

# Sistema de Gestión de Procesos

Docente:

Medina Flores, Jimmy Roberto

Integrantes:

- Alarcon Chiquillan Mileydi Diana
- Ruiz Cardenas Angeli Mayli
- Chahuayla Castro Kenny Smith



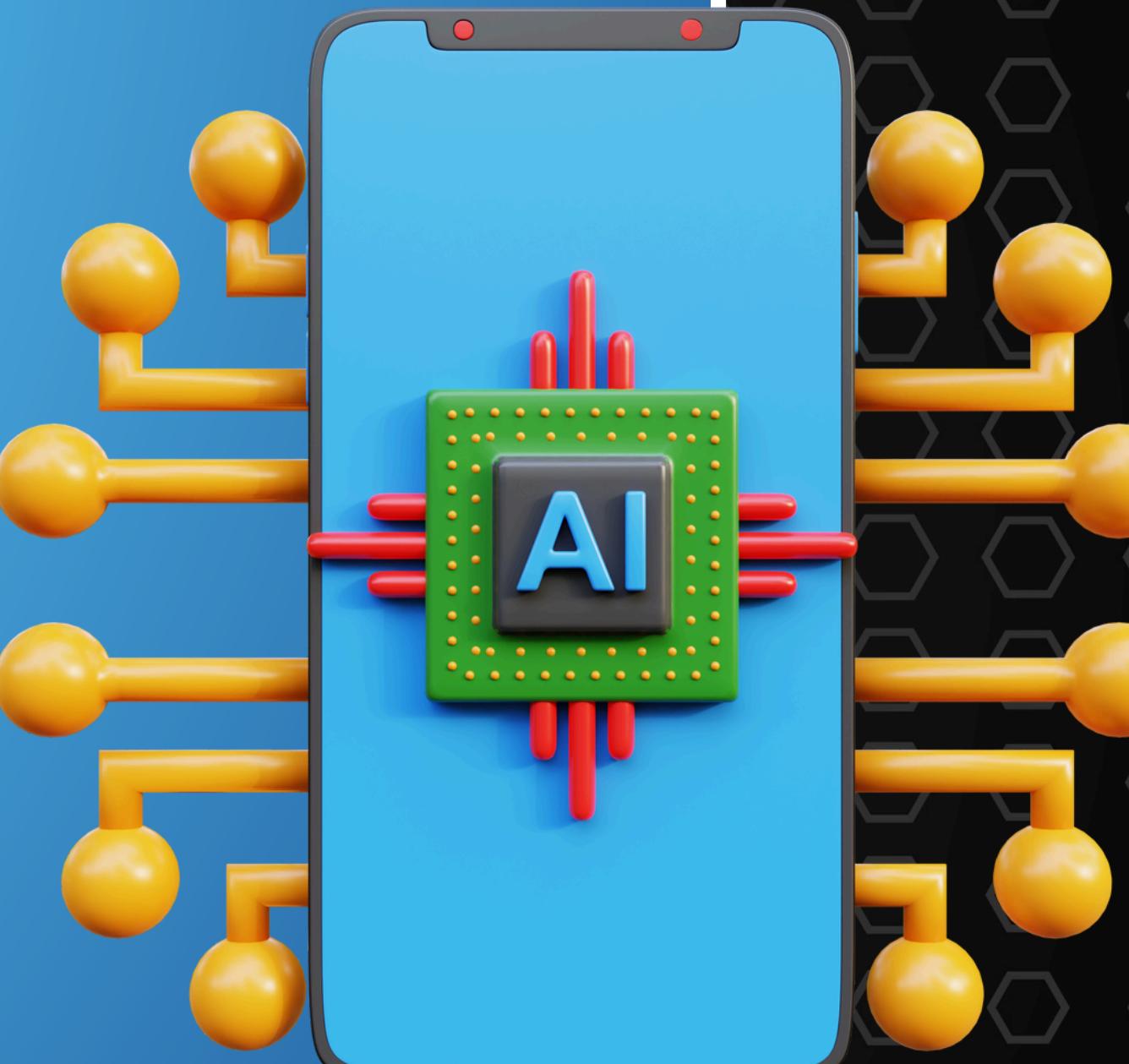
# Contenido

- Análisis del problema
- Diseño de la solución
- Solución final



# Descripción

En la actualidad, los sistemas operativos requieren una administración eficiente de los procesos para asegurar un buen desempeño y una distribución adecuada de los recursos disponibles. Este proyecto busca replicar ese proceso mediante el desarrollo de un sistema capaz de registrar procesos, organizarlos según su nivel de prioridad y gestionar la memoria que se les asigna. El propósito principal es reforzar el uso práctico de estructuras dinámicas lineales y aplicar conocimientos de programación en la solución de situaciones similares a las del mundo real.



