

SISTEMAS OPERATIVOS

TEMA DE UNIDAD: ADMINISTRACION DE ARCHIVOS

OBJETIVO DE LA ACTIVIDAD:

User GitHub: <https://github.com/yireel22/proyecto-integrador>

FECHA: 28/11/2025

Elaborar un simulador de gestor de procesos que implementa diferentes algoritmos de planificación de procesos en un sistema operativo. El simulador permite crear, ejecutar, suspender y terminar procesos, además de gestionar recursos del sistema y comunicación entre procesos, usando los conocimientos obtenidos durante la impartición de catedra de la materia de sistemas operativos impartida por el profesor Muñoz Quintero Dante Adolfo.

PARTICIPANTES:

- ANDRADE NIETO ISAAC YIREEL
- TINAJERO GUZMAN MARCO AXEL
- GARCÍA SALAS YAHIR MISael
- CRUZ HERNÁNDEZ KEVIN EFRÉN"
- SUAREZ MARTINEZ MACIEL FRANCISCO

SEMESTRE: SEXTO GRADO

GRUPO: J



Windows Vista

Mac



PRODUCTO INTEGRADOR

SIMULADOR DE GESTOR DE PROCESOS

Por: EQUIPO VI

28/11/2025

Gestor de Procesos:

Un gestor de procesos constituye el núcleo fundamental de cualquier sistema operativo moderno, funcionando como una entidad de coordinación sofisticada que supervisa el ciclo de vida completo de los procesos. Su arquitectura engloba mecanismos para la creación, planificación, ejecución, sincronización y terminación de procesos, operando como un intermediario crítico entre el hardware y el software. Este componente gestiona meticulosamente la asignación de recursos del sistema, incluyendo tiempo de CPU, espacio de memoria y acceso a dispositivos de E/S, mientras implementa protocolos de comunicación interprocesos que permiten la colaboración y el intercambio de información entre aplicaciones concurrentes. A través de algoritmos de planificación avanzados, el gestor optimiza el rendimiento del sistema, maximizando el throughput y minimizando los tiempos de respuesta, todo mientras mantiene la estabilidad y equidad en la distribución de recursos computacionales.

Algoritmos de Planificación

Los algoritmos de planificación representan el cerebro decisario del sistema operativo, determinando la secuencia temporal en que los procesos acceden a los recursos de procesamiento. Estas estrategias se clasifican según su comportamiento y objetivos de optimización:

- **FCFS (First-Come, First-Served):** Implementa una política de cola simple donde los procesos se ejecutan en estricto orden de llegada, manteniéndose en ejecución hasta su finalización completa. Este enfoque, aunque conceptualmente simple, puede generar el fenómeno de "convoy effect" donde procesos largos retrasan significativamente a procesos cortos.
- **SJFS (Shortest Job First):** Opera bajo el principio de optimalidad, priorizando la ejecución de procesos con menor duración estimada. Existe en dos variantes: no apropiativa (una vez iniciado un proceso, ejecuta hasta completarse) y apropiativa (si llega un proceso más corto, interrumpe el actual). Minimiza el tiempo promedio de espera, pero requiere conocimiento previo del tiempo de ejecución.



- **Round Robin:** Emplea un esquema de tiempo compartido basado en quantums temporales, donde cada proceso recibe una porción equitativa de tiempo de CPU antes de ser reinsertado al final de la cola de preparados. Este algoritmo garantiza equidad en la respuesta del sistema y es fundamental en entornos interactivos.
- Planificación por Prioridades: Asigna niveles de importancia jerárquica a cada proceso, ejecutando siempre el de mayor prioridad disponible. Puede presentar variantes estáticas (prioridades fijas) o dinámicas (prioridades ajustables según comportamiento). Requiere mecanismos de aging para prevenir inanición de procesos de baja prioridad.

Detección de Interbloqueos:

La detección de interbloqueos representa un subsistema crítico que identifica estados de bloqueo mutuo donde múltiples procesos permanecen suspendidos indefinidamente, esperando recursos retenidos por otros procesos en la misma situación de espera. Este mecanismo emplea modelado matemático mediante grafos de asignación de recursos, donde los procesos y recursos se representan como nodos y las asignaciones y solicitudes como arcos dirigidos. La detección se realiza mediante algoritmos de búsqueda de ciclos en estos grafos, donde la existencia de un ciclo cerrado indica necesariamente un interbloqueo. Los sistemas implementan regularmente exploraciones periódicas del estado del sistema, aplicando técnicas como el algoritmo del banquero para evaluación preventiva o métodos de detección reactiva que identifican interbloqueos existentes para posterior recuperación mediante aborto selectivo de procesos o rollback de transacciones.

Método Productor-Consumidor

El problema productor-consumidor encapsula un paradigma fundamental de programación concurrente donde dos entidades asíncronas (productor y consumidor) interactúan a través de un buffer compartido de capacidad limitada. El productor genera elementos de datos y los inserta en el buffer, mientras el consumidor los extrae y procesa. La complejidad radica en la sincronización precisa requerida para mantener la integridad de los datos y evitar condiciones de carrera. La solución implementa ^{典型}mente semáforos binarios para exclusión mutua del buffer y semáforos contadores para gestionar los espacios vacíos y llenos. Este modelo requiere manejo exhaustivo de situaciones límite: buffer lleno (productor debe bloquearse), buffer vacío (consumidor debe bloquearse) y acceso concurrente al buffer (requiere exclusión mutua). El paradigma extiende su aplicabilidad a numerosos escenarios incluyendo pipes del sistema operativo, colas de mensajería, buffers de E/S y sistemas de comunicación distribuida.



Funcionalidad:

Interfaz de usuario principal:

podemos encontrar el algoritmo seleccionado, el quantum base que dará el “sistema operativo” a los procesos, y justo debajo una platilla automática de procesos generados, su PID, el tipo y nombre de proceso, su tiempo y el recurso de memoria que ocupa.

