

FACULTADES DE INGENIERÍA EN SISTEMAS E INFORMATICA

TEMA:

Desarrollo de un "Sistema de Gestión de Procesos"

ASIGNATURA:

ESTRUCTURA DE DATOS

DOCENTE:

OSORIO CONTRERAS, ROSARIO DELIA

ESTUDIANTES:

Huaman Brañez, Jose Antonio
García Quijada Álvaro

Huancayo - Perú, 2025

CAPITULO 1: Análisis del Problema

1. Descripción del problema:

En entornos donde no existe un sistema operativo o un gestor de procesos centralizado (como en sistemas embebidos, microcontroladores o aplicaciones especializadas), la administración manual de procesos y recursos computacionales se vuelve ineficiente y propensa a errores. Esto genera los siguientes problemas:

Falta de Gestión Óptima de Procesos:

- No hay un mecanismo estructurado para crear, pausar, reanudar o terminar procesos.
- Los procesos compiten por recursos sin un planificador que priorice tareas críticas, lo que puede causar inanición (algunos procesos nunca se ejecutan) o bloqueos (procesos que esperan indefinidamente).

Asignación Ineficiente de Memoria:

- Sin un gestor de memoria, los programas asignan y liberan bloques de forma desorganizada, llevando a fragmentación (espacio desperdiciado) o corrupción de datos (sobreescritura accidental).

Ausencia de Persistencia y Recuperación:

- Los datos de procesos no se guardan automáticamente, por lo que fallos del sistema implican pérdida de información.
- No hay registros claros de estados de procesos (ej., "en ejecución", "bloqueado"), lo que complica la detección de errores.

2. Requerimientos del sistema

Requerimientos funcionales:

1. Gestión de Procesos

- o Crear, eliminar y listar procesos con atributos: **ID, nombre, prioridad y estado**.
- o Cambiar el estado de un proceso (ej., *listo, en ejecución, bloqueado*).
- o Búsqueda de procesos por **ID** o nombre.

2. Planificación de CPU

- o Ejecutar procesos en orden de **prioridad** (mayor a menor).
- o Implementar un algoritmo **SJF (Shortest Job First)** para evitar inanición.
- o Soporte para **interrupción manual** de procesos en ejecución.

3. Gestión de Memoria

- o Asignar y liberar bloques de memoria mediante una **pila (LIFO)**.
- o Mostrar el **estado actual de la memoria** (bloques ocupados/libres).
- o Manejar **fragmentación** mediante reubicación de bloques (opcional).

4. Persistencia de Datos

- o Guardar y cargar la lista de procesos en un **archivo de texto** o binario.
- o Registrar un **historial de operaciones** (log) para auditoría.

5. Interfaz de Usuario

- o Menú interactivo en **consola** con opciones claras.
- o Validación de entrada para evitar errores (ej., IDs duplicados).

Requerimientos No Funcionales

Estos requisitos garantizan la calidad y eficiencia del sistema:

1. **Rendimiento**

- Operaciones de inserción/eliminación en lista enlazada: **$O(1)$** .
- Búsqueda de procesos: **$O(n)$** (aceptable para el alcance del proyecto).

2. **Confiabilidad**

- Manejo de errores: Procesos inexistentes, memoria llena, archivos corruptos.
- Recuperación ante fallos: Carga automática del último estado guardado.

3. **Usabilidad**

- Interfaz intuitiva con mensajes de ayuda.
- Compatibilidad con sistemas **Windows/Linux** (consola estándar).

4. **Mantenibilidad**

- Código modular (separación en **gestor de procesos, planificador, memoria**).
- Comentarios y documentación clara en el repositorio.

5. **Seguridad Básica**

- Validación de datos para evitar inyección de comandos.
- Archivos de persistencia con permisos restringidos (solo lectura/escritura para el sistema).

3. **Estructuras de datos propuestas:**

- **Lista Enlazada (Gestor de Procesos)**

Propósito:

Almacenar de manera dinámica todos los procesos activos en el sistema, permitiendo su creación, eliminación y búsqueda eficiente.

Atributos de cada Proceso:

- ID: Identificador único (entero).
- Nombre: Descripción del proceso (cadena de texto).
- Prioridad: Número que determina su orden de ejecución (entero, mayor valor = mayor prioridad).
- Estado: Puede ser Listo, En Ejecución, Bloqueado o Terminado.

Operaciones Clave:

1. insertarProceso(id, nombre, prioridad)
 - Añade un nuevo nodo al final de la lista.
 - Complejidad: $O(1)$ si se mantiene un puntero al último nodo.
2. eliminarProceso(id)
 - Busca el proceso por ID, lo elimina y reajusta los punteros del nodo anterior y siguiente.
 - Complejidad: $O(n)$ (búsqueda secuencial).
3. buscarPorId(id)
 - Recorre la lista hasta encontrar el ID solicitado.
 - Complejidad: $O(n)$.
4. mostrarProcesos()
 - Imprime en consola todos los procesos con sus atributos.

-**Cola de Prioridad (Planificador CPU)**

Propósito:

Decidir el orden de ejecución de los procesos basado en su prioridad (el de mayor prioridad se ejecuta primero).

Implementación:

- Utiliza una lista enlazada ordenada de mayor a menor prioridad.
- Si dos procesos tienen la misma prioridad, se usa FIFO (First-In-First-Out).

Operaciones Clave:

1. encolarPrioridad(proceso)
 - Inserta el proceso en la posición correcta según su prioridad.
 - Complejidad: $O(n)$ (peor caso: recorrer toda la lista).
2. desencolar()
 - Extrae y retorna el primer proceso de la cola (el de mayor prioridad).
 - Complejidad: $O(1)$.
3. mostrarCola()
 - Muestra los procesos en orden de ejecución.