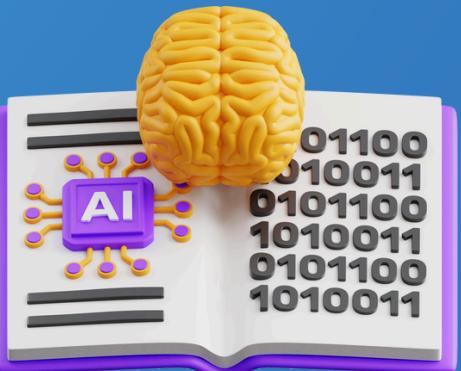
# Requerimientos del sistema

## 1.2.1 Funcionales:

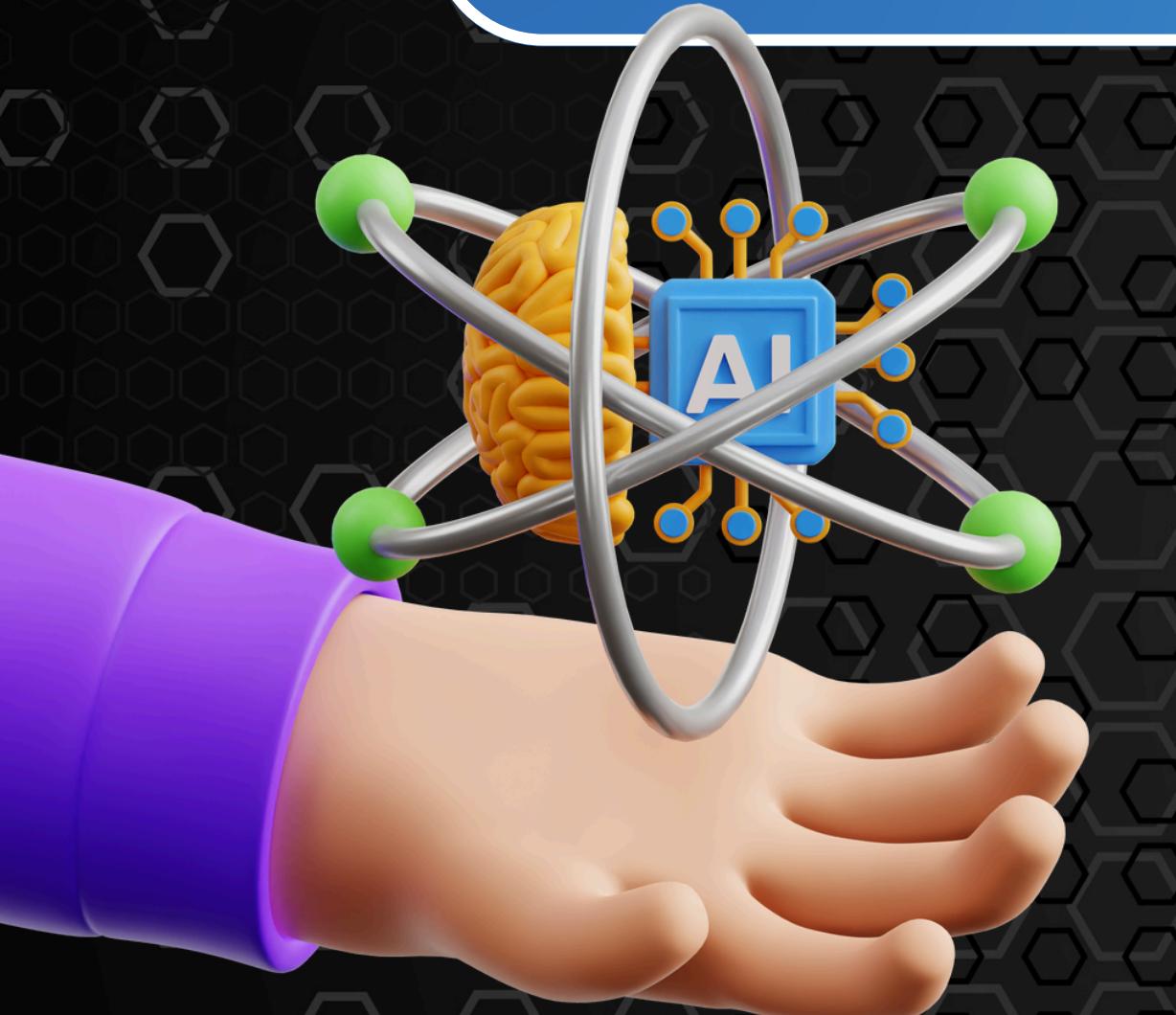
- Permitir el registro de procesos con ID, nombre y prioridad.
- Planificar la ejecución de procesos en base a su prioridad.
- Gestionar el uso de la memoria de forma dinámica por proceso.
- Mostrar visualmente el estado de cada estructura (lista, cola y pila).
- Brindar opciones para modificar, eliminar o buscar procesos.

