UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS

IMPLEMENTACIÓN DE UN SIMULADOR DE SISTEMA OPERATIVO

ASIGNATURA:
Sistemas operativos

SECCIÓN:
Grupo A

Estudiante:
Maykol David Espinoza Kquerare 2025-11902c
Luz Marina Flores Carbajal 2025-11901c

FECHA:
Tacna, diciembre del 2025

ÍNDICE

I. Introducción

El presente informe documenta el desarrollo, funcionamiento y análisis de un simulador de sistema operativo, diseñado con el propósito de representar de manera didáctica y controlada el comportamiento interno de un sistema operativo real. La simulación se centra en tres áreas fundamentales: gestión de procesos, planificación de la CPU y gestión de memoria. Estos componentes conforman el núcleo operativo de todo sistema moderno y determinan en gran medida la eficiencia, estabilidad y capacidad de respuesta del entorno computacional.

La gestión de procesos permite crear, controlar y finalizar tareas, garantizando que cada proceso disponga de los recursos necesarios para su ejecución. La planificación de la CPU, por su parte, define el orden y la prioridad con la que los procesos acceden al procesador, afectando directamente indicadores críticos como el tiempo de espera, el tiempo de respuesta, la equidad entre procesos y el rendimiento global del sistema. Finalmente, la gestión de memoria regula la asignación y liberación del espacio de memoria física, optimizando su uso y permitiendo la coexistencia de múltiples programas.

Debido a la complejidad intrínseca de estos mecanismos y a la dificultad de observarlos directamente en sistemas operativos reales, los simuladores se vuelven herramientas esenciales en el ámbito académico y en el análisis experimental. Permiten recrear condiciones específicas, probar algoritmos de administración de recursos y comprender el impacto de las decisiones internas sin comprometer entornos productivos.

En este contexto, el simulador desarrollado constituye una plataforma que facilita la visualización, evaluación y comparación del comportamiento de algoritmos fundamentales de la administración de procesos, la planificación de la CPU y la gestión de memoria, aportando una experiencia práctica que complementa la teoría del funcionamiento de los sistemas operativos.

II. Problema a abordar

Los sistemas operativos deben decidir continuamente qué proceso recibe la CPU, durante cuánto tiempo y qué parte de la memoria principal puede utilizar para cada proceso esto se debe principalmente a que dentro de la computadora.

- Los recursos del sistema operativo son limitados.
- Los procesos tienen necesidades diferentes.
- Las políticas de asignación influyen directamente en el rendimiento global.

El problema que presentamos en este informe consiste en evaluar cómo los diferentes algoritmos de planificación y técnicas de asignación de memoria influyen en la eficiencia, equidad y rendimiento de un sistema bajo condiciones controladas.

El problema a abordar es importante debido a que:

- Permite comprender por qué ciertos algoritmos son usados en sistemas reales.
- Ayuda a evaluar ventajas y desventajas de cada algoritmo.
- Facilita la selección de algoritmos según la carga de trabajo.
- Permite medir cuantitativamente el impacto de decisiones de diseño del sistema operativo.

III. Objetivo

Cuando iniciamos este proyecto partimos con el objetivo de realizar un simulador de sistema operativo que permita analizar, comprender y avaluar el comportamiento de diferentes tipos de algoritmos con respecto a su eficiencia dentro de la CPU, a su vez que realizamos técnicas de asignación a memoria, manteniendo todo esto bajo un entorno controlado el cual permita analizar los resultados exactos a los que se puede observar en un entorno completamente normal, a través del cálculo de métricas de desempeño, organización y estructura de resultados.

IV. Alcance y supuestos

1. Entorno monoprocesador

Se simula una única CPU, lo cual simplifica el estudio y permite enfocarse en los algoritmos, sin considerar concurrencia multinúcleo debido a que esto podría generar una variación en los resultados que se llegaran a obtener.

2. Simulación discreta por tiempo

Cada paso de tiempo representa un milisegundo, lo cual en programación es una medida de tiempo adecuada debido a que estamos trabajando con proceso que trabajan con bits, de esta manera puede modelarse las siguientes medidas:

- Llegadas en tiempo exacto
- Expropiación
- Finalización
- Cambios en memoria

En un sistema operativo real, cambiar de un proceso a otro implica guardar registros, estado, entre muchas otras acciones, lo que requiere un costo que afecta el rendimiento de la computadora.

