tope de la pila a -1 para indicar que está vacía
    cout << "Todos los procesos han sido quitados de la memoria (pila) exitosamente." << endl;
    vaciarCola(); // <-- sincroniza la cola
}

// Funcion para visualizar los procesos en la pila

void visualizarPila() {
    if (tope == -1) { // Verifica si la pila está vacía
        cout << "La pila de memoria está vacía." << endl; // Mensaje si la pila está vacía
    } else {
        cout << "Procesos en la pila de memoria:" << endl;
        for (int i = 0; i <= tope; i++) { // Recorre la pila e imprime los IDs de los procesos
            cout << "ID: " << pila[i]->id << ", Nombre: " << pila[i]->nombre << ", Prioridad: " <<
pila[i]->prioridad << endl;
        }
    }
}

// Función para agregar un nuevo proceso al final de la lista

void agregarAlFinal(Proceso*& listaProcesos, int ID, string Nombre, int Prioridad) {
    Proceso *nuevoProceso = new Proceso(ID, Nombre, Prioridad); // Crea un nuevo proceso
con los datos proporcionados

    if (listaProcesos == NULL) { // Verifica si la lista está vacía
        listaProcesos = nuevoProceso; // Si está vacía, el nuevo proceso se convierte en el primer
elemento de la lista
    } else {
        Proceso* temp = listaProcesos; // Crea un puntero temporal para recorrer la lista
        while (temp->siguiente) { // Recorre la lista hasta el final
            temp = temp->siguiente;
        }
        temp->siguiente = nuevoProceso; // Asigna el nuevo proceso al final de la lista
        cout << "Proceso agregado al final de la lista.\n";
    }
}

// Funcion para imprimir todos los procesos en la lista

void imprimirProcesos(Proceso* listaProcesos) {
    Proceso* temp = listaProcesos; // Crea un puntero temporal para recorrer la lista

    if (!temp) { // Verifica si la lista está vacía
```

```
        cout << "No hay procesos registrados.\n"; // Mensaje si la lista está vacía
        return;
    }
    cout << "Lista de procesos:\n";
    while (temp != NULL) { // Recorre la lista e imprime los datos de cada proceso
        cout << "ID: " << temp->id << ", Nombre: " << temp->nombre << ", Prioridad: " <<
temp->prioridad << endl;
        temp = temp->siguiente; // Avanza al siguiente proceso
    }
    cout << endl;
}

void eliminarProceso(Proceso*& listaProcesos, int id) {
    Proceso* temp = listaProcesos;
    // Creamos un puntero que buscara el proceso que indiquemos
    Proceso* anterior = NULL;
    // Creamos otro puntero que guardará el nodo anterior que deseqamos eliminar
    while (temp && temp->id != id) {
        anterior = temp;
        temp = temp->siguiente;
    }
    // Con esto podemos eliminar un proceso de la lista cuando el ID coincide
    if (!temp) {
        cout << "Proceso no encontrado.\n";
        return;
    }
    // Esto informa cuando no se encuentre ningun proceso con ese ID y termina la función
    if (!anterior) listaProcesos = temp->siguiente;
    else anterior->siguiente = temp->siguiente;
    delete temp;
    // Esto hace que libere la memoria del proceso eliminado
    cout << "Proceso eliminado.\n";
}
void buscarProceso(Proceso*& listaProcesos, int id) {
    Proceso* temp = listaProcesos;
    // Indicamos que comienza desde el inicio de la lista
    while (temp) {
        if (temp->id == id) {
            cout << "ID: " << temp->id << ", Nombre: " << temp->nombre << ", Prioridad: " <<
temp->prioridad << endl;
            return;
        }
        temp = temp->siguiente;
    }
```

