Universidad Nacional Autónoma de México
Facultad de Ciencias

PROYECTO

SISTEMAS OPERATIVOS

2026-1

Briceño Espinoza Alejandro - 420141215

Equipo 16

25 de Noviembre de 2025

1. Introducción

La planificación de la CPU es uno de los componentes centrales de un sistema operativo. Cada vez que varios procesos compiten por el procesador, el planificador debe decidir a qué proceso asignar la CPU y durante cuánto tiempo. De estas decisiones dependen métricas clave como el tiempo de respuesta percibido por el usuario, el tiempo de espera en la cola de listos y el aprovechamiento del procesador.

En este proyecto se implementó un simulador de planificación de CPU en Python capaz de ejecutar distintos algoritmos clásicos sobre conjuntos de procesos controlados. El objetivo no es reproducir todos los detalles de un sistema operativo real, sino contar con una herramienta sencilla y reproducible para comparar políticas bajo escenarios específicos de carga.

A partir de este simulador se realizaron experimentos con varios algoritmos no expulsivos y expulsivos, midiendo su impacto en el tiempo de espera, tiempo de retorno (*turnaround*) y tiempo de respuesta promedio. Los resultados permiten observar en la práctica los compromisos teóricos de cada estrategia y discutir en qué contextos resulta más adecuada.

2. Objetivos

El proyecto tiene como objetivo general:

- Diseñar e implementar un simulador de planificación de CPU que permita comparar algoritmos clásicos bajo diferentes escenarios de carga.

A partir de este objetivo general se plantean los siguientes objetivos específicos:

- Implementar en Python los algoritmos de planificación FCFS, SJF no expulsivo y Round Robin con quantum fijo.
- Definir un conjunto de escenarios de procesos (deterministas y pseudoaleatorios) que representen distintos patrones de carga.
- Medir para cada algoritmo y escenario las métricas de tiempo de espera, tiempo de retorno y tiempo de respuesta promedio.
- Automatizar la ejecución de los experimentos y la generación de tablas de resultados para facilitar el análisis.
- Validar el comportamiento del simulador mediante pruebas unitarias sobre los algoritmos y los escenarios definidos.

3. Marco teórico

En un sistema operativo multiprogramado, varios procesos pueden encontrarse en estado listo (*ready*) esperando a que el planificador de la CPU les asigne el procesador. Cada proceso se caracteriza, entre otros parámetros, por su tiempo de llegada al sistema, su duración de ráfaga de CPU y los instantes en que entra y sale de la CPU. A partir de estos datos se definen métricas clásicas:

- **Tiempo de espera** (W_i): tiempo total que el proceso i pasa en la cola de listos, es decir, el tiempo transcurrido desde su llegada hasta que está efectivamente ejecutándose, excluyendo el tiempo de CPU.
- **Tiempo de retorno o *turnaround*** (T_i): tiempo total desde la llegada del proceso hasta su finalización. Se calcula como $T_i = C_i - A_i$, donde A_i es el tiempo de llegada y C_i el tiempo de terminación.
- **Tiempo de respuesta** (R_i): tiempo desde la llegada del proceso hasta la primera vez que recibe CPU. En sistemas interactivos es una métrica especialmente relevante, pues determina cuánto tarda el usuario en percibir que el sistema reaccionó.

En este trabajo se consideran tres algoritmos de planificación de CPU:

First-Come, First-Served (FCFS)

FCFS es la política más simple: los procesos se atienden en el orden en que llegan a la cola de listos. Una vez que un proceso entra a la CPU, se mantiene allí hasta que termina su ráfaga; por tanto, es un algoritmo no expulsivo. Su implementación suele basarse en una cola FIFO. Aunque FCFS es fácil de entender y de implementar, puede producir tiempos de espera muy altos cuando un proceso largo se coloca al frente de la cola (*convoy effect*).

