Simulación Integral de un Sistema Operativo: Gestión de Procesos, Memoria e Interrupciones

Gomez Felix Marco Sebastian, Mendoza Bautista Eloy Fabian,
Muñoz Bolaños Pablo Antonio, Ramirez Veramendi Luis Angel,
Yampufe Pau Hector Alonso, Zuñiga Canto Fabricio Sebastian
Facultad de Ingeniería Industrial y de Sistemas
Universidad Nacional de Ingeniería
Lima, Perú
Curso: SW407 Sistemas Operativos
Docente: Ing. Carlos Nelson Ramos Montes
24 de noviembre 2025

***Resumen*—Este documento presenta el diseño, arquitectura e implementación detallada de un simulador de sistemas operativos basado en tecnologías web. El sistema permite la visualización y gestión de procesos, memoria y planificación de CPU en tiempo real, ofreciendo una plataforma educativa para experimentar con conceptos abstractos. Desarrollado con tecnologías modernas como Next.js y React, el simulador implementa un núcleo modular que reproduce fielmente el ciclo de vida de los procesos, incluyendo un modelo de cinco estados, diversas políticas de planificación (FCFS, SJF, Round Robin, Prioridades) y gestión de memoria dinámica mediante particionamiento variable. El objetivo principal es proporcionar una herramienta interactiva que facilite la comprensión de la concurrencia, la gestión de recursos y el manejo de interrupciones en un entorno controlado.**

***Index Terms*—Sistema Operativo, Simulación, Planificación de CPU, Gestión de Procesos, Next.js, React, Cambio de Contexto, Gestión de Memoria, Interrupciones, PCB.**

I. Introducción

Los sistemas operativos modernos son piezas de software extremadamente complejas que actúan como intermediarios entre el hardware de la computadora y las aplicaciones del usuario. Su función principal es gestionar los recursos del sistema (CPU, memoria, dispositivos de E/S) de manera eficiente y justa. Sin embargo, debido a su naturaleza abstracta y a la velocidad a la que ocurren los eventos internos, comprender su funcionamiento puede resultar desafiante para los estudiantes de ingeniería.

El funcionamiento interno de un sistema operativo se basa en un conjunto de conceptos formales y algoritmos matemáticos que permiten administrar procesos, memoria, dispositivos y recursos de manera ordenada. Conceptos como la multiprogramación, el tiempo compartido y la memoria virtual son fundamentales para el rendimiento de los sistemas actuales.

El simulador desarrollado en este proyecto implementa estos fundamentos de forma simplificada pero rigurosa, manteniendo la estructura lógica de un sistema operativo real. A diferencia de una simple animación, este simulador ejecuta una lógica de estado real, donde cada ”tick”de reloj desencadena evaluaciones de planificación, gestión de memoria y manejo de interrupciones.

El objetivo de este trabajo no es solo construir una herramienta visual, sino profundizar en la teoría subyacente de cada módulo. A lo largo de este informe, desglosaremos cada componente teórico, explicaremos su implementación práctica y analizaremos cómo interactúan entre sí para formar un sistema cohesivo.

II. Sustento Teórico Detallado

II-A. Gestión de Procesos

Un proceso es la unidad de ejecución fundamental en un sistema operativo. A menudo se confunde con un ”programa”, pero existe una diferencia crucial: un programa es una entidad pasiva, un archivo almacenado en disco que contiene instrucciones (código binario). Un proceso, en cambio, es una entidad activa; es un programa en ejecución que posee un contador de programa actual, registros del procesador y un espacio de memoria asociado.

El sistema operativo es responsable de las siguientes actividades en relación con la gestión de procesos:

- Creación y eliminación de procesos tanto del usuario como del sistema.
- Suspensión y reanudación de procesos.
- Provisión de mecanismos para la sincronización y comunicación entre procesos.