The screenshot shows the PyCharm IDE interface. The project is named 'miAppGUI'. A terminal window is open with the command: C:\Users\yiree\PycharmProjects\miAppGUI\venv\Scripts\python.exe "C:\Users\yiree\PycharmProjects\miAppGUI\taller de python\examen1.py". The output in the terminal is as follows:

```
C:\Users\yiree\PycharmProjects\miAppGUI\venv\Scripts\python.exe "C:\Users\yiree\PycharmProjects\miAppGUI\taller de python\examen1.py"
== Simulador de Gestor de Procesos ==
Algoritmos disponibles: FCFS, SJF, RR, Prioridades
Selecione algoritmo (FCFS): rr
Quantum para Round Robin (3): 5
[Sat Nov 29 11:13:55 2025] Proceso creado: PID 1: Navegador (Listo) | Prioridad: 1 | Tiempo: 6 | Recursos: [memoria:1024]
[Sat Nov 29 11:13:55 2025] Proceso creado: PID 2: Editor (Listo) | Prioridad: 2 | Tiempo: 4 | Recursos: [memoria:512]
[Sat Nov 29 11:13:55 2025] Proceso creado: PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:2048]

== Menú Principal ==
Algoritmo actual: RR (Quantum: 5)
1. Crear proceso
2. Listar procesos
3. Suspender proceso
4. Reanudar proceso
5. Terminar proceso
6. Ejecutar ciclo
7. Ejecutar 5 ciclos automáticos
8. Demostración Productor-Consumidor
9. Ver historial de proceso
10. Cambiar algoritmo de planificación
0. Salir
6. presentacion de participantes
Seleccione una opción: |
```

Podemos ver debajo un menú selector en donde podemos elegir el tipo de orden que queremos dar las cuales son:

1.- CREAR PROCESO

```
Seleccione una opción: 1
Nombre del proceso: chrome
Tiempo de ejecución: 4
Prioridad (0=normal): 0
Memoria requerida (MB): 120
[Sat Nov 29 11:19:57 2025] Proceso creado: PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]
```

podemos observar que al seleccionar la opción 1 nos permite darle nombre al proceso, su tiempo de ejecución, la prioridad, aunque en el caso actual Round Robin no es necesaria, y la memoria requerida, terminando despliega la impresión del detalle del proceso creado.



2.- LISTAR PROCESOS

```
Seleccione una opción: 2
[]
== Estado del Sistema (Ciclo 0) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
    Ninguno

Cola de listos:
    PID 1: Navegador (Listo) | Prioridad: 1 | Tiempo: 6 | Recursos: [memoria:1024]
    PID 2: Editor (Listo) | Prioridad: 2 | Tiempo: 4 | Recursos: [memoria:512]
    PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:2048]
    PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
    Ninguno

Recursos disponibles:
    CPU: 1/1
    memoria: 392/4096
```

3.- SUSPENDER PROCESO

Antes:

```
--- Ciclo 1 (Algoritmo: RR) ---
[Sat Nov 29 12:16:12 2025] Proceso 1 en ejecución
[]
== Estado del Sistema (Ciclo 1) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
    PID 1: Navegador (Ejecutando) | Prioridad: 1 | Tiempo: 6 | Recursos: [CPU:1, memoria:1024]

Cola de listos:
    PID 2: Editor (Listo) | Prioridad: 2 | Tiempo: 4 | Recursos: [memoria:512]
    PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:2048]
    PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
    Ninguno

Recursos disponibles:
    CPU: 0/1
    memoria: 392/4096
```



Suspensión del pid 1:

```
Seleccione una opción: 3
PID del proceso a suspender: 1
[Sat Nov 29 12:16:51 2025] Proceso 1 suspendido
[Sat Nov 29 12:16:51 2025] Proceso 2 en ejecución
```

Después:

```
== Estado del Sistema (Ciclo 1) ===
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 2: Editor (Ejecutando) | Prioridad: 2 | Tiempo: 4 | Recursos: [CPU:1, memoria:512]

Cola de listos:
PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:2048]
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]
PID 1: Navegador (Listo) | Prioridad: 1 | Tiempo: 6 | Recursos: [memoria:1024]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 392/4096
```

Como pueden observar el funcionamiento es sencillo se inicializa el gestor y durante el ciclo 1 pedimos la suspensión con el menú selector y pasa al siguiente proceso exitosamente.

4.- REANUDAR PROCESO

De forma rápida le pediremos que lo vuelva a reintegrar

```
== Estado del Sistema (Ciclo 3) ===
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 1: Navegador (Ejecutando) | Prioridad: 1 | Tiempo: 6 | Recursos: [CPU:1, memoria:1024]

Cola de listos:
PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:2048]
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 904/4096
```



5.- terminar proceso

```
Seleccione una opción: 5
PID del proceso a terminar: 1
[Sat Nov 29 12:28:05 2025] Proceso 1 terminado. Causa: Terminado por usuario
[Sat Nov 29 12:28:05 2025] Proceso 3 en ejecución
```

Solo con seleccionar el pid que queremos terminar podemos sacarlo de la cola de procesos y al volver a listar veremos algo así:

```
== Estado del Sistema (Ciclo 5) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 3: Servidor (Ejecutando) | Prioridad: 0 | Tiempo: 8 | Recursos: [CPU:1, memoria:2048]

Cola de listos:
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 1928/4096
```

6.- ejecutar un ciclo, no tiene ciencia solo avanza un ciclo manualmente

```
Seleccione una opción: 6
[Sat Nov 29 12:29:48 2025]
--- Ciclo 6 (Algoritmo: RR) ---
[Sat Nov 29 12:29:48 2025] Proceso 3 ejecutando... Tiempo restante: 7
[]
== Estado del Sistema (Ciclo 6) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 3: Servidor (Ejecutando) | Prioridad: 0 | Tiempo: 7 | Recursos: [CPU:1, memoria:2048]

Cola de listos:
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 1928/4096
```