Shortest Job First (SJF) no expulsivo

SJF selecciona siempre el proceso con la ráfaga de CPU más corta entre los que están listos. En su versión no expulsiva, una vez que el proceso más corto entra a la CPU se ejecuta hasta terminar. Bajo ciertas suposiciones, SJF minimiza el tiempo de espera promedio, pero requiere conocer o estimar de alguna forma la duración de las ráfagas de CPU. Además, en su versión pura puede provocar inanición de procesos largos si llegan constantemente nuevos procesos cortos.

Shortest Remaining Time First (SRTF)

Shortest Remaining Time First es la versión expulsiva de SJF. En lugar de elegir una única vez al proceso con ráfaga más corta, SRTF reconsidera la decisión en cada instante de tiempo: en todo momento se ejecuta el proceso listo con menor *tiempo restante* de CPU. Si mientras un proceso está ejecutándose llega otro con una ráfaga restante más pequeña, el planificador expulsa al proceso actual y entrega la CPU al nuevo.

Bajo las mismas suposiciones ideales que SJF, SRTF minimiza el tiempo de espera promedio y también el tiempo de turnaround. Sin embargo, comparte sus limitaciones prácticas: requiere conocer o estimar la duración de las ráfagas de CPU y puede provocar inanición de procesos largos cuando llegan constantemente procesos más cortos. Además, al ser expulsivo introduce más cambios de contexto, lo que en un sistema real implicaría un may

Round Robin (RR)

Round Robin es un algoritmo expulsivo diseñado para sistemas de tiempo compartido. Cada proceso listo recibe la CPU durante un intervalo máximo llamado *quantum*. Si su ráfaga no termina en ese intervalo, el proceso es expulsado y colocado al final de la cola de listos, dando oportunidad a los demás. Un quantum muy grande hace que RR se parezca a FCFS, mientras que un quantum muy pequeño incrementa mucho el número de cambios de contexto. RR tiende a mejorar el tiempo de respuesta percibido por el usuario, a costa de empeorar el tiempo total de finalización de los procesos en algunas cargas.

4. Desarrollo del simulador

El simulador se implementó en Python 3.12 utilizando únicamente la biblioteca estándar. La estructura del proyecto se organizó en varios módulos para separar la lógica de planificación, la definición de escenarios de prueba y la automatización de experimentos.

Modelo de proceso y algoritmos

En el módulo `src/cpu_scheduler.py` se define una estructura de datos `Process` (como *dataclass*) que almacena el identificador del proceso, tiempo de llegada, ráfaga de CPU, prioridad y campos auxiliares para registrar su tiempo de inicio, término y ráfaga restante. Sobre este modelo se implementan las funciones:

- `simulate_fcfs`: simula la política FCFS recorriendo los procesos en orden de llegada y construye una línea de tiempo de ejecución.
- `simulate_sjf`: implementa SJF no expulsivo, seleccionando en cada paso el proceso listo con ráfaga más corta.
- `simulate_srtf`: implementa *Shortest Remaining Time First*, la versión expulsiva de SJF, eligiendo en cada instante el proceso listo con menor tiempo restante y permitiendo expulsar al proceso en ejecución cuando llega uno más corto.
- `simulate_rr`: implementa Round Robin con un quantum configurable, mediante una cola circular de procesos listos y un contador de ráfaga restante.

Cada función de simulación devuelve un diccionario con el nombre del algoritmo, la línea de tiempo y las métricas agregadas: tiempo de espera promedio, tiempo de turnaround promedio y tiempo de respuesta promedio. La función auxiliar `compute_metrics` calcula estos valores a partir de los tiempos de llegada, inicio y finalización de cada proceso.

Además, el módulo incluye una función `compare_algorithms` que ejecuta todos los algoritmos sobre un mismo conjunto de procesos de ejemplo y construye un *ranking* por métrica (1 = mejor) y un puntaje global (suma de rangos). Esta comparación sirve como herramienta interactiva para visualizar, en la terminal, qué política resulta más adecuada en un escenario concreto.