-Pila (Gestor de Memoria)**Propósito:**

Simular la asignación y liberación de bloques de memoria usando el principio LIFO (Last-In-First-Out).

Estructura de un Bloque de Memoria:

- Dirección: Número que simula una dirección de memoria (ej., 0x1000).
- Tamaño: Tamaño del bloque asignado (en bytes).

Operaciones Clave:

1. push(dirección, tamaño)
 - Reserva un nuevo bloque y lo apila.
 - Complejidad: $O(1)$.
2. pop()
 - Libera el último bloque asignado (cima de la pila).
 - Complejidad: $O(1)$.
3. estadoMemoria()
 - Muestra los bloques asignados y sus direcciones.

4. Justificación de la elección:**- Lista Enlazada para Gestión de Procesos:**

- **Ventajas:**
 - Inserción rápida ($O(1)$) al final de la lista.
 - Flexibilidad: Permite eliminar procesos intermedios sin reorganizar toda la estructura (solo ajuste de punteros).
 - Uso óptimo de memoria: Asigna espacio solo cuando se crea un proceso.

- Cola de Prioridad para Planificación:

- **Ventajas:**

- Priorización eficiente: Garantiza que el proceso más crítico (prioridad alta) se ejecute primero.
- Prevención de inanición: Al combinar prioridad y SJF, procesos largos no monopolizan la CPU.

- **Pila para Gestión de Memoria:**

- **Ventajas:**

- Simplicidad: LIFO es ideal para sistemas sin MMU (Memory Management Unit), ya que libera memoria en orden inverso a su asignación.
- Bajo overhead: Operaciones push/pop son $O(1)$.

Capítulo 2: Diseño de la Solución

1. Descripción de estructuras de datos y operaciones:

a. Lista enlazada (Gestor de Procesos):

Propósito: Almacenar y gestionar todos los procesos del sistema con sus metadatos.

Estructura:

```
class NodoProceso {
public:
    int id; // Identificador único
    string nombre; // Nombre del proceso
    int prioridad; // Nivel de prioridad (0-100)
    NodoProceso* siguiente; // Puntero al siguiente nodo
};
```

Operación	Descripción
insertarProceso ()	Añade procesos al final de la lista
eliminarProceso ()	Elimina por ID con reajuste de punteros
buscarPorId ()	Búsqueda secuencial por ID
mostrar ()	Recorrido completo para visualización

b. Cola de Prioridad (Planificador CPU):

Propósito: Gestionar el orden de ejecución de procesos basado en prioridades.

Estructura:

```
class ColaPrioridad {
private:
    struct NodoCola {
        NodoProceso* proceso; // Referencia al proceso
        NodoCola* siguiente; // Puntero al siguiente nodo
    };
    NodoCola* frente; // Puntero al proceso de mayor prioridad
};
```

Operación	Descripción
encolarPrioridad()	Inserta ordenado por prioridad (mayor a menor)
desencolar()	Extrae el proceso de mayor prioridad
mostrar()	Muestra ID y prioridad de procesos en cola

c. Pila (Gestor de Memoria):

Propósito: Simular la asignación/liberación de bloques de memoria.

Estructura:

```
class PilaMemoria {
private:
    struct NodoMemoria {
        int direccion;           // Dirección de memoria
        NodoMemoria* abajo;     // Puntero al nodo inferior
    };
    NodoMemoria* tope;          // Último bloque asignado
    int capacidad;              // Límite máximo de bloques
    int contador;               // Bloques actualmente usados
};
```

Operación	Descripción
push()	Asigna memoria (LIFO)
pop()	Libera último bloque asignado
estadoMemoria()	Muestra uso y direcciones



2. Algoritmos principales (pseudocódigo):

2.1 Lista Enlazada (Gestor de Procesos):

- Inicialización:

```
1  Algoritmo inicializarLista():
2      cabeza ? NULO
3      tamaño ? 0
4  Fin Algoritmo
```

- Insertar Proceso:

```
1  Algoritmo insertarProceso(id, nombre, prioridad):
2      nuevo ? CrearNodo(id, nombre, prioridad)
3
4      Si cabeza = NULO entonces
5          cabeza ? nuevo
6      Sino
7          actual ? cabeza
8          Mientras actual.siguiente ? NULO hacer
9              actual ? actual.siguiente
10         Fin Mientras
11         actual.siguiente ? nuevo
12     Fin Si
13     tamaño ? tamaño + 1
14     Mostrar "Proceso " + id + " creado"
15 Fin Algoritmo
```

- Buscar y Eliminar Proceso por ID:

```
1  Algoritmo buscarPorId(id):
2      actual ? cabeza
3      Mientras actual ? NULO hacer
4          Si actual.id = id entonces
5              Retornar actual
6          Fin Si
7          actual ? actual.siguiente
8      Fin Mientras
9      Retornar NULO
10 Fin Algoritmo

1  Algoritmo eliminarProceso(id):
2      Si cabeza = NULO entonces
3          Mostrar "Error: Lista vacía"
4          Retornar FALSO
5      Fin Si
6
7      Si cabeza.id = id entonces
8          temp ? cabeza
9          cabeza ? cabeza.siguiente
10         Liberar(temp)
11         tamaño ? tamaño - 1
12         Mostrar "Proceso eliminado"
13         Retornar VERDADERO
14     Fin Si
15
16     anterior ? cabeza
17     actual ? cabeza.siguiente
18     Mientras actual ? NULO hacer
19         Si actual.id = id entonces
20             anterior.siguiente ? actual.siguiente
21             Liberar(actual)
22             tamaño ? tamaño - 1
23             Mostrar "Proceso eliminado"
24             Retornar VERDADERO
25         Fin Si
26         anterior ? actual
27         actual ? actual.siguiente
28     Fin Mientras
29
30     Mostrar "Error: Proceso no encontrado"
31     Retornar FALSO
32 Fin Algoritmo
```