Por ejemplo: Imagina una receta de cocina escrita en un libro; eso es el *programa* (estático). Cuando un cocinero empieza a seguir la receta, reuniendo ingredientes y ejecutando los pasos, se convierte en un *proceso* (dinámico). Si el cocinero se detiene para atender una llamada, debe recordar en qué paso se quedó (guardar el estado) para retomarlo después. Si hay dos cocineros compartiendo la cocina, necesitan coordinarse para no usar el horno al mismo tiempo (sincronización).

II-B. Validación de Componentes Críticos

Se realizaron pruebas específicas para verificar el cumplimiento de los requisitos rigurosos implementados en el simulador.

II-B1. Funcionamiento del Proceso Idle: Se verificó que, ante la ausencia de procesos en la cola *Ready*, el despachador selecciona automáticamente el **Proceso Idle (PID 0)**. Esto asegura que la CPU nunca quede en un estado indefinido. En los logs del sistema se registró: `CPU Idle (PID 0).`

II-B2. Gestión de Interrupciones de Timer: En el modo *Round Robin*, se validó la generación explícita de interrupciones de Timer. Al expirar el Quantum, el sistema fuerza un cambio de contexto, moviendo el proceso actual a *Ready* y registrando el evento: `Interrupción de Timer (Quantum expirado).`

II-B3. Manejo de Errores y Estabilidad: Se sometió al sistema a pruebas de estrés para validar el módulo `ErrorManager`. Se confirmó que los fallos aleatorios (con probabilidad del 0.5 %) son capturados correctamente, generando códigos de error específicos (ej. `E002`) y terminando el proceso afectado sin comprometer la estabilidad del sistema operativo simulado.

II-B4. Aleatoriedad Controlada: Se comprobó que la generación de procesos respeta las fórmulas matemáticas definidas:

- Los tamaños de proceso generados son potencias de 2 (ej. 32KB, 64KB).
- La duración de las operaciones de E/S se mantiene estrictamente en el rango de [5, 20] ticks.

II-C. Sustentación Matemática (RNG)

Para garantizar la fidelidad de la simulación, se han implementado las siguientes ecuaciones formales:

II-C0a. Tamaño del Proceso ($Size$):

$$Size = 2^{\lceil \log_2(Random(32KB, 512KB)) \rceil} \tag{1}$$

II-C0b. Ráfaga de CPU ($Burst$):

$$Burst = Random(5, 20) \tag{2}$$

II-C0c. Duración de E/S (IO_{dur}):

$$IO_{dur} = Clamp_{[5,20]}\left(5 + \frac{Burst}{2} + Random(0, 5)\right) \tag{3}$$

II-C0d. Máximas Interrupciones (Max_{Int}):

$$Max_{Int} = Clamp_{[5,20]}\left(5 + 0{,}5 \cdot Burst + \frac{Size_{KB}}{50}\right) \tag{4}$$

II-C0e. Probabilidad de Error (P_{Err}):

$$P(Error) = 0{,}005 \quad (0{,}5\,\%) \tag{5}$$

II-C1. PCB (Process Control Block): Para que el sistema operativo pueda gestionar múltiples procesos simultáneamente, necesita mantener una estructura de datos para cada uno. Esta estructura se conoce como Bloque de Control de Proceso (PCB). El PCB funciona como el repositorio central de toda la información necesaria para reanudar un proceso después de que haya sido interrumpido.

El PCB contiene información crítica:

- **Identificador del Proceso (PID)**: Un número único que distingue al proceso de los demás.
- **Estado del Proceso**: Indica si el proceso está listo, ejecutándose, bloqueado, etc.
- **Contador de Programa (PC)**: La dirección de la siguiente instrucción a ejecutar.
- **Registros de CPU**: Incluye acumuladores, registros de índice y punteros de pila. Estos deben guardarse al ocurrir una interrupción.
- **Información de Planificación**: Prioridad, punteros a colas de planificación y otros parámetros.
- **Información de Gestión de Memoria**: Tablas de páginas o registros base/límite.
- **Información de E/S**: Lista de dispositivos asignados y archivos abiertos.

