Actividad: Simulación de Planificación de Procesos y Gestión de Memoria

Objetivo:

Implementar un simulador de planificación de procesos que incluya algoritmos de planificación tradicionales (FCFS, SJF, Round Robin, Prioridad) y un algoritmo de gestión de memoria (FMS). Los estudiantes analizarán cómo la disponibilidad de memoria afecta la ejecución de los procesos.

Instrucciones para los Estudiantes:

1. Conceptos Básicos

Antes de comenzar, repasen los siguientes conceptos:

- **Proceso**: Un programa en ejecución. Tiene un ID, un tiempo de llegada, un tiempo de ejecución (burst time), un requisito de memoria y, opcionalmente, una prioridad.
- **Estados de un proceso**: Nuevo, listo, en ejecución, espera y terminado.
- **Algoritmos de Planificación**:
  - **FCFS (First-Come, First-Served)**: El primer proceso que llega es el primero en ejecutarse.
  - **SJF (Shortest Job First)**: El proceso con el menor tiempo de ejecución se ejecuta primero.
  - **Round Robin**: Cada proceso tiene un tiempo limitado (quantum) para ejecutarse antes de pasar al siguiente.
  - **Prioridad**: Los procesos se ejecutan según su prioridad (el de mayor prioridad primero).
  - **FMS (First-Memory-Served)**: El primer proceso que llega y puede ser cargado en la memoria disponible se ejecuta primero.

---

2. Implementación del Simulador

Deben implementar un programa en Python que simule la ejecución de procesos utilizando los algoritmos de planificación mencionados. El programa debe:

1. Permitir al usuario ingresar una lista de procesos con los siguientes datos:
   - ID del proceso.
   - Tiempo de llegada.
   - Tiempo de ejecución (burst time).
   - Requisito de memoria.
   - Prioridad (opcional, para el algoritmo de prioridad).
2. Implementar los algoritmos de planificación:
   - FCFS.
   - SJF.
   - Round Robin (con un quantum definido por el usuario).
   - Prioridad.
   - FMS (First-Memory-Served).
3. Mostrar los siguientes resultados para cada algoritmo:

- Orden de ejecución de los procesos.
- Tiempo de espera de cada proceso.
- Tiempo de retorno (turnaround time) de cada proceso.
- Tiempo promedio de espera y de retorno.

---

3. Ejemplo de Código Base

Aquí tienes un ejemplo de código en Python para empezar:

```
class Proceso:
    def __init__(self, id, llegada, burst, memoria=None, prioridad=None):
        self.id = id
        self.llegada = llegada
        self.burst = burst
        self.memoria = memoria  # Requisito de memoria del proceso
        self.prioridad = prioridad
        self.tiempo_espera = 0
        self.tiempo_retorno = 0

def fcfs(procesos):
    tiempo_actual = 0
    for p in sorted(procesos, key=lambda x: x.llegada):
        p.tiempo_espera = tiempo_actual - p.llegada
        tiempo_actual += p.burst
        p.tiempo_retorno = tiempo_actual - p.llegada
    return procesos

def sjf(procesos):
    tiempo_actual = 0
    cola = sorted(procesos, key=lambda x: (x.llegada, x.burst))
    while cola:
        p = next((p for p in cola if p.llegada <= tiempo_actual), None)
        if not p:
            tiempo_actual += 1
            continue
        p.tiempo_espera = tiempo_actual - p.llegada
        tiempo_actual += p.burst
        p.tiempo_retorno = tiempo_actual - p.llegada
        cola.remove(p)
    return procesos

def round_robin(procesos, quantum):
    tiempo_actual = 0
    cola = list(procesos)
    while cola:
        p = cola.pop(0)
        if p.burst > quantum:
            tiempo_actual += quantum
            p.burst -= quantum
            cola.append(p)
        else:
```

```
            tiempo_actual += p.burst
            p.tiempo_retorno = tiempo_actual - p.llegada
            p.tiempo_espera = p.tiempo_retorno - p.burst
    return procesos

def prioridad(procesos):
    tiempo_actual = 0
    cola = sorted(procesos, key=lambda x: (x.llegada, x.prioridad))
    while cola:
        p = next((p for p in cola if p.llegada <= tiempo_actual), None)
        if not p:
            tiempo_actual += 1
            continue
        p.tiempo_espera = tiempo_actual - p.llegada
        tiempo_actual += p.burst
        p.tiempo_retorno = tiempo_actual - p.llegada
        cola.remove(p)
    return procesos

def fms(procesos, memoria_disponible):
    tiempo_actual = 0
    cola = sorted(procesos, key=lambda x: x.llegada)
    while cola:
        p = next((p for p in cola if p.llegada <= tiempo_actual and
p.memoria <= memoria_disponible), None)
        if not p:
            tiempo_actual += 1
            continue
        memoria_disponible -= p.memoria  # Asignar memoria al proceso
        p.tiempo_espera = tiempo_actual - p.llegada
        tiempo_actual += p.burst
        p.tiempo_retorno = tiempo_actual - p.llegada
        memoria_disponible += p.memoria  # Liberar memoria después de la
ejecución
        cola.remove(p)
    return procesos

# Ejemplo de uso
procesos = [
    Proceso(1, 0, 5, memoria=100),
    Proceso(2, 1, 3, memoria=200),
    Proceso(3, 2, 8, memoria=150)
]

memoria_disponible = 300  # Memoria total disponible en el sistema

print("FCFS:")
resultados = fcfs(procesos)
for p in resultados:
    print(f"Proceso {p.id}: Espera = {p.tiempo_espera}, Retorno =
{p.tiempo_retorno}")

print("\nSJF:")
resultados = sjf(procesos)
```

```
for p in resultados:
    print(f"Proceso {p.id}: Espera = {p.tiempo_espera}, Retorno =
{p.tiempo_retorno}")

print("\nRound Robin (Quantum = 2):")
resultados = round_robin(procesos, 2)
for p in resultados:
    print(f"Proceso {p.id}: Espera = {p.tiempo_espera}, Retorno =
{p.tiempo_retorno}")

print("\nPrioridad:")
procesos[0].prioridad = 2
procesos[1].prioridad = 1
procesos[2].prioridad = 3
resultados = prioridad(procesos)
for p in resultados:
    print(f"Proceso {p.id}: Espera = {p.tiempo_espera}, Retorno =
{p.tiempo_retorno}")

print("\nFMS (First-Memory-Served):")
resultados = fms(procesos, memoria_disponible)
for p in resultados:
    print(f"Proceso {p.id}: Espera = {p.tiempo_espera}, Retorno =
{p.tiempo_retorno}")
```

4. Experimentación y Análisis

- Ejecuten el programa con diferentes conjuntos de procesos y valores de memoria disponible.
- Comparen los resultados de los algoritmos y respondan las siguientes preguntas:
- ¿Cómo afecta la disponibilidad de memoria en la ejecución de procesos con FMS?
- ¿Qué sucede si un proceso requiere más memoria de la disponible en el sistema?
- ¿En qué situaciones sería útil combinar FMS con otros algoritmos de planificación?

5. Entrega

- Código Fuente: El programa completo en Python.
- Informe: Un documento que incluya:
- - Explicación del código.
- - Resultados obtenidos con diferentes conjuntos de procesos.
- - Respuestas a las preguntas de reflexión.

Evaluación:

- Funcionalidad (50%): El programa debe ejecutarse correctamente y cumplir con los requisitos.
- Análisis (30%): Claridad y profundidad del informe.
- Creatividad (20%): Mejoras adicionales, como simular la fragmentación de memoria o implementar una interfaz gráfica.