- **Mostrar todos los procesos:**

```

1  Algoritmo mostrarProcesos():
2      Si cabeza = NULO entonces
3          Mostrar "No hay procesos activos"
4          Retornar
5      Fin Si
6
7      actual ? cabeza
8      Mostrar "--- LISTA DE PROCESOS ---"
9      Mientras actual ? NULO hacer
10         Mostrar "ID: " + actual.id + " | Nombre: " + actual.nombre + " | Prioridad: " + actual.prioridad
11         actual ? actual.siguiente
12     Fin Mientras
13 Fin Algoritmo

```

2.2 Cola de Prioridad (Planificador de CPU):

- **Inicialización:**

```

1  Algoritmo inicializarCola():
2      frente ? NULO
3      final ? NULO
4      tamaño ? 0
5  Fin Algoritmo

```

- **Encolar por Prioridad:**

```

1  Algoritmo encolarPrioridad(proceso):
2      nuevo ? CrearNodoCola(proceso)
3
4      Si frente = NULO entonces
5          frente ? nuevo
6          final ? nuevo
7      Sino Si proceso.prioridad > frente.proceso.prioridad entonces
8          nuevo.siguiente ? frente
9          frente ? nuevo
10     Sino
11         actual ? frente
12         Mientras actual.siguiente ? NULO Y
13             actual.siguiente.proceso.prioridad >= proceso.prioridad hacer
14             actual ? actual.siguiente
15         Fin Mientras
16         nuevo.siguiente ? actual.siguiente
17         actual.siguiente ? nuevo
18         Si nuevo.siguiente = NULO entonces
19             final ? nuevo
20         Fin Si
21     Fin Si
22     tamaño ? tamaño + 1
23     Mostrar "Proceso " + proceso.id + " encolado (Prioridad: " + proceso.prioridad + ")"
24 Fin Algoritmo

```

- **Desencolar Proceso:**

```

1  Algoritmo desencolar():
2      Si frente = NULO entonces
3          Mostrar "Error: Cola vacía"
4          Retornar NULO
5      Fin Si
6
7      temp ? frente
8      proceso ? frente.proceso
9      frente ? frente.siguiente
10     Si frente = NULO entonces
11         final ? NULO
12     Fin Si
13     Liberar(temp)
14     tamaño ? tamaño - 1
15     Mostrar "Ejecutando proceso " + proceso.id
16     Retornar proceso
17 Fin Algoritmo

```


- **Mostrar Cola:**

```
1  Algoritmo mostrarCola():
2      Si frente = NULO entonces
3          Mostrar "Cola vacía"
4          Retornar
5      Fin Si
6
7      actual ? frente
8      Mostrar "--- COLA DE PRIORIDAD ---"
9      Mientras actual ? NULO hacer
10         Mostrar "ID: " + actual.proceso.id + " | Prioridad: " + actual.proceso.prioridad
11         actual ? actual.siguiete
12     Fin Mientras
13 Fin Algoritmo
```

2.3 Pila (Gestor de Memoria):

- **Inicialización:**

```
1  Algoritmo inicializarPila(capacidadMax):
2      tope ? NULO
3      capacidad ? capacidadMax
4      contador ? 0
5  Fin Algoritmo
```

- **Push (Asignar Memoria):**

```
1  Algoritmo push(direccion):
2      Si contador >= capacidad entonces
3          Mostrar "Error: Memoria llena"
4          Retornar FALSO
5      Fin Si
6
7      nuevo ? CrearNodoMemoria(direccion)
8      nuevo.abajo ? tope
9      tope ? nuevo
10     contador ? contador + 1
11     Mostrar "Memoria asignada en " + direccion
12     Retornar VERDADERO
13 Fin Algoritmo
```