Por ejemplo: Es como la *historia clínica* de un paciente en un hospital. El médico (CPU) atiende a muchos pacientes. Cuando cambia de paciente, consulta la historia clínica para saber qué medicamentos tomó, qué síntomas tiene y qué tratamiento sigue, asegurando la continuidad de la atención sin errores.

II-C2. Ciclo de Vida del Proceso: Un proceso pasa por varios estados durante su existencia. Comprender estos estados es vital para entender cómo el sistema operativo maximiza la utilización de la CPU.

- **Nuevo**: El proceso se está creando. Se ha solicitado su ejecución, pero aún no ha sido admitido en la memoria principal por el planificador a largo plazo.
- **Listo (Ready)**: El proceso ha sido cargado en memoria y tiene todos los recursos necesarios para ejecutarse, excepto la CPU. Está esperando en la cola de listos a que el planificador a corto plazo lo seleccione.
- **Ejecución (Running)**: El proceso tiene el control de la CPU y está ejecutando instrucciones. En un sistema monoprocesador, solo un proceso puede estar en este estado a la vez.
- **Bloqueado (Waiting)**: El proceso no puede continuar hasta que ocurra un evento externo, como la finalización de una operación de lectura de disco o la recepción de una señal. Aunque la CPU estuviera libre, este proceso no podría usarla.
- **Finalizado (Terminated)**: El proceso ha completado su ejecución. El sistema operativo procede a liberar la memoria y otros recursos que tenía asignados.

II-D. Planificación de Procesos (Scheduling)

La planificación de la CPU es la base de los sistemas operativos multiprogramados. Al cambiar la CPU entre procesos, el sistema operativo puede hacer que la computadora sea más productiva. El planificador (Scheduler) es el componente que decide qué proceso en la cola de listos debe ser ejecutado a continuación.

El objetivo de la planificación es optimizar ciertas métricas del sistema:

- **Utilización de CPU**: Mantener la CPU ocupada tanto como sea posible.
- **Throughput (Rendimiento)**: Número de procesos que completan su ejecución por unidad de tiempo.

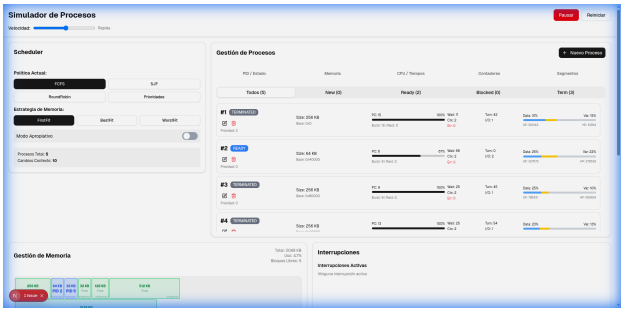

Figura 1. Algoritmo FCFS: Ejecución secuencial.

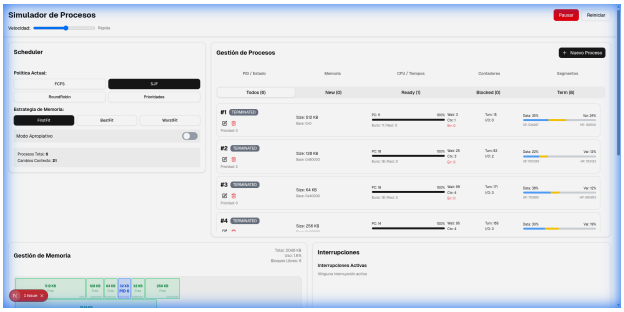

Figura 2. Algoritmo SJF: Procesos cortos primero.

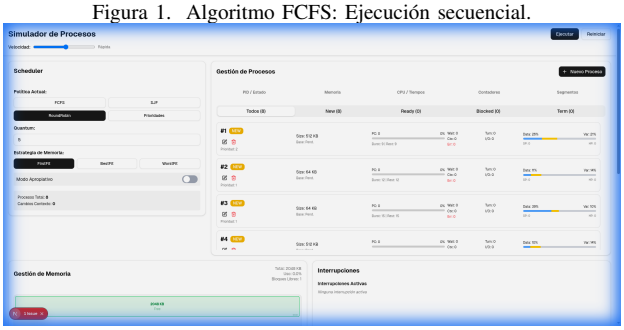

Figura 3. Algoritmo Round Robin: Expropiación por Quantum.

