UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

“Proyecto de unidad U01”

Curso: SISTEMAS OPERATIVOS I

Docente: Ing. Hugo Manuel Barraza Vizcarra

Dongo Palza,Manuel Andree (2023076842)

Tacna – Perú
2025

INDICE

INTRODUCCIÓN

El estudio de los sistemas operativos es importante porque son el fundamental que permite la relación entre el hardware de la computadora y las aplicaciones de usuario. Los sistemas operativos no solo gestionan los recursos de hardware como la CPU, la memoria y los dispositivos de entrada y de salida sino que también garantizan que las aplicaciones funcionen de una forma eficiente y, sobre todo, de una forma segura. En la práctica, la comprensión del comportamiento interno de un sistema operativo real es difícil, pues son muchos los procesos que se realizan simultáneamente, las interrupciones correspondientes de hardware y los mecanismos de sincronización. Por todo ello, es necesario construir entornos controlados (simuladores, por ejemplo) que permitan comprender y experimentar los principios fundamentales de su funcionamiento.

La finalidad del presente proyecto consiste en la creación de un simulador de sistema operativo muy básico que persigue ayudar a la comprensión en términos de cómo se realizan los aprender los conceptos teóricos que aparecen en el documento de la Unidad 01 del curso Sistemas Operativos, el simulador tendrá como eje tres componentes básicos de la gestión de procesos, la planificación de CPU y la gestión de memoria principal, lo cual proporcionará la posibilidad de que el estudiante pueda observar cómo los diferentes algoritmos de planificación afectan los tiempos de respuesta, espera y retorno de los procesos y como las políticas de asignación de memoria afectan la utilización de los recursos.

I. INFORMACIÓN GENERAL

- **Objetivo General:**

  Diseñar e implementar un simulador de sistema operativo simple que permita modelar la gestión de procesos mediante un PCB básico, aplicar algoritmos clásicos de planificación de CPU y estrategias de asignación de memoria, así como calcular métricas de desempeño e interpretar críticamente los resultados obtenidos.

  -Específicos

  1. Implementar un simulador que represente procesos con un PCB mínimo (PID, llegada, servicio, inicio, fin) y calcule métricas individuales (respuesta, espera, retorno) y globales (promedios y throughput).

  2. Incorporar y comparar los algoritmos de planificación de CPU FCFS, SPN y Round Robin, evaluando su rendimiento bajo las mismas condiciones de carga.

  3. Implementar las estrategias de asignación de memoria First-Fit y Best-Fit, observando su impacto en la utilización y fragmentación de la memoria.

  4. Permitir que el estudiante ejecute el simulador, analice las salidas generadas y desarrolle una interpretación crítica de los resultados obtenidos en cada experimento.

```
PS C:\WINDOWS\system32> cd $HOME\Documents
PS C:\Users\Manuel_Dongo\Documents> mkdir sim-unidad01


    Directorio: C:\Users\Manuel_Dongo\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----     17/09/2025  11:03 a. m.                sim-unidad01


PS C:\Users\Manuel_Dongo\Documents> cd sim-unidad01
PS C:\Users\Manuel_Dongo\Documents\sim-unidad01> python -m venv .venv
PS C:\Users\Manuel_Dongo\Documents\sim-unidad01> .\.venv\Scripts\Activate.ps1
(.venv) PS C:\Users\Manuel_Dongo\Documents\sim-unidad01>
```

En la Figura 1 se observa la creación y activación del entorno virtual en Python. Este paso garantiza un espacio aislado para instalar dependencias y ejecutar el simulador de manera controlada.

2. MARCO CONCEPTUAL

2.1 Procesos y estados

Un proceso es mucho más que un simple programa en ejecución; es una entidad dinámica que necesita constantemente recursos para avanzar en su ciclo de vida. Mientras que el programa almacenado en disco es pasivo, el proceso adquiere “vida” al cargarse en memoria y ser atendido por la CPU. Para que el sistema operativo pueda administrarlo, se utiliza una estructura llamada PCB (Process Control Block) que centraliza toda la información crítica del proceso. Allí se guardan datos como el identificador único de proceso (PID), el contador de programa que indica la siguiente instrucción a ejecutar, los registros actuales de CPU, y campos relacionados con su gestión como el tiempo de llegada o de servicio restante.