- **Pop (Liberar Memoria):**

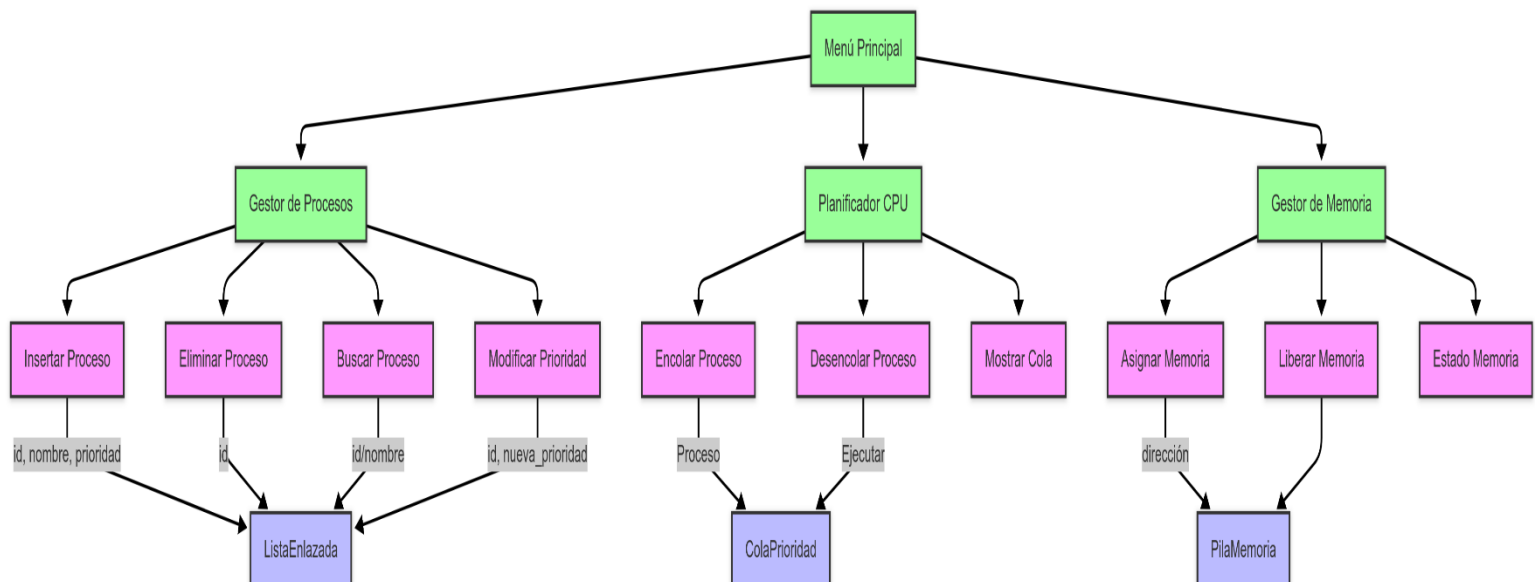
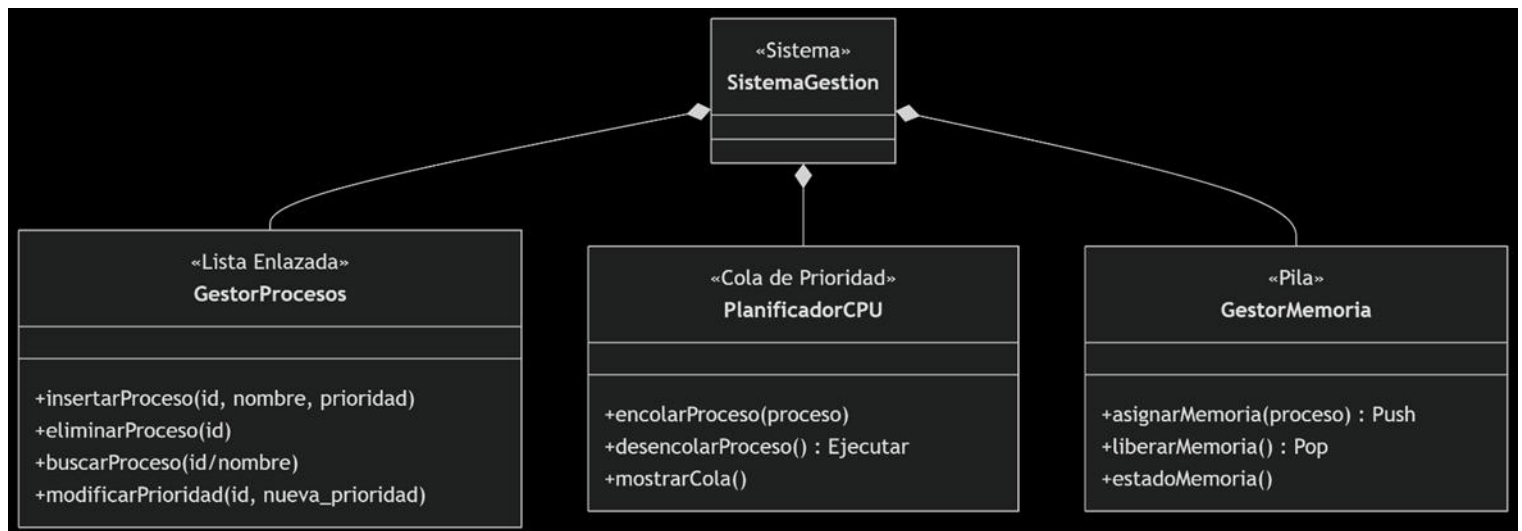
```
1  Algoritmo pop():
2      Si tope = NULO entonces
3          Mostrar "Error: Pila vacía"
4          Retornar FALSO
5      Fin Si
6
7      temp ? tope
8      direccion ? tope.direccion
9      tope ? tope.abajo
10     Liberar(temp)
11     contador ? contador - 1
12     Mostrar "Memoria liberada en " + direccion
13     Retornar VERDADERO
14 Fin Algoritmo
```

- **Mostrar estado de Memoria:**

```

1  Algoritmo mostrarMemoria():
2      Mostrar "--- ESTADO DE MEMORIA ---"
3      Mostrar "Bloques usados: " + contador + "/" + capacidad
4
5      Si tope = NULO entonces
6          Mostrar "No hay bloques asignados"
7          Retornar
8      Fin Si
9
10     actual ? tope
11     Mostrar "Bloques activos (Tope primero):"
12     Mientras actual ? NULO hacer
13         Mostrar "Dirección: " + actual.direccion
14         actual ? actual.abajo
15     Fin Mientras
16 Fin Algoritmo
    
```

3. Diagramas de Flujo



4. Justificación del diseño:

El diseño del sistema se fundamenta en el uso de **clases y estructuras de datos** para lograr un equilibrio entre eficiencia, modularidad y claridad del código. La implementación con clases permite encapsular la lógica de cada componente (gestión de procesos, planificación y memoria) en unidades independientes, facilitando el mantenimiento y la escalabilidad. Cada clase representa una entidad bien definida con responsabilidades específicas, lo que sigue el principio de separación de preocupaciones.

Elección de Estructuras de Datos

1. Lista Enlazada (Gestor de Procesos):

Se optó por una lista enlazada simple para gestionar los procesos debido a su flexibilidad en inserciones y eliminaciones dinámicas, operaciones comunes en un sistema de gestión de procesos. Aunque la búsqueda es lineal ($O(n)$), esto resulta aceptable dado que el sistema está diseñado para manejar un número moderado de procesos (decenas o cientos, no miles). La lista enlazada también evita el problema de redimensionamiento que enfrentarían los arrays, y su implementación con punteros garantiza un uso eficiente de la memoria.

