

GRUPO 06

SISTEMA DE GESTIÓN DE PROCESOS

E S T R U C T U R A D E D A T O S

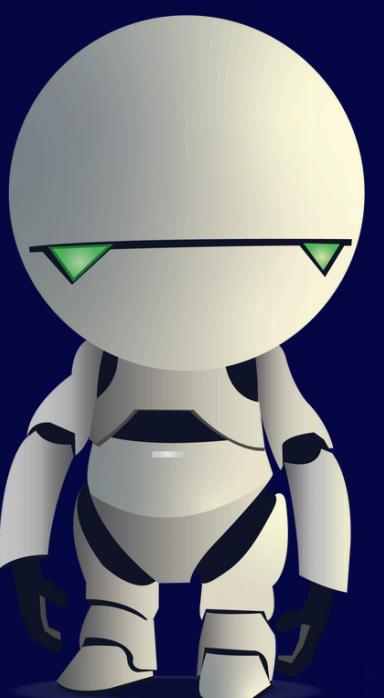
INTEGRANTES

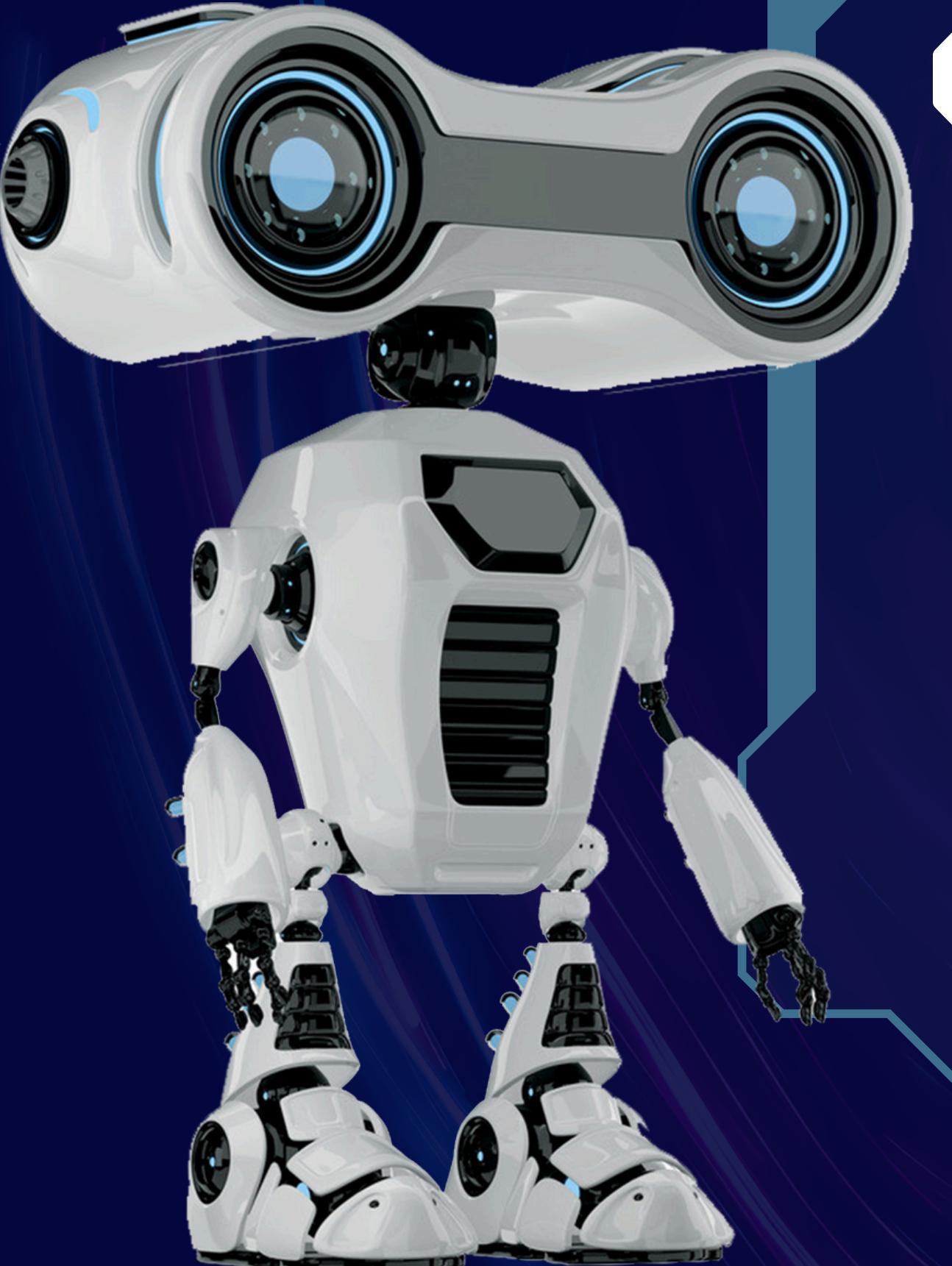
Merino Abarca Nidia Evelyn

Muñoz de la Cruz Marco Antonio

Paitan Rios Shamira Shantal

Chuquimantari Avila Jairo Sam





OBJETIVOS DEL PROGRAMA

El código simula la gestión de procesos en un sistema operativo básico:

- Crea procesos y los almacena en una lista enlazada.
- Permite asignar y liberar memoria con una pila .
- Ejecuta procesos por orden con una cola .

PROCESO

En esta parte del código podemos identificar lo siguiente:

- El **int id**, nos da un número para identificar el proceso.
- El **string nombre**, nos permite nombrar al proceso(Ejemplo: Chrome)
- int prioridad**: nos ayuda a priorizar la ejecución
- Proceso* siguiente**: Es un puntero para el siguiente nodo el cual nos ayuda a enlazar los procesos en la lista

```
6 // ===== Estructura de un proceso =====
7 struct Proceso {
8     int id;
9     string nombre;
10    int prioridad;
11    Proceso* siguiente;
12 };
13
```

Ejemplo:

- Se crea Proceso 1: ID=1, nombre="Word", prioridad=3.
- Se crea Proceso 2: ID=2, nombre="Chrome", prioridad=1

LISTA

- Proceso* inicio: El cual marca dónde comienza la lista (primer proceso).
- Proceso* fin: Este apunta al último proceso para agregar nuevos procesos eficientemente.