El ciclo de vida de un proceso se puede dividir en diferentes estados. En el estado nuevo, el proceso ha sido creado pero aún no está listo para ejecutar. Cuando sus condiciones lo permiten, pasa al estado listo, donde espera su turno en la cola para acceder a la CPU. Al ser seleccionado, entra en estado ejecución, y si necesita esperar un evento externo como E/S, pasa a bloqueado. Finalmente, cuando termina, entra al estado terminado. Este modelo, aunque simplificado en nuestro simulador, refleja la esencia de cómo un sistema operativo administra múltiples programas simultáneamente.

Comprender los estados es crucial porque de ellos depende la planificación de CPU. Cada transición entre estados genera decisiones de asignación de recursos. Por ejemplo, cuando hay varios procesos en la cola de listos, el planificador debe decidir cuál se ejecutará a continuación, influyendo directamente en métricas como tiempo de respuesta, tiempo de espera y tiempo de retorno. Estas métricas permiten evaluar la calidad del algoritmo utilizado y constituyen la base de los experimentos realizados en el proyecto.

2.2 Algoritmos de planificación

Los algoritmos de planificación buscan determinar cuál proceso debe ejecutarse en cada momento para aprovechar mejor la CPU. El más simple es First-Come, First-Served (FCFS), que respeta el orden de llegada de los procesos. Su ventaja radica en su simplicidad, ya que se implementa como una cola FIFO. Sin embargo, presenta el fenómeno conocido como efecto convoy: un proceso de larga duración puede retrasar a muchos procesos cortos que llegaron después, aumentando así los tiempos de espera promedio.

Un algoritmo más eficiente en promedio es Shortest Process Next (SPN), también denominado Shortest Job First (SJF). Este selecciona el proceso con menor tiempo de servicio. Teóricamente es el

algoritmo que minimiza el tiempo de espera promedio, pero tiene limitaciones importantes: exige conocer de antemano el tiempo de servicio, lo cual en sistemas reales es poco realista, y puede producir inanición si llegan continuamente procesos cortos que postergan a los largos indefinidamente.

Por último, el Round Robin (RR) está orientado a sistemas multiusuario e interactivos. En este algoritmo, la CPU se reparte en intervalos de tiempo llamados quantum. Cada proceso recibe un quantum; si no finaliza en ese tiempo, se interrumpe y regresa al final de la cola de listos. La equidad es la principal ventaja del RR, ya que todos los procesos reciben atención temprana. Sin embargo, su eficacia depende del valor del quantum: si es demasiado pequeño, genera sobrecarga por excesivos cambios de contexto; si es demasiado grande, el algoritmo termina comportándose como FCFS.

2.3 Estrategias de memoria

La memoria principal es un recurso limitado y fundamental. Su gestión determina si los procesos pueden ejecutarse o quedan en espera por falta de espacio. En este proyecto, la memoria se modela como un espacio lineal y contiguo con un tamaño configurable (ejemplo: 1 MiB). El simulador implementa dos estrategias de asignación: First-Fit y Best-Fit, ambas aplicadas sobre listas de bloques libres y ocupados.

La política First-Fit consiste en recorrer la lista de bloques libres desde el inicio y asignar el primer bloque suficientemente grande. Su gran ventaja es la rapidez, ya que evita un recorrido completo. Sin embargo, tiene como efecto secundario dejar huecos grandes hacia el inicio de la memoria, lo que con el tiempo puede provocar una fragmentación externa importante, aunque de forma moderada.

La política Best-Fit, por el contrario, intenta aprovechar al máximo la memoria en el corto plazo. Busca el bloque libre que más se ajuste al tamaño solicitado. Esto evita desperdicio inmediato, pero genera muchos fragmentos diminutos que no se podrán usar más adelante, aumentando la fragmentación externa. En la práctica, Best-Fit tiende a ser menos eficiente a largo plazo que First-Fit, aunque en casos específicos puede resultar útil para acomodar procesos de tamaños variados.

config > {} ejemplo_rr.json > ...

```json
{
  "cpu": { "algoritmo": "RR", "quantum": 4 },
  "procesos": [
    { "pid": 1, "llegada": 0, "servicio": 12 },
    { "pid": 2, "llegada": 1, "servicio": 5 },
    { "pid": 3, "llegada": 2, "servicio": 8 }
  ],
  "memoria": { "tam": 1048576, "estrategia": "first-fit" },
  "solicitudes_mem": [
    { "pid": 1, "tam": 120000 },
    { "pid": 2, "tam": 64000 }
  ]
}
```

La Figura 2 muestra un ejemplo de archivo JSON de configuración, donde se definen el algoritmo de planificación, el quantum en caso de Round Robin, la lista de procesos y las solicitudes de memoria.