2. Cola de Prioridad (Planificador CPU):

La cola de prioridad se implementó como una lista enlazada ordenada para garantizar que los procesos con mayor prioridad se ejecuten primero. Aunque un *heap* binario podría ofrecer mejor rendimiento teórico ($O(\log n)$ para encolar), la lista enlazada fue elegida por su simplicidad de implementación y porque, en la práctica, el número de procesos encolados no justifica la complejidad adicional. La política FIFO para procesos con igual prioridad asegura equidad.

3. Pila (Gestor de Memoria):

La pila LIFO (último en entrar, primero en salir) simula la asignación y liberación de memoria de manera intuitiva y eficiente ($O(1)$ para push y pop). Esta estructura es ideal para sistemas simples donde no se requiere gestión avanzada de fragmentación. Aunque una lista libre (*free list*) permitiría reutilizar bloques intermedios, la pila garantiza predictibilidad y bajo overhead, adecuado para una simulación didáctica.

Ventajas del Enfoque Orientado a Objetos

- **Encapsulación:**

Cada clase oculta sus detalles internos (como punteros o contadores), exponiendo solo métodos públicos (ej.: `insertarProceso()`, `desencolar()`). Esto protege la integridad de los datos y reduce errores.

- **Modularidad:**

El sistema está dividido en componentes intercambiables. Por ejemplo, el Planificador CPU no necesita conocer cómo se almacenan los procesos, solo interactúa con sus punteros. Esto facilita futuras mejoras (ej.: reemplazar la lista por un árbol).

- **Claridad:**

El código es autoexplicativo gracias a:

- Nombres descriptivos de clases y métodos (ColaPrioridad, estadoMemoria()).
- Comentarios que explican la lógica crítica (ej.: reorganización de punteros al eliminar nodos).

Eficiencia y Limitaciones:

- **Rendimiento:**

Las operaciones más frecuentes (encolar, desencolar, asignar memoria) son $O(1)$ o $O(n)$ en el peor caso, lo que es aceptable para el alcance académico del proyecto. En un entorno real con miles de procesos, podrían usarse estructuras más complejas (tablas hash para búsquedas o heaps para planificación).

- **Trade-offs:**

- Fragmentación de memoria: La pila LIFO no reutiliza bloques liberados en el medio, pero esto se asume como una limitación aceptable.
- Búsqueda lineal: La lista enlazada no es óptima para búsquedas frecuentes, pero se compensa con su simplicidad.

Capítulo 3: Solución Final

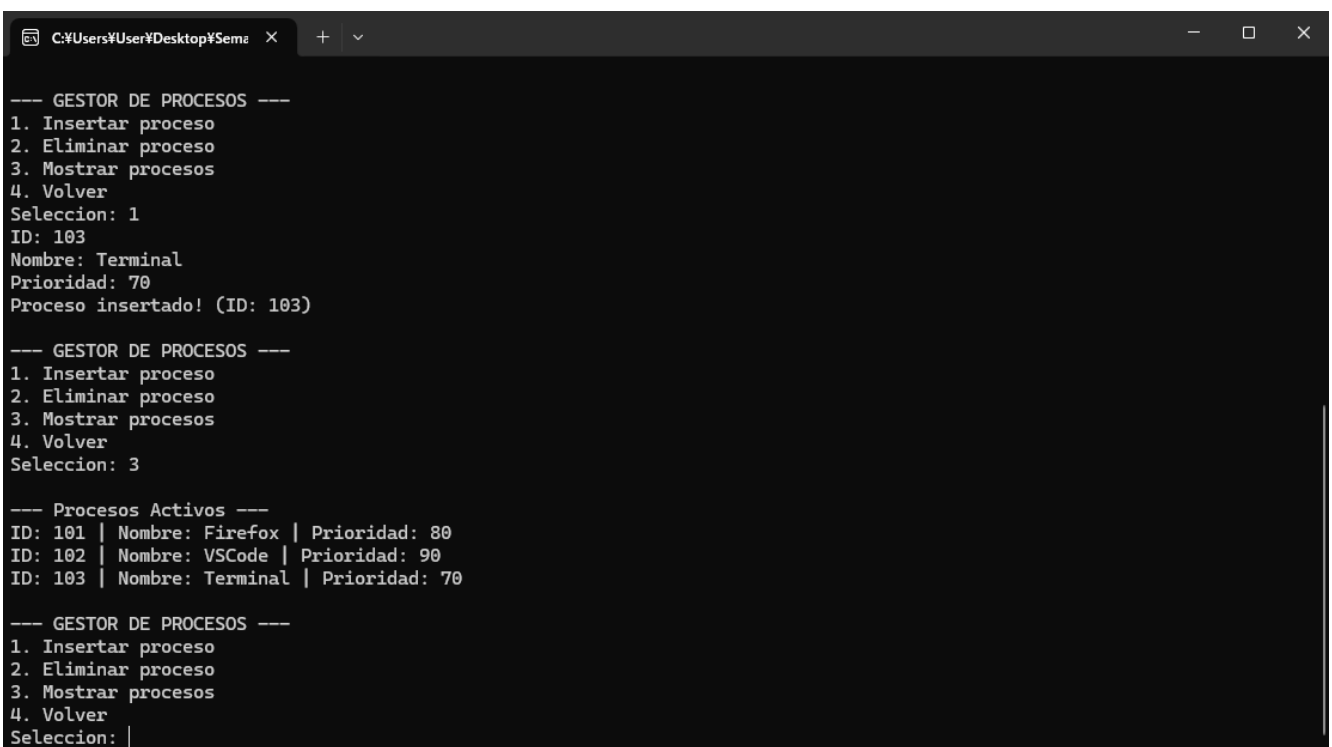
1. Código limpio, bien comentado y estructurado.