Escenarios y módulo de experimentos

El módulo `experiments/scenarios.py` encapsula la definición de escenarios de carga. Se incluyen tres escenarios deterministas (trabajos *batch*, llegadas escalonadas y carga interactiva) y dos escenarios pseudoaleatorios. Estos últimos se generan con un generador de números aleatorios local inicializado con una semilla fija, lo cual garantiza que los experimentos sean reproducibles entre ejecuciones.

Sobre estos escenarios se construyó el módulo `experiments/run_experiments.py`, que recorre automáticamente todos los escenarios y algoritmos, ejecuta las simulaciones y genera un resumen de resultados. El script produce tanto una tabla en formato Markdown, útil para incluir directamente en el reporte, como un archivo `data/results/summary.csv` que se puede utilizar posteriormente para graficar o realizar análisis adicionales.

Pruebas unitarias

Para aumentar la confiabilidad del simulador se añadieron pruebas unitarias utilizando `pytest`. Los archivos `test_fcfs.py`, `test_sjf.py`, `test_srtf.py` y `test_rr.py` validan el orden de ejecución y las métricas calculadas por cada algoritmo en escenarios pequeños donde se conoce de antemano el resultado esperado. Adicionalmente, los archivos `test_scenarios.py` y `test_experiments.py` comprueban que los escenarios estén bien formados, que los escenarios aleatorios sean deterministas y que el módulo de experimentos genere exactamente una fila de resultados por combinación escenario–algoritmo.

Estas pruebas permiten refactorizar el código con mayor confianza y sirven como documentación ejecutable del comportamiento esperado del simulador.

5. Experimentos y resultados

5.1. Metodología experimental

Para evaluar el comportamiento de los algoritmos implementados se diseñó un conjunto de escenarios de carga y se ejecutaron de forma automática los cuatro planificadores del simulador: *First-Come, First-Served* (FCFS), *Shortest Job First* no expulsivo (SJF), *Shortest Remaining Time First* (SRTF) y *Round Robin* (RR) con quantum fijo $q = 2$. En todos los casos se utilizó el mismo conjunto de procesos por escenario, de modo que las diferencias observadas provienen únicamente del algoritmo de planificación.

El simulador registra, para cada proceso, el tiempo de inicio de servicio, tiempo de finalización, tiempo de espera, tiempo de retorno (*turnaround*) y tiempo de respuesta. A partir de estos valores se calculan los promedios por algoritmo: tiempo de espera promedio, tiempo de turnaround promedio y tiempo de respuesta promedio. No se modeló el costo del cambio de contexto, por lo que el foco del análisis está en el efecto "ideal" de cada política sobre estas métricas. Tampoco se modelan operaciones de entrada/salida ni estados bloqueados: se supone que cada proceso consiste en una única ráfaga de CPU.

Para evitar errores manuales y garantizar reproducibilidad, se implementó un módulo `experiments` que recorre todos los escenarios, ejecuta cada algoritmo y genera automáticamente un resumen en formato CSV y una tabla en Markdown. La Tabla 1 muestra los resultados promedio obtenidos.

Adicionalmente, a partir del archivo `data/results/summary.csv` se implementó un pequeño script en Python para generar gráficas de barras por escenario, comparando los tiempos de espera, turnaround y respuesta promedio de cada algoritmo. Estas visualizaciones complementan la tabla numérica y facilitan identificar de forma cualitativa qué políticas dominan a las demás en cada carga de trabajo.