7.- ejecutar 5 ciclos

```
Seleccione una opción: 7
[Sat Nov 29 12:31:35 2025]
--- Ciclo 7 (Algoritmo: RR) ---
[Sat Nov 29 12:31:35 2025] Proceso 3 ejecutando... Tiempo restante: 6
[]
== Estado del Sistema (Ciclo 7) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 3: Servidor (Ejecutando) | Prioridad: 0 | Tiempo: 6 | Recursos: [CPU:1, memoria:2048]

Cola de listos:
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 1928/4096
[Sat Nov 29 12:31:36 2025]
--- Ciclo 8 (Algoritmo: RR) ---
[Sat Nov 29 12:31:36 2025] Proceso 3 ejecutando... Tiempo restante: 5
[]
== Estado del Sistema (Ciclo 8) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 3: Servidor (Ejecutando) | Prioridad: 0 | Tiempo: 5 | Recursos: [CPU:1, memoria:2048]

Cola de listos:
PID 4: chrome (Listo) | Prioridad: 0 | Tiempo: 4 | Recursos: [memoria:120]

Procesos bloqueados:
Ninguno
```

Desplegará automáticamente 5 ciclos seguidos imprimiendo claramente el avance de los procesos.



8.- demostración productor-consumidor

```
Seleccione una opción: 8
[Sat Nov 29 12:34:02 2025] Proceso creado: PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:256]
[Sat Nov 29 12:34:02 2025] Proceso creado: PID 6: Consumidor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:256]
Productor 5 escribió: Dato-0-de-5
Consumidor 6 leyó: Dato-0-de-5
[Sat Nov 29 12:34:02 2025]
--- Ciclo 12 (Algoritmo: RR) ---
[Sat Nov 29 12:34:02 2025] Proceso 4 ejecutando... Tiempo restante: 2
[]

== Estado del Sistema (Ciclo 12) ==
Algoritmo: RR (Quantum: 5)

Proceso en ejecución:
PID 4: chrome (Ejecutando) | Prioridad: 0 | Tiempo: 2 | Recursos: [CPU:1, memoria:120]
|
Cola de listos:
PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 3 | Recursos: [memoria:2048]
PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:256]
PID 6: Consumidor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:256]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 1416/4096
Productor 5 escribió: Dato-1-de-5
Consumidor 6 leyó: Dato-1-de-5
Productor 5 escribió: Dato-2-de-5
[Sat Nov 29 12:34:03 2025]
--- Ciclo 13 (Algoritmo: RR) ---
[Sat Nov 29 12:34:03 2025] Proceso 4 ejecutando... Tiempo restante: 1
[]
```

Como podemos observar La simulación ilustra el funcionamiento de la planificación Round Robin (turnos de CPU limitados por un Quantum) en un entorno de multiprogramación que ejecuta la lógica del Productor-Consumidor. Los procesos de alta prioridad (como chrome y Servidor) son ejecutados, y en el tiempo de CPU que no ocupan o durante su ejecución, los procesos Productor y Consumidor realizan sus tareas de forma concurrente, escribiendo y leyendo datos de una estructura compartida, lo que requiere mecanismos de sincronización (implícitos en las operaciones de escribió y leyó) para que funcionen correctamente.



```
Cola de listos:  
PID 3: Servidor (Listo) | Prioridad: 0 | Tiempo: 3 | Recursos: [memoria:26  
PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:  
PID 6: Consumidor (Listo) | Prioridad: 1 | Tiempo: 10 | Recursos: [memoria:  
  
Procesos bloqueados:  
Ninguno  
  
Recursos disponibles:  
CPU: 0/1  
memoria: 1416/4096  
Consumidor 6 leyó: Dato-2-de-5  
Productor 5 escribió: Dato-3-de-5  
Productor 5 escribió: Dato-4-de-5  
[Sat Nov 29 12:34:04 2025]  
--- Ciclo 14 (Algoritmo: RR) ---  
[Sat Nov 29 12:34:04 2025] Proceso 4 ejecutando... Tiempo restante: 0  
[Sat Nov 29 12:34:04 2025] Proceso 4 terminado. Causa: Finalización normal  
[Sat Nov 29 12:34:04 2025] Proceso 3 en ejecución  
[]
```

Procesos del Productor-Consumidor

Los procesos que interactúan son:

- **PID 5: Productor (Listo | Prioridad: 1):** Genera datos (Dato-1-de-5, Dato-2-de-5, etc.) y los escribe en un **búfer compartido** (implícito en el flujo de "escribió" / "leyó").
- **PID 6: Consumidor (Listo | Prioridad: 1):** Lee los datos que el Productor ha colocado en el búfer compartido.
- **PID 3: Servidor (Listo | Prioridad: 0):** Un proceso de alta prioridad que también está esperando.

Así hasta llegar al ciclo 25, ¿pero por qué se detiene en el ciclo 26?:



Estado Inicial (Ciclo 14 - 25)

En el inicio del **Ciclo 25 al final del avance de ciclos**, el sistema presenta la siguiente configuración:

- **Algoritmo:** Round Robin (RR) con Quantum: 5.
- **Proceso en ejecución (CPU): PID 6: Consumidor * Prioridad: 1**
 - **Tiempo restante (total del proceso):** 7
 - **Recursos:** CPU:1, memoria:256 (Está usando la única CPU disponible).
- **Cola de listos: PID 5: Productor**
 - **Prioridad:** 1
 - **Tiempo restante (total del proceso):** 5
 - **Recursos:** memoria:256
- **Recursos disponibles:**
 - **CPU:** 0/1 (La CPU está ocupada por el Consumidor).
 - **Memoria:** 3584/4096 (Hay 3584 unidades de memoria libres).

Ejecución del Ciclo 25

1. **El Consumidor (PID 6)** está ejecutando y recibe la CPU.
2. **El planificador (RR con Quantum=5)** le da un turno.
3. **El sistema avanza al Ciclo 26.**
4. **La línea Proceso 6 ejecutando... Tiempo restante: 6** indica que, al entrar al Ciclo 26, el **PID 6** sigue en ejecución, y su tiempo de servicio total restante ha disminuido de 7 a 6.



