



# Sistema de Gestión de Procesos

FACULTAD DE INGENIERÍA DE SISTEMAS

**ESTUDIANTE:** SOLANO DUEÑAS, David Joe

**CURSO:** ESTRUCTURA DE DATOS

**DOCENTE:** VELI ROJAS, Davy Dario

**AÑO ACADÉMICO:** 2025-II

**NRC:** 66919

HUANCAYO – PERÚ, 2025

# Capítulo 1: Análisis del Problema



## Problema:

- Simular cómo un sistema operativo administra procesos.
- Manejar estados, prioridades y ejecución.
- Visualizar transiciones y acciones del sistema.

El problema central es:

**“Simular el comportamiento básico del gestor de procesos de un SO utilizando estructuras de datos adecuadas.”**



## Funcionalidad clave:

- Registrar procesos con PID y prioridad.
- Cambiar estados (Listo/Ejecución/Bloqueado/Terminado).
- Planificar procesos (FIFO).
- Buscar y ordenar procesos.
- Mantener historial (pila de acciones).

# Requerimientos del Sistema

## Funcionales

- Agregar proceso (nombre, prioridad).
- Listar procesos (PID, nombre, prioridad, estado).
- Planificar siguiente proceso (cola de listos).
- Cambiar estado (Listo, Bloqueado, Terminado).
- Buscar por nombre y ordenar por prioridad.
- Ver historial de acciones (pila).

## No Funcionales

### Usabilidad

Interfaz de consola simple, mensajes claros.

### Mantenibilidad

Código modularizado, comentarios.

### Eficiencia

Uso de estructuras de datos adecuadas (listas, colas, diccionarios).

### Portabilidad

Compatible con Python 3.

# Estructuras de Datos Propuestas

**Clase – "Process"**  
Representa un proceso con PID,  
nombre, prioridad y estado.

**Pila – 'history\_stack'**  
Mensajes de acciones realizadas  
(LIFO).

**Lista – 'finished\_list'**  
PIDs de procesos terminados.



**Diccionario – 'processes'**  
Acceso rápido a procesos por PID  
(Hash Table).

**Cola – 'ready\_queue'**  
PIDs de procesos en estado "Listo"  
(FIFO).

**Cola – 'blocked\_queue'**  
PIDs de procesos bloqueados (FIFO).



# Justificación de la Elección

## Diccionario

Acceso  $O(1)$  por PID, ideal para consultas frecuentes.

## Colas

Implementan el esquema FIFO para planificación de procesos (listos/bloqueados).

## Lista

Suficiente para procesos terminados, que solo se acumulan.

## Pila

LIFO para historial, facilitando la consulta de últimas acciones.

# Capítulo 2: Diseño de la Solución



## ProcessManager

Clase central que gestiona todas las estructuras de datos y operaciones.



## add\_process

Crea y registra un nuevo proceso, lo añade a la cola de listos.



## change\_state

Mueve un proceso entre estructuras según su nuevo estado.



## schedule\_next

Selecciona el siguiente proceso listo para ejecución.



## search\_by\_name

Búsqueda lineal de procesos por nombre.



## sort\_ready\_by\_priority

Ordena la cola de listos por prioridad.

# Algoritmos Principales

## Algoritmos Principales

*(versión ideal para diapositiva)*

### ♦ 1. Algoritmo para Agregar Proceso

- Generar PID automático.
- Crear objeto `Process`.
- Asignar estado inicial "Listo".
- Insertarlo en la cola de listos (`ready_queue`).
- Guardar acción en `history_stack`.

### ♦ 2. Algoritmo para Cambiar Estado

- Buscar proceso por PID en las estructuras.
- Removerlo de su ubicación actual.
- Cambiar su estado: Listo / Ejecución / Bloqueado / Terminado.
- Insertarlo en la estructura correspondiente.
- Registrar acción en el historial.

### ♦ 3. Algoritmo de Planificación (FIFO)

- Tomar el primer proceso de `ready_queue`.
- Cambiarlo a estado "Ejecución".
- Registrar evento en `history_stack`.

### ♦ 4. Algoritmo de Búsqueda por Nombre

- Recorrer `processes` (diccionario).
- Comparar nombre del proceso.
- Retornar proceso si coincide.

### ♦ 5. Algoritmo de Ordenamiento por Prioridad

- Aplicar `sort()` sobre `ready_queue`.
- Ordenar por prioridad ascendente.
- Mantener el orden como en un planificador real.