3. Asignación contigua

La memoria solo puede ser asignada de manera lineal en el presente trabajo no se realiza procesos de paginación ni segmentación esto debido a que además de ser métodos demasiado avanzados y tediosos de implementar el propósito principal a estudiar en el documento es realizar la:

- Fragmentación externa
- Estrategias de búsqueda
- efecto del tamaño del proceso

V. Marco conceptual

1. Proceso y estados

Silberschatz, Galvin P. y Gagne G. (2018) mencionan que:

> "Un proceso es la entidad que requiere CPU y memoria para ejecutar un programa; su información esencial se agrupa en un Process Control Block (PCB) que suele incluir al menos: PID, tiempos de llegada y servicio, estado, punteros de memoria y métricas de ejecución como pueden ser el inicio, fin, respuesta, espera y retorno. El PCB es la estructura fundamental que permiten a un sistema operativo gestionar y conmutar procesos."

Mientras que Tanenbaum A. y Bos H. (2015) expresan que:

> "El modelado de los estados facilita la abstracción de la multiprogramación y el análisis de métricas temporales, como respuesta y turnaround"

Los estados simulados en este trabajo son:

- Nuevo (New): Es el proceso que acaba de ser leído desde el archivo JSON que todavía no participa en el sistema.
- Listo (Ready): El proceso está esperando en la cola de listos a recibir CPU. En este estado compite con los demás procesos dependiendo del algoritmo activo.
- Ejecución (Running): El proceso se encuentra usando la CPU. Su tiempo restante disminuye cada ciclo.
- Terminado (Exit): El proceso finaliza y libera su memoria. Sus métricas quedan registradas.

2. Planificación de CPU

Stallings W. (2018) manifiesta lo siguiente:

> "Los algoritmos de planificación son aquellos encargados de definir el orden de despacho, dentro de estos algoritmos los más usados, sencillos y eficientes que podemos encontrar son el FCFS que significa First Come First-Served, siendo un algoritmo no expropiativo, el SPN/SJF o Shortest Process Next / Shortest Job First, perteneciendo al grupo de algoritmos no expropiativo y por último, pero no menos importante el algoritmo Round Robin o RR perteneciente al grupo de algoritmos expropiativos. Round Robin es el encargado de asignar a cada proceso una porción fija de tiempo el cual se conoce como quantum y lo cicla en una cola circular, la vetaja que tiene sobre las demás es que prioriza

la equidad y ausencia de inanición, pero su desempeño depende fuertemente del quantum elegido para su ejecución siendo trade-off entre interactividad y costo por cambios de contexto.

- Expropiación (Preemption): Es la capacidad del sistema para detener un proceso antes de que termine su servicio en el caso del algoritmo de Round Robin se utiliza expropiación según el quantum que se elija.
- Quantum: Es el lapso máximo que un proceso puede ejecutarse antes de ser reencolado, un quantum muy pequeño simula sistemas interactivos o mejor conocido como las terminales mientras que un que un quantum grande simula el procesamiento por lotes."

3. Gestión de memoria

El simulador que presentamos en este entregable usa el particionamiento dinámico, similar a sistemas antiguos o embebidos, esto debido a la forma en que maneja la repartición de memoria el cual se trata principalmente de asignar memoria a los procesos según el tamaño que estos mismo tengan evitando problemas como los frames pequeños que podrían asignarse hacia un proceso grande cosa que se hace en una memoria de particionamiento fijo; sin embargo, los problemas que encontramos en ente tipo de particionamiento son los siguientes:

Los autores de Tanenbaum A. y Bos H. (2015) aplican lo siguiente:

- "Fragmentación externa
  Ocurre cuando el espacio total libre es suficiente, pero está dividido en partes pequeñas que no pueden alojar un proceso completo, lo que hace imposible que el proceso ingrese a esos espacios quedando inutilizables, siento este uno de los problemas más estudiados en asignación contigua siendo resuelto por el siguiente método:
- Coalescing o compactación
  Esto se realiza a medida que los procesos terminan, lo que pasa es que se libera el bloque que dicho proceso ocupada y lo que sucede es que, si los bloques vecinos a este también están libres, el sistema los une en uno mayor, lo que trae los siguientes beneficios:
    - Reduce fragmentación externa
    - Permite alojar procesos grandes
    - Mejora la eficiencia de First-Fit y Best-Fit"

4. Métricas clave.

El autor Silberschatz et al (2018) menciona que la forma de calcular las métricas clave de cualquier proceso bajo cualquier algoritmo que se encuentre es la siguiente:

“Por proceso:

- Respuesta = inicio − llegada
- Espera = fin − llegada − servicio
- Retorno (turnaround) = fin − llegada

Lo que se muestra anteriormente son los promedios de las métricas y throughput realizando el siguiente calculo el cual es “procesos completados” / tiempo total. Estas métricas son las que permiten comparar los diferentes algoritmos en cargas distintas.”