5.2. Escenarios evaluados

Se consideraron cinco escenarios representativos:

- **batch_jobs**: todos los procesos llegan en el tiempo 0 con ráfagas de CPU muy distintas (2, 4, 8, 16 unidades). Modela una carga tipo *batch* clásica.
- **staggered_arrivals**: los procesos llegan de forma escalonada y algunos trabajos cortos aparecen después de uno largo. Permite observar cómo SJF y RR reaccionan ante arribos tardíos de tareas cortas.
- **interactive_like**: varios procesos con ráfagas pequeñas y medianas, con llegadas cercanas en el tiempo. Es una aproximación a un sistema interactivo con múltiples tareas simultáneas.
- **random_light_load**: escenario aleatorio ligero, con cinco procesos cuyas llegadas y ráfagas se generan de manera uniforme en rangos acotados. Se utiliza una semilla fija para hacer el experimento reproducible.
- **random_heavy_load**: escenario aleatorio pesado, con diez procesos, ráfagas más grandes y llegadas dispersas en una ventana de tiempo más amplia. También se construye con una semilla fija para conservar la reproducibilidad.

Tabla 1: Resultados promedio por escenario y algoritmo.

Escenario	Algoritmo	Espera prom.	Turnaround prom.	Respuesta prom.
batch_jobs	FCFS	5.50	13.00	5.50
batch_jobs	SJF	5.50	13.00	5.50
batch_jobs	SRTF	5.50	13.00	5.50
batch_jobs	RR_q2	8.00	15.50	3.00
staggered_arrivals	FCFS	5.25	9.25	5.25
staggered_arrivals	SJF	4.25	8.25	4.25
staggered_arrivals	SRTF	2.50	6.50	0.25
staggered_arrivals	RR_q2	4.50	8.50	1.25
interactive_like	FCFS	4.75	8.25	4.75
interactive_like	SJF	4.00	7.50	4.00
interactive_like	SRTF	3.50	7.00	2.25
interactive_like	RR_q2	7.00	10.50	2.00
random_light_load	FCFS	4.40	6.80	4.40
random_light_load	SJF	3.60	6.00	3.60
random_light_load	SRTF	1.60	4.00	0.40
random_light_load	RR_q2	3.20	5.60	2.00
random_heavy_load	FCFS	28.90	36.70	28.90
random_heavy_load	SJF	22.10	29.90	22.10
random_heavy_load	SRTF	22.10	29.90	22.10
random_heavy_load	RR_q2	43.80	51.60	7.80

5.3. Análisis de resultados

En el escenario **batch_jobs**, FCFS, SJF y SRTF producen exactamente los mismos tiempos promedio (espera y turnaround de 5.50 y 13.00 unidades, respectivamente), ya que todos los procesos llegan en el tiempo 0 y el orden de las ráfagas coincide con el orden óptimo de corto a largo. En este caso ni SJF ni SRTF aportan beneficio adicional respecto a FCFS. Round Robin, con $q = 2$, empeora la espera y el turnaround (8.00 y 15.50), porque divide la ejecución de los trabajos, pero mejora el tiempo de respuesta promedio (3.00 frente a 5.50) al hacer que todos los procesos toquen la CPU relativamente pronto.

En **staggered_arrivals**, donde algunos trabajos cortos llegan después de uno largo, SJF reduce los tiempos promedio respecto a FCFS (espera de 4.25 vs. 5.25 y turnaround de 8.25 vs. 9.25) al dar prioridad a los procesos con ráfagas menores una vez que están listos. SRTF va un paso más allá: al ser expulsivo, puede interrumpir al proceso largo en ejecución cuando llegan trabajos cortos, y obtiene así los mejores valores de todas las métricas (espera de 2.50, turnaround de 6.50 y respuesta de 0.25). Round Robin alcanza un tiempo de respuesta bajo (1.25), pero sus tiempos de espera y turnaround quedan por encima de SRTF.

En el escenario **interactive_like** se observa un patrón similar: SJF mejora ligeramente a FCFS en espera y turnaround (4.00 y 7.50 frente a 4.75 y 8.25). SRTF vuelve a posicionarse como un compromiso atractivo: reduce tanto la espera como el turnaround (3.50 y 7.00) y mejora de manera significativa el tiempo de respuesta (2.25). Round Robin sacrifica de nuevo espera y turnaround (7.00 y 10.50), pero obtiene el menor tiempo de respuesta promedio (2.00), coincidiendo con su uso típico en sistemas interactivos donde la sensación de respuesta rápida es prioritaria.