Estado Final (Ciclo 26)

Al final del **Ciclo 26**, justo antes de que el planificador decida el siguiente proceso:

- **Proceso en ejecución: PID 6: Consumidor**
 - **Prioridad:** 1
 - **Tiempo restante:** 6 (Aún no ha agotado el Quantum del ciclo actual).
 - **Recursos:** CPU:1, memoria:256
- **Cola de listos: PID 5: Productor**
 - **Prioridad:** 1
 - **Tiempo restante:** 5

Análisis de la Planificación

El **Consumidor (PID 6)** inició su ejecución al comienzo del Ciclo 25. Debido a que el **Quantum es 5**, puede ejecutar por hasta 5 ciclos de reloj.

- Si el **Consumidor** no se bloquea antes (por ejemplo, al intentar leer de un búfer vacío), seguirá usando la CPU hasta que su tiempo restante baje de **6 a 2** ($7 - 5 = 2$).
- En este punto (después de 5 ciclos de uso de CPU), su Quantum expira, y el planificador lo desalojará (preemption) y lo enviará al final de la cola de listos.
- El siguiente proceso en la cola (el **Productor, PID 5**) tomará la CPU.

Conclusión:

El **Consumidor (PID 6)** está actualmente monopolizando la CPU. Si no se bloquea, recibirá un turno completo de **5 ciclos** (el Quantum). Después de este turno, el **Productor (PID 5)** tendrá la oportunidad de ejecutarse.



9.- *historial de proceso, volvamos con nuestro querido amigo el PID 1 y con esta opción podremos ver que fue de el*

```
Seleccione una opción: 9
PID del proceso a consultar: 1

Historial del proceso 1:
[Sat Nov 29 11:13:55 2025] Asignado 1024 de memoria
[Sat Nov 29 11:13:55 2025] Agregado a cola de listos
[Sat Nov 29 12:16:12 2025] Asignado 1 de CPU
[Sat Nov 29 12:16:51 2025] Liberado 1 de CPU
[Sat Nov 29 12:16:51 2025] Agregado a cola de listos
[Sat Nov 29 12:16:51 2025] Proceso suspendido
[Sat Nov 29 12:26:34 2025] Asignado 1 de CPU
[Sat Nov 29 12:28:05 2025] Liberado 1 de CPU
[Sat Nov 29 12:28:05 2025] Liberado 1024 de memoria
[Sat Nov 29 12:28:05 2025] Proceso terminado. Causa: Terminado por usuario
```

10.- *cambio de algoritmo*

```
Seleccione una opción: 10

Algoritmos disponibles:
1. FCFS (First Come, First Served)
2. SJF (Shortest Job First)
3. RR (Round Robin)
4. Prioridades
Seleccione el algoritmo: |
```

Al seleccionar la opción nos permitirá cambiar el tipo de algoritmo que está usando el sistema o el gestor de procesos, no habrá mucho cambio más que en su naturaleza de procesar lo cual ya describimos un poco arriba en el documento



Si seleccionamos FCFS , continuara el funcionamiento normal a excepción que a diferencia de RR, este no cambiara de pid hasta terminar el tiempo de ejecución del proceso actual en la sección critica

```
[Sat Nov 29 12:49:44 2025]
--- Ciclo 27 (Algoritmo: FCFS) ---
[Sat Nov 29 12:49:44 2025] Proceso 6 ejecutando... Tiempo restante: 5
[]
==== Estado del Sistema (Ciclo 27) ====
Algoritmo: FCFS

Proceso en ejecución:
    PID 6: Consumidor (Ejecutando) | Prioridad: 1 | Tiempo: 5 | Recursos: [CPU:1, memoria:256]

Cola de listos:
    PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 5 | Recursos: [memoria:256]

Procesos bloqueados:
    Ninguno

Recursos disponibles:
    CPU: 0/1
    memoria: 3584/4096
```



Si seleccionamos sjf, cambiará de modo que cuando el proceso que ya está en la sección critica termine el siguiente proceso en entrar será el del tiempo más corto, volví a crear a crhome ahora con un tiempo de proceso menor que el proceso existente para la demostración:

```
== Estado del Sistema (Ciclo 31) ==
Algoritmo: SJF

Proceso en ejecución:
PID 6: Consumidor (Ejecutando) | Prioridad: 1 | Tiempo: 1 | Recursos: [CPU:1, memoria:256]

Cola de listos:
PID 7: crhome (Listo) | Prioridad: 0 | Tiempo: 2 | Recursos: [memoria:50]
PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 5 | Recursos: [memoria:256]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 3534/4096
[Sat Nov 29 12:54:33 2025]
--- Ciclo 32 (Algoritmo: SJF) ---
[Sat Nov 29 12:54:33 2025] Proceso 6 ejecutando... Tiempo restante: 0
[Sat Nov 29 12:54:33 2025] Proceso 6 terminado. Causa: Finalización normal
[Sat Nov 29 12:54:33 2025] Proceso 7 en ejecución
[]
== Estado del Sistema (Ciclo 32) ==
Algoritmo: SJF

Proceso en ejecución:
PID 7: crhome (Ejecutando) | Prioridad: 0 | Tiempo: 2 | Recursos: [CPU:1, memoria:50]

Cola de listos:
PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 5 | Recursos: [memoria:256]

Procesos bloqueados:
Ninguno
```



Si seleccionamos prioridades, como su nombre lo dice cuando termine el proceso que está en sección critica el gestor seleccionara el siguiente proceso, por su prioridad. Cree un nuevo proceso llamado Edge y uno llamado Microsoft store y les di un tiempo de proceso diferente alto y bajo y además una prioridad más alta del proceso aun en cola para la demostración:

```
Seleccione una opción: 7
[Sat Nov 29 13:12:00 2025]
--- Ciclo 35 (Algoritmo: PRIORIDADES) ---
[Sat Nov 29 13:12:00 2025] Proceso 8 ejecutando... Tiempo restante: 3
[]
== Estado del Sistema (Ciclo 35) ==
Algoritmo: PRIORIDADES

Proceso en ejecución:
PID 8: edge (Ejecutando) | Prioridad: 0 | Tiempo: 3 | Recursos: [CPU:1, memoria:25]

Cola de listos:
PID 5: Productor (Listo) | Prioridad: 1 | Tiempo: 5 | Recursos: [memoria:256]
PID 9: microsoft store (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:150]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 3665/4096
[Sat Nov 29 13:12:01 2025]
--- Ciclo 36 (Algoritmo: PRIORIDADES) ---
[Sat Nov 29 13:12:01 2025] Proceso 8 ejecutando... Tiempo restante: 2
```