5. Formato de archivo JSON

Son métricas de entrada para el simulador, además se utilizará para la parte de resultados.

```
{
  "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} ]
}
```

VI. Diseño del simulador

1. Estructuras de datos

PCB (Process Control Block): El PCB es la estructura más importante, ya que almacena:

- Identificador (PID)
- Tiempos los cuales se mientras como llegada, servicio total, restante
- Estado actual
- Dirección de memoria asignada
- Tamaño de memoria requerida
- Métricas resultantes

Es un modelo simplificado de los PCB reales utilizados por sistemas operativos.

- NodoProceso: Permite construir colas enlazadas para las políticas.
- BloqueMemoria: Modelo de lista enlazada con campos:
  - inicio
  - tamaño

- id_proceso (−1 si libre)

Permite representar la memoria física como bloques dinámicos.

2. Módulos de colas

- Encolar / desencolar (FCFS y Round Robin)
- Representan colas FIFO tradicionales, útiles para:
  - FCFS (orden de llegada)
  - Round Robin (cola circular)
- Insertar_spn

Inserta el proceso en orden según tiempo restante. Y lo que hace es que este comportamiento es emular el ordenamiento dinámico propio de SPN.

3. Manejo de llegada y servicio

El simulador se ejecuta paso a paso:

- Paso 1: Llegada de procesos
  Cuando tiempo_actual coincide con la llegada de un proceso, este se mueve a la cola de listos. Esto permite modelar concurrencia temporal realista.
- Paso 2: Despacho
  Si la CPU está libre, se elige un proceso según la política:
  - FCFS elige el proceso que se encuentra primero en la cola
  - SPN elige el proceso con el menor tiempo restante
  - Round Robin elige el siguiente proceso en la cola circular
- Paso 3: Ejecución
  - Ejemplo de ejecución del algoritmo FCFS
    P1: llegada 0, servicio 10, significa que se despacha primero.
    - Inicio = 0
    - Fin = 0 + 10 = 10
    - Respuesta = inicio − llegada = 0 − 0 = 0
    - Espera = fin − llegada − servicio = 10 − 0 − 10 = 0
    - Retorno = fin − llegada = 10 − 0 = 10

    P2: llegada 1, servicio 10, significa que queda listo hasta que P1 termine.
    - Inicio = 10
    - Fin = 10 + 10 = 20
    - Respuesta = inicio − llegada = 10 − 1 = 9
    - Espera = fin − llegada − servicio = 20 − 1 − 10 = 9
    - Retorno = fin − llegada = 20 − 1 = 19
  - Implementación de Round Robin

El algoritmo Round Robin permite agrega complejidad a la simulación debido a que requiere las siguientes métricas que son:

- Necesidad de rastrear el quantum
- Expropiar procesos innecesarios
- Reencolar los procesos
- Manejar cambios frecuentes durante la ejecución

Esto se puede observar en la variable local de quantum_restante_actual la cual se encuentra dentro del archivo simulador.c dentro de la función inciar_simulacion dentro del código adjuntado al este archivo, lo que se realiza es reiniciar esta variable cada vez que un proceso entra en la CPU y cuando llega a cero se realiza una evaluación y se elige una de la de las siguientes opciones:

- Si el proceso aún no termina, el proceso vuelve al final de la cola
- Si el proceso termina, se libera el CPU y memoria

Lo que realiza Round Robin en este caso es equilibra equidad y rendimiento, especialmente en procesos con interactividad simulada.

VII. Metodología de experimentos

1. Casos de prueba

Se eligió un conjunto de 10 ejemplos repartidos en diversos archivos JSON los cuales son ejemplos representativos de procesos, todos diferentes y ninguno igual teniendo las siguientes características:

- Diferentes tiempos de llegada
- Diferentes tiempos de servicio
- Diferentes tamaños de memoria

Lo que nos permite observar de manera más amplia, elevada y completa:

- Cómo el algoritmo SPN favorece procesos cortos
- Cómo el algoritmo FCFS puede crear convoy
- Cómo Round Robin depende del quantum
- Cómo los algoritmos de First-Fit y Best-Fit asignan memoria de forma distinta

2. Parámetros de rendimiento

- Tiempo de respuesta
  Esto es importante para diversos sistemas interactivos debido a que si se muestra un tiempo de respuesta bajo significa que el sistema reacciona rápido y es efectivo y eficiente.
- Tiempo de espera

Se interpreta como "tiempo de CPU desperdiciado", y mientras más alto más alto sea este menos eficiente es el algoritmo permitiéndonos ver de qué manera los algoritmos reaccionan a diversos tipos de entradas.

