Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas Operativos

Proyecto Final

“Simulador de Sistema Operativo Básico”

Profesor: José Albero Avalos Vélez

Grupo: 02

Integrantes:

Flores Torres Ángel Joel

Morales Soto Fernando

Muciño Hernández Zara Paulina

Hernández Hernández Oscar Mario

Fecha de entrega: 20 de noviembre de 2024

Semestre: 2025-1

Índice

Introducción

Un sistema operativo es el software base que gestiona los recursos de hardware y software de un sistema informático. Este proyecto tiene como objetivo simular tres de sus componentes fundamentales: la planificación de procesos, la administración de memoria y el sistema de archivos. Con este simulador, se busca facilitar el aprendizaje práctico y la comprensión de cómo estas funcionalidades trabajan juntas para garantizar el rendimiento y la estabilidad del sistema. La combinación de una interfaz gráfica intuitiva y una implementación lógica sólida proporciona un entorno educativo ideal para explorar los conceptos básicos de los sistemas operativos.

Objetivos

- Comprender y aplicar los conceptos de planificación de procesos.
- Implementar un sistema básico de administración de memoria.
- Simular un sistema de archivos simple con comandos básicos de manipulación de archivos y directorios.

Componentes del proyecto

1. **Planificación de Procesos**
    - Implementar varios algoritmos de planificación como FIFO, Round Robin, SJF (Shortest Job First).
    - El sistema debe permitir al usuario crear "procesos" con distintos tiempos de ejecución y prioridades, y mostrar el orden de ejecución según el algoritmo elegido.
    - Incluir estadísticas como el tiempo promedio de espera y tiempo de retorno para analizar la eficiencia de cada algoritmo.

2. **Administración de Memoria**
    - Simular un esquema de paginación o segmentación en memoria, donde los procesos se carguen en "bloques" o "segmentos".

- Implementar algoritmos de reemplazo de páginas, como FIFO o LRU (Least Recently Used), para gestionar el reemplazo en caso de fallos de página.
- Mostrar la tabla de páginas o el estado de la memoria tras cada carga de proceso.

3. **Sistema de Archivos**
    - Crear un sistema de archivos básico que permita crear, leer, y eliminar archivos y directorios.
    - Implementar comandos como mkdir, touch, rm, y ls, para que el usuario pueda manipular los archivos en el sistema.
    - Cada archivo puede contener texto, permitiendo operaciones de lectura y escritura en el mismo.

Desarrollo

El proyecto se divide en dos archivos principales:

1. **simulador.py**: Contiene toda la lógica de backend que implementa las funcionalidades del sistema operativo.
2. **interfaz.py**: Proporciona una interfaz gráfica de usuario (GUI) mediante Tkinter para interactuar con las funcionalidades del simulador.

Descripción del Archivo simulador.py

Planificación de Procesos

La clase ProcessScheduler es responsable de gestionar los procesos y aplicar diferentes algoritmos de planificación:

- **FIFO (First In, First Out)**:
    - Los procesos se ejecutan en el orden en que llegan a la cola.
    - Este algoritmo es simple y fácil de implementar, pero puede sufrir del problema conocido como "Convoy Effect".

- **SJF (Shortest Job First)**:
    - Los procesos con menor tiempo de ráfaga son priorizados.
    - Minimiza el tiempo promedio de espera, pero puede llevar a inanición de procesos largos.

- **Round Robin**:
    - Divide el tiempo de CPU en intervalos llamados "quantum".
    - Proporciona equidad a todos los procesos, pero puede aumentar el overhead si el quantum es muy pequeño.

Funciones principales de ProcessScheduler:

- **add_process(name, burst_time, priority)**:
    - Agrega un proceso con un nombre, tiempo de ráfaga y prioridad a la cola de procesos.

- **fifo()**:
    - Aplica el algoritmo FIFO, calcula el tiempo de espera y de retorno para cada proceso, y devuelve los promedios.

- **sjf()**:
    - Ordena los procesos por tiempo de ráfaga, los ejecuta y calcula las métricas como en FIFO.

- **round_robin(quantum)**:
    - Aplica el algoritmo Round Robin con el quantum especificado, gestionando los tiempos de ejecución de los procesos.

Administración de Memoria

La clase MemoryManager simula un esquema de paginación. Divide la memoria física en bloques de tamaño fijo llamados frames, y los procesos se dividen en páginas de igual tamaño.

Características principales:

- Los procesos ocupan tantos frames como páginas tengan.
- Si no hay frames libres, se aplica reemplazo de páginas (FIFO).
- Se proporciona una visualización del estado actual de los frames de memoria.