Como dato final toda la primera presentación de funciones como pude notar fue hecha en Round Robin y cuando volvamos a cambiar a RR nos volverá a pedir un quantum como designa su algoritmo y respetará este nuevo Quantum para los procesos existentes.

```
Seleccione una opción: 10

Algoritmos disponibles:
1. FCFS (First Come, First Served)
2. SJF (Shortest Job First)
3. RR (Round Robin)
4. Prioridades
Seleccione el algoritmo: 3
Ingrese el quantum para Round Robin (3): 6
```

Antes de los ciclos:

```
== Estado del Sistema (Ciclo 39) ==
Algoritmo: RR (Quantum: 6)

Proceso en ejecución:
PID 5: Productor (Ejecutando) | Prioridad: 1 | Tiempo: 4 | Recursos: [CPU:1, memoria:256]

Cola de listos:
PID 9: microsoft store (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:150]
PID 10: canva (Listo) | Prioridad: 0 | Tiempo: 5 | Recursos: [memoria:152]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 3538/4096
```



Después de los ciclos (cree un nuevo proceso llamado canva para la demostración):

```
== Estado del Sistema (Ciclo 42) ==
Algoritmo: RR (Quantum: 6)

Proceso en ejecución:
PID 5: Productor (Ejecutando) | Prioridad: 1 | Tiempo: 1 | Recursos: [CPU:1, memoria:256]

Cola de listos:
PID 9: microsoft store (Listo) | Prioridad: 0 | Tiempo: 8 | Recursos: [memoria:150]
PID 10: canva (Listo) | Prioridad: 0 | Tiempo: 5 | Recursos: [memoria:152]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 3538/4096
[Sat Nov 29 13:24:14 2025]
--- Ciclo 43 (Algoritmo: RR) ---
[Sat Nov 29 13:24:14 2025] Proceso 5 ejecutando... Tiempo restante: 0
[Sat Nov 29 13:24:14 2025] Proceso 5 terminado. Causa: Finalización normal
[Sat Nov 29 13:24:14 2025] Proceso 9 en ejecución
[]
== Estado del Sistema (Ciclo 43) ==
Algoritmo: RR (Quantum: 6)

Proceso en ejecución:
PID 9: microsoft store (Ejecutando) | Prioridad: 0 | Tiempo: 8 | Recursos: [CPU:1, memoria:150]

Cola de listos:
PID 10: canva (Listo) | Prioridad: 0 | Tiempo: 5 | Recursos: [memoria:152]
```

Acabo con Edge y continuo con Microsoft como ya habíamos dicho, y ahora le daremos los 6 ciclos y hará el cambio automáticamente requerido.

```
== Estado del Sistema (Ciclo 49) ==
Algoritmo: RR (Quantum: 6)

Proceso en ejecución:
PID 10: canva (Ejecutando) | Prioridad: 0 | Tiempo: 5 | Recursos: [CPU:1, memoria:152]

Cola de listos:
PID 9: microsoft store (Listo) | Prioridad: 0 | Tiempo: 2 | Recursos: [memoria:150]

Procesos bloqueados:
Ninguno

Recursos disponibles:
CPU: 0/1
memoria: 3794/4096
```

