Implementación de un Simulador de Sistema Operativo Simplificado con Planificación de CPU, E/S y Administración de Memoria

1er Alvaro Raúl Quispe Condori
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
aquispecondo@unsa.edu.pe

2do Christian Raul Mestas Zegarra
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
cmestasz@unsa.edu.pe

3er Luis Gustavo Sequeiros Condori
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
lsequeiros@unsa.edu.pe

4to Yenaro Joel Noa Camino
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
ynoa@unsa.edu.pe

***Resumen*—Este trabajo presenta un simulador de sistema operativo que integra planificación de unidad central de procesamiento (CPU), gestión de entrada/salida (E/S) y administración de memoria virtual mediante paginación. El sistema está implementado en C++17 con ejecución concurrente, donde cada proceso se ejecuta en un hilo independiente. Se incluyen cuatro algoritmos de planificación de CPU, cuatro algoritmos de reemplazo de páginas y dos algoritmos de planificación de E/S. Un módulo complementario en Python genera visualizaciones para el análisis del comportamiento del sistema. El diseño modular facilita la extensión de componentes y permite evaluar combinaciones de algoritmos bajo diferentes cargas de trabajo.**

***Index Terms*—planificación de procesos, memoria virtual, paginación, simulación de sistemas operativos, algoritmos de reemplazo**

I. Introducción

Referencia al azar [1].

Los sistemas operativos son responsables de gestionar los recursos de hardware y proporcionar servicios a las aplicaciones que se ejecutan sobre ellos. Entre las funciones principales de un sistema operativo se encuentran la planificación de procesos para el uso del procesador, la administración de memoria para la asignación de espacio a los programas, y la coordinación de operaciones de E/S con dispositivos periféricos. La interacción entre estos tres subsistemas determina en gran medida el rendimiento del sistema.

El desarrollo de simuladores permite estudiar el comportamiento de estos mecanismos sin requerir acceso a hardware dedicado. Un simulador puede ejecutar diferentes configuraciones de algoritmos y cargas de trabajo, facilitando la comparación de estrategias de planificación y gestión de recursos. Esta capacidad resulta valiosa tanto para fines educativos como para la investigación de nuevas políticas de administración.

Este trabajo presenta un simulador que reproduce el comportamiento de los tres subsistemas mencionados. El componente de planificación de CPU implementa los algoritmos First Come First Served (FCFS), Shortest Job First (SJF), Round Robin y planificación por prioridad. El administrador de memoria utiliza paginación con algoritmos de reemplazo First In First Out (FIFO), Least Recently Used (LRU), Not Recently Used (NRU) y el algoritmo óptimo. El planificador de E/S soporta FCFS y Round Robin. Cada proceso se ejecuta en un hilo independiente, lo que replica el comportamiento concurrente de un sistema operativo.

El documento se organiza de la siguiente manera: la Sección II describe la metodología empleada, incluyendo la arquitectura del sistema, los detalles de implementación y los algoritmos utilizados; la Sección III presenta los resultados experimentales; y la Sección IV expone las conclusiones del trabajo.

II. Metodología

Esta sección describe el diseño del simulador, los detalles de implementación de cada módulo y los algoritmos empleados para planificación y reemplazo de páginas.

II-A. Arquitectura del sistema

El sistema se compone de dos módulos principales: un simulador desarrollado en C++17 y un visualizador implementado en Python. El simulador ejecuta la lógica del sistema operativo, mientras que el visualizador procesa los datos generados para producir gráficos que facilitan el análisis del comportamiento.

El simulador se subdivide en cinco componentes. El módulo Core contiene las clases fundamentales que representan procesos y ráfagas de ejecución. El módulo CPU implementa el planificador central y los algoritmos de selección de procesos. El módulo IO gestiona los dispositivos de E/S y sus planificadores. El módulo Memory administra los marcos físicos y los algoritmos de reemplazo de páginas. El módulo Metrics recopila eventos durante la simulación para su posterior análisis.

Figura 1. Arquitectura general del sistema, mostrando la relación entre el simulador en C++ y el visualizador en Python.

El visualizador procesa archivos en formato JavaScript Object Notation Lines (JSONL) generados por el simulador. Cada generador especializado produce un tipo de gráfico: diagramas de Gantt para CPU y E/S, evolución temporal de las colas de procesos, uso de memoria, estado de las tablas de páginas, asignación de marcos físicos, cambios de contexto y distribución de estados.