## 1.2.2 No funcionales:

- El sistema debe ser rápido y eficiente en la ejecución de operaciones.
- Debe tener una interfaz amigable por consola.
- Debe permitir la persistencia de datos (guardar y recuperar estados).
- El código debe estar bien estructurado, modular y comentado



# Descripción de estructuras de datos



## Lista enlazada

La lista enlazada permite mantener un registro dinámico de todos los procesos activos en el sistema. Cada nodo de la lista contiene la información de un proceso (ID, nombre, prioridad) y un puntero al siguiente nodo.

## Cola

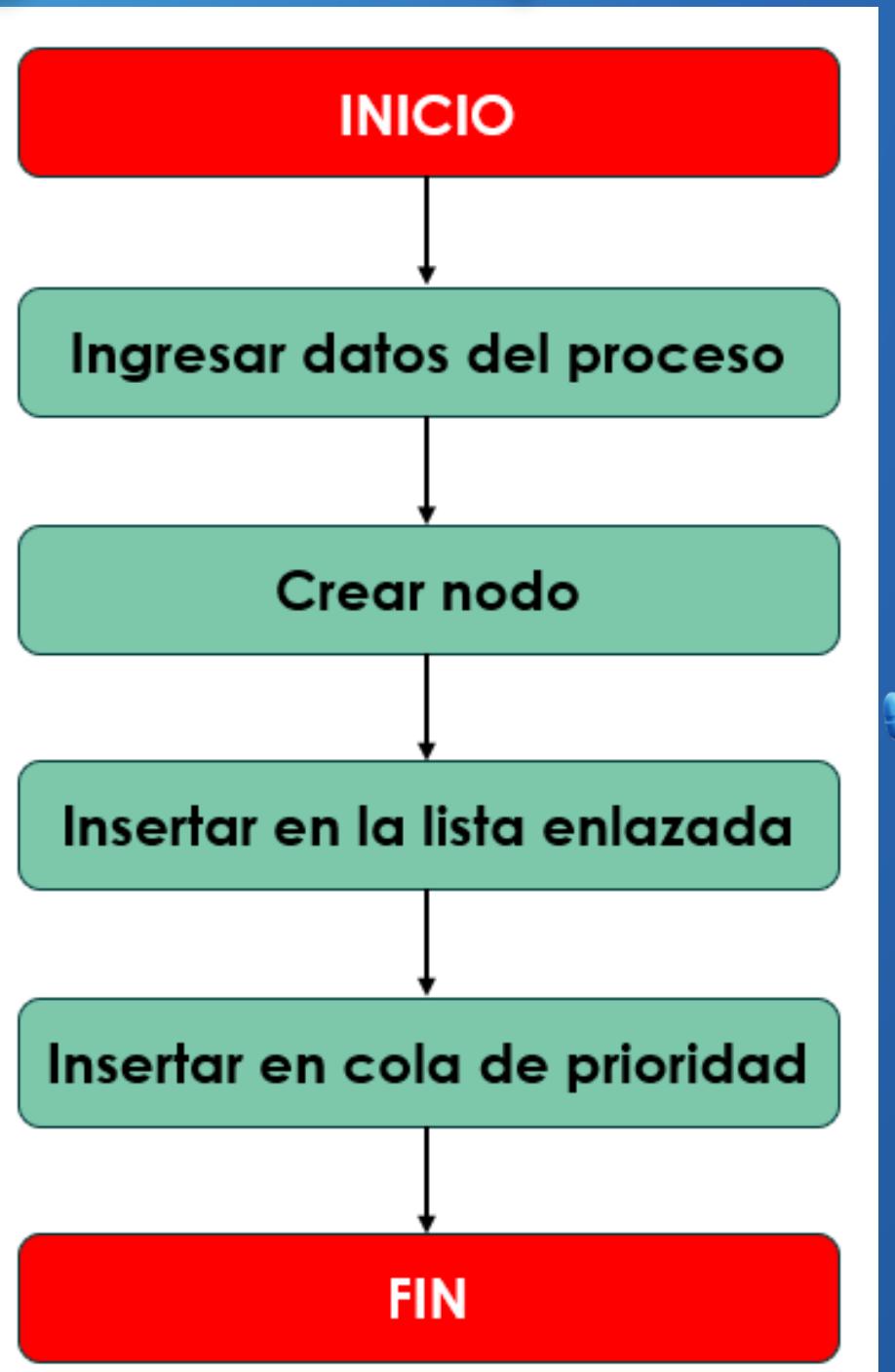
Esta estructura organiza los procesos pendientes de ejecución según su prioridad. Los procesos con mayor prioridad (valor numérico menor) se atienden primero, simulando el comportamiento de un planificador de CPU real.