**CODIGO FUENTE:**

```
#librerias
import queue
import time
import threading
from enum import Enum
from collections import deque
import os

# Enumeraciones
class Estado(Enum):
    LISTO = "Listo"
    EJECUTANDO = "Ejecutando"
    ESPERANDO = "Esperando"
    TERMINADO = "Terminado"

class CausaTerminacion(Enum):
    NORMAL = "Finalización normal"
    FORZADA = "Terminado por usuario"
    ERROR = "Error en ejecución"
    INTERBLOQUEO = "Interbloqueo detectado"
    SIN_MEMORIA = "Memoria insuficiente"

# Estructura PCB
class PCB:
    def __init__(self, pid, nombre, prioridad=0):
        self.pid = pid
        self.nombre = nombre
        self.estado = Estado.LISTO
        self.prioridad = prioridad
        self.recursos_asignados = {
            "CPU": False,
            "memoria": 0
        }
        self.tiempo_restante = 0
        self.tiempo_creacion = time.time()
        self.mensajes = queue.Queue()
        self.causa_terminacion = None
        self.historial = []

    def registrar_evento(self, evento):
        self.historial.append(f"[{time.ctime()}] {evento}")

    def __str__(self):
        recursos = ", ".join([f"{k}:{v}" for k, v in
        self.recursos_asignados.items() if v])
        estado = f"{self.estado.value}"
        if self.estado == Estado.TERMINADO and self.causa_terminacion:
            estado += f" ({self.causa_terminacion.value})"
        return f"PID {self.pid}: {self.nombre} ({estado}) | Prioridad: {self.prioridad} | Tiempo: {self.tiempo_restante} | Recursos: [{recursos}]"
```



```
# Recurso del sistema
class Recurso:
    def __init__(self, tipo, cantidad_total):
        self.tipo = tipo
        self.cantidad_total = cantidad_total
        self.cantidad_disponible = cantidad_total
        self.procesos_esperando = queue.Queue()

    def asignar(self, proceso, cantidad):
        if self.cantidad_disponible >= cantidad:
            self.cantidad_disponible -= cantidad
            proceso.recurso_asignado[self.tipo] += cantidad
            proceso.registrar_evento(f"Asignado {cantidad} de {self.tipo}")
            return True
        proceso.estado = Estado.ESPERANDO
        self.procesos_esperando.put((proceso, cantidad))
        proceso.registrar_evento(f"Esperando {cantidad} de {self.tipo}")
        return False

    def liberar(self, proceso, cantidad):
        self.cantidad_disponible += cantidad
        proceso.recurso_asignado[self.tipo] -= cantidad
        proceso.registrar_evento(f"Librado {cantidad} de {self.tipo}")

    # Intentar asignar a procesos en espera
    if not self.procesos_esperando.empty():
        prox_proceso, prox_cantidad = self.procesos_esperando.get()
        if self.asignar(prox_proceso, prox_cantidad):
            prox_proceso.estado = Estado.LISTO
            return prox_proceso
    return None

# Memoria compartida para productor-consumidor
class MemoriaCompartida:
    def __init__(self, capacidad=5):
        self.buffer = []
        self.capacidad = capacidad
        self.mutex = threading.Semaphore(1)
        self.lleno = threading.Semaphore(0)
        self.vacio = threading.Semaphore(capacidad)

    def escribir(self, dato, proceso):
        self.vacio.acquire()
        self.mutex.acquire()

        self.buffer.append(dato)
        proceso.registrar_evento(f"Productor escribió: {dato}")
        print(f"Productor {proceso.pid} escribió: {dato}")

        self.mutex.release()
        self.lleno.release()

    def leer(self, proceso):
```



```
        self.lleno.acquire()
        self.mutex.acquire()

        dato = self.buffer.pop(0)
        proceso.registrar_evento(f"Consumidor leyó: {dato}")
        print(f"Consumidor {proceso.pid} leyó: {dato}")

        self.mutex.release()
        self.vacio.release()
        return dato

# Gestor de Procesos completo
#no se por que algoritmo y quantum tienen una advertencia... revisar
despues
class GestorProcesos:
    def __init__(self, algoritmo="FCFS", quantum=3):
        self.procesos = {}
        self.cola_listos = deque()
        self.recursos = {
            "CPU": Recurso("CPU", 1),
            "memoria": Recurso("memoria", 4096) # 4GB
        }
        self.pid_counter = 1
        self.proceso_ejecutando = None
        self.algoritmo = algoritmo.upper() # Asegurar mayúsculas
        self.quantum = quantum
        self.contador_quantum = 0
        self.memoria_compartida = MemoriaCompartida()
        self.reloj = 0
        self.log_sistema = queue.Queue()
        self.eventos = threading.Event()
        self.running = True

    def log_evento(self, mensaje):
        entrada = f"[{time.ctime()}] {mensaje}"
        self.log_sistema.put(entrada)
        print(entrada)

    def crear_proceso(self, nombre, tiempo_ejecucion, prioridad=0,
memoria_necesaria=512):
        pid = self.pid_counter
        self.pid_counter += 1

        pcb = PCB(pid, nombre, prioridad)
        pcb.tiempo_restante = tiempo_ejecucion

        if not self.recursos["memoria"].asignar(pcb, memoria_necesaria):
            self.log_evento(f"Error: Memoria insuficiente para crear
proceso {nombre}")
            pcb.causa_terminacion = CausaTerminacion.SIN_MEMORIA
            pcb.estado = Estado.TERMINADO
            self.procesos[pid] = pcb
            return None

        self.procesos[pid] = pcb
        self.agregar_a_cola_listos(pcb)
```



```
        self.log_evento(f"Proceso creado: {pcb}")
        return pid

    def agregar_a_cola_listos(self, proceso):
        if self.algoritmo == "SJF":
            # Insertar ordenado por tiempo restante (menor primero)
            i = 0
            while i < len(self.cola_listos):
                if proceso.tiempo_restante <
self.cola_listos[i].tiempo_restante:
                    break
                i += 1
            self.cola_listos.insert(i, proceso)
        elif self.algoritmo == "PRIORIDADES":
            # Insertar ordenado por prioridad (mayor primero)
            i = 0
            while i < len(self.cola_listos):
                if proceso.prioridad > self.cola_listos[i].prioridad:
                    break
                i += 1
            self.cola_listos.insert(i, proceso)
        else: # FCFS y Round Robin
            self.cola_listos.append(proceso)

        proceso.estado = Estado.LISTO
        proceso.registrar_evento("Agregado a cola de listos")

    def planificar(self):
        if self.proceso_ejecutando is None and self.cola_listos:
            siguiente = self.cola_listos.popleft()

            if self.recursos["CPU"].asignar(siguiente, 1):
                siguiente.estado = Estado.EJECUTANDO
                self.proceso_ejecutando = siguiente
                self.contador_quantum = 0
                self.log_evento(f"Proceso {siguiente.pid} en ejecución")
            else:
                self.agregar_a_cola_listos(siguiente)

    def ejecutar_ciclo(self):
        self.reloj += 1
        self.log_evento(f"\n--- Ciclo {self.reloj} (Algoritmo:
{self.algoritmo}) ---")

        if self.proceso_ejecutando:
            self.proceso_ejecutando.tiempo_restante -= 1
            self.contador_quantum += 1

            self.log_evento(
                f"Proceso {self.proceso_ejecutando.pid} ejecutando...
Tiempo restante: {self.proceso_ejecutando.tiempo_restante}")

            # Verificar si el proceso terminó
            if self.proceso_ejecutando.tiempo_restante <= 0:
                self.terminar_proceso(self.proceso_ejecutando.pid,
CausaTerminacion.NORMAL)
                # Round Robin: Verificar quantum solo si estamos usando RR
```



```
        elif self.algoritmo == "RR" and self.contador_quantum >=
self.quantum:
            self.log_evento(f"Quantum agotado para proceso
{self.proceso_ejecutando.pid}")
            self.suspender_proceso(self.proceso_ejecutando.pid)

            self.planificar()
            self.detectar_interbloqueos()
            self.mostrar_estado()

def suspender_proceso(self, pid):
    if pid in self.procesos:
        proceso = self.procesos[pid]
        if proceso.estado == Estado.EJECUTANDO:
            proceso.estado = Estado.LISTO
            self.reursos["CPU"].liberar(proceso, 1)
            self.agregar_a_cola_listos(proceso)
            self.proceso_ejecutando = None
            proceso.registrar_evento("Proceso suspendido")
            self.log_evento(f"Proceso {pid} suspendido")
            self.planificar()

def reanudar_proceso(self, pid):
    if pid in self.procesos:
        proceso = self.procesos[pid]
        if proceso.estado == Estado.ESPERANDO:
            proceso.estado = Estado.LISTO
            self.agregar_a_cola_listos(proceso)
            proceso.registrar_evento("Proceso reanudado")
            self.log_evento(f"Proceso {pid} reanudado")
            return True
    return False

def terminar_proceso(self, pid, causa=CausaTerminacion.FORZADA):
    if pid in self.procesos:
        proceso = self.procesos[pid]

        # Liberar todos los recursos
        if proceso.reursos_asignados["CPU"] > 0:
            self.reursos["CPU"].liberar(proceso, 1)