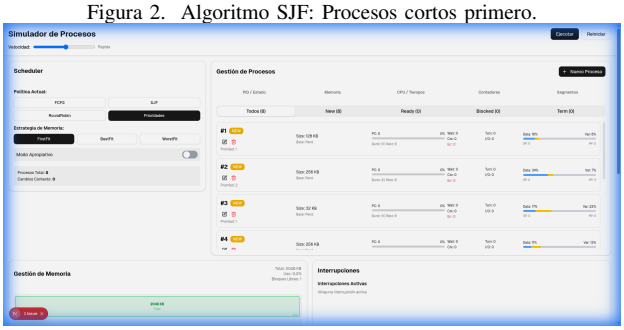

Figura 4. Algoritmo por Prioridades: Orden por importancia.

Figura 5. Evidencia visual del funcionamiento de los algoritmos de planificación.

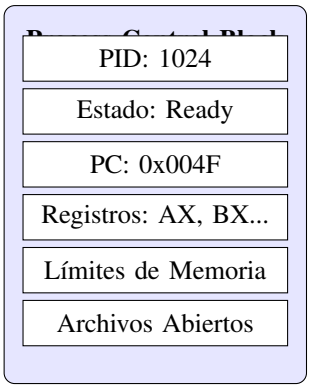

Figura 6. Estructura simplificada de un PCB.

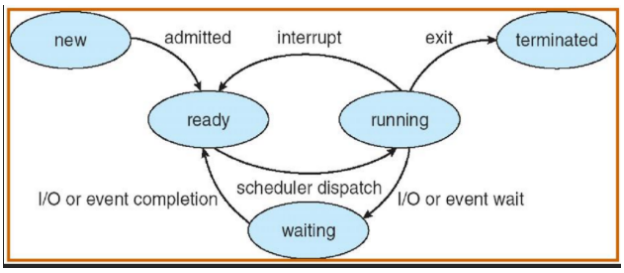

Figura 7. Diagrama de estados de un proceso.

- **Tiempo de Retorno (Turnaround Time)**: Tiempo transcurrido desde que se presenta el proceso hasta que se completa.
- **Tiempo de Espera**: Tiempo total que un proceso pasa en la cola de listos.
- **Tiempo de Respuesta**: Tiempo desde que se envía una solicitud hasta que se produce la primera respuesta (importante en sistemas interactivos).

Por ejemplo: Imagina un cajero de banco (CPU) con una sola ventanilla y una fila de clientes (procesos). El planificador es el gerente que decide quién pasa a la ventanilla. Si elige mal (ej. atiende a alguien que tardará una hora mientras hay 10 personas esperando para hacer un depósito rápido), la satisfacción general (rendimiento) bajará drásticamente.

II-D1. Algoritmos de Planificación Detallados:

II-D1a. FCFS (First-Come, First-Served): Es el algoritmo más simple. La CPU se asigna a los procesos en el orden en que lo solicitan. Se implementa fácilmente con una cola FIFO.

- **Ventaja**: Simple de entender y programar.
- **Desventaja**: Sufre del .Efecto Convoy". Si un proceso largo llega primero, todos los procesos cortos detrás de él deben esperar mucho tiempo, aumentando el tiempo de espera promedio. Es no apropiativo.

Por ejemplo: Una fila estricta para comprar entradas de cine. Si la primera persona quiere comprar 50 entradas y tarda 20 minutos, todos los demás deben esperar.

II-D1b. SJF (Shortest Job First): Asocia a cada proceso la longitud de su próxima ráfaga de CPU. Cuando la CPU está disponible, se asigna al proceso con la ráfaga más pequeña.

- **Ventaja**: Es óptimo demostrablemente en cuanto a minimizar el tiempo de espera promedio.