## Pila

La pila se utiliza para simular la asignación y liberación de bloques de memoria por proceso. Funciona bajo el principio LIFO (Last In, First Out), apropiado para representar estructuras como la pila de llamadas de funciones en un sistema operativo.

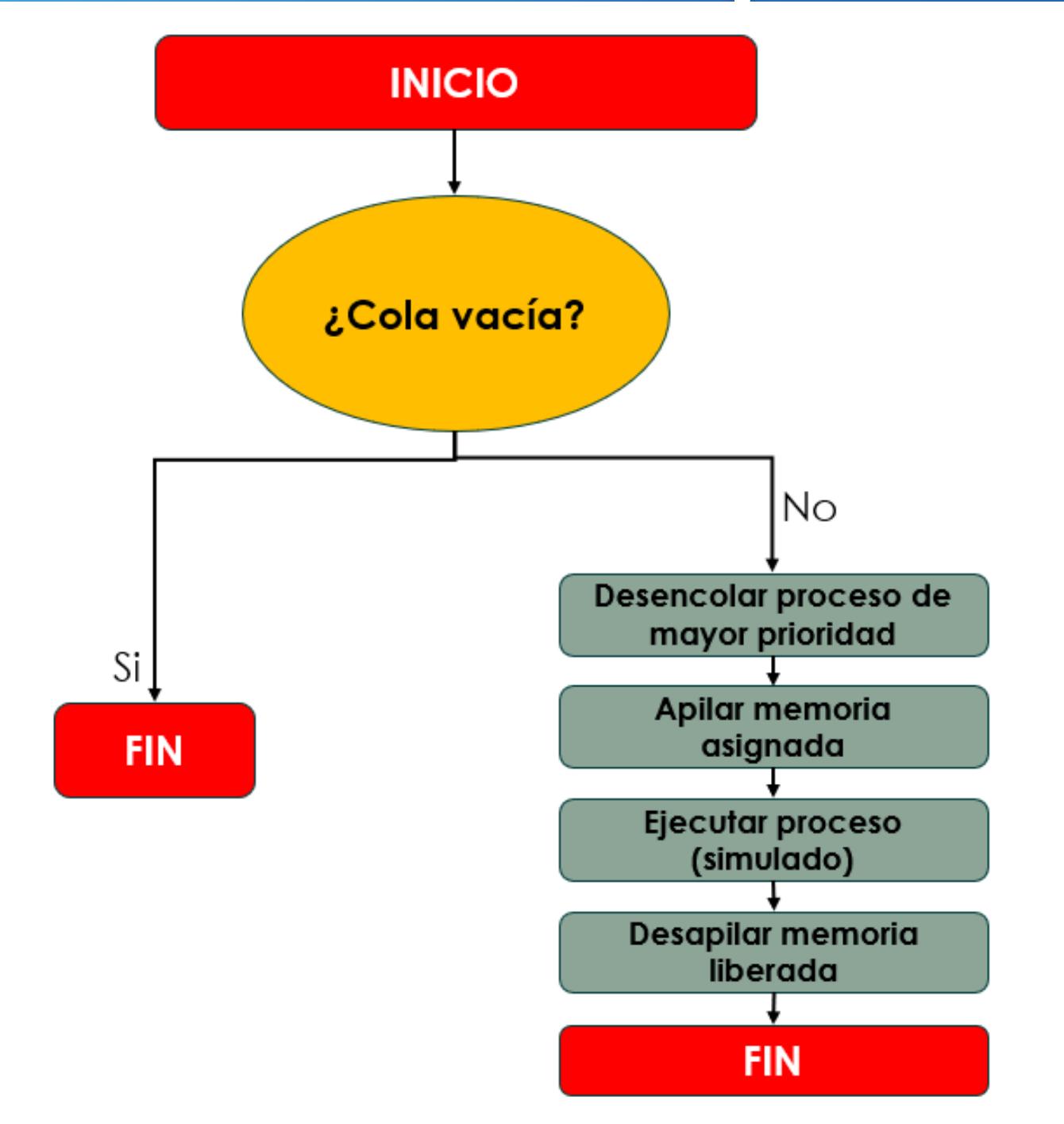
# Diagramas de flujo

## Registro de procesos



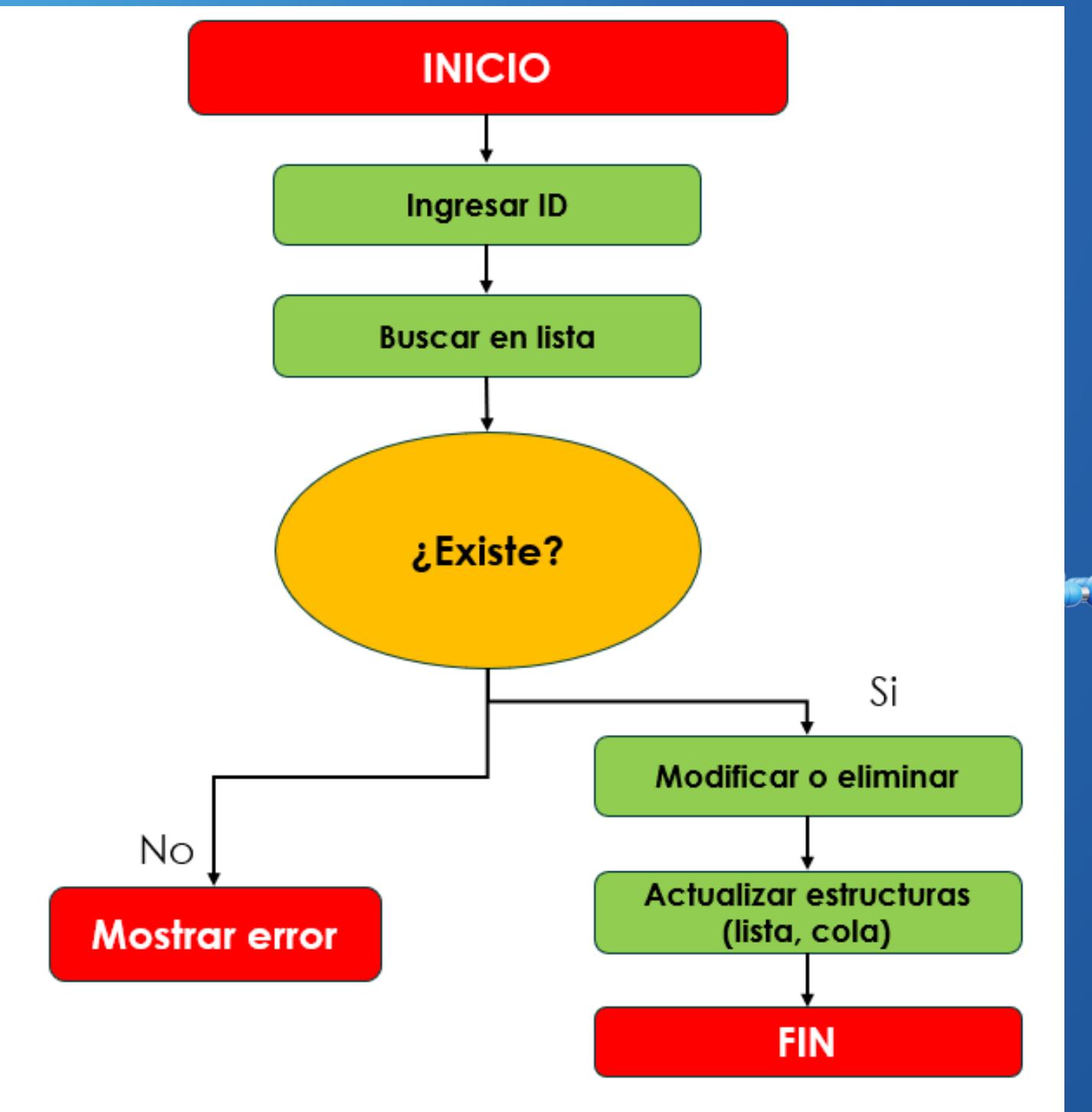
# Diagramas de flujo

## Planificación de proceso



# Diagramas de flujo

Modificación y/o eliminación de proceso:



# ALGORITMOS CLAVE

01

a) Registro de proceso  
**crearNodo(ID, nombre, prioridad);**  
**insertarEnLista(nuevoNodo);**  
**insertarEnColaPrioridad(nuevoNodo);**

02

b) Ejecución de proceso  
**proceso = desencolarPrioridad();**  
**if (proceso != NULL) {**  
    **asignarMemoria(proceso);**  
    **ejecutar(proceso);**  
    **liberarMemoria(proceso);**  
**}**