- int siguienteID: Guarda el próximo ID que se asignará automáticamente.

Ejemplo:

- La lista inicialmente está vacía: inicio = null, fin = null, siguienteID = 1
- Se crea proceso "Word" → inicio y fin apuntan a este nodo, siguienteID incrementa a 2.
- Se crea proceso "Chrome" → se enlaza al final, fin apunta a "Chrome", siguienteID es 3.

```
14 // ===== Lista de procesos (lista enlazada) =====  
15 struct Lista {  
16     Proceso* inicio;  
17     Proceso* fin;  
18     int siguienteID;  
19 }
```

NODOCOLA - COLA

```

21 // ===== Nodo para la cola =====
22 struct NodoCola {
23     Proceso* proceso;
24     NodoCola* siguiente;
25 };

```

- Proceso* proceso: Cada nodo almacena un proceso para ejecutar.
- NodoCola* siguiente: Enlaza los nodos en la cola.
- NodoCola* frente: Primer nodo que será ejecutado.
- NodoCola* final: Último nodo para insertar nuevos procesos en cola.

```

27 // ===== Cola de procesos =====
28 struct Cola {
29     NodoCola* frente;
30     NodoCola* final;
31 };

```

- Ejemplo dinámico:
- Cola vacía: frente=null, final=null.
 - Encolar proceso ID 1 → frente y final apuntan a este nodo.
 - Encolar proceso ID 2 → nuevo nodo añadido al final, final actualizado.

NODOPILA

- int idProceso: Identificador del proceso que está "en memoria".
- NodoPila* siguiente: Apunta al nodo debajo en la pila .