```
    // Cuando encuentre el ID que busquemos lo imprimirá
    cout << "Proceso no encontrado.\n";
    // Nos informará si no lo encuentra
}

void modificarProceso(Proceso*& listaProcesos, int id) {
    Proceso* temp = listaProcesos;
    // Creamos un puntero que buscara el proceso que indiquemos
    while (temp) { // Recorremos la lista de procesos
        if (temp->id == id) { // Verificamos si el ID del proceso coincide con el ID buscado
            // Si encontramos el proceso, solicitamos los nuevos datos
            cout << "Ingrese el nuevo nombre del proceso: ";
            cin >> temp->nombre;
            cout << "Ingrese la nueva prioridad del proceso: ";
            cin >> temp->prioridad;
            cout << "Proceso modificado exitosamente.\n";
            return;
        }
        temp = temp->siguiente; // Avanzamos al siguiente proceso en la listaS
    }
    // Cuando encuentre el ID que busquemos lo modificará
    cout << "Proceso no encontrado.\n";
    // Nos informará si no lo encuentra
}

void Menu() {
    cout << "\n***Menu de Gestion***" << endl;
    cout << "[1]. Gestion de Procesos" << endl;
    cout << "[2]. Gestion de Memoria" << endl;
    cout << "[3]. Gestion de CPU" << endl;
    cout << "[4]. Salir" << endl;
}

void Menu2() {
    cout << "\n***Gestion de Procesos***" << endl;
    cout << "[1]. Registrar proceso" << endl;
    cout << "[2]. Imprimir procesos" << endl;
    cout << "[3]. Buscar procesos" << endl;
    cout << "[4]. Eliminar proceso" << endl;
    cout << "[5]. Modificar proceso" << endl;
    cout << "[6]. Salir" << endl;
    cout << "**************************" << endl;
}
```

```
void Menu3() {
    cout << "\n***Gestion de Memoria***" << endl;
    cout << "[1]. Asignar a memoria" << endl;
    cout << "[2]. Eliminar de memoria" << endl;
    cout << "[3]. Liberar memoria" << endl;
    cout << "[4]. Visualizar memoria" << endl;
    cout << "[5]. Salir" << endl;
    cout << "**************************" << endl;
}

void Menu4() {
    cout << "\n***Gestion de CPU***" << endl;
    cout << "[1]. Verificar orden de ejecucion" << endl;
    cout << "[2]. Ejecutar procesos" << endl;
    cout << "[3]. Salir" << endl;
    cout << "**************************" << endl;
}

int main() {

    Proceso* listaProcesos = NULL; // Inicializa la lista de procesos como vacía

    int op, op1, op2, op3, op4, NumEl, id, prioridad;
    string nombre;

    do {

        Menu(); // Muestra el menú de opciones al usuario

        do {

            cout << "Ingrese una opcion: "; cin >> op; // Solicita al usuario que ingrese una opción
            if (op < 1 || op > 4) { // Verifica si la opción es válida
                cout << "Opcion invalida. Por favor, intente de nuevo." << endl; // Mensaje de error si
la opción no es válida
            }

        } while (op < 1 || op > 4); // Bucle para asegurar que la opción ingresada sea válida

        switch (op) { // Estructura de control que maneja las opciones del menú
            case 1:
                do {
                    Menu2(); // Muestra el menú de gestión de procesos
```

```
        do {

            cout << "Ingrese una opcion: "; cin >> op2; // Solicita al usuario que ingrese una
opción

            if (op2 < 1 || op2 > 6) { // Verifica si la opción es válida
                cout << "Opcion invalida. Por favor, intente de nuevo." << endl; // Mensaje de
error si la opción no es válida
                }

        } while (op2 < 1 || op2 > 6); // Bucle para asegurar que la opción ingresada sea
válida

        switch (op2) {
            case 1:
                // Pedimos al usuario cuantos procesos desea registrar
                cout << "Cuantos procesos desea registrar? "; cin >> NumEl;

