UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

SISTEMAS OPERATIVOS

Profesora: Dr. Gunnar Eyal Wolf Iszaevich

Alumno: Echevarria Aguilar Luis Angel
No. Cuenta: 320236235

TAREA 2. Comparación de planificadores.

Semestre: 2026-1 **Fecha de entrega:** 28 / 10 / 2025

INFORME TÉCNICO DE ENTREGA

DESCRIPCIÓN GENERAL DE LA TAREA

La Tarea 2 consistió en el desarrollo de un programa en Python (compara_planif.py) capaz de simular y comparar el rendimiento de diversos algoritmos de planificación de procesos. Esta tarea se basa en el análisis de rendimiento discutido en clase, ejemplificado por las pruebas de Raphael Finkel.
Los algoritmos base requeridos para la simulación fueron:

- FCFS (First-Come, First-Served)
- RR (Round Robin), con distintos quantums (q=1 y q=4)
- SPN (Shortest Process Next)

Para alcanzar la calificación máxima (10), la tarea solicitaba dos características adicionales:

1. Esquema Visual: La implementación de una línea de tiempo textual (similar a un diagrama de Gantt) que mostrara qué proceso se ejecutaba en cada pulso de reloj.
2. Algoritmo Avanzado: El desarrollo de un algoritmo de colas múltiples, para el cual se eligió FB (Retroalimentación Multinivel).

El objetivo final era generar cinco ejecuciones con cargas aleatorias y comparar las métricas clave de rendimiento (Tiempo de Retorno, Tiempo de Espera y Proporción de Penalización) para todos los algoritmos bajo las mismas condiciones.

LENGUAJE, ENTORNO Y EJECUCIÓN

Lenguaje y Entorno: La solución fue desarrollada en Python 3.8 o superior. Se utilizó exclusivamente la biblioteca estándar threading, que viene incluida con cualquier instalación de Python, por lo que no se requieren dependencias externas. El uso de Python se eligió por su sintaxis clara y su modelo de hilos accesible, lo que permite centrarse en la complejidad de la lógica de sincronización en lugar de en la gestión de memoria o la configuración del compilador.

Instrucciones de Ejecución: Para ejecutar el programa, solo se necesita un intérprete de Python 3 instalado en el sistema. Abra una terminal o línea de comandos, navegue con el comando “cd” hasta el directorio donde se encuentra el archivo y ejecute el siguiente comando:

python compara_planif.py

Al ejecutar el script compara_planif.py en la terminal, el programa realiza una simulación completa en dos fases:

1. Fase 1: Pruebas del Profesor
    - El script primero imprime el encabezado --- PRUEBAS DEL PROFESOR ---.
    - Ejecuta las dos rondas de prueba especificadas en la tarea: "Primera ronda (ejemplo profe)" y "Segunda ronda (ejemplo 'huecos')".
    - Para cada una de estas rondas, imprime un bloque de resultados que contiene la ejecución de los 5 planificadores (FCFS, RR1, RR4, SPN y FB). Cada resultado muestra las métricas T, E, y P, seguidas de su correspondiente línea de tiempo visual.
2. Fase 2: Ejecuciones Aleatorias
    - A continuación, el script imprime el encabezado --- 5 EJECUCIONES ALEATORIAS ---.

- Entra en un bucle que se repite 5 veces para cumplir con el requisito de la tarea.
- En cada iteración (ej. "- Ronda Aleatoria 1:"):
  - Imprime la definición de los procesos generados aleatoriamente (ej. A: 0, t=3; B: 2, t=5...).
  - Ejecuta la simulación completa para FCFS, RR1, RR4, SPN y FB con esa carga específica.
  - Imprime el bloque de resultados (métricas y línea de tiempo) para cada uno.

La salida total es un reporte detallado que permite la comparación directa de los algoritmos bajo 7 conjuntos de procesos diferentes (2 fijos y 5 aleatorios).