Ejemplo:

- Pila vacía: cima = null.
- Push ID 1 → cima apunta a nodo ID 1.
- Push ID 2 → cima apunta a nodo ID 2, que apunta al nodo ID 1.
- Pop → cima se mueve al nodo ID 1, sacando el ID 2.

```
14 // ===== Lista de procesos (lista enlazada) =====  
15 struct Lista {  
16     Proceso* inicio;  
17     Proceso* fin;  
18     int siguienteID;  
19 };
```

VOID INICIARLISTA

```
39     // === Funciones para Lista de procesos ===
40     void inicializarLista(Lista& lista) {
41         lista.inicio = nullptr;
42         lista.fin = nullptr;
43         lista.siguienteID = 1;
44     }
```

Inicializa la lista vacía con un contador de IDs automático.

CREAR Y AGREGAR PROCESO

```
46     // Crear y agregar un nuevo proceso a la lista
47     Proceso* crearProceso(Lista& lista, string nombre, int prioridad) {
48         Proceso* nuevo = new Proceso{ lista.siguienteID++, nombre, prioridad, nullptr };
49         if (!lista.inicio)
50             lista.inicio = lista.fin = nuevo;
51         else {
52             lista.fin->siguiente = nuevo;
53             lista.fin = nuevo;
54         }
55         cout << "Proceso creado con ID: " << nuevo->id << endl;
56         return nuevo;
57     }
```

Aquí podemos observar que:

- Crea un nuevo proceso.
- Le asigna un ID automático.
- Lo coloca al final de la lista.

Ejemplo:

Supongamos que creamos 2 procesos:
crearProceso(lista, "Editor", 2);
crearProceso(lista, "Navegador", 1);

La lista quedaría así:

[ID: 1, Editor, Prio: 2] -> [ID: 2, Navegador, Prio: 1] -> NULL

BUSCAR PROCESO

```
59 // Buscar un proceso por ID
60 Proceso* buscarProceso(Lista& lista, int id) {
61     Proceso* actual = lista.inicio;
62     while (actual) {
63         if (actual->id == id) return actual;
64         actual = actual->siguiente;
65     }
66     return nullptr;
67 }
```

Busca un proceso en la lista por su ID.

ELIMINAR PROCESO

```
69 // Eliminar un proceso por ID
70 bool eliminarProceso(Lista& lista, int id) {
71     Proceso* actual = lista.inicio;
72     Proceso* anterior = nullptr;
73     while (actual) {
74         if (actual->id == id) {
75             if (anterior)
76                 anterior->siguiente = actual->siguiente;
77             else
78                 lista.inicio = actual->siguiente;
79             if (actual == lista.fin)
80                 lista.fin = anterior;
81             delete actual;
82             return true;
83         }
84         anterior = actual;
85         actual = actual->siguiente;
86     }
87     return false;
88 }
```

Realiza:

- Busca el proceso por ID.
- Lo elimina, ajustando los punteros de la lista.

Visual antes y después:

Antes:

[ID: 1] -> [ID: 2] -> [ID: 3] -> NULL

Después de eliminarProceso(lista, 2);
[ID: 1] -> [ID: 3] -> NULL

```
90 // Mostrar todos los procesos registrados
91 void mostrarLista(Lista& lista) {
92     Proceso* actual = lista.inicio;
93     cout << "\nProcesos registrados:\n";
94     while (actual) {
95         cout << "ID: " << actual->id << ", Nombre: " << actual->nombre << ", Prioridad: " << actual->prioridad << "\n";
96         actual = actual->siguiente;
97     }
98 }
```

Imprime todos los procesos registrados.

```
100 // ===== Funciones para Cola =====
101 void inicializarCola(Cola& cola) {
102     cola.frente = cola.final = nullptr;
103 }
```

Inicializa la cola vacía.

```
105 // Encolar un proceso
106 void encolar(Cola& cola, Proceso* p) {
107     NodoCola* nuevo = new NodoCola{ p, nullptr };
108     if (!cola.frente) cola.frente = cola.final = nuevo;
109     else {
110         cola.final->siguiente = nuevo;
111         cola.final = nuevo;
112     }
113 }
```

Agrega un proceso al final de la cola.