3. DISEÑO DEL SIMULADOR

El simulador fue construido en Python, aprovechando su sintaxis clara y la facilidad de manejo de estructuras de datos. Para organizar el proyecto, se optó por una arquitectura modular que separa las responsabilidades de procesos, planificación y memoria. Esto permite mantener el código más limpio y facilita futuras extensiones como la inclusión de prioridades o memoria virtual.

En primer lugar, se implementó la clase PCB (Process Control Block), que guarda información como PID, tiempo de llegada, tiempo de servicio requerido, tiempo de inicio y fin de ejecución. Además, a partir de estos datos se calculan automáticamente métricas como tiempo de respuesta, espera y retorno. Esta centralización simplifica el análisis final.

En segundo lugar, se diseñaron las colas de procesos. La cola de listos es la pieza central del planificador, pues ahí esperan los procesos su turno en CPU. Su comportamiento depende del algoritmo seleccionado: en FCFS se mantiene como FIFO, en SPN se ordena dinámicamente por servicio restante, y en RR se aplica un esquema rotatorio que respeta el quantum. A su vez, se implementó una lista de bloques de memoria para asignar y liberar espacio en función de First-Fit o Best-Fit.

El núcleo del simulador es un ciclo de ticks de tiempo. En cada tick se revisa si llegan nuevos procesos, si deben asignarse a la cola de listos y si la CPU está libre. Si hay un proceso en ejecución, se actualiza su tiempo restante y se comprueba si terminó. En el caso de RR, también se descuenta el

quantum. Una vez finalizado un proceso, se liberan sus recursos de memoria y se calculan sus métricas. El ciclo continúa hasta que todos los procesos han finalizado.

```
(.venv) PS C:\Users\Manuel_Dongo\Desktop\sim-unidad01> dir -Force


    Directorio: C:\Users\Manuel_Dongo\Desktop\sim-unidad01


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d--h--        17/09/2025  11:37 a. m.                .git
d-----        17/09/2025  11:25 a. m.                config
-a----        17/09/2025  11:17 a. m.             40 .gitignore
-a----        17/09/2025  11:25 a. m.              0 GUIA_USUARIO.md
-a----        17/09/2025  11:54 a. m.           2756 README.md
-a----        17/09/2025  11:45 a. m.           8802 sim_so.py
```

La Figura 3 evidencia la organización modular del proyecto, donde se separan archivos de configuración, documentación y código fuente en diferentes carpetas.

4. METODOLOGÍA DE EXPERIMENTOS

Para evaluar el simulador de manera objetiva, se diseñó una metodología que garantiza comparaciones justas entre algoritmos y estrategias de memoria. El primer paso fue la creación de archivos JSON de entrada, que incluyen todos los procesos de un experimento con sus parámetros básicos: PID, tiempo de llegada y tiempo de servicio. Esta definición explícita de entrada asegura que las pruebas sean repetibles.

En la primera etapa de experimentación se compararon los tres algoritmos de planificación de CPU: FCFS, SPN y RR. Cada conjunto de procesos definido se ejecutó bajo las tres políticas, manteniendo iguales las condiciones iniciales. Para Round Robin, se realizaron varias ejecuciones cambiando el quantum a valores de 2, 4 y 8, con el fin de observar cómo afecta este parámetro a las métricas de rendimiento.

En la segunda etapa, se probó la gestión de memoria con First-Fit y Best-Fit. Se introdujeron solicitudes de memoria de diferentes tamaños, algunas grandes y otras pequeñas, para generar escenarios con fragmentación. De esta forma se pudo analizar la eficiencia de cada política y sus

limitaciones. Todo el proceso se documentó con capturas de pantalla y tablas de resultados para que las comparaciones sean claras.

```
(.venv) PS C:\Users\Manuel_Dongo\Desktop\sim-unidad01> python sim_so.py --config config/ejemplo_rr.json
PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
------------------------------------------------------------------------------
  1 |       0 |       12 |      0 |  25 |         0 |     13 |      25
  2 |       1 |        5 |      4 |  17 |         3 |     11 |      16
  3 |       2 |        8 |      8 |  21 |         6 |     11 |      19
```

