SISTEMAS OPERATIVOS

ACTIVIDAD DE APRENDIZAJE 14
PROGRAMA 7

Profesor:
VIOLETA DEL ROCIO BECERRA VELAZQUEZ

VERDUZCO ROSALES LUIS ENRIQUE
223992388
Ingenieria en Computación

23/11/25

CUCEI DIVTIC D04

ÍNDICE

Paginación Simple y Round Robin

Objetivo de la actividad

Implementar la técnica de administración de memoria de Paginación Simple sobre un simulador de planificación de procesos Round Robin (RR). El propósito es gestionar la memoria dividiéndola en marcos de tamaño fijo y los procesos en páginas, condicionando la admisión de procesos a la disponibilidad de marcos libres, y visualizando gráficamente la asignación de memoria, la fragmentación interna y las tablas de páginas correspondientes.

Introducción

El estudio de los sistemas operativos requiere no solo de la comprensión teórica de conceptos abstractos como la planificación de procesos y la gestión de memoria, sino también de la visualización práctica de estos mecanismos. En este proyecto, se desarrolló un simulador que implementa el algoritmo de planificación Round Robin (RR) conjuntamente con un esquema de gestión de memoria basado en Paginación Simple.

Para la construcción de este software, se seleccionó un entorno de desarrollo web estándar compuesto por HTML, CSS y JavaScript. Adelante veremos las razones detrás de esta elección tecnológica, sus ventajas, los desafíos técnicos enfrentados durante la codificación y las soluciones implementadas.

Elección del Entorno de Desarrollo Web

La decisión de utilizar tecnologías web (HTML, CSS y JS) en lugar de lenguajes compilados tradicionales (como C++ o Java con Swing) se fundamentó en tres pilares: accesibilidad, capacidad de visualización y arquitectura orientada a eventos.

JavaScript, al ser el lenguaje nativo del navegador, permite una ejecución inmediata sin necesidad de instalar compiladores o entornos complejos en la máquina del usuario. Además, el Modelo de Objetos del Documento (DOM) ofrece una interfaz gráfica nativa que facilita la representación dinámica de estructuras de datos, como las colas de procesos y, crucialmente, el mapa de memoria de 48 marcos solicitado en la actividad.

Ventajas Principales

El uso de este stack tecnológico aportó beneficios significativos al desarrollo del simulador:

1. Renderizado Dinámico con CSS Grid:
   La visualización de la memoria (240 unidades divididas en 48 marcos) se simplificó enormemente gracias a CSS Grid. En lugar de dibujar píxel por píxel, se pudieron generar divs dinámicamente y organizarlos en una retícula, permitiendo representar estados (Libre, Listo, Ejecución, Bloqueado, SO) simplemente alternando clases CSS (.so, .listo, .bloqueado).

2. Manejo de Eventos Asíncronos:
   La naturaleza de JavaScript basada en el Event Loop se alinea perfectamente con la simulación de interrupciones de hardware. El uso de addEventListener para capturar teclas ('E', 'W', 'P', 'C') simula de manera efectiva las interrupciones de E/S y errores, sin bloquear el hilo principal de ejecución visual.

3. Separación de Intereses:
   La estructura de archivos separados permitió mantener un código limpio. La lógica de negocio (cálculo de Quantum, paginación) se mantuvo aislada de la lógica de presentación.

Problemas Encontrados y Soluciones

Durante el desarrollo del código, surgieron varios desafíos técnicos inherentes a la lógica de sistemas operativos y a las peculiaridades de JavaScript:

<u>1. Sincronización del "Reloj" y el Bucle de Eventos</u>

Problema:
Un sistema operativo real tiene un reloj de hardware constante. JavaScript es asíncrono y monohilo, por lo que un bucle while infinito congelaría la interfaz gráfica, impidiendo ver la simulación.

Solución:
Se implementó la función setInterval para crear un "tick" artificial cada 1000ms. Esto permite que la función mainEjecucion() actúe como el despachador del SO, ejecutando

un ciclo de instrucción y luego liberando el hilo para que el navegador renderice los cambios en el DOM.

2. Gestión de la Memoria y Fragmentación Interna

Problema:
Representar visualmente la fragmentación interna fue difícil. No bastaba con marcar un marco como "ocupado"; era necesario mostrar qué porción de ese marco (tamaño 5) estaba siendo utilizada realmente por la última página de un proceso.

Solución:
Se diseñó cada marco visual en HTML como un contenedor con 5 divs internos (slots). En la lógica de JavaScript (asignarMemoria), se calcula el residuo del tamaño del proceso. Si un proceso requiere 3 unidades en su última página, el script asigna la clase de color solo a los primeros 3 slots del marco, dejando los otros 2 en blanco, visualizando así la fragmentación de manera precisa.

3. Conflicto de Clases CSS en la Transición de Pantallas

Problema:
Al finalizar la simulación, la pantalla de ejecución no desaparecía correctamente para dar paso al reporte final, a pesar de aplicar las clases de ocultamiento.

Solución:
Se identificó un problema de especificidad y cascada en CSS donde las clases .enableCont y .disableCont entraban en conflicto. La solución fue lógica: asegurar en JavaScript que antes de añadir la clase disableCont, se eliminará explícitamente la clase enableCont utilizando classList.remove(), garantizando una transición limpia entre estados.

4. Lógica de Admisión (Paginación)

Problema:
En los requerimiento se pedía que un proceso sólo pudiera entrar a la cola de listos si existían suficientes marcos libres (Planificador a Largo Plazo).

Solución:
Se implementó un arreglo auxiliar marcosLibres que actúa como una pila o lista de huecos disponibles. Antes de admitir un proceso de la cola de nuevos, se verifica if (proc.numPaginas <= marcosLibres.length). Esta validación previa evita errores de desbordamiento de memoria y cumple con la restricción de admisión estricta.

Conclusión

El desarrollo de este simulador en JavaScript me ayudó a demostrar que las tecnologías web son herramientas poderosas para el aprendizaje de conceptos complejos de ingeniería de sistemas. Aunque JavaScript abstrae la gestión real de memoria (punteros y direcciones físicas), su flexibilidad permite recrear lógicamente la tabla de páginas y la asignación de marcos con total fidelidad a los algoritmos teóricos.

El haber logrado implementar la Paginación Simple sobre un esquema Round Robin me ayudó a darme cuenta de la importancia de estructuras de datos claras (arreglos de objetos para los procesos y mapas para la memoria) y una manipulación precisa del DOM para la retroalimentación visual.

Código Fuente: https://github.com/UnluckyLuren/SisOperativos

Ejecutable: https://unluckyluren.github.io/SisOperativos/