Ejemplo:

Cola:

[ID: 2] -> [ID: 3] -> [ID: 5] -> NULL

```
115 // Desencolar el proceso del frente  
116 Proceso* desencolar(Cola& cola) {  
117     if (!cola.frente) return nullptr;  
118     NodoCola* temp = cola.frente;  
119     Proceso* p = temp->proceso;  
120     cola.frente = cola.frente->siguiente;  
121     if (!cola.frente) cola.final = nullptr;  
122     delete temp;  
123     return p;  
124 }
```

Saca el proceso del frente de la cola.

```
126 // Verificar si la cola esta vacia  
127 bool colavacia(Cola& cola) {  
128     return cola.frente == nullptr;  
129 }
```

Retorna true si no hay elementos en la cola.

```

131 // ===== Funciones para Pila =====
132 void push(NodoPila*& cima, int idProceso) {
133     NodoPila* nuevo = new NodoPila{ idProceso, cima };
134     cima = nuevo;
135 }
136
137 int pop(NodoPila*& cima) {
138     if (!cima) return -1;
139     int id = cima->idProceso;
140     NodoPila* temp = cima;
141     cima = cima->siguiente;
142     delete temp;
143     return id;
144 }
145
146 bool pilaVacia(NodoPila* cima) {
147     return cima == nullptr;
148 }
149
150 // Mostrar la pila de procesos en memoria
151 void mostrarPila(NodoPila* cima) {
152     cout << "\nMemoria (Pila):\n";
153     while (cima) {
154         cout << "ID Proceso: " << cima->idProceso << endl;
155         cima = cima->siguiente;
156     }
157 }
```

El **void Push** carga un proceso en memoria (al inicio de la pila).

Ejemplo:

Pila (memoria):

TOP -> [ID: 5] -> [ID: 3] -> [ID: 2] -> NULL

Por otro lado:

int pop

Libera el proceso más reciente en la pila.

En el :

bool pilaVacia

pregunta:

¿La pila está vacía? es decir verifica si hay procesos cargados en memoria.

En el:

void mostrarPila

Muestra todos los procesos en la pila.

```

159 // === Funcion principal (main) ===
160 int main() {
161     Lista lista;
162     Cola cola;
163     NodoPila* pila = nullptr;
164
165     inicializarLista(lista);
166     inicializarCola(cola);
167
168     int opcion;
169     do {
170         // Menu de opciones para el usuario
171         cout << "\n==== GESTION DE PROCESOS ===\n";
172         cout << "1. Crear Proceso\n";
173         cout << "2. Eliminar Proceso\n";
174         cout << "3. Buscar Proceso\n";
175         cout << "4. Actualizar Prioridad\n";
176         cout << "5. Asignar Memoria (Push)\n";
177         cout << "6. Liberar Memoria (Pop)\n";
178         cout << "7. Ver Procesos\n";
179         cout << "8. Ejecutar Procesos (CPU)\n";
180         cout << "9. Ver Memoria\n";
181         cout << "0. Salir\n";
182         cout << "Seleccione una opcion: ";
183         cin >> opcion;
184
185         if (opcion == 1) {
186             // Crear un nuevo proceso
187             string nombre;
188             int prioridad;
189             cout << "Nombre del proceso: ";
190             cin >> nombre;
191             cout << "Prioridad: ";
192             cin >> prioridad;

```

En nuestro main, vamos a encontrar varias opciones como lo son:

Opción 1 – Crear proceso
crearProceso(lista, "Chrome", 2)

Opción 2 – Eliminar proceso
eliminarProceso(lista, 2)

Opción 3 – Buscar proceso
buscarProceso(lista, 1)

Opción 4 – Actualizar prioridad
Modifica el atributo prioridad del proceso.

Opción 5 – Asignar Memoria (PUSH)
push(pila, 1)- Carga en memoria el proceso ID 1

Opción 6 – Liberar Memoria (POP)
int id = pop(pila)- Saca de memoria el último proceso

Opción 7 – Ver procesos
mostrarLista(lista)

Opción 8 – Ejecutar procesos (cola)
Encola todos los procesos.
Ejecuta uno a uno.
Si el usuario responde "s", se elimina el proceso y se libera memoria.

Opción 9 – Ver pila (memoria)
mostrarPila(pila)

GRACIAS