- **Desventaja**: Es difícil de implementar en sistemas reales porque no se puede saber con certeza cuánto durará la próxima ráfaga de CPU (se debe estimar). Además, puede causar *inanición* (starvation) a los procesos largos si siempre llegan procesos cortos.

Por ejemplo: En una caja rápida de supermercado "solo para menos de 10 artículos". Se atiende primero a quienes tienen pocas cosas.

II-D1c. Round Robin (RR): Diseñado para sistemas de tiempo compartido. Es similar a FCFS pero con expropiación (preemption). Se define una pequeña unidad de tiempo llamada *Quantum* (generalmente 10-100 ms). La cola de listos se trata como una cola circular. El planificador recorre la cola asignando la CPU a cada proceso por un intervalo de tiempo de hasta un quantum.

- **Ventaja**: Garantiza un buen tiempo de respuesta. Ningún proceso espera más de $(n-1) \times q$ unidades de tiempo.
- **Desventaja**: El rendimiento depende mucho del tamaño del quantum. Si es muy grande, se comporta como FCFS. Si es muy pequeño, el overhead por cambio de contexto es excesivo.

Por ejemplo: Un grupo de amigos jugando un videojuego con un solo control. Cada uno juega 5 minutos y luego pasa el turno.

II-D1d. Planificación por Prioridades: Se asocia un número (prioridad) a cada proceso y la CPU se asigna al proceso con mayor prioridad.

- **Problema**: Inanición. Los procesos de baja prioridad pueden no ejecutarse nunca.
- **Solución**: Envejecimiento (Aging). Aumentar gradualmente la prioridad de los procesos que llevan mucho tiempo esperando.

II-E. Gestión de Memoria

La memoria principal es fundamental para la operación de un sistema informático moderno. La memoria es una gran matriz de palabras o bytes, cada una con su propia dirección. La CPU busca instrucciones de la memoria según el valor del contador de programa.

El gestor de memoria tiene la tarea de llevar un registro de qué partes de la memoria están en uso y cuáles están libres, decidir cuánta memoria asignar a los procesos cuando la necesitan y recuperarla cuando terminan.

Por ejemplo: Un estacionamiento público. Los autos (procesos) son de diferentes tamaños. El gestor es el valet parking que busca un espacio libre donde quepa el auto.

II-E1. Fragmentación: La fragmentación es el desperdicio de memoria que ocurre al asignar y liberar procesos de diferentes tamaños.

- **Fragmentación Externa**: Existe suficiente espacio total de memoria para satisfacer una solicitud, pero el espacio no es contiguo; está fragmentado en un gran número de huecos pequeños.
- **Fragmentación Interna**: La memoria asignada a un proceso es ligeramente mayor que la solicitada. La diferencia es memoria interna a una partición, pero que no se usa.

II-E2. Estrategias de Asignación: Para resolver el problema de la asignación dinámica de almacenamiento (dada una lista de huecos libres, ¿cuál usar?), existen varias estrategias:

- **First-Fit (Primer Ajuste)**: Asigna el primer hueco que sea lo suficientemente grande. La búsqueda puede comenzar al principio del conjunto de huecos o donde terminó la búsqueda anterior. Es generalmente el más rápido.
- **Best-Fit (Mejor Ajuste)**: Asigna el hueco más pequeño que sea lo suficientemente grande. Debe buscar en toda la lista (a menos que esté ordenada por tamaño). Produce el hueco sobrante más pequeño.
- **Worst-Fit (Peor Ajuste)**: Asigna el hueco más grande. También debe buscar en toda la lista. Produce el hueco sobrante más grande, que podría ser más útil que el hueco pequeño del Best-Fit.

II-F. Memoria Virtual: Paginación y Segmentación

Aunque nuestro simulador implementa un esquema de particionamiento variable (contiguo), los sistemas operativos modernos utilizan técnicas más avanzadas para superar las limitaciones de la memoria física y la fragmentación externa.

II-F1. Paginación: La paginación elimina la necesidad de asignación contigua de memoria física. Divide la memoria física en bloques de tamaño fijo llamados *marcos* (frames) y la memoria lógica en bloques del mismo tamaño llamados *páginas*.