II-B. Modelo de procesos

Un proceso en el simulador se define mediante su identificador, tiempo de llegada, secuencia de ráfagas, prioridad y número de páginas requeridas. La secuencia de ráfagas alterna entre períodos de uso de CPU y operaciones de E/S. Cada proceso atraviesa seis estados durante su ciclo de vida: nuevo, listo, en espera de memoria, en ejecución, bloqueado por E/S y terminado.

Figura 2. Diagrama de transición de estados de un proceso en el simulador.

Cuando un proceso llega al sistema, se encuentra en estado nuevo. El planificador lo mueve a la cola de listos cuando corresponde según su tiempo de llegada. Al ser seleccionado para ejecución, el administrador de memoria verifica que sus páginas estén cargadas. Si alguna página no está presente, el proceso pasa a estado de espera de memoria hasta que el fallo se resuelva. Una vez cargadas las páginas, el proceso entra en ejecución y consume su ráfaga de CPU. Si la ráfaga siguiente es de E/S, el proceso se bloquea hasta que la operación complete. Al finalizar todas sus ráfagas, el proceso termina.

II-C. Flujo de ejecución

El simulador avanza mediante un reloj virtual discreto. En cada unidad de tiempo, denominada tick, se ejecuta una secuencia de operaciones coordinadas. Primero se verifican las llegadas de nuevos procesos según los tiempos definidos en el archivo de entrada. Luego el planificador de CPU selecciona el siguiente proceso a ejecutar de acuerdo con el algoritmo configurado. Si el proceso seleccionado requiere páginas que no están en memoria, se encola una solicitud de carga y el proceso se bloquea.

El administrador de memoria resuelve un fallo de página por tick. Cuando todas las páginas de un proceso están cargadas, este puede continuar su ejecución. Durante la ejecución, el proceso consume una unidad de tiempo de su ráfaga actual. Si completa una ráfaga de CPU y la siguiente es de E/S, se envía una solicitud al planificador de E/S y el proceso se bloquea. El planificador de E/S atiende las solicitudes según su algoritmo configurado. Al completar una operación de E/S, el proceso regresa a la cola de listos.

Figura 3. Flujo de ejecución de un tick en el simulador, mostrando la interacción entre los módulos de CPU, memoria y E/S.

II-D. Sincronización concurrente

Cada proceso se ejecuta en un hilo independiente para replicar el comportamiento concurrente de un sistema operativo. Esta decisión de diseño introduce la necesidad de mecanismos de sincronización para mantener la consistencia del estado global.

El simulador utiliza una función central protegida por un mutex que coordina la ejecución. Esta función avanza el reloj virtual, notifica al proceso activo que puede ejecutar su porción de trabajo y espera mediante una variable de condición hasta que el proceso señale que completó su tick. Posteriormente, la función actualiza los módulos de memoria y E/S, y registra los eventos correspondientes en el colector de métricas.

Las variables compartidas entre hilos utilizan tipos atómicos de C++ cuando las operaciones son simples lecturas o escrituras. Para operaciones más complejas que requieren consistencia entre múltiples variables, se emplean secciones protegidas por mutex.

II-E. Prevención de inanición por memoria

Un problema que puede surgir durante la simulación es la inanición de un proceso que espera cargar sus páginas en

memoria. Cuando un proceso es seleccionado para ejecución, el administrador de memoria intenta cargar sus páginas. Si no hay marcos disponibles, el algoritmo de reemplazo selecciona páginas víctima. Sin embargo, mientras el proceso espera, el planificador puede seleccionar otro proceso, cuyas páginas podrían reemplazar las que el primer proceso acaba de cargar.

Para evitar este escenario, el simulador implementa un mecanismo basado en el bit de referencia. Cuando las páginas de un proceso se cargan en preparación para su ejecución, se marcan como protegidas. Estas páginas no pueden ser seleccionadas como víctimas hasta que el proceso termine su ráfaga actual, ya sea por completar su tiempo de CPU, por un bloqueo de E/S o por preempción en algoritmos apropiativos. Una vez que el proceso sale del estado de ejecución, sus páginas se desprotegen y quedan disponibles para reemplazo.

II-F. Algoritmos de planificación de CPU

El simulador implementa cuatro algoritmos de planificación de CPU. FCFS atiende los procesos en el orden en que llegan a la cola de listos. Este algoritmo no es apropiativo, por lo que un proceso en ejecución continúa hasta completar su ráfaga. SJF selecciona el proceso cuya próxima ráfaga de CPU sea la más corta. En la implementación actual, SJF tampoco es apropiativo.