Enlace de Github con el archivo cpp:

- Link: <https://github.com/JBranetz/Simulacion-de-Sistema-Operativo-Estructura-de-Datos-.git>

2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos:

Inserción y mostrado de procesos:



```
--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Selección: 1
ID: 103
Nombre: Terminal
Prioridad: 70
Proceso insertado! (ID: 103)

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Selección: 3

--- Procesos Activos ---
ID: 101 | Nombre: Firefox | Prioridad: 80
ID: 102 | Nombre: VSCode | Prioridad: 90
ID: 103 | Nombre: Terminal | Prioridad: 70

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Selección: |
```

Eliminar proceso existente e inexistente:

```
C:\Users\User\Desktop\Semr x + v
Prioridad: 70
Proceso insertado! (ID: 103)

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 3

--- Procesos Activos ---
ID: 101 | Nombre: Firefox | Prioridad: 80
ID: 102 | Nombre: VSCode | Prioridad: 90
ID: 103 | Nombre: Terminal | Prioridad: 70

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 101
Proceso eliminado!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: |
```

```
C:\Users\User\Desktop\Semr x + v

--- Procesos Activos ---
ID: 101 | Nombre: Firefox | Prioridad: 80
ID: 102 | Nombre: VSCode | Prioridad: 90
ID: 103 | Nombre: Terminal | Prioridad: 70

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 101
Proceso eliminado!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 999
Proceso no encontrado!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: |
```

Eliminar lista vacía:

```
C:\Users\User\Desktop\Sema x + v
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 102
Proceso eliminado!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 103
Proceso eliminado!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: 2
ID a eliminar: 101
Lista vacia!

--- GESTOR DE PROCESOS ---
1. Insertar proceso
2. Eliminar proceso
3. Mostrar procesos
4. Volver
Seleccion: |
```

Encolar y mostrar procesos:

```
C:\Users\User\Desktop\Sema x + v
ID del proceso: 102
Proceso encolado! (ID: 102)

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: 1
ID del proceso: 103
Proceso encolado! (ID: 103)

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: 3

--- Cola de Prioridad ---
ID: 101 | Prioridad: 80
ID: 102 | Prioridad: 80
ID: 103 | Prioridad: 70

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: |
```

Ejecutar Procesos:

```
C:\Users\User\Desktop\Sema x + v
--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: 2
Ejecutando proceso ID: 101

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: 2
Ejecutando proceso ID: 102

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: 2
Ejecutando proceso ID: 103

--- PLANIFICADOR CPU ---
1. Encolar proceso
2. Ejecutar proceso
3. Mostrar cola
4. Volver
Seleccion: |
```

Asignación y mostrado de memoria:

```
C:\Users\User\Desktop\Sema x + v
Seleccion: 1
Direccion: 102
Memoria asignada! (Dir: 102)

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Seleccion: 1
Direccion: 103
Memoria asignada! (Dir: 103)

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Seleccion: 3

--- Estado Memoria ---
Espacio usado: 3/3
Direcciones (tope primero): 103 102 101

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Seleccion: |
```

Liberación de Memoria:

```
C:\Users\User\Desktop\Sema x + v
--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Selección: 2
Memoria liberada! (Dir: 103)

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Selección: 2
Memoria liberada! (Dir: 102)

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Selección: 2
Memoria liberada! (Dir: 101)

--- GESTOR DE MEMORIA ---
1. Asignar memoria
2. Liberar memoria
3. Estado memoria
4. Volver
Selección: |
```

3. Manual de usuario:

Introducción:

El Sistema Operativo Mini v2.0 es una aplicación de simulación que permite gestionar procesos, planificar su ejecución en una CPU y administrar memoria de forma básica. Está diseñado como una herramienta educativa para entender conceptos fundamentales de sistemas operativos. El programa ofrece una interfaz de línea de comandos organizada en menús interactivos que guían al usuario a través de sus funcionalidades.

Requisitos del Sistema

Para ejecutar el Sistema Operativo Mini v2.0 se necesita:

- Un ordenador con sistema operativo Windows, Linux o macOS
- Compilador C++ compatible con C++11 o superior
- Mínimo 100 MB de espacio en disco
- Terminal o consola que soporte caracteres UTF-8

Funcionalidades Principales:

1. Gestor de Procesos

El módulo de gestión de procesos permite crear, eliminar y visualizar procesos. Cada proceso tiene:

- Un ID numérico único (entero positivo)
- Un nombre descriptivo (cadena de texto)

- Un nivel de prioridad (0-100, donde 100 es máxima prioridad)

Los procesos se guardan automáticamente al salir del programa y se cargan al iniciarlo nuevamente.