                // Ingresamos los datos de cada proceso
                for (int i = 0; i < NumEl; i++) {
                cout << "\nRegistro del proceso " << (i + 1) << endl;
                // Pedimos al usuario que ingrese los datos del proceso
                cout << "Ingrese el ID del proceso: "; cin >> id;
                cout << "Ingrese el nombre del proceso: "; cin >> nombre;
                cout << "Ingrese la prioridad del proceso: "; cin >> prioridad;

                // Llamamos a la función para agregar el proceso a la lista
                agregarAlFinal(listaProcesos, id, nombre, prioridad);
                }
                break;
            case 2:
                cout << "Imprimiendo procesos registrados..." << endl;
                imprimirProcesos(listaProcesos);
                break;
            case 3:
                cout << "Ingrese el ID del proceso a buscar: "; cin >> id;
                buscarProceso(listaProcesos, id);
                break;
            case 4:
                cout << "Ingrese el ID del proceso a eliminar: "; cin >> id;
                eliminarProceso(listaProcesos, id);
                break;
            case 5:
                cout << "Ingrese el ID del proceso a modificar: "; cin >> id;
```

```
                modificarProceso(listaProcesos, id);
                break;
            case 6:
                cout << "Saliendo del menu de gestion de procesos." << endl;
                break;
        }

    } while (op2 != 6); // Bucle para continuar mostrando el menú de gestión de procesos hasta que se elija salir

        break;
    case 2:
        do {
            Menu3();
            do {
                cout << "Ingrese una opcion: "; cin >> op3;
                if (op3 < 1 || op3 > 5) {
                    cout << "Opcion invalida. Por favor, intente de nuevo." << endl;
                }
            } while (op3 < 1 || op3 > 5);
            switch (op3) {
                case 1:
                    cout << "\n" << endl;
                    imprimirProcesos(listaProcesos);
                    insertar(listaProcesos);
                    break;
                case 2:
                    eliminarDePila();
                    break;
                case 3:
                    liberarMemoria();
                    cout << "Memoria liberada." << endl;
                    break;
                case 4:
                    visualizarPila();
                    break;
                case 5:
                    cout << "Saliendo del menu de gestion de memoria." << endl;
            }
        } while (op3 != 5);
        break;
    case 3:
        do {
            Menu4();
```

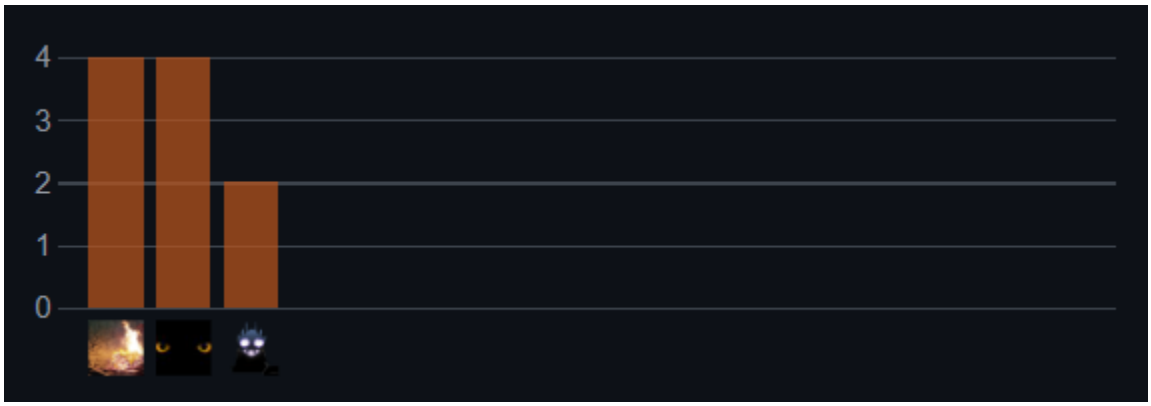
```
                do {
                    cout << "Ingrese una opcion: "; cin >> op4;
                    if (op4 < 1 || op4 > 3) {
                        cout << "Opcion invalida. Por favor, intente de nuevo." << endl;
                    }
                } while (op4 < 1 || op4 > 3);
                switch (op4) {
                    case 1:
                        visualizarCola();
                        break;
                    case 2:
                        ejecutarProcesosEnCola();
                        break;
                    case 3:
                        cout << "Saliendo del menu de gestion de memoria." << endl;
                        break;
                }
            } while (op4 != 3);
            break;
        case 4:
            cout << "Saliendo del sistema de gestion de procesos." << endl;
            break;
    }

} while (op != 4);

}
```

```
***Menu de Gestion***
[1]. Gestion de Procesos
[2]. Gestion de Memoria
[3]. Gestion de CPU
[4]. Salir
Ingrese una opcion: 1

***Gestion de Procesos***
[1]. Registrar proceso
[2]. Imprimir procesos
[3]. Buscar procesos
[4]. Eliminar proceso
[5]. Modificar proceso
[6]. Salir
**************************
Ingrese una opcion: 7
Opcion invalida. Por favor, intente de nuevo.
Ingrese una opcion:
```

```
***Menu de Gestion***
[1]. Gestion de Procesos
[2]. Gestion de Memoria
[3]. Gestion de CPU
[4]. Salir
Ingrese una opcion: 6
Opcion invalida. Por favor, intente de nuevo.
Ingrese una opcion:
```

```
***Menu de Gestion***
[1]. Gestion de Procesos
[2]. Gestion de Memoria
[3]. Gestion de CPU
[4]. Salir
Ingrese una opcion: 1

***Gestion de Procesos***
[1]. Registrar proceso
[2]. Imprimir procesos
[3]. Buscar procesos
[4]. Eliminar proceso
[5]. Modificar proceso
[6]. Salir
**************************
Ingrese una opcion: 1
Cuantos procesos desea registrar? 2

Registro del proceso 1
Ingrese el ID del proceso: 00
El ID debe ser un número positivo. Intente de nuevo.
Ingrese el ID del proceso: 12
Ingrese el nombre del proceso: Proceso_1
Ingrese la prioridad del proceso (1-10): -4
La prioridad debe estar entre 1 y 10. Intente de nuevo.
Ingrese la prioridad del proceso (1-10): 80
La prioridad debe estar entre 1 y 10. Intente de nuevo.
Ingrese la prioridad del proceso (1-10): 9
```

9. Aporte de cada integrante al código implementado



Excluding merges, **3 authors** have pushed **9 commits** to main and **10 commits** to all branches. On main, **1 file** has changed and there have been 360 **additions** and 42 **deletions**.

- **Limas Chagua Jair**

```
250  void eliminarProceso(Proceso*& listaProcesos, int id) {
251      Proceso* temp = listaProcesos;
252      // Creamos un puntero que buscara el proceso que indiquemos
253      Proceso* anterior = NULL;
254      // Creamos otro puntero que guardará el nodo anterior que deseqamos eliminar
255      while (temp && temp->id != id) {
256          anterior = temp;
257          temp = temp->siguiente;
258      }
259      // Con esto podemos eliminar un proceso de la lista cuando el ID coincide
260      if (!temp) {
261          cout << "Proceso no encontrado.\n";
262          return;
263      }
264      // Esto informa cuando no se encuentre ningun proceso con ese ID y termina la función
265      if (!anterior) listaProcesos = temp->siguiente;
266      else anterior->siguiente = temp->siguiente;
267      delete temp;
268      // Esto hace que libere la memoria del proceso eliminado
269      cout << "Proceso eliminado.\n";
270  }
271  void buscarProceso(Proceso*& listaProcesos, int id) {
272      Proceso* temp = listaProcesos;
273      // Indicamos que comienza desde el inicio de la lista
274      while (temp) {
275          if (temp->id == id) {
276              cout << "ID: " << temp->id << ", Nombre: " << temp->nombre << ", Prioridad: " << temp->prioridad << endl;
277              return;
278          }
279          temp = temp->siguiente;
280      }
281      // Cuando encuentre el ID que busquemos lo imprimirá
282      cout << "Proceso no encontrado.\n";
283      // Nos informará si no lo encuentra
284  }
```