- Tiempo de retorno
  Se observa la eficiencia total desde que el proceso llega hasta que termina, es clave para sistemas por lotes o sistemas batch ya que podemos ver cómo se comportó el algoritmo durante todo el procedimiento pudiendo observar sus tiempos más altos y bajos y como estos se desenvuelven.

3. **Repetibilidad**

Al estar basado los algoritmos de entra en un archivo JSON tienen las siguientes ventajas las cuales son que:

- Puede repetirse infinitamente con los mismos resultados.
- Permite modificar parámetros aislados y observar efectos directos.
- Facilita comparaciones justas entre algoritmos.

VIII. Resultados

La tabla obtenida con Round Rodin (q=4 ms) muestra:

- P1 inicia inmediatamente porque llega primero.
- P2, aunque llega después llegar a obtener CPU en el segundo turno gracias a Round Robin.
- P3, aunque tiene un servicio mayor que P2, se beneficia de la rotación justa.

Esto demuestra:

- Round Robin evita inanición
- Round Robin reduce el efecto convoy
- Round Robin equilibra espera entre procesos

Las métricas promedio reflejan un compromiso razonable entre rapidez y equidad.

```
MEMORIA: PID 1 asignado en 0x0 (Tam: 120000). Bloque elegido: 120000
MEMORIA: PID 2 asignado en 0x1D4C0 (Tam: 64000). Bloque elegido: 64000

INICIANDO SIMULACIÓN (RR)

None
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
```

XI. Discusión y análisis de Trade-offs

1. Efecto convoy

FCFS es simple, pero sufre un problema crítico:

- Si un proceso largo llega primero, obliga a todos los demás a esperar.
- Este efecto se vuelve grave cuando:
    - Existe la heterogeneidad entre los tiempos de servicio
    - Los procesos cortos son interactivos
    - Hay múltiples llegadas cercanas

2. Falta de conocimiento perfecto en SPN

SPN o mejor conocido como el SJF no expropiativo, requiere saber el tiempo total de servicio. En sistemas reales:

- Esto no es posible
- Se suele estimar por promedios exponenciales
- La estimación puede fallar lo cual traería como consecuencia la penalización a procesos largos

En el simulador presentado, el algoritmo SPN representa el caso ideal.

3. Quantum de Round Robin

El quantum dentro del algoritmo de Round Robin determina:

- Cuánta CPU recibe un proceso de una sola vez
- Cuántas veces el sistema debe cambiar de contexto
- Qué tan fluida es la rotación entre procesos

Un quantum demasiado grande hace que el algoritmo Round Robin se convierta en un algoritmo FCFS y si no hay alternancia los procesos cortos terminan esperando mucho.

En caso de que el quantum es demasiado pequeño se hacen demasiadas expropiaciones en, un sistema operativo real esto sería muy costoso por lo que se trabaja con un quantum intermedio como de 4 ms lo que le da un equilibrio razonable y a su vez hace que al algoritmo tenga una buena alternancia a la hora de distribuir procesos sin tener que llegar a una excesiva penalización y sin sobrecarga injustificada

X. Conclusiones y recomendaciones

En síntesis, el simulador demuestra los siguientes resultados:

- El algoritmo Round Robin es adecuado para cargas mixtas donde se prioriza equidad.

- El algoritmo de SPN minimiza tiempos de espera lo que es ideal para cargas con trabajos pequeños.
- El algoritmo FCFS es simple pero ineficiente ante procesos de tamaño diverso.
- El uso de Coalescing es esencial para evitar degradación en memoria contigua.

Lo que en conjunto significa que el simulador reproduce comportamientos coherentes con la teoría clásica de sistemas operativos.

En cuanto a las recomendaciones, este algoritmo sería muy eficaz para ayudar y permitir estudiar el efecto real del quantum pequeño por lo cual sería muy interesante poder implementar el algoritmo SRTF o un topo de algoritmo SJF expropiativo. Además de que provee una comparación más rica entre algoritmos preemptivos, ya que permite modelar procesos con ráfagas de E/S.

Se puede llegar a calibrar mejor la conducta real de sistemas multitarea para poder incluir diversos graficos como pueden ser:

- línea de tiempo de CPU
- ocupación de memoria
- comparación visual de métricas

Una recomendación importante seria extender el archivo JSON para permitir cargas más complejas para poder observar de manera detallada los procesos con varias ráfagas CPU E/S.

XI. Referencias bibliográficas

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson.
- Stallings, W. (2018). Operating Systems: Internals and Design Principles (9th ed.). Pearson.
- Shore, J. E. (1975). On the external storage fragmentation produced by first-fit and best-fit allocation strategies. Communications of the ACM, 18(1), 54–58.