Los escenarios aleatorios refuerzan estas tendencias. En **random_light_load**, SJF domina a FCFS en espera y turnaround, y SRTF mejora aún más las tres métricas (espera de 1.60, turnaround de 4.00 y respuesta de 0.40). Round Robin con $q = 2$ consigue un rendimiento intermedio entre SRTF y SJF. En **random_heavy_load**, SJF y SRTF obtienen los mismos resultados promedio, lo que indica que en esta carga particular las decisiones expulsivas de SRTF no cambian el orden efectivo de ejecución. En este caso, RR presenta nuevamente el menor tiempo de respuesta (7.80), pero a costa de tiempos de espera y turnaround muy elevados (43.80 y 51.60).

En conjunto, los experimentos muestran que SJF y, sobre todo, SRTF tienden a minimizar los tiempos promedio de espera y turnaround cuando se conoce de antemano la duración de las ráfagas, mientras que Round Robin favorece sistemáticamente el tiempo de respuesta, especialmente en cargas interactivas, a costa de penalizar los tiempos de finalización en escenarios muy cargados. FCFS sirve como línea base simple, pero suele ser dominado por SJF, SRTF y RR en la mayoría de las métricas.

6. Conclusiones

El desarrollo del simulador permitió observar de manera concreta cómo las distintas políticas de planificación de CPU afectan el desempeño del sistema bajo diferentes patrones de carga. En los escenarios diseñados, el algoritmo SJF no expulsivo tendió a minimizar el tiempo de espera y el tiempo de turnaround promedio cuando las ráfagas de CPU son conocidas, confirmando su propiedad teórica, mientras que SRTF, al permitir expulsión, logró en varios casos mejorar todavía más estas métricas al interrumpir oportunamente a los procesos largos. Sin embargo, su aplicación práctica se ve limitada por la necesidad de estimar dichas ráfagas y por el riesgo de inanición de procesos largos.

Round Robin con quantum fijo mostró un comportamiento más adecuado para cargas interactivas: aunque en algunos escenarios incrementó el tiempo de finalización de los procesos, mejoró de forma notable el tiempo de respuesta promedio, lo que se traduce en una mejor percepción de rapidez por parte del usuario. Bajo cargas muy pesadas, los experimentos evidenciaron que un quantum demasiado pequeño puede provocar tiempos de espera y turnaround elevados, lo que sugiere la importancia de elegir cuidadosamente este parámetro.

Finalmente, el proyecto ilustra la utilidad de contar con herramientas experimentales para estudiar conceptos de sistemas operativos. La combinación de una implementación modular en Python, escenarios controlados, un módulo de experimentos automatizado y un conjunto de pruebas unitarias permitió explorar, con relativamente poco código, las ventajas y desventajas de cada algoritmo de planificación. Como trabajo futuro se podría extender el simulador para incluir SRTF, diferentes valores de quantum y métricas adicionales como el número de cambios de contexto o la utilización de la CPU.

Referencias

[1] A. S. Tanenbaum y H. Bos, *Modern Operating Systems*, 4a edición. Pearson, 2015.

[2] A. Silberschatz, P. Galvin y G. Gagne, *Operating System Concepts*, 10a edición. Wiley, 2018.

[3] GeeksforGeeks. “CPU Scheduling in Operating Systems.” Disponible en: `https://www.geeksforgeeks.org/operating-systems/cpu-scheduling-in-operating-systems/`.

[4] GeeksforGeeks. “Round Robin Scheduling in Operating System.” Disponible en: `https://www.geeksforgeeks.org/operating-systems/round-robin-scheduling-in-operating-system/`.

[5] Wikipedia. “Round-robin scheduling.” Disponible en: `https://en.wikipedia.org/wiki/Round-robin_scheduling`.