2. Planificador de CPU

El planificador organiza los procesos en una cola de prioridad para determinar el orden de ejecución. Los procesos con mayor prioridad se ejecutan primero. Puede:

- Añadir procesos existentes a la cola de ejecución
- Ejecutar el próximo proceso en la cola
- Visualizar los procesos en espera de ejecución

3. Gestor de Memoria

Simula la asignación y liberación de bloques de memoria usando un enfoque de pila (LIFO). Permite:

- Asignar bloques de memoria especificando una dirección
- Liberar el último bloque asignado
- Visualizar el estado actual de la memoria

Uso del Sistema:

Al iniciar el programa, se presenta un menú principal con cuatro opciones. Use las teclas numéricas (1-4) para navegar:

1. **Gestor de Procesos:**
 - Insertar: Crea nuevos procesos ingresando ID, nombre y prioridad
 - Eliminar: Borra procesos existentes por su ID
 - Mostrar: Lista todos los procesos activos con sus detalles
2. **Planificador CPU:**
 - Encolar: Añade un proceso existente a la cola de ejecución
 - Ejecutar: Extrae y "ejecuta" el proceso con mayor prioridad
 - Mostrar: Visualiza los procesos en cola de ejecución
3. **Gestor de Memoria:**
 - Asignar: Reserva un nuevo bloque de memoria
 - Liberar: Libera el último bloque asignado
 - Estado: Muestra el uso actual de memoria
4. **Salir:** Finaliza el programa guardando automáticamente los procesos

Consideraciones Importantes:

- Los IDs de proceso deben ser únicos. El sistema rechazará IDs duplicados.
- La prioridad debe estar entre 0 y 100. Valores fuera de este rango generarán error.
- La memoria tiene capacidad limitada (3 bloques en esta versión).
- Al eliminar un proceso, asegúrese de que no esté en la cola de ejecución.
- Todos los errores se registran en el archivo errors.log para diagnóstico.

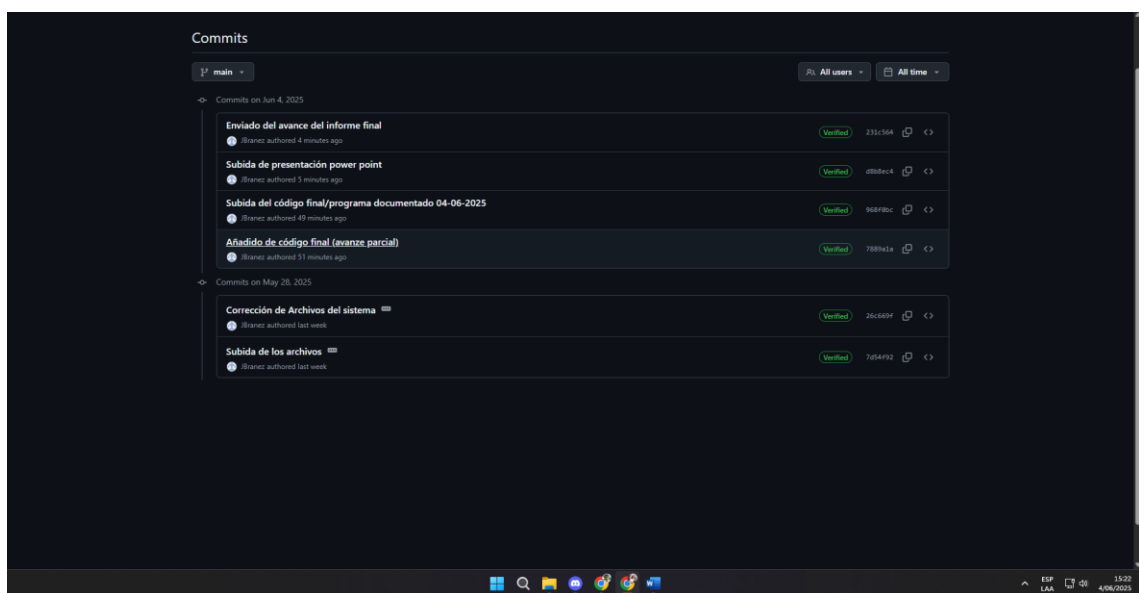
Ejemplo de Flujo de Trabajo:

1. Crear varios procesos en el Gestor de Procesos
2. Asignar algunos a la cola de ejecución en el Planificador CPU
3. Ejecutar procesos según su prioridad
4. Asignar y liberar memoria según sea necesario
5. Verificar el estado de los procesos y memoria periódicamente
6. Salir del programa cuando finalice la simulación

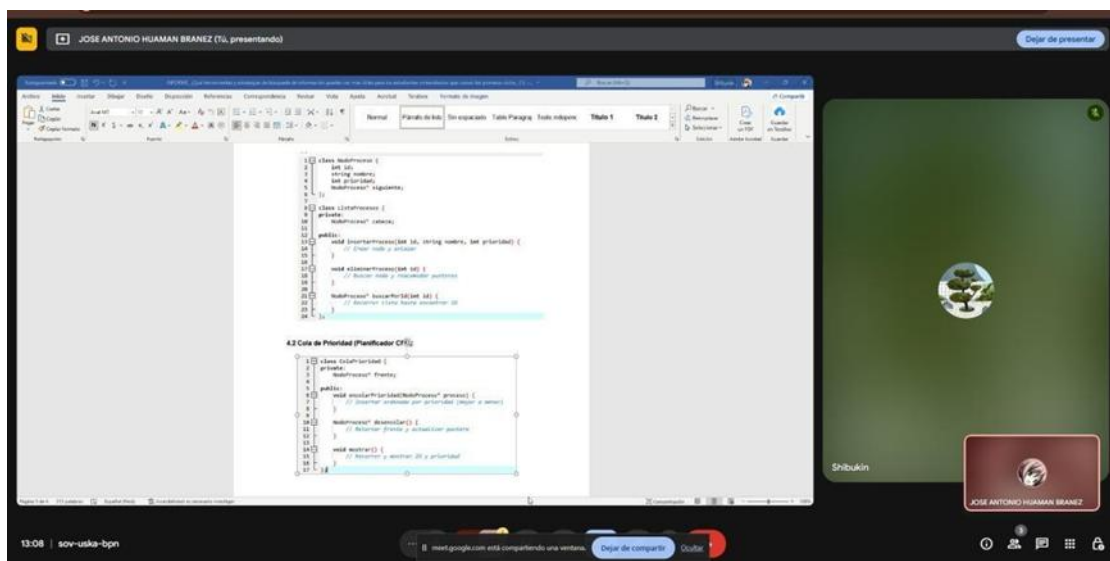
Capítulo 4: Evidencias de Trabajo en Equipo

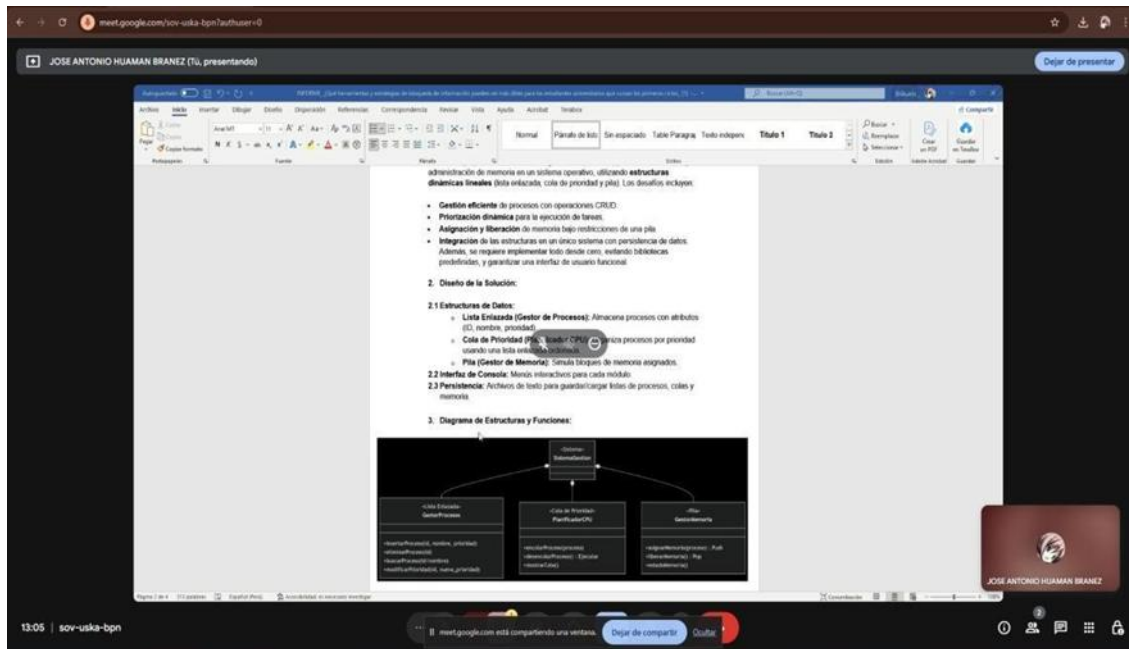
1. Repositorio con Control de Versiones (Capturas de Pantalla)

- Registro de commits claros y significativos que evidencien aportes individuales (proactividad).



- Evidencia por cada integrante del equipo.





Link: <https://github.com/JBranetz/Simulacion-de-Sistema-Operativo-Estructura-de-Datos-git>

2. Plan de Trabajo y Roles Asignados

- Documento inicial donde se asignan tareas y responsabilidades:

5. Planificación de tareas y responsabilidades por integrante:

5.1 García Quijada Álvaro:

- Módulo 1: Gestor de Procesos
 - Implementar toda la lógica relacionada con la lista enlazada de procesos (creación, eliminación, búsqueda).



5.2 Huamán Brañez, José Antonio:

- Módulo 2: Planificador de CPU
 - Implementar la cola de prioridad para ejecutar procesos.
- Módulo 3: Gestor de Memoria
 - Implementar la pila para asignar y liberar memoria.

5.3 Tareas Compartidas

- Diseñar la interfaz de usuario (menús de consola).
- Integrar los módulos para que funcionen juntos.
- Pruebas generales y documentación final.

- Registro de reuniones o comunicación del equipo (Actas de reuniones.).

ACTA DE REUNIÓN N° 1

Asignatura	ESTRUCTURA DE DATOS	Fecha:	20/05/25
Responsable de grupo	Huamán Brañez José Antonio	Hora de inicio:	10:00 am
Modalidad de Reunión	Presencial	Hora de fin	11:00 am

Integrantes:

Apellidos y nombres	Asistió (Si/No)	% Participación	Firma
1. Huamán Brañez José Antonio	SI	100%	
2. García Quijada Álvaro	SI	100%	
3.			
4.			

Temas tratados	Acuerdos	Responsables	Fecha de entrega
Planificación y diseño	Creación de un sistema de gestión de procesos consistente en 3 módulos	Todos los integrantes	22/05/25
Repartición de responsabilidades	Modulo 1 perteneciente a Álvaro, modulo 2 y 3 a José y tareas compartidas	Todos los integrantes	22/05/25
Creación de Diagramas y Pseudocódigo		Todos los integrantes	22/05/25

ACTA DE REUNIÓN N ' 2

Asignatura	ESTRUCTURA DE DATOS	Fecha:	27/05/25
Responsable de grupo	Huamán Brañez José Antonio	Hora de inicio:	10:00 am
Modalidad de Reunión	Presencial	Hora de fin	11:00 am

Integrantes:

Apellidos y nombres	Asistió (Si/No)	% Participación	Firma
1. Huamán Brañez José Antonio	SI	100%	
2. García Quijada Álvaro	No	50%	
3.			
4.			

Temas tratados	Acuerdos	Responsables	Fecha de entrega
Separación de código a escribir	Se escribió el código fuente	Todos los integrantes	28/05/25
Corrección de puntos del anterior informe	Se hizo un mejor análisis de problemas y el diagrama de flujo	Todos los integrantes	28/05/25
Descripción del código fuente	Se escribió el código al mismo tiempo que se hacía el informe	Todos los integrantes	28/05/25