# ALGORITMOS CLAVE

03

c) Gestión de memoria  
**void asignarMemoria(Proceso p) {**  
    apilar(p.ID, bloqueMemoria);  
**}**  
**void liberarMemoria(Proceso p) {**  
    desapilar();  
**}**

04

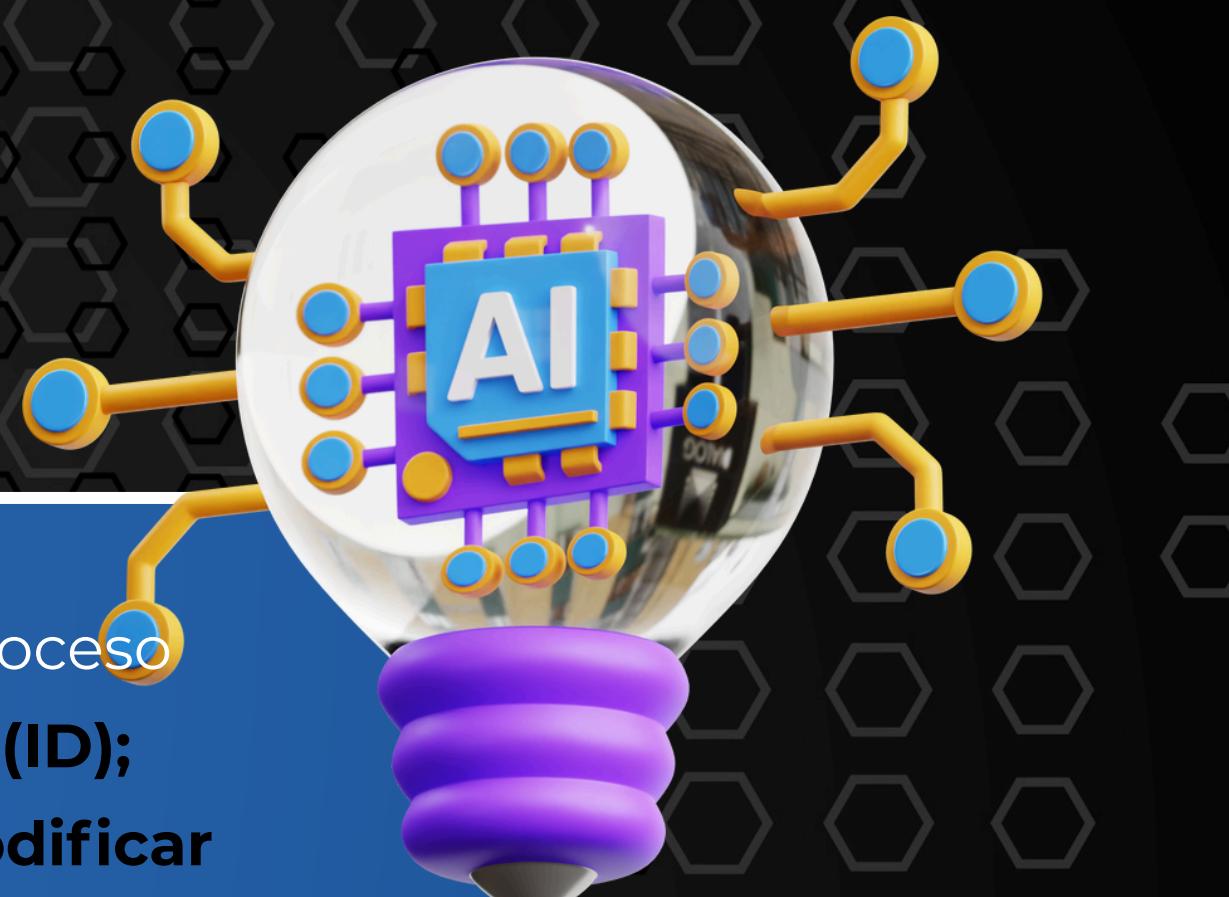
d) Buscar proceso  
**recorrerLista(ID o nombre);**  
**mostrarProceso();**

05

e) Modificar proceso  
**buscarEnLista(ID);**  
si existe -> modificar  
nombre o prioridad;

06

f) Eliminar proceso  
**buscarYEliminarDeLista(ID);**  
**eliminarDeCola(ID);**  
si en ejecución -> liberarMemoria();