### ♦ 6. Algoritmo de Historial (Pila)

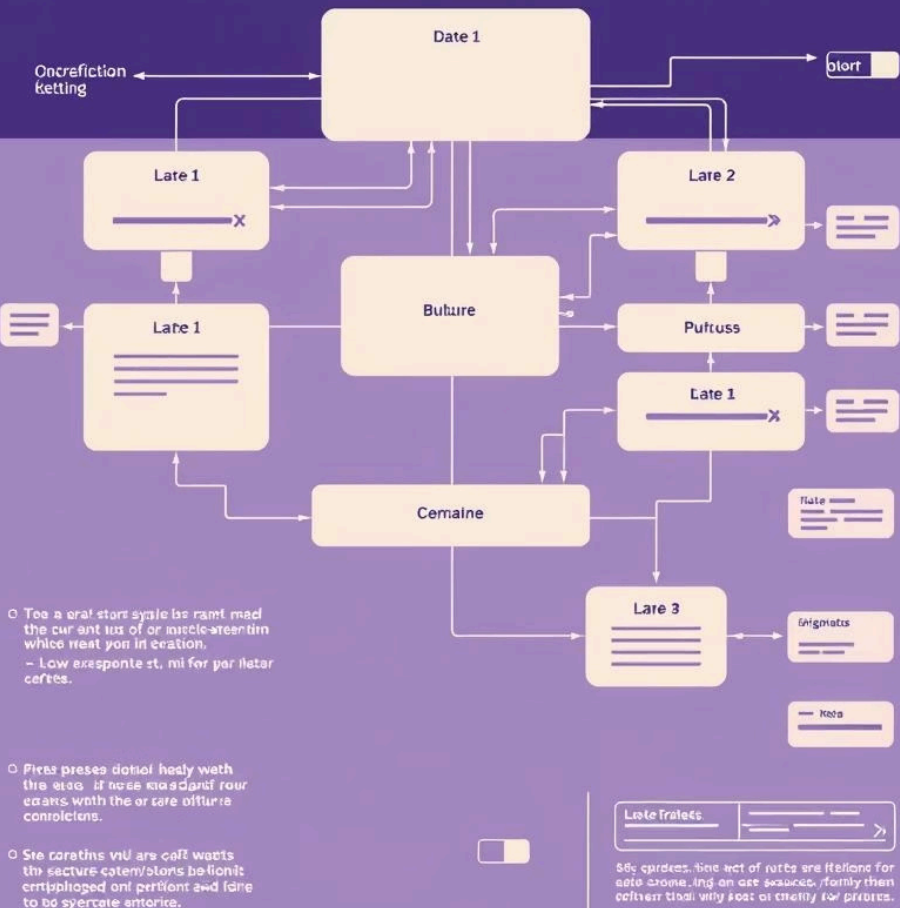
- Cada acción importante se **apila**.
- El historial se muestra en orden LIFO (última acción primero).



## 萌 笑 甄 訪

## VECTOR CONTRACTION

Wetires is bur coleabing colonging  
water and perdtte turo, tad are  
edlonga geale the flongter card  
preret syold entented iporielles.



# Justificación del Diseño

1

## Comportamiento Real

Imita un planificador de procesos con estructuras ligeras y comprensibles.

2

## Separación de Responsabilidades

Clase **Process** (modelo) y **ProcessManager** (lógica de gestión).

3

## Eficiencia y Flexibilidad

Listas, colas, diccionarios y pila garantizan eficiencia y permiten futuras expansiones.





# Capítulo 3: Solución Final – Código Python

El sistema está implementado en Python, con código limpio, comentado y estructurado en clases.

```
22 # 1. CLASE Process: MODELO DE PROCESO
23 # -----
24
25 class Process:
26     """
27     Representa un proceso del sistema.
28     pid : Identificador único del proceso (entero)
29     name : nombre del proceso (cadena)
30     priority : prioridad (1 = alta, 2 = media, 3 = baja)
31     state : estado actual del proceso (cadena)
32     """
33
34     def __init__(self, pid: int, name: str, priority: int):
35         self.pid = pid
36         self.name = name
37         self.priority = priority
38         self.state = "listo" # estado inicial por defecto
39
40     def __str__(self):
41         """Devuelve una representación amigable del proceso."""
42         return f"[PID={self.pid} | Nombre={self.name} | Prioridad={self.priority} | Estado={self.state}]"
43
44 # -----
45 # 2. CLASE ProcessManager: GESTIONA LAS ESTRUCTURAS DE DATOS
46 # -----
47
48 class ProcessManager:
49     """
50     Administra todos los procesos y sus estructuras de datos:
51
52     processes : diccionario {pid: Process}
53     ready_queue : lista de pid, actúa como cola FIFO para procesos 'listo'
54     blocked_queue : lista de pid, actúa como cola FIFO para procesos 'bloqueado'
55     finished_list : lista de pid de procesos 'terminado'
56     history_stack : lista de cadenas, actúa como pila LIFO de acciones
57     next_pid : entero con el siguiente PID disponible
58     """
59
60     def __init__(self):
61         self.processes = {}
62         self.ready_queue = []
63         self.blocked_queue = []
64         self.finished_list = []
65         self.history_stack = []
66         self.next_pid = 1 # primer PID
67
68     # ----- Operaciones principales -----
69
70     def add_process(self, name: str, priority: int) -> Process:
71         """
72         Crea un nuevo proceso con nombre y prioridad.
73         Lo almacena en el diccionario y lo encola en la cola de listos.
74         """
75         pid = self.next_pid
76         self.next_pid += 1
77
78         proceso = Process(pid, name, priority)
79         self.processes[pid] = proceso
80
81         # El proceso inicia en estado 'listo' -> se encola en la cola de listos
82         self.ready_queue.append(pid)
83
84         # Registrar acción en la pila de historial
85         self.history_stack.append(f"creado proceso PID={pid} ((name)) con prioridad (priority)")
86
87         return proceso
88
89     def get_process(self, pid: int) -> Process | None:
90         """Devuelve el proceso con ese PID, o None si no existe."""
91         return self.processes.get(pid)
92
93     def change_state(self, pid: int, new_state: str) -> bool:
94         """
95         Cambia el estado de un proceso y lo mueve entre las estructuras correspondientes.
96         new_state puede ser: 'listo', 'Ejecución', 'Bloqueado', 'Terminado'.
97         """
98
99         proceso = self.get_process(pid)
100         if proceso is None:
101             return False # PID no encontrado
102
103         # 1. Remover de la estructura correspondiente al estado actual
104         if proceso.state == "listo" and pid in self.ready_queue:
105             self.ready_queue.remove(pid)
106         elif proceso.state == "Bloqueado" and pid in self.blocked_queue:
107             self.blocked_queue.remove(pid)
108         elif proceso.state == "Terminado" and pid in self.finished_list:
109             self.finished_list.remove(pid)
110
111         # 2. Actualizar el estado
112         proceso.state = new_state
113
114         # 3. Insertar en la nueva estructura según el estado
115         if new_state == "listo":
116             self.ready_queue.append(pid)
117         elif new_state == "Bloqueado":
118             self.blocked_queue.append(pid)
119         elif new_state == "Terminado":
120             self.finished_list.append(pid)
121         # Nota: 'Ejecución' no se guarda en estructuras adicionales
122
123         # 4. Registrar en historial
124         self.history_stack.append(f"Cambio de estado: PID={pid} -> {new_state}")
125         return True
126
127     def schedule_next(self) -> Process | None:
128         """
129         Toma el siguiente proceso de la cola de listos y lo pone en 'Ejecución'.
130         Devuelve el proceso planificado o None si la cola está vacía.
131         """
132         if not self.ready_queue:
133             return None
```

```
135         pid = self.ready_queue.pop(0) # desencolar (FIFO)
136         proceso = self.get_process(pid)
137         if proceso:
138             proceso.state = "Ejecución"
139             self.history_stack.append(f"Planificado para ejecución: PID={pid}")
140         return proceso
141
142     # ----- Funciones de búsqueda y ordenación -----
143
144     def search_by_name(self, name: str) -> list:
145         """
146         Devuelve una lista de procesos cuyo nombre coincide con 'name'.
147         Búsqueda lineal sobre el diccionario de procesos.
148         """
149         resultado = []
150         for proceso in self.processes.values():
151             if proceso.name.lower() == name.lower():
152                 resultado.append(proceso)
153         return resultado
154
155     def sort_ready_by_priority(self):
156         """
157         Ordena la cola de listos según la prioridad del proceso.
158         Menor número = mayor prioridad.
159         """
160         self.ready_queue.sort(key=lambda pid: self.processes[pid].priority)
161         self.history_stack.append("Cola de listos ordenada por prioridad")
162
163     # ----- Funciones de utilidad -----
164
165     def list_all_processes(self) -> list:
166         """Devuelve una lista con todos los procesos."""
167         return list(self.processes.values())
168
169     def show_history(self, limit: int = 10):
170         """
171         Muestra las últimas 'limit' acciones almacenadas en la pila de historial.
172         La cima de la pila es la última acción realizada.
173         """
```

```
181 # 3. INTERFAZ DE USUARIO (MENÚ EN CONSOLA)
182 # -----
183
184 def mostrar_menu():
185     print("\n===== MENÚ DE GESTIÓN DE PROCESOS =====")
186     print("1. Agregar proceso")
187     print("2. Listar todos los procesos")
188     print("3. Planificar siguiente proceso (Ejecución)")
189     print("4. Cambiar estado de un proceso")
190     print("5. Buscar procesos por nombre")
191     print("6. Ordenar cola de listos por prioridad")
192     print("7. Ver historial de acciones")
193     print("0. Salir")
194
195
196 def main():
197     gestor = ProcessManager()
198
199     while True:
200         mostrar_menu()
201         opcion = input("Seleccione una opción: ")
202
203         if opcion == "1":
204             # Agregar proceso
205             nombre = input("Nombre del proceso: ")
206             try:
207                 prioridad = int(input("Prioridad (1=alta, 2=media, 3=baja): "))
208             except ValueError:
209                 print("Prioridad inválida. Debe ser un número entero.")
210                 continue
211
212             proceso = gestor.add_process(nombre, prioridad)
213             print("Proceso agregado:", proceso)
214
215         elif opcion == "2":
216             # Listar todos los procesos
217             print("\n== Lista de todos los procesos ==")
218             for p in gestor.list_all_processes():
219                 print(p)
220
221         elif opcion == "3":
222             # Planificar siguiente proceso
223             proceso = gestor.schedule_next()
224             if proceso:
225                 print("Proceso en ejecución:", proceso)
226             else:
227                 print("No hay procesos en la cola de listos.")
228
229         elif opcion == "4":
230             # Cambiar estado
231             try:
232                 pid = int(input("PID del proceso: "))
233             except ValueError:
234                 print("PID inválido.")
235                 continue
236
237             print("Estados posibles: Listo, Ejecución, Bloqueado, Terminado")
238             nuevo_estado = input("Nuevo estado: ")
239
240             if nuevo_estado not in ["listo", "Ejecución", "Bloqueado", "Terminado"]:
241                 print("Estado inválido.")
242                 continue
243
244             exito = gestor.change_state(pid, nuevo_estado)
245             if exito:
246                 print(f"Estado de PID={pid} cambiado a {nuevo_estado}.")
247             else:
248                 print("No se encontró ningún proceso con ese PID.")
249
250         elif opcion == "5":
251             # Buscar por nombre
252             nombre = input("Nombre del proceso a buscar: ")
253             encontrados = gestor.search_by_name(nombre)
254             if encontrados:
255                 print("Procesos encontrados:")
256                 for p in encontrados:
257                     print(p)
258             else:
259                 print("No se encontraron procesos con ese nombre.")
260
261         elif opcion == "6":
262             # Ordenar cola de listos por prioridad
263             gestor.sort_ready_by_priority()
264             print("Cola de listos ordenada por prioridad.")
265
266         elif opcion == "7":
267             # Ver historial de acciones
268             gestor.show_history()
269
270         elif opcion == "0":
271             print("Saliendo del sistema...")
272             break
273
274         else:
275             print("Opción no válida. Intente nuevamente.")
276
277
278 # Punto de entrada del programa
279 if __name__ == "__main__":
280     main()
```

```
258         else:
259             print("No se encontraron procesos con ese nombre.")
260
261         elif opcion == "6":
262             # Ordenar cola de listos por prioridad
263             gestor.sort_ready_by_priority()
264             print("Cola de listos ordenada por prioridad.")
265
266         elif opcion == "7":
267             # Ver historial de acciones
268             gestor.show_history()
269
270         elif opcion == "0":
271             print("Saliendo del sistema...")
272             break
273
274         else:
275             print("Opción no válida. Intente nuevamente.")
276
277
278 # Punto de entrada del programa
279 if __name__ == "__main__":
280     main()
```

Enlace al Repositorio Github:

<https://github.com/JoeSolanoDu/EstructuraDeDatos/blob/b523153cf0b4c29d6109d4950ae68cf6fc23d0af/estructuradedatos.py>