Funciones principales de MemoryManager:

- **allocate(process_id, process_size)**:
    - Intenta asignar memoria a un proceso. Devuelve True si se logra la asignación, o False si no hay espacio suficiente.

- **load_page(process_id, page_number)**:
    - Carga una página de un proceso en un frame. Si no hay espacio, realiza un reemplazo siguiendo el algoritmo FIFO.

- **display_memory()**:
    - Imprime el estado actual de los frames, indicando qué proceso ocupa cada uno.

Sistema de Archivos

La clase FileSystem implementa operaciones básicas de un sistema de archivos:

1. Crear directorios y archivos.
2. Escribir y leer archivos.
3. Eliminar archivos o directorios.
4. Listar contenidos del directorio raíz.

Funciones principales de FileSystem:

- **mkdir(name)**:
    - Crea un directorio con el nombre especificado.
- **touch(name)**:
    - Crea un archivo vacío con el nombre especificado.
- **write(name, content)**:

    - Escribe contenido en un archivo existente.
- **read(name)**:
    - Lee y devuelve el contenido de un archivo.
- **rm(name)**:
    - Elimina un archivo o directorio existente.
- **ls()**:
    - Lista todos los archivos y directorios en el directorio raíz.

Descripción del Archivo interfaz.py

Estructura de la Interfaz

El archivo interfaz.py utiliza Tkinter para crear una interfaz gráfica que permite al usuario interactuar con el simulador. Está organizada en tres módulos principales:

1. Planificación de Procesos.
2. Administración de Memoria.
3. Sistema de Archivos.

Funcionalidades

- **Módulo de Planificación de Procesos:**
    - Permite agregar procesos mediante diálogos de entrada.
    - Ejecuta los algoritmos FIFO, SJF y Round Robin.
    - Muestra los resultados en un área de texto.

- **Módulo de Administración de Memoria:**
    - Permite visualizar el estado actual de los frames de memoria.
    - Muestra información sobre qué proceso ocupa cada frame.

- **Módulo de Sistema de Archivos:**
    - Permite crear, leer, escribir y eliminar archivos y directorios.
    - Lista los contenidos del directorio raíz.

Algoritmos Utilizados

FIFO

- **Descripción:** Los procesos se ejecutan en el orden en que llegan.
- **Ventaja:** Simple de implementar.
- **Desventaja:** Puede causar tiempos de espera altos para procesos largos.

SJF

- **Descripción:** Los procesos con menor tiempo de ráfaga tienen prioridad.
- **Ventaja:** Minimiza el tiempo promedio de espera.
- **Desventaja:** Los procesos largos pueden sufrir inanición.

Round Robin

- **Descripción:** Divide el tiempo de CPU en intervalos llamados "quantum".
- **Ventaja:** Proporciona equidad entre los procesos.
- **Desventaja:** Puede aumentar el overhead si el quantum es demasiado pequeño.

Interacción entre Módulos

La interfaz gráfica se comunica con las clases de simulador.py para ejecutar las funcionalidades del simulador. Cada botón de la interfaz llama a una función específica que interactúa con las clases ProcessScheduler, MemoryManager o FileSystem.

Conclusiones

Flores Torres Ángel Joel.

Este proyecto nos permitió comprender de manera práctica cómo funcionan los componentes fundamentales de un sistema operativo, como la planificación de procesos, la administración de memoria y el sistema de archivos. Además, la implementación de algoritmos como FIFO, SJF y Round Robin nos ayudó a analizar las ventajas y desventajas de cada enfoque.

Morales Soto Fernando.

La integración de una interfaz gráfica hizo que la interacción con el simulador fuera más accesible, lo que permitió no solo probar, sino también visualizar el comportamiento de un sistema operativo. Esto fue clave para comprender cómo las decisiones en la planificación y la memoria afectan el rendimiento general.

Muciño Hernández Zara Paulina.

Trabajar en este simulador nos enseñó cómo los sistemas operativos manejan recursos de manera eficiente. Particularmente, implementar la paginación en la memoria y los algoritmos de reemplazo nos dio una visión detallada de cómo se optimizan los recursos en tiempo real.

Hernández Hernández Oscar Mario.

Este proyecto no solo fue una práctica de programación, sino también una forma de explorar la teoría detrás de los sistemas operativos. La aplicación de conceptos como la equidad en Round Robin o la eficiencia de SJF nos permitió conectar la teoría con la práctica de una manera significativa.