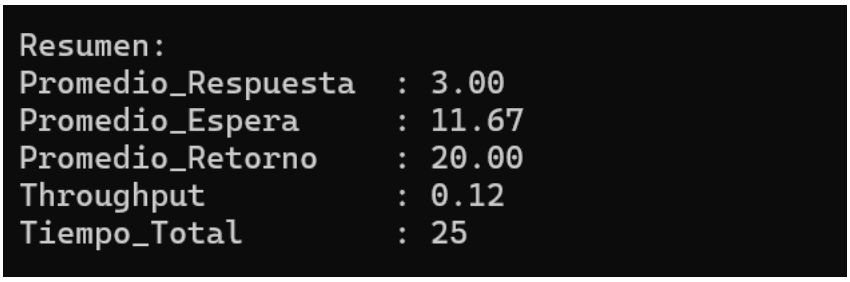
La Figura 4 corresponde a la ejecución del simulador con un archivo JSON de configuración. Se observa la tabla con los resultados de cada proceso y sus métricas de planificación.

5. RESULTADOS

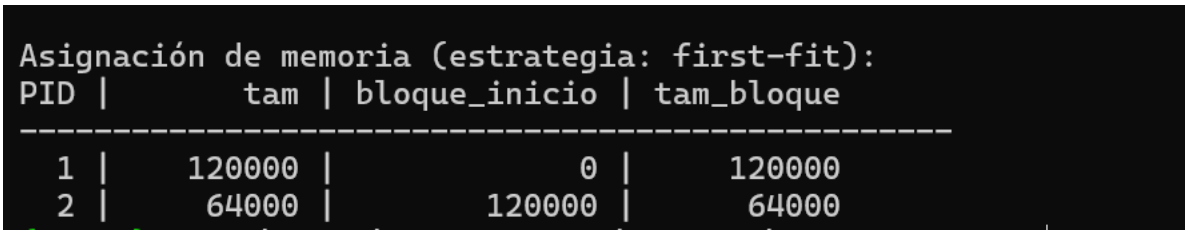
En este capítulo se presentan y analizan los resultados obtenidos tras la ejecución del simulador bajo distintos escenarios. El objetivo no es únicamente mostrar números, sino también comprender qué significan y cómo cada algoritmo o estrategia impacta en el rendimiento del sistema. Para ello, se utilizaron configuraciones con diferentes algoritmos de planificación (FCFS, SPN y RR con quantum configurable) y se evaluaron métricas como tiempo de respuesta, tiempo de espera, tiempo de retorno, throughput y asignación de memoria.

Es importante señalar que las métricas calculadas por el simulador permiten medir de manera objetiva el comportamiento de cada política. Mientras que el tiempo de respuesta refleja la rapidez con la que un proceso recibe atención inicial, el tiempo de espera indica cuánto tiempo pasó bloqueado en la cola sin ejecutar, y el tiempo de retorno sintetiza la experiencia total de un proceso dentro del sistema. El throughput en cambio refleja la productividad global: cuántos procesos terminan por unidad de tiempo.

En la siguiente sección se detallan los resultados paso a paso para cada algoritmo utilizando un conjunto base de cuatro procesos. Esta metodología controlada permite identificar claramente las diferencias. Posteriormente, se complementan los resultados con la asignación de memoria, que muestra cómo las políticas de First-Fit y Best-Fit responden a solicitudes concretas.



La Figura 5 muestra el resumen global de métricas calculadas por el simulador, donde se incluyen los promedios de respuesta, espera y retorno, así como el throughput y el tiempo total de ejecución.



La Figura 6 presenta la asignación de memoria generada por el simulador. Se observa cómo los procesos ocupan bloques contiguos en memoria según la estrategia elegida (First-Fit o Best-Fit).

5.1 Resultados con FCFS

El algoritmo First-Come, First-Served atiende a los procesos en orden de llegada. Con los datos de entrada P1=(0,6), P2=(1,3), P3=(2,4), P4=(3,2), la secuencia fue: P1 → P2 → P3 → P4. El diagrama de Gantt quedó de la siguiente manera:
P1 [0–6] → P2 [6–9] → P3 [9–13] → P4 [13–15].

Los cálculos muestran que P1, al llegar primero y ser atendido de inmediato, tiene métricas excelentes: tiempo de respuesta = 0, espera = 0, retorno = 6. Pero P2, P3 y P4 deben esperar la finalización de P1, elevando significativamente sus tiempos de respuesta y retorno. El promedio de respuesta resultó 5.5 u.t., mientras que el promedio de retorno alcanzó 9.25 u.t.