```
134  void insertar(Proceso*& listaProcesos){
135      if (tope == MAX - 1) { // Verifica si la pila está llena
136          cout << "Memoria llena. No se puede insertar mas elementos." << endl;
137      } else {
138          int id2;
139          cout << "Ingrese el ID del proceso a asignar a la memoria: "; cin >> id2;
140          Proceso* temp = listaProcesos; // Crea un puntero temporal para recorrer la lista de procesos
141          while (temp != NULL) { // Recorre la lista de procesos
142              if (temp->id == id2) { // Verifica si el ID del proceso coincide con el ID ingresado
143                  pila[++tope] = temp; // Agrega el proceso a la pila
144                  cout << "Proceso con ID " << id2 << " agregado a la memoria." << endl;
145                  encolar(); // Llama a la función para encolar el proceso en la cola de memoria
146                  return; // Sale de la función una vez que se ha agregado el proceso
147              }
148              temp = temp->siguiente; // Avanza al siguiente proceso en la lista
149          }
150          cout << "No se encontró un proceso con ese ID." << endl;
151      }
152  }
```

```cpp
void eliminarDePila() {
    if (tope == -1) { // Verifica si la pila está vacía
        cout << "La pila de memoria está vacía. No se puede eliminar ningún proceso." << endl;
        return;
    }
    cout << "Procesos en la pila de memoria:" << endl;
    for (int i = 0; i <= tope; i++) { // Recorre la pila e imprime los IDs de los procesos
        cout << "ID: " << pila[i]->id << ", Nombre: " << pila[i]->nombre << ", Prioridad: " << pila[i]->prioridad << endl;
    }
    int idEliminar;
    cout << "Ingrese el ID del proceso que desea quitar de la pila: ";
    cin >> idEliminar;

    int pos = -1; // Variable para almacenar la posición del proceso a eliminar
    for (int i = 0; i <= tope; i++) { // Recorre la pila desde el inicio hasta el tope
        if (pila[i]->id == idEliminar) { // Busca el proceso con el ID especificado
            pos = i; // Guarda la posición del proceso a eliminar
            break;
        }
    }
    if (pos == -1) {
        cout << "No se encontró un proceso con ese ID en la pila." << endl; // Mensaje si no se encuentra el proceso
        return;
    }
    Proceso* procesoEliminado = pila[pos]; // Guarda el proceso que se va a eliminar
    // Desplazar los elementos para llenar el hueco
    for (int i = pos; i < tope; i++) { // Desplaza los elementos hacia la izquierda
        pila[i] = pila[i + 1]; // Mueve los elementos hacia la izquierda
    }
    tope--; // Decrementa el índice del tope de la pila
    cout << "Proceso con ID " << procesoEliminado->id << " eliminado de la memoria." << endl;
    eliminarDeColaPorID(idEliminar); // <-- sincroniza la cola
}
```

- **Solis Sedano Anderson**

```
Proceso* listaProcesos void registrarProceso() {
    // Aqui esta donde se registrara el proceso de una manera breve y un poco basica por ahora
   void registrarProceso() {
    Proceso* nuevo = new Proceso();

    cout<<"Ingrese ID del proceso: ";
    cin>>nuevo->id;
    cout<<"Ingrese nombre del proceso (una sola palabra): ";
    cin>>nuevo->nombre;
    cout<<"Ingrese prioridad del proceso: ";
    cin>>nuevo->prioridad;
    nuevo->siguiente = listaProcesos;
    listaProcesos = nuevo;
    cout<<"Proceso registrado correctamente."<<"\n";
}= NULL;
```