ESTRATEGÍA DE COMPARACIÓN DE PLANIFICADORES

La estrategia central fue la simulación empírica y la medición directa. Dado que la tarea especifica que "resulta iluso comparar a estos distintos mecanismos únicamente a partir de un único conjunto de procesos", el programa se diseñó para validar esta afirmación.
La metodología de comparación fue la siguiente:

1. Generación de Cargas: El programa genera 5 conjuntos distintos de procesos. Cada proceso tiene un tiempo de llegada y una duración de ráfaga (t) generados aleatoriamente.
2. Ejecución Idéntica (Piso Parejo): Cada uno de estos 5 conjuntos de procesos (o "Rondas") se somete a todos los algoritmos implementados: FCFS, RR(q=1), RR(q=4), SPN y FB. Esto es crucial, ya que permite una comparación directa de cómo cada algoritmo maneja la misma carga de trabajo.
3. Manejo de "Huecos": La simulación avanza pulso a pulso, por lo que es capaz de manejar correctamente el tiempo de CPU ocioso (-). Esto ocurre si un proceso termina y no hay otros en la cola de listos, o si ningún proceso ha llegado al inicio de la simulación.
4. Medición de Métricas: Para cada ejecución, el programa calcula tres métricas promedio:
   - T (Tiempo de Retorno): T = TiempoFin - TiempoLlegada
   - E (Tiempo de Espera): E = T - RáfagaTotal
   - P (Proporción de Penalización): P = T /RáfagaTotal
5. Visualización: Junto a las métricas, se presenta el esquema visual de la CPU. Esto permite una verificación cualitativa inmediata de la "justicia" (como en RR) o la "inanición" (un riesgo en SPN) del algoritmo.

Esta estrategia nos permite observar directamente cómo las decisiones de diseño de cada planificador (apropiativo vs. no apropiativo, prioridad estática vs. dinámica, cuántum corto vs. largo) impactan el rendimiento general del sistema.

CONCLUSIONES

Este proyecto fue un ejercicio sumamente revelador sobre las complejidades de la planificación de procesos. Si bien los conceptos teóricos de FCFS o SPN son directos, la implementación de simuladores apropiativos precisos expuso una gran cantidad de desafíos sutiles.
El principal problema encontrado durante el desarrollo fue un error persistente en las métricas de Round Robin (RR1 y RR4). Aunque los esquemas visuales parecían casi correctos (a menudo solo diferían por un "-" al final), las métricas T, E y P no coincidían con las proporcionadas en el ejemplo de la tarea, siendo consistentemente más altas.

El error era un "off-by-one" en el bucle de simulación principal. La causa fue la condición de salida original del bucle (while len(procesos_terminados) < n_procesos).

1. El Problema: Cuando el último proceso terminaba (ej. en t=19), el simulador registraba su tiempo_fin (como t=20) y lo añadía a procesos_terminados. El tiempo avanzaba a t=20.
2. El Ciclo Fantasma: El bucle while se evaluaba de nuevo. Dado que procesos_terminados se actualizó dentro del bucle, la condición len(...) < n_procesos seguía siendo verdadera en el inicio de este nuevo ciclo.
3. La Falla: El simulador ejecutaba un ciclo "fantasma" (de t=20 a t=21). Al no encontrar procesos, añadía un "-" al visual y avanzaba tiempo_actual a 21. Solo entonces el bucle terminaba.
4. El Resultado: Este ciclo fantasma no afectaba el tiempo_fin del último proceso (que ya se había registrado), pero causaba que el siguiente algoritmo (ej. RR4) comenzara con un tiempo_actual incorrecto o que la lógica de huecos fallara, propagando el error.

Este error fue extremadamente difícil de depurar por varias razones:

- Las métricas estaban cerca, pero no eran correctas, sugiriendo un error de redondeo o un problema menor, no un fallo de bucle.
- Los intentos de "arreglar" los huecos (por ejemplo, con lógicas de continue o saltos de tiempo) solo empeoraron el problema, rompiendo el algoritmo FB (que, irónicamente, funcionaba bien debido a su lógica de preemption que evitaba el ciclo fantasma).
- Se modificó constantemente el manejo de huecos y la lógica de avance del tiempo, sin éxito, porque el error no estaba ahí, sino en la condición de salida fundamental del bucle while.

La Solución: El error se corrigió reestructurando el bucle a un while True: y moviendo la condición de salida al inicio de la fase de selección de procesos (if proceso_en_ejecucion is None and not colas... break). Esto asegura que el bucle termina exactamente en el pulso de reloj donde ya no hay trabajo, sin ejecutar un ciclo fantasma.

En conclusión, la tarea demostró que, si bien SPN es óptimo en papel (como se vio en las rondas aleatorias, donde casi siempre tuvo el mejor T y E), los algoritmos apropiativos como FB ofrecen un equilibrio mucho mejor entre respuesta rápida (para procesos cortos) y rendimiento general, justificando su complejidad de implementación.