Esto refleja la gran desventaja de FCFS: cuando hay procesos largos, los cortos que llegan después deben esperar, generándose el fenómeno conocido como convoy effect. En este efecto, un “camión” (proceso largo) arrastra detrás a procesos pequeños que podrían haber terminado rápidamente, disminuyendo la eficiencia global percibida.

```
(.venv) PS C:\Users\Manuel_Dongo\Desktop\sim-unidad01> python sim_so.py --config config/ejemplo_fcfs.json
PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
------------------------------------------------------------------------------------
  1 |       0 |       12 |      0 |  12 |         0 |      0 |      12
  2 |       1 |        5 |     12 |  17 |        11 |     11 |      16
  3 |       2 |        8 |     17 |  25 |        15 |     15 |      23

Resumen:
Promedio_Respuesta  : 8.67
Promedio_Espera     : 8.67
Promedio_Retorno    : 17.00
Throughput          : 0.12
Tiempo_Total        : 25

Asignación de memoria (estrategia: best-fit):
PID |       tam | bloque_inicio | tam_bloque
-----------------------------------------------
  3 |    700000 |             0 |     700000
  1 |    200000 |        700000 |     200000
```

La Figura 7 muestra la ejecución del simulador con el algoritmo First-Come, First-Served (FCFS).

5.2 Resultados con SPN

En el algoritmo Shortest Process Next, el planificador elige siempre al proceso con menor tiempo de servicio. La secuencia en este caso fue: P1 [0–6] → P4 [6–8] → P2 [8–11] → P3 [11–15]. El orden cambió radicalmente respecto a FCFS, ya que después de P1, el sistema seleccionó a P4 (el más corto de los listos).

Los resultados muestran que el promedio de espera bajó a 4.75 u.t., y el promedio de retorno a 8.5 u.t., mejores que en FCFS. Sin embargo, esta mejora se da a costa de la equidad: P3, el proceso más largo de los que llegaron tarde, se vio postergado hasta el final, acumulando un tiempo de espera de 9 u.t. Este tipo de comportamiento puede generar inanición en sistemas donde llegan constantemente procesos cortos, dejando indefinidamente bloqueados a los largos.

Un aspecto interesante es que SPN requiere conocer de antemano el tiempo de servicio, algo que no siempre es posible en sistemas reales. Esto significa que, aunque los resultados promedio son mejores, el algoritmo es difícil de implementar en la práctica sin mecanismos predictivos.

```
(.venv) PS C:\Users\Manuel_Dongo\Desktop\sim-unidad01> python sim_so.py --config config/ejemplo_spn.json
PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
------------------------------------------------------------------------------
  1 |       0 |       12 |      0 |  12 |         0 |      0 |      12
  2 |       1 |        5 |     12 |  17 |        11 |     11 |      16
  3 |       2 |        8 |     17 |  25 |        15 |     15 |      23

Resumen:
Promedio_Respuesta  : 8.67
Promedio_Espera     : 8.67
Promedio_Retorno    : 17.00
Throughput          : 0.12
Tiempo_Total        : 25

Asignación de memoria (estrategia: best-fit):
PID |       tam | bloque_inicio | tam_bloque
-------------------------------------------------
  1 |    120000 |             0 |     120000
  2 |     64000 |        120000 |      64000
```

La Figura 8 corresponde a la ejecución del simulador con el algoritmo Shortest Process Next (SPN).

5.3 Resultados con Round Robin (Q=4)

El algoritmo Round Robin divide la CPU en quanta de tiempo fijo. Con Q=4, la secuencia fue: P1 [0–4] → P2 [4–7] → P3 [7–11] → P4 [11–13] → P1 [13–15]. Aquí todos los procesos recibieron CPU temprano, reduciendo la espera inicial y mejorando la equidad.

El promedio de respuesta fue de 4.0 u.t., inferior a FCFS y cercano a SPN. Sin embargo, el promedio de retorno fue de 10.0 u.t., más alto que en SPN, porque dividir los procesos largos en múltiples tramos genera pausas adicionales entre quanta. El throughput fue similar en los tres algoritmos (0.27 procesos/u.t.) porque el tiempo total de servicio era el mismo (15 u.t.) y no se consideró costo de cambio de contexto.