```
4     using namespace std;
5     //ESTRUCTURA PRINCIPAL
6
7
8     //Representa un proceso con ID, nombre, prioridad y puntero al siguiente
9     struct Proceso
10    {
11        int id;
12        string nombre;
13        int prioridad;
14        Proceso* siguiente;
15    };
16
17    Proceso* listaProcesos = NULL;
```

```cpp
//Creamos la funcion encolar con la prioridad de mayor
void encolarCPU(Proceso* proceso) {
    NodoCPU* nuevo = new NodoCPU(proceso);

    if (!frente || proceso->prioridad > frente->proceso->prioridad) {
        // Insertar al inicio si la cola está vacía o la prioridad es mayor que el primero
        nuevo->siguiente = frente;
        frente = nuevo;
        if (!fin) fin = nuevo;
    } else {
        // Insertar en la posición correcta según prioridad
        NodoCPU* actual = frente;
        while (actual->siguiente && actual->siguiente->proceso->prioridad >= proceso->prioridad) {
            actual = actual->siguiente;
        }
        nuevo->siguiente = actual->siguiente;
        actual->siguiente = nuevo;
        if (!nuevo->siguiente) fin = nuevo;  // Si se inserta al final, actualizar fin
    }

    cout << "Proceso encolado a la CPU segun su prioridad.\n";
}
```

```cpp
void mostrarOrdenEjecucion() {
    if (!frente) {
        cout << "No hay procesos en la cola de la CPU.\n";
        return;
    }
    NodoCPU* temp = frente;
    cout << "Orden de ejecucion de procesos:\n";
    while (temp) {
        cout << "ID: " << temp->proceso->id << ", Nombre: " << temp->proceso->nombre << ", Prioridad: " << temp->proceso->prioridad << endl;
        temp = temp->siguiente;
    }
}

void ejecutarProcesos() {
    if (!frente) {
        cout << "No hay procesos para ejecutar.\n";
        return;
    }
    cout << "Ejecutando procesos...\n";
    while (frente) {
        cout << "Ejecutando proceso ID: " << frente->proceso->id << " (" << frente->proceso->nombre << ")\n";
        NodoCPU* temp = frente;
        frente = frente->siguiente;
        delete temp;
    }
    fin = NULL;
    cout << "Todos los procesos han sido ejecutados.\n";
}
```

- **Quispe Linares Antoni**

```
306     void Menu() {
307         cout << "\n***Menu de Gestion***" << endl;
308         cout << "[1]. Gestion de Procesos" << endl;
309         cout << "[2]. Gestion de Memoria" << endl;
310         cout << "[3]. Gestion de CPU" << endl;
311         cout << "[4]. Salir" << endl;
312     }
313
314     void Menu2() {
315         cout << "\n***Gestion de Procesos***" << endl;
316         cout << "[1]. Registrar proceso" << endl;
317         cout << "[2]. Imprimir procesos" << endl;
318         cout << "[3]. Buscar procesos" << endl;
319         cout << "[4]. Eliminar proceso" << endl;
320         cout << "[5]. Modificar proceso" << endl;
321         cout << "[6]. Salir" << endl;
322         cout << "************************" << endl;
323     }
324
325     void Menu3() {
326         cout << "\n***Gestion de Memoria***" << endl;
327         cout << "[1]. Asignar a memoria" << endl;
328         cout << "[2]. Eliminar de memoria" << endl;
329         cout << "[3]. Liberar memoria" << endl;
330         cout << "[4]. Visualizar memoria" << endl;
331         cout << "[5]. Salir" << endl;
332         cout << "************************" << endl;
333     }
334
335     void Menu4() {
336         cout << "\n***Gestion de CPU***" << endl;
337         cout << "[1]. Verificar orden de ejecucion" << endl;
338         cout << "[2]. Ejecutar procesos" << endl;
339         cout << "[3]. Salir" << endl;
340         cout << "************************" << endl;
341     }
```