Round Robin asigna a cada proceso un intervalo de tiempo fijo denominado quantum. Cuando un proceso agota su quantum sin completar su ráfaga, es preemptado y reinsertado al final de la cola de listos. El planificador por prioridad selecciona el proceso con mayor prioridad, representada por el menor valor numérico. Este algoritmo es apropiativo: si llega un proceso con mayor prioridad que el actual, se produce una preempción.

Cuadro I
ALGORITMOS DE PLANIFICACIÓN DE CPU IMPLEMENTADOS.

Algoritmo	Criterio de selección	Apropiativo
FCFS	Orden de llegada	No
SJF	Menor ráfaga	No
Round Robin	Turno con quantum	Sí
Por prioridad	Mayor prioridad	Sí

II-G. Algoritmos de reemplazo de páginas

El administrador de memoria implementa cuatro algoritmos de reemplazo. FIFO mantiene un registro del orden de llegada de las páginas a memoria y selecciona la más antigua para reemplazo. LRU registra el último acceso a cada página y selecciona aquella que no ha sido utilizada durante más tiempo.

NRU clasifica las páginas según sus bits de referencia y modificación. En la implementación actual, dado que la simulación no contempla modificaciones a memoria, efectivamente solo existen dos categorías: páginas referenciadas y no referenciadas. El algoritmo selecciona una página de la categoría más baja disponible.

El algoritmo óptimo requiere conocimiento del futuro, lo cual no está disponible en sistemas reales. La implementación utiliza una heurística que aproxima este comportamiento: primero considera como víctimas las páginas de procesos terminados, luego las de procesos bloqueados por E/S ordenados por tiempo restante de mayor a menor, y finalmente cualquier página que no esté protegida.

Cuadro II
ALGORITMOS DE REEMPLAZO DE PÁGINAS IMPLEMENTADOS.

Algoritmo	Criterio de selección
FIFO	Página con mayor tiempo en memoria
LRU	Página no accedida por más tiempo
NRU	Página con menor categoría según bit de referencia
Óptimo	Heurística basada en estado de procesos

II-H. Planificación de E/S

El módulo de E/S gestiona las solicitudes de acceso a dispositivos periféricos. La implementación actual utiliza un único dispositivo denominado disco. Cuando un proceso completa una ráfaga de CPU y su siguiente ráfaga es de E/S, se genera una solicitud que se encola en el planificador del dispositivo.

Los algoritmos disponibles son FCFS y Round Robin, con funcionamiento análogo a sus contrapartes de CPU. En FCFS, las solicitudes se atienden en orden de llegada y cada operación se completa antes de comenzar la siguiente. En Round Robin, cada solicitud recibe un quantum de tiempo configurable; si no completa en ese intervalo, se reencola y se atiende la siguiente solicitud pendiente.

La arquitectura del módulo de E/S está diseñada para soportar múltiples dispositivos, aunque la configuración actual emplea únicamente uno. Cada dispositivo mantiene su propia cola de solicitudes pendientes y puede configurarse con un planificador independiente. Esta separación permite que futuras extensiones del simulador incorporen dispositivos con diferentes características de latencia y throughput.

II-I. Recolección de métricas

El módulo de métricas registra eventos durante toda la simulación. Los eventos se agrupan por tick e incluyen información sobre el estado de la CPU, operaciones de memoria, actividad de E/S y transiciones de estado de los procesos. Al finalizar la simulación, los datos se exportan en formato JSONL, donde cada línea representa un objeto JSON independiente correspondiente a un tick.

El visualizador en Python lee estos archivos y genera gráficos mediante la biblioteca Seaborn. Los diagramas de Gantt muestran la asignación temporal de CPU y dispositivos de E/S a los procesos. Los gráficos de evolución de colas, como el ilustrado en la Figura 4, muestran cómo cambia la cantidad de procesos en cada estado a lo largo del tiempo. Los mapas de memoria presentan la asignación de marcos físicos a las páginas de cada proceso.

Figura 4. Ejemplo de gráfico de evolución de colas, mostrando la cantidad de procesos en cada estado a lo largo del tiempo.

Figura 5. Ejemplo de diagrama de Gantt generado por el visualizador, mostrando la asignación de CPU a tres procesos.