# CODIGO PRINCIPAL

```
1 #include <iostream>
2 using namespace std;
3 #include "proceso.h"
4 #include "lista_enlazada.h"
5 #include "cola_prioridad.h"
6 #include "pila_memoria.h"
7
8 int main() {
9     setlocale(LC_CTYPE,"Spanish");
10    ListaEnlazada lista;
11    ColaPrioridad cola;
12    PilaMemoria pila;
13    int opcion;
14    do {
15        cout << "\n--- MENÚ PRINCIPAL ---\n";
16        cout << "1. Registrar proceso\n";
17        cout << "2. Ejecutar proceso\n";
18        cout << "3. Liberar memoria\n";
19        cout << "4. Mostrar estructuras\n";
20        cout << "5. Salir\n";
21        cout << "Ingrese opción: ";
22        cin >> opcion;
```

```
23     switch(opcion) {
24         case 1: {
25             int id, prioridad;
26             string nombre;
27             cout << "ID: "; std::cin >> id;
28             cout << "Nombre: "; std::cin >> nombre;
29             cout << "Prioridad (1=Alta, 2=Media, 3=Baja): "; std::cin >> prioridad;
30             Proceso p(id, nombre, prioridad);
31             lista.insertar(p);
32             cola.encolar(p);
33             pila.asignarMemoria(id);
34             break;
35         }
36         case 2:
37             cola.ejecutar();
38             break;
39         case 3:
40             pila.liberarMemoria();
41             break;
42         case 4:
43             lista.mostrar();
44             cola.mostrar();
45             pila.mostrar();
46             break;
47     }
48 } while (opcion != 5);
49 return 0;
50 }
```

# cola\_prioridad

```
1 #ifndef COLA_PRIORIDAD_H
2 #define COLA_PRIORIDAD_H
3
4 #include <queue>
5 #include <vector>
6 #include <iostream>
7 #include "proceso.h"
8
9 class ColaPrioridad {
10 private:
11     struct Comparador {
12         bool operator()(Proceso a, Proceso b) {
13             return a.prioridad > b.prioridad; // menor valor = mayor prioridad
14         }
15     };
16     std::priority_queue<Proceso, std::vector<Proceso>, Comparador> cola;
17 public:
18     void encolar(Proceso p) {
19         cola.push(p);
20     }
21     void ejecutar() {
22         if (!cola.empty()) {
23             Proceso p = cola.top();
24             cola.pop();
25             std::cout << "Ejecutando proceso ID " << p.id << ":" << p.nombre << "\n";
26         } else {
27             std::cout << "No hay procesos por ejecutar\n";
28         }
29     }
30     void mostrar() {
31         std::cout << "Procesos listos para ejecuciÃ³n: " << cola.size() << "\n";
32     }
33 };
34
35 #endif
```

# lista\_enlazada

```
1 #ifndef LISTA_ENLAZADA_H
2 #define LISTA_ENLAZADA_H
3
4 #include <iostream>
5 #include "proceso.h"
6
7 using namespace std;
8
9 struct Nodo {
10     Proceso proceso;
11     Nodo* siguiente;
12
13     Nodo(Proceso p) : proceso(p), siguiente(NULL) {} //
14 };
15
16 class ListaEnlazada {
17 private:
18     Nodo* cabeza;
19
20 public:
21     ListaEnlazada() : cabeza(NULL) {} //
22
23     void insertar(Proceso p) {
24         Nodo* nuevo = new Nodo(p);
25         if (!cabeza) {
26             cabeza = nuevo;
27         } else {
28             Nodo* temp = cabeza;
```

```
28         Nodo* temp = cabeza;
29         while (temp->siguiente)
30             temp = temp->siguiente;
31         temp->siguiente = nuevo;
32     }
33 }
34
35 void mostrar() {
36     Nodo* temp = cabeza;
37     cout << "\n--- Lista de procesos ---\n";
38     while (temp) {
39         cout << "ID: " << temp->proceso.id
40             << ", Nombre: " << temp->proceso.nombre
41             << ", Prioridad: " << temp->proceso.prioridad << endl;
42         temp = temp->siguiente;
43     }
44 }
45
46 Proceso* buscar(int id) {
47     Nodo* temp = cabeza;
48     while (temp) {
49         if (temp->proceso.id == id)
50             return &temp->proceso;
51         temp = temp->siguiente;
52     }
53     return NULL;
54 }
```

```
--  
56     bool eliminar(int id) {  
57         Nodo* temp = cabeza;  
58         Nodo* anterior = NULL;  
59         while (temp) {  
60             if (temp->proceso.id == id) {  
61                 if (anterior)  
62                     anterior->siguiente = temp->siguiente;  
63                 else  
64                     cabeza = temp->siguiente;  
65                 delete temp;  
66                 return true;  
67             }  
68             anterior = temp;  
69             temp = temp->siguiente;  
70         }  
71         return false;  
72     }  
73  
74     ~ListaEnlazada() {  
75         while (cabeza) {  
76             Nodo* temp = cabeza;  
77             cabeza = cabeza->siguiente;  
78             delete temp;  
79         }  
80     }  
81 };  
82  
83 #endif
```