Estos resultados ilustran la ventaja de RR: aunque no minimiza los promedios como SPN, ofrece una experiencia más justa para todos los procesos, evitando que uno largo bloquee a los demás. Su desempeño depende críticamente de la elección del quantum: con Q demasiado grande se comporta como FCFS, y con Q demasiado pequeño genera sobrecarga.

```
(.venv) PS C:\Users\Manuel_Dongo\Desktop\sim-unidad01> python sim_so.py --config config/ejemplo_rr.json
PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
------------------------------------------------------------------------------
  1 |       0 |       12 |      0 |  25 |         0 |     13 |      25
  2 |       1 |        5 |      4 |  17 |         3 |     11 |      16
  3 |       2 |        8 |      8 |  21 |         6 |     11 |      19

Resumen:
Promedio_Respuesta  : 3.00
Promedio_Espera     : 11.67
Promedio_Retorno    : 20.00
Throughput          : 0.12
Tiempo_Total        : 25

Asignación de memoria (estrategia: first-fit):
PID |       tam | bloque_inicio | tam_bloque
-----------------------------------------------
  1 |    120000 |             0 |     120000
  2 |     64000 |        120000 |      64000
```

La Figura 9 presenta la ejecución con el algoritmo Round Robin (RR) utilizando un quantum de 4 unidades de tiempo.

5.4 Asignación de Memoria

La salida del simulador también muestra cómo se asignaron bloques de memoria según la estrategia seleccionada. Por ejemplo, con solicitudes de 120000 y 64000 unidades, la estrategia First-Fit asignó los primeros bloques contiguos disponibles, mientras que Best-Fit buscó bloques que se ajustaran mejor.

Los resultados evidencian que First-Fit suele ser más rápido al recorrer la lista, pero puede dejar huecos grandes hacia el inicio de la memoria. Best-Fit, en cambio, tiende a producir fragmentos pequeños e inutilizables. En escenarios de alta carga de memoria, estas diferencias se vuelven críticas, ya que pueden determinar si un proceso puede ser admitido o no.

```
Asignación de memoria (estrategia: best-fit):
PID |       tam | bloque_inicio | tam_bloque
-----------------------------------------------
  3 |    700000 |             0 |     700000
  1 |    200000 |        700000 |     200000
```

La Figura 10 presenta la salida de la asignación de memoria simulada.

En este caso, se observa cómo los procesos se ubican en bloques contiguos de la memoria principal según la estrategia seleccionada. Con First-Fit, el asignador toma siempre el primer bloque libre suficientemente grande, lo cual es más rápido pero puede dejar huecos grandes en la parte inicial de la memoria. Con Best-Fit, en cambio, se selecciona el bloque más ajustado posible al tamaño solicitado, lo que reduce el desperdicio inmediato de memoria, aunque conlleva la generación de

fragmentos muy pequeños que no podrán reutilizarse. Estos resultados evidencian las diferencias prácticas entre ambas estrategias y permiten evaluar su impacto en la fragmentación y eficiencia global del sistema.

CONCLUSIONES

- La implementación del simulador permitió comprender de manera práctica los conceptos fundamentales de gestión de procesos, planificación de CPU y asignación de memoria, acercando la teoría de sistemas operativos a la experimentación directa.

- Se comprobó que el algoritmo FCFS, aunque sencillo y justo en cuanto al orden de llegada, presenta el efecto convoy cuando existen procesos largos al inicio de la cola, lo que eleva los tiempos de espera promedio.

- El algoritmo SPN (SJF) mostró mejores resultados en los promedios de espera y retorno, confirmando su eficiencia teórica, pero evidenció el riesgo de inanición para procesos largos, lo que lo hace poco viable en sistemas interactivos reales.

- La política Round Robin equilibró mejor la atención a todos los procesos, reduciendo los tiempos de respuesta inicial. Sin embargo, su rendimiento depende directamente del valor del quantum, lo que demuestra la importancia de calibrarlo adecuadamente en sistemas reales.

- En la gestión de memoria, First-Fit resultó más rápido y sencillo de aplicar, mientras que Best-Fit, aunque en teoría más eficiente, generó fragmentación externa significativa. Este comportamiento refleja que la elección de la política depende de los objetivos: velocidad de asignación o aprovechamiento de memoria.

- El proyecto fortaleció la capacidad de análisis crítico de los algoritmos y políticas, destacando que no existe una solución perfecta, sino que cada técnica debe aplicarse según el contexto y los requerimientos del sistema operativo.

BIBLIOGRAFÍA

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
- Stallings, W. (2018). Operating Systems: Internals and Design Principles (9th ed.). Pearson.
- Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson.
- Bovet, D. P., & Cesati, M. (2005). Understanding the Linux Kernel (3rd ed.). O'Reilly Media.
- Powershell Documentation. Microsoft Docs. Disponible en: https://docs.microsoft.com/powershell/
- Python Documentation. Python Software Foundation. Disponible en: https://docs.python.org/3/