        if proceso.reursos_asignados["memoria"] > 0:
            self.reursos["memoria"].liberar(proceso,
proceso.reursos_asignados["memoria"])

        proceso.estado = Estado.TERMINADO
        proceso.causa_terminacion = causa
        proceso.registrar_evento(f"Proceso terminado. Causa:
{causa.value}")
        self.log_evento(f"Proceso {pid} terminado. Causa:
{causa.value}")

        if self.proceso_ejecutando and self.proceso_ejecutando.pid ==
pid:
            self.proceso_ejecutando = None
            self.planificar()
            return True
    return False
```



```
def detectar_interbloqueos(self):
    # Detección simple: procesos esperando recursos en ciclo
    grafo = {}
    for pid, proceso in self.procesos.items():
        if proceso.estado == Estado.ESPERANDO:
            recursos_necesarios = []
            if proceso.recursos_asignados["CPU"] == 0:
                recursos_necesarios.append("CPU")
            grafo[pid] = recursos_necesarios

    # Si hay ciclos en el grafo, hay interbloqueo
    if grafo: # Implementación simplificada
        for pid in list(self.procesos.keys()):
            if self.procesos[pid].estado == Estado.ESPERANDO:
                self.log_evento(f"Possible interbloqueo detectado en proceso {pid}")
                self.terminar_proceso(pid,
CausaTerminacion.INTERBLOQUEO)
                break

    # Comunicación entre procesos
def enviar_mensaje(self, pid_emisor, pid_receptor, mensaje):
    if pid_receptor in self.procesos:
        self.procesos[pid_receptor].mensajes.put(mensaje)
        self.log_evento(f"Mensaje enviado de {pid_emisor} a {pid_receptor}: {mensaje}")
        return True
    return False

def recibir_mensaje(self, pid):
    if pid in self.procesos:
        try:
            mensaje = self.procesos[pid].mensajes.get_nowait()
            self.log_evento(f"Proceso {pid} recibió mensaje: {mensaje}")
            return mensaje
        except queue.Empty:
            self.log_evento(f"Proceso {pid} no tiene mensajes")
            return None
    return None

# Productor-Consumidor
def productor(self, pid, items):
    proceso = self.procesos.get(pid)
    if proceso:
        for i in range(items):
            dato = f"Dato-{i}-de-{pid}"
            self.memoria_compartida.escribir(dato, proceso)
            time.sleep(0.5)
        self.log_evento(f"Productor {pid} terminó")

def consumidor(self, pid, items):
    proceso = self.procesos.get(pid)
    if proceso:
        for _ in range(items):
            dato = self.memoria_compartida.leer(proceso)
```



```
        time.sleep(0.7)
        self.log_evento(f"Consumidor {pid} terminó")

    # Visualización del estado
    def mostrar_estado(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        print(f"\n==== Estado del Sistema (Ciclo {self.reloj}) ====")
        print(f"Algoritmo: {self.algoritmo}", end="")
        if self.algoritmo == "RR":
            print(f" (Quantum: {self.quantum})")
        else:
            print()

        print("\nProceso en ejecución:")
        print(f"  {self.proceso_ejecutando}" if self.proceso_ejecutando
else " Ninguno")

        print("\nCola de listos:")
        for proceso in self.cola_listos:
            print(f"  {proceso}")
        if not self.cola_listos:
            print("  Vacía")

        print("\nProcesos bloqueados:")
        bloqueados = [p for p in self.procesos.values() if p.estado ==
Estado.ESPERANDO]
        for proceso in bloqueados:
            print(f"  {proceso}")
        if not bloqueados:
            print("  Ninguno")

        print("\nRecursos disponibles:")
        for nombre, recurso in self.recursos.items():
            print(f"  {nombre}: {recurso.cantidad_disponible}/{recurso.cantidad_total}")

    # Interfaz de usuario
    def menu_principal(self):
        while self.running:
            print("\n==== Menú Principal ====")
            print(f"Algoritmo actual: {self.algoritmo}", end="")
            if self.algoritmo == "RR":
                print(f" (Quantum: {self.quantum})")
            else:
                print()
            print("1. Crear proceso")
            print("2. Listar procesos")
            print("3. Suspender proceso")
            print("4. Reanudar proceso")
            print("5. Terminar proceso")
            print("6. Ejecutar ciclo")
            print("7. Ejecutar 5 ciclos automáticos")
            print("8. Demostración Productor-Consumidor")
            print("9. Ver historial de proceso")
            print("10. Cambiar algoritmo de planificación")
            print("0. Salir")
            print("G. presentacion de participantes")
```



```
opcion = input("Seleccione una opción: ")

try:
    if opcion == "1":
        self.crear_proceso_interactivo()
    elif opcion == "2":
        self.mostrar_estado()
    elif opcion == "3":
        pid = int(input("PID del proceso a suspender: "))
        self.suspender_proceso(pid)
    elif opcion == "4":
        pid = int(input("PID del proceso a reanudar: "))
        self.reanudar_proceso(pid)
    elif opcion == "5":
        pid = int(input("PID del proceso a terminar: "))
        self.terminar_proceso(pid)
    elif opcion == "6":
        self.ejecutar_ciclo()
    elif opcion == "7":
        for _ in range(5):
            if not self.running:
                break
            self.ejecutar_ciclo()
            time.sleep(1)
    elif opcion == "8":
        self.demostacion_productor_consumidor()
    elif opcion == "9":
        pid = int(input("PID del proceso a consultar: "))
        self.mostrar_historial(pid)
    elif opcion == "10":
        self.cambiar_algoritmo()
    elif opcion == "0":
        self.running = False
    elif opcion == "g":
        print("\nMuchas gracias por parte del equipo 6")
        por utilizar nuestro software de simulación de gestor de procesos"
        print("Integrantes del equipo:")
        print("- Andrade Nieto Isaac Yireel")
        print("- Suarez Martinez Maciel Francisco")
        print("- Tinajero Guzman Marco Axel")
        print("- García Salas Yahir Misael")
        print("- Cruz Hernández Kevin Efrén")
        opcion = input("precione ENTER para regresar al
menu.")

    else:
        print("Opción no válida")
except ValueError:
    print("Entrada inválida. Ingrese un número.")

def cambiar_algoritmo(self):
    print("\nAlgoritmos disponibles:")
    print("1. FCFS (First Come, First Served)")
    print("2. SJF (Shortest Job First)")
    print("3. RR (Round Robin)")
    print("4. Prioridades")
```



```
opcion = input("Seleccione el algoritmo: ")

if opcion == "1":
    self.algoritmo = "FCFS"
    print("Algoritmo cambiado a FCFS")
elif opcion == "2":
    self.algoritmo = "SJF"
    print("Algoritmo cambiado a SJF")
elif opcion == "3":
    self.algoritmo = "RR"
    self.quantum = int(input("Ingrese el quantum para Round Robin (3): ") or 3)
    print(f"Algoritmo cambiado a Round Robin con quantum {self.quantum}")
elif opcion == "4":
    self.algoritmo = "PRIORIDADES"
    print("Algoritmo cambiado a Prioridades")
else:
    print("Opción no válida")

# Reorganizar la cola de listos con el nuevo algoritmo
procesos_a_reorganizar = list(self.cola_listos)
self.cola_listos = deque()
for proceso in procesos_a_reorganizar:
    self.agregar_a_cola_listos(proceso)

def crear_proceso_interactivo(self):
    nombre = input("Nombre del proceso: ")
    tiempo = int(input("Tiempo de ejecución: "))
    prioridad = int(input("Prioridad (0=normal): "))
    memoria = int(input("Memoria requerida (MB): "))
    self.crear_proceso(nombre, tiempo, prioridad, memoria)

def mostrar_historial(self, pid):
    if pid in self.procesos:
        print(f"\nHistorial del proceso {pid}:")
        for evento in self.procesos[pid].historial:
            print(f"  {evento}")
    else:
        print("PID no válido")

def demostracion_productor_consumidor(self):
    prod = self.crear_proceso("Productor", 10, 1, 256)
    cons = self.crear_proceso("Consumidor", 10, 1, 256)

    # Hilos para ejecutar concurrentemente
    hilo_prod = threading.Thread(target=self.productor, args=(prod, 5))
    hilo_cons = threading.Thread(target=self.consumidor, args=(cons, 5))

    hilo_prod.start()
    hilo_cons.start()

    # Ejecutar ciclos mientras trabajan
    for _ in range(15):
        if not self.running:
```



```
        break
    self.ejecutar_ciclo()
    time.sleep(1)

    hilo_prod.join()
    hilo_cons.join()

# Punto de entrada
if __name__ == "__main__":
    print("== Simulador de Gestor de Procesos ==")
    print("Algoritmos disponibles: FCFS, SJF, RR, Prioridades")
    algoritmo = input("Seleccione algoritmo (FCFS): ").upper() or "FCFS"
    quantum = 3
    if algoritmo == "RR":
        quantum = int(input("Quantum para Round Robin (3): ") or "3")

    gestor = GestorProcesos(algoritmo=algoritmo, quantum=quantum)

    # Crear algunos procesos de ejemplo
    gestor.crear_proceso("Navegador", 6, 1, 1024)
    gestor.crear_proceso("Editor", 4, 2, 512)
    gestor.crear_proceso("Servidor", 8, 0, 2048)

    # Iniciar interfaz
    gestor.menu_principal()
```



CONCLUSIÓN DEL PROYECTO:

- *Implementación de algoritmos de planificación: FCFS, SJF, Round Robin y Prioridades*
- *Gestión de recursos del sistema (CPU y memoria)*
- *Comunicación entre procesos mediante mensajes*
- *Simulación de productor-consumidor con memoria compartida*
- *Detección básica de interbloqueos*
- *Interfaz de usuario interactiva*

La implementación desarrollada en Python constituye un simulador integral de sistemas operativos que materializa exitosamente los conceptos teóricos fundamentales abordados en el curso. El proyecto demuestra una comprensión profunda de la gestión de procesos mediante la implementación completa de los cuatro algoritmos de planificación estudiados: FCFS (First-Come, First-Served) para escenarios secuenciales básicos, SJF (Shortest Job First) para optimización de tiempos de espera, Round Robin para sistemas de tiempo compartido y el algoritmo por Prioridades para entornos con distintos niveles de urgencia.

La arquitectura del sistema incorpora una gestión avanzada de recursos que simula de manera realista la administración concurrente de CPU y memoria, implementando mecanismos eficientes para la asignación, liberación y control de estos recursos críticos. En el ámbito de la comunicación interprocesos, el proyecto integra tanto el modelo de paso de mensajes como el esquema de memoria compartida, este último ejemplificado a través de una simulación robusta del problema productor-consumidor que maneja adecuadamente la sincronización, los buffers compartidos y la prevención de condiciones de carrera.

Un componente significativo lo representa el módulo de detección de interbloqueos, que implementa algoritmos para identificar situaciones de bloqueo mutuo mediante el análisis de grafos de asignación de recursos y la aplicación de estrategias de prevención. Este proyecto permite observar métricas de rendimiento, estados de los procesos y la dinámica de asignación de recursos, dando un reflejo de una plataforma educativa completa que valida la comprensión de los mecanismos internos de los sistemas operativos, desde la planificación de procesos hasta la gestión avanzada de recursos y la comunicación interprocesos, estableciendo las bases para el desarrollo de sistemas más complejos y el análisis de optimizaciones futuras.