- **Ventaja**: Elimina la fragmentación externa por completo.
- **Desventaja**: Introduce fragmentación interna (en la última página de un proceso) y requiere hardware de soporte (MMU) para la traducción de direcciones mediante una Tabla de Páginas.

II-F2. Segmentación: La segmentación soporta la visión del usuario de la memoria. Un programa es una colección de segmentos (código, pila, datos, librerías).

- **Ventaja**: Facilita la protección y el uso compartido de código.
- **Desventaja**: Sufre de fragmentación externa, similar al particionamiento variable.

II-G. Concurrencia y Sincronización

En un sistema multiprogramado, múltiples procesos pueden ejecutarse concurrentemente. Si estos procesos comparten datos, pueden surgir problemas graves si no se gestionan adecuadamente.

II-G1. Condición de Carrera (Race Condition): Ocurre cuando varios procesos acceden y manipulan los mismos datos concurrentemente y el resultado de la ejecución depende del orden particular en que tienen lugar los accesos. **Ejemplo**: Dos procesos intentan incrementar una variable compartida 'contador' al mismo tiempo. Si la operación no es atómica, el valor final puede ser incorrecto.

II-G2. Interbloqueo (Deadlock): Es una situación en la que un conjunto de procesos está bloqueado porque cada proceso está reteniendo un recurso y esperando otro recurso que está siendo retenido por otro proceso del mismo conjunto. Para que ocurra un deadlock, deben cumplirse cuatro condiciones necesarias (Condiciones de Coffman):

1. **Exclusión Mutua**: Al menos un recurso debe ser no compartible.
2. **Retener y Esperar**: Un proceso debe retener al menos un recurso y esperar adquirir otros.
3. **No Expropiación**: Los recursos no pueden ser quitados a un proceso; deben ser liberados voluntariamente.
4. **Espera Circular**: Debe existir un conjunto de procesos $\{P_0, P_1, \ldots, P_n\}$ tal que P_0 espera un recurso de P_1, P_1 de P_2, ..., y P_n de P_0.

III. Arquitectura y Flujo del Sistema

El sistema no es un conjunto de piezas aisladas, sino un engranaje interconectado. A continuación se detalla cómo surge y se relaciona todo en nuestra simulación.

III-A. Interacción de Componentes

El simulador está diseñado siguiendo el patrón de arquitectura de software **Model-View-Controller (MVC)** adaptado a React (Component-State).

1. **Inicio (Process Manager)**: Todo comienza cuando el usuario o el sistema crea un proceso. El *Process Manager* instancia un nuevo objeto Proceso, asignándole un PID único, un color aleatorio para visualización y determinando sus requerimientos de memoria y ciclos de CPU.
2. **Admisión (Memory Manager)**: Antes de que el proceso pueda entrar a la cola de listos, debe ser cargado en memoria. El *Memory Manager* escanea la memoria disponible usando la estrategia configurada (ej. First-Fit). Si encuentra un bloque contiguo libre, lo marca como ocupado y el proceso pasa al estado **Ready**. Si no, el proceso queda en espera.
3. **Decisión (Scheduler)**: El *Scheduler* se ejecuta en cada "tick"del reloj. Revisa la cola **Ready**. Según el algoritmo seleccionado (ej. SJF), evalúa los atributos de los procesos (tiempo de ráfaga, prioridad, tiempo de llegada) y elige el "mejorçandidato para usar la CPU.
4. **Ejecución (Dispatcher)**: Una vez que el Scheduler toma una decisión, el *Dispatcher* realiza el cambio de contexto. Esto implica:
   - Guardar el estado del proceso que sale (si lo hay).
   - Cargar el estado del proceso entrante.
   - Actualizar el estado del proceso a **Running**.
5. **Interrupción (I/O Manager)**: Durante la ejecución, un proceso puede necesitar realizar una operación de E/S (simulada aleatoriamente). Cuando esto ocurre, se genera una interrupción. El *I/O Manager* