Proyecto Simulador

Universidad del Valle de Guatemala

Sistemas Operativos - Sección 10

Michelle Angel de María Mejía Villela, 22596
Silvia Alejandra Illescas Fernández, 22376
Diederich Josué Emidio Solís López, 22952

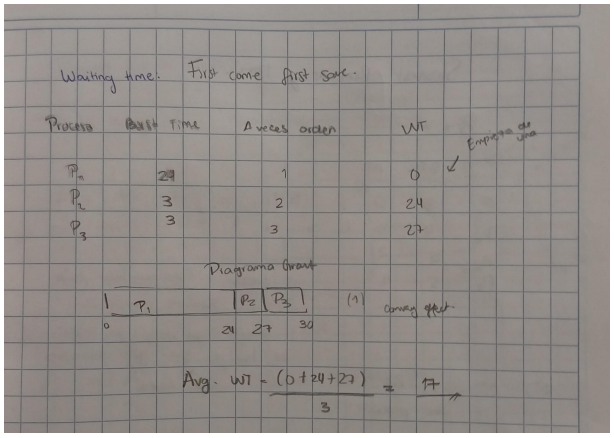
Guatemala, 04 de junio del 2025

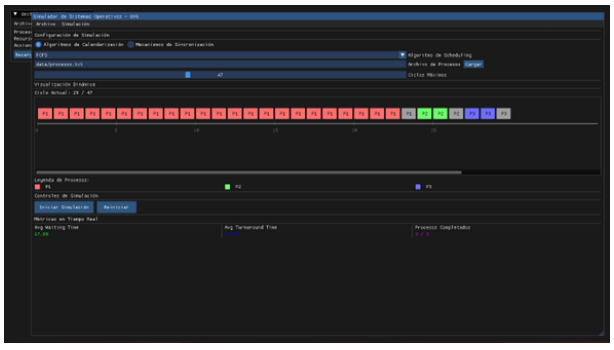
Introducción

Este proyecto simula diversos algoritmos de planificación de procesos y mecanismos de sincronización en sistemas operativos, utilizando el lenguaje de programación C++. El objetivo principal es comprender y analizar el comportamiento de estos algoritmos y mecanismos en un entorno controlado, permitiendo evaluar su eficiencia, ventajas y desventajas.

Algoritmos de Planificación de Procesos

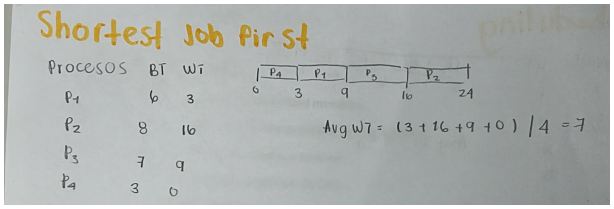
1. **FCFS (First Come, First Served)**
    - **Descripción:** Atiende los procesos en el orden en que llegan al sistema.
    - **Tipo:** No apropiativo.
    - **Ventajas:** Fácil de implementar.
    - **Desventajas:** Puede causar tiempos de espera elevados si un proceso con una ráfaga de CPU larga llega antes que otros más cortos.

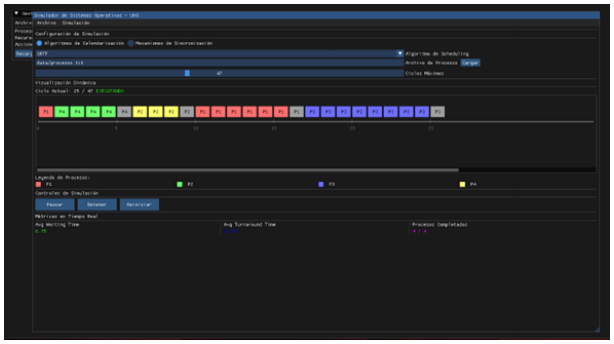




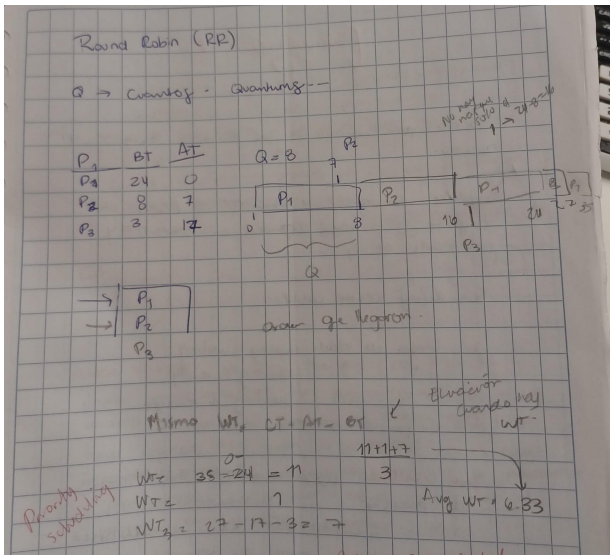
2. **SJF (Shortest Job First)**
    - **Descripción:** Selecciona el proceso con la menor ráfaga de CPU estimada.