II-J. Configuración del sistema

El simulador lee dos archivos de configuración al inicio. El archivo de configuración general especifica parámetros globales como el número de marcos de memoria disponibles, el tamaño de cada marco, los algoritmos a utilizar y los valores de quantum. El archivo de procesos define cada proceso con su identificador, tiempo de llegada, secuencia de ráfagas, prioridad y número de páginas requeridas.

```
total_memory_frames=64
frame_size=4096
scheduling_algorithm=RoundRobin
page_replacement_algorithm=LRU
io_scheduling_algorithm=FCFS
quantum=4
io_quantum=4
```

Listing 1. Ejemplo de archivo de configuración del simulador.

El Listado 1 muestra un ejemplo de archivo de configuración. En este caso, el sistema dispone de 64 marcos de memoria de 4096 bytes cada uno, utiliza Round Robin para planificación de CPU con quantum de 4 unidades, LRU para reemplazo de páginas y FCFS para planificación de E/S.

```
# PID llegada rafagas prioridad paginas
P1 0 CPU(4),E/S(3),CPU(5) 1 4
P2 2 CPU(6),E/S(2),CPU(3) 2 5
P3 4 CPU(8) 3 6
```

Listing 2. Ejemplo de archivo de definición de procesos.

El Listado 2 ilustra la definición de tres procesos. El proceso P1 llega en el tiempo 0, tiene prioridad 1, requiere 4 páginas y ejecuta una ráfaga de CPU de 4 unidades, seguida de una operación de E/S de 3 unidades y una ráfaga final de CPU de 5 unidades.

II-K. Interfaz de línea de comandos

El simulador se ejecuta desde la terminal y acepta parámetros para especificar los archivos de entrada y salida. La opción -f indica el archivo de procesos, -c el archivo de configuración y -m el archivo donde se guardarán las métricas. El visualizador se invoca como un módulo de Python y soporta procesamiento individual de archivos o por lotes de un directorio completo.

El proyecto incluye pruebas unitarias implementadas con la biblioteca Catch2 que verifican el correcto funcionamiento de los algoritmos de planificación y reemplazo. La documentación del código se genera mediante Doxygen y está disponible en formato HTML y PDF. El código fuente del proyecto se encuentra disponible en un repositorio público de GitHub.

III. Resultados

III-A. Casos de prueba

III-B. Análisis de resultados

IV. Conclusiones

El desarrollo de este simulador permitió integrar los conceptos de planificación de CPU, administración de memoria virtual y gestión de E/S en un sistema funcional. La arquitectura modular del simulador facilita la modificación de algoritmos sin afectar otros componentes del sistema. Esta característica resulta útil para comparar el desempeño de diferentes estrategias de planificación y reemplazo bajo las mismas condiciones de carga.

La implementación de procesos como hilos independientes introduce la complejidad de la sincronización concurrente, pero replica de manera más fiel el comportamiento de un sistema operativo real. El uso de mutex y variables de condición garantiza la consistencia del estado global mientras permite que múltiples procesos existan simultáneamente en memoria. El mecanismo de protección mediante el bit de referencia demostró ser efectivo para prevenir situaciones de inanición cuando un proceso espera la carga de sus páginas en memoria.

La integración entre los módulos de CPU, memoria y E/S requiere una coordinación cuidadosa para mantener la coherencia del sistema. El diseño centralizado del planificador de CPU como coordinador principal simplifica esta tarea al concentrar las decisiones de estado en un único punto de control. Esta aproximación facilita el debugging y la extensión del sistema.

El módulo de visualización complementa al simulador proporcionando representaciones gráficas que facilitan la comprensión del comportamiento del sistema. Los diagramas de Gantt y las gráficas de evolución de colas permiten identificar patrones de ejecución y posibles cuellos de botella. El formato

JSONL para la exportación de métricas permite el procesamiento flexible de los datos con herramientas externas.

Como trabajo futuro, se propone extender el simulador para soportar múltiples dispositivos de E/S con diferentes latencias, implementar algoritmos de planificación multinivel con retroalimentación, e incorporar métricas adicionales que permitan analizar la localidad de referencia de los accesos a memoria. También sería interesante agregar una interfaz gráfica interactiva que permita visualizar la simulación en tiempo real.

REFERENCIAS

[1] "AMS-StyleGuide-online.pdf," American Mathematical Society, published by the American Mathematical Society.