# pila\_memoria

```
1 #ifndef PILA_MEMORIA_H
2 #define PILA_MEMORIA_H
3
4 #include <stack>
5 #include <iostream>
6 using namespace std;
7 class PilaMemoria {
8 private:
9     stack<int> memoria;
10 public:
11     void asignarMemoria(int procesoID) {
12         memoria.push(procesoID);
13         cout << "Memoria asignada al proceso ID " << procesoID << "\n";
14     }
15     void liberarMemoria() {
16         if (!memoria.empty()) {
17             int id = memoria.top();
18             memoria.pop();
19             cout << "Memoria liberada del proceso ID " << id << "\n";
20         } else {
21             cout << "No hay memoria para liberar\n";
22         }
23     }
24     void mostrar() {
25         cout << "Procesos con memoria asignada: " << memoria.size() << "\n";
26     }
27 };
28
29#endif
```

# proceso

```
1 #ifndef PROCESO_H
2 #define PROCESO_H
3
4 #include <string>
5
6 struct Proceso {
7     int id;
8     std::string nombre;
9     int prioridad;
10    Proceso(int i, std::string n, int p) : id(i), nombre(n), prioridad(p) {}
11 };
12
13 #endif
```

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 1

ID: 487

Nombre: panda

Prioridad (1=Alta, 2=Media, 3=Baja): 1

Memoria asignada al proceso ID 487

--- MENÚ PRINCIPAL ---

- .. Registrar proceso
- .. Ejecutar proceso
- .. Liberar memoria
- .. Mostrar estructuras
- .. Salir

Ingrese opción: 1

ID: 744

Nombre: youtube

Prioridad (1=Alta, 2=Media, 3=Baja): 2

Memoria asignada al proceso ID 744

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 1

ID: 784

Nombre: facebook

Prioridad (1=Alta, 2=Media, 3=Baja): 1

Memoria asignada al proceso ID 784

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 2

Ejecutando proceso ID 487: panda

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 4

--- Lista de procesos ---

ID: 487, Nombre: panda, Prioridad: 1  
ID: 744, Nombre: youtube, Prioridad: 2  
ID: 784, Nombre: facebook, Prioridad: 1  
Procesos listos para ejecución: 2  
Procesos con memoria asignada: 3

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 2

Ejecutando proceso ID 784: facebook

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 2

Ejecutando proceso ID 744: youtube

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 4

--- Lista de procesos ---

ID: 487, Nombre: panda, Prioridad: 1  
ID: 744, Nombre: youtube, Prioridad: 2  
ID: 784, Nombre: facebook, Prioridad: 1  
Procesos listos para ejecución: 0  
Procesos con memoria asignada: 3

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 3

Memoria liberada del proceso ID 784

--- MENÚ PRINCIPAL ---

- 1. Registrar proceso
- 2. Ejecutar proceso
- 3. Liberar memoria
- 4. Mostrar estructuras
- 5. Salir

Ingrese opción: 4

--- Lista de procesos ---

ID: 487, Nombre: panda, Prioridad: 1  
ID: 744, Nombre: youtube, Prioridad: 2  
ID: 784, Nombre: facebook, Prioridad: 1  
Procesos listos para ejecución: 0  
Procesos con memoria asignada: 2

--- MENÚ PRINCIPAL ---

1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir

Ingrese opción: 4

--- Lista de procesos ---

ID: 487, Nombre: panda, Prioridad: 1  
ID: 744, Nombre: youtube, Prioridad: 2  
ID: 784, Nombre: facebook, Prioridad: 1

Procesos listos para ejecución: 0

Procesos con memoria asignada: 2

--- MENÚ PRINCIPAL ---

1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir

Ingrese opción: 3

Memoria liberada del proceso ID 744

--- MENÚ PRINCIPAL ---

1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir

Ingrese opción: 3

Memoria liberada del proceso ID 487

--- MENÚ PRINCIPAL ---

1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir

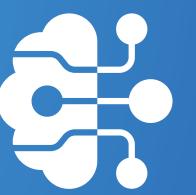
Ingrese opción: 4

--- Lista de procesos ---

ID: 487, Nombre: panda, Prioridad: 1  
ID: 744, Nombre: youtube, Prioridad: 2  
ID: 784, Nombre: facebook, Prioridad: 1

Procesos listos para ejecución: 0

Procesos con memoria asignada: 0



Studio  
shodwe

# Gacias