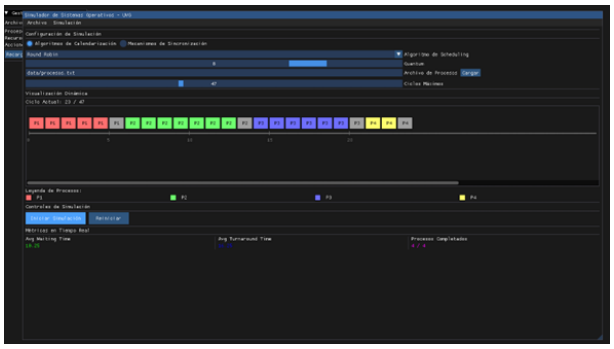
- **Tipo:** No apropiativo.
- **Ventajas:** Minimiza el tiempo promedio de espera.
- **Desventajas:** Puede causar inanición si llegan continuamente procesos cortos.



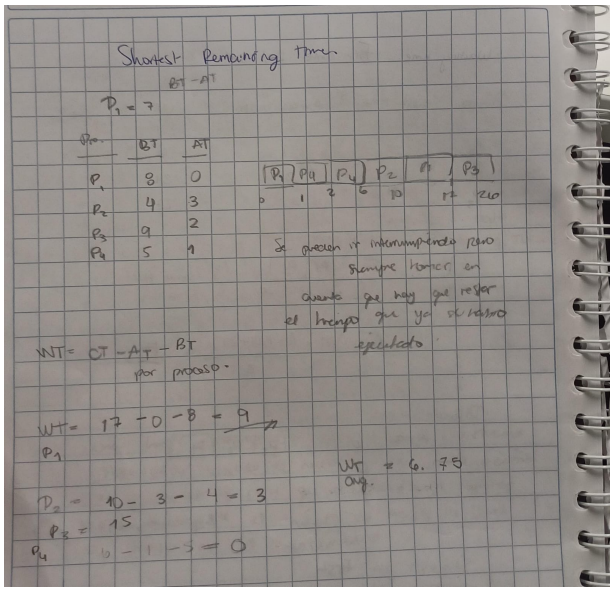


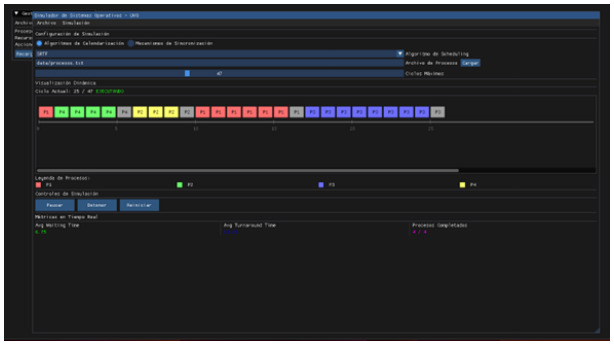
3. **Round Robin (RR)**
    - **Descripción:** Asigna a cada proceso un tiempo fijo (quantum) de CPU en orden circular.
    - **Tipo:** Apropiativo.
    - **Ventajas:** Justo y adecuado para sistemas de tiempo compartido.
    - **Desventajas:** El rendimiento depende del tamaño del quantum; un quantum muy pequeño puede causar sobrecarga.





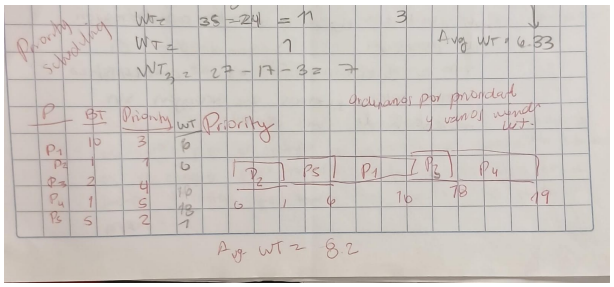
4. **SRTF (Shortest Remaining Time First)**
   - **Descripción:** Versión apropiativa de SJF; selecciona el proceso con el menor tiempo restante de ejecución.
   - **Tipo:** Apropiativo.
   - **Ventajas:** Minimiza el tiempo promedio de espera y retorno.
   - **Desventajas:** Puede causar inanición para procesos con ráfagas de CPU largas.