```cpp
int main() {

    Proceso* listaProcesos = NULL; // Inicializa la lista de procesos como vacía

    int op, op1, op2, op3, op4, NumEl, id, prioridad;
    string nombre;

    do {

        Menu(); // Muestra el menú de opciones al usuario

        do {

            cout << "Ingrese una opcion: "; cin >> op; // Solicita al usuario que ingrese una opción
            if (op < 1 || op > 4) { // Verifica si la opción es válida
                cout << "Opcion invalida. Por favor, intente de nuevo." << endl; // Mensaje de error si la opción no es válida
            }

        } while (op < 1 || op > 4); // Bucle para asegurar que la opción ingresada sea válida

        switch (op) { // Estructura de control que maneja las opciones del menú
            case 1:
                do {
                    Menu2(); // Muestra el menú de gestión de procesos

                    do {

                        cout << "Ingrese una opcion: "; cin >> op2; // Solicita al usuario que ingrese una opción

                        if (op2 < 1 || op2 > 6) { // Verifica si la opción es válida
                            cout << "Opcion invalida. Por favor, intente de nuevo." << endl; // Mensaje de error si la opción no es válida
                        }

                    } while (op2 < 1 || op2 > 6); // Bucle para asegurar que la opción ingresada sea válida

                    switch (op2) {
                        case 1:
                            // Pedimos al usuario cuantos procesos desea registrar
                            cout << "Cuantos procesos desea registrar? "; cin >> NumEl;

                            // Ingresamos los datos de cada proceso
                            for (int i = 0; i < NumEl; i++) {
                            cout << "\nRegistro del proceso " << (i + 1) << endl;
                            // Pedimos al usuario que ingrese los datos del proceso
                            cout << "Ingrese el ID del proceso: "; cin >> id;
                            cout << "Ingrese el nombre del proceso: "; cin >> nombre;
                            cout << "Ingrese la prioridad del proceso: "; cin >> prioridad;

                            // Llamamos a la función para agregar el proceso a la lista
                            agregarAlFinal(listaProcesos, id, nombre, prioridad);
                            }
                            break;
                        case 2:
                            cout << "Imprimiendo procesos registrados..." << endl;
                            imprimirProcesos(listaProcesos);
                            break;
                        case 3:
                            cout << "Ingrese el ID del proceso a buscar: "; cin >> id;
                            buscarProceso(listaProcesos, id);
                            break;
                        case 4:
                            cout << "Ingrese el ID del proceso a eliminar: "; cin >> id;
                            eliminarProceso(listaProcesos, id);
                            break;
                        case 5:
                            cout << "Ingrese el ID del proceso a modificar: "; cin >> id;
                            modificarProceso(listaProcesos, id);
                            break;
                        case 6:
```

```
412                             cout << "Saliendo del menu de gestion de procesos." << endl;
413                             break;
414                     }
415
416                 } while (op2 != 6); // Bucle para continuar mostrando el menú de gestión de procesos hasta que se elija salir
417
418                 break;
419             case 2:
420                 do {
421                     Menu3();
422                     do {
423                         cout << "Ingrese una opcion: "; cin >> op3;
424                         if (op3 < 1 || op3 > 5) {
425                             cout << "Opcion invalida. Por favor, intente de nuevo." << endl;
426                         }
427                     } while (op3 < 1 || op3 > 5);
428                     switch (op3) {
429                         case 1:
430                             cout << "\n" << endl;
431                             imprimirProcesos(listaProcesos);
432                             insertar(listaProcesos);
433                             break;
434                         case 2:
435                             eliminarDePila();
436                             break;
437                         case 3:
438                             liberarMemoria();
439                             cout << "Memoria liberada." << endl;
440                             break;
441                         case 4:
442                             visualizarPila();
443                             break;
444                         case 5:
445                             cout << "Saliendo del menu de gestion de memoria." << endl;
446                     }
```