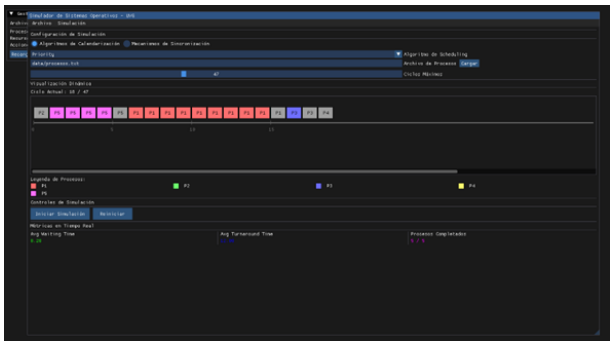




5. **Priority Scheduling**

   - **Descripción:** Asigna prioridad a cada proceso y selecciona el de mayor prioridad para ejecutarse.
   - **Tipo:** Puede ser apropiativo o no apropiativo.
   - **Ventajas:** Permite un control más fino sobre la ejecución de procesos importantes.
   - **Desventajas:** Puede causar inanición para procesos de baja prioridad.





Mecanismos de Sincronización

1. **Mutex (Mutual Exclusion)**
   - **Descripción:** Mecanismo que permite que solo un hilo o proceso acceda a una sección crítica o recurso compartido a la vez.
   - **Funcionamiento:** Un proceso debe adquirir el mutex antes de entrar a la sección crítica y liberarlo al salir.
   - **Ventajas:** Previene condiciones de carrera.
   - **Desventajas:** Puede causar bloqueo si no se maneja adecuadamente.
2. **Semáforo**
   - **Descripción:** Variable que controla el acceso a recursos compartidos mediante operaciones de espera (wait) y señalización (signal).
   - **Tipos:**
     - Semáforo binario: Valores 0 o 1, similar a un mutex.
     - Semáforo contable: Valores mayores a 1, permite múltiples accesos simultáneos.
   - **Ventajas:** Flexible para manejar múltiples recursos.
   - **Desventajas:** Mayor complejidad y posibilidad de errores como la inversión de prioridades.

Estructura del Proyecto

El proyecto está organizado de forma modular para facilitar su mantenimiento y comprensión. A continuación se detalla la estructura general:

- **include/**: Archivos de cabecera (`.hpp`) donde se definen las clases, estructuras y funciones del sistema.
- **src/**: Código fuente principal en C++. Contiene los módulos de planificación, sincronización, manejo de archivos, lógica del simulador y la interfaz gráfica basada en Dear ImGui con SFML.

- **data/**: Archivos de entrada como `procesos.txt`, `recursos.txt` y `acciones.txt`, necesarios para simular diferentes escenarios.
- **build/**: Carpeta destinada a la salida de la compilación generada por CMake.
- **output/**: Contiene archivos de salida en formato `.csv` con resultados como diagramas de Gantt o métricas de sincronización.
- **CMakeLists.txt**: Script de configuración para compilar el proyecto fácilmente.
- **configurar_proyecto.sh**: Script de instalación automática de dependencias y compilación (en sistemas compatibles).

Compilación y Ejecución

Requisitos

Para compilar y ejecutar el simulador se requieren las siguientes herramientas y bibliotecas:

- Compilador C++ compatible con C++17
- CMake versión 3.20 o superior
- SFML 2.5+
- Dear ImGui

Instalación Automática (macOS/Linux)

```
chmod +x configurar_proyecto.sh
./configurar_proyecto.sh
```

Compilación Manual

```
mkdir build
cd build
cmake ..
make
```

Ejecución

```
cd build
./SimuladorSO
```

Al iniciar el programa, se despliega una interfaz gráfica que permite seleccionar entre distintos algoritmos de planificación (FCFS, SJF, SRTF, Round Robin, Prioridades) y mecanismos de sincronización (mutex, semáforos). La UI permite además configurar parámetros y observar métricas en tiempo real.

Salida y Visualización

El simulador genera automáticamente archivos en formato CSV con los resultados obtenidos, tales como:

- `gantt_fcfs.csv`, `gantt_rr.csv`, etc.: Diagramas de Gantt para cada algoritmo.
- `mutex_resultado.csv`, `semaforo_resultado.csv`: Resultados de sincronización.
- Métricas visualizadas directamente en la UI: tiempo promedio de espera, turnaround, accesos a recursos, etc.

Además, la interfaz gráfica muestra una línea de tiempo interactiva que facilita la visualización del comportamiento del sistema.

Conclusión

Este proyecto permitió desarrollar un simulador funcional e interactivo de algoritmos de planificación y mecanismos de sincronización en sistemas operativos. Inicialmente diseñado para consola, se logró evolucionar a una aplicación con interfaz gráfica utilizando Dear ImGui y SFML, mejorando significativamente la experiencia del usuario y la comprensión visual de los conceptos. Las simulaciones ejecutadas demostraron el correcto funcionamiento del sistema, cumpliendo con los objetivos propuestos del curso.