```
447                 } while (op3 != 5);
448                 break;
449             case 3:
450                 do {
451                     Menu4();
452                     do {
453                         cout << "Ingrese una opcion: "; cin >> op4;
454                         if (op4 < 1 || op4 > 3) {
455                             cout << "Opcion invalida. Por favor, intente de nuevo." << endl;
456                         }
457                     } while (op4 < 1 || op4 > 3);
458                     switch (op4) {
459                         case 1:
460                             visualizarCola();
461                             break;
462                         case 2:
463                             ejecutarProcesosEnCola();
464                             break;
465                         case 3:
466                             cout << "Saliendo del menu de gestion de memoria." << endl;
467                             break;
468                     }
469                 } while (op4 != 3);
470                 break;
471             case 4:
472                 cout << "Saliendo del sistema de gestion de procesos." << endl;
473                 break;
474         }
475
476     } while (op != 4);
477
478 }
```

```
192  void liberarMemoria() {
193      if (tope == -1) { // Verifica si la pila está vacía
194          cout << "La pila de memoria está vacía. No hay procesos para liberar." << endl;
195          return;
196      }
197      // Solo vacía la pila, no elimina los procesos de la lista enlazada
198      tope = -1; // Reinicia el tope de la pila a -1 para indicar que está vacía
199      cout << "Todos los procesos han sido quitados de la memoria (pila) exitosamente." << endl;
200      vaciarCola(); // <-- sincroniza la cola
201  }
202
203  // Funcion para visualizar los procesos en la pila
204
205  void visualizarPila() {
206      if (tope == -1) { // Verifica si la pila está vacía
207          cout << "La pila de memoria está vacía." << endl; // Mensaje si la pila está vacía
208      } else {
209          cout << "Procesos en la pila de memoria:" << endl;
210          for (int i = 0; i <= tope; i++) { // Recorre la pila e imprime los IDs de los procesos
211              cout << "ID: " << pila[i]->id << ", Nombre: " << pila[i]->nombre << ", Prioridad: " << pila[i]->prioridad << endl;
212          }
213      }
214  }
```

- ANTHONY EMANUEL CASTRO SOLANO

```
void modificarProceso(Proceso*& listaProcesos, int id) {
    Proceso* temp = listaProcesos;
    // Creamos un puntero que buscara el proceso que indiquemos
    while (temp) { // Recorremos la lista de procesos
        if (temp->id == id) { // Verificamos si el ID del proceso coincide con el ID buscado
            // Si encontramos el proceso, solicitamos los nuevos datos
            cout << "Ingrese el nuevo nombre del proceso: ";
            cin >> temp->nombre;
            cout << "Ingrese la nueva prioridad del proceso: ";
            cin >> temp->prioridad;
            cout << "Proceso modificado exitosamente.\n";
            return;
        }
        temp = temp->siguiente; // Avanzamos al siguiente proceso en la lista
    }
    // Cuando encuentre el ID que busquemos lo modificará
    cout << "Proceso no encontrado.\n";
    // Nos informará si no lo encuentra
}
```

```
168              case 5:
169                  cout << "Ingrese el ID del proceso a modificar: "; cin >> id;
170                  modificarProceso(listaProcesos, id);
171                  break;
```


```
168              case 5:
169                  cout << "Ingrese el ID del proceso a modificar: "; cin >> id;
170                  modificarProceso(listaProcesos, id);
171                  break;
```


Link Del Github

https://github.com/AntoniRaul/Sistema_de_Gestion_de_Proceso

Link Del La Presentación

Presentacion Publica –> https://bit.ly/4kOdKdT

Presentacion Editable –> https://bit.ly/3FLEVHv

Anexo:

https://drive.google.com/drive/folders/1j8_K-XcuYOyb6u5L6rFK3j_ujDV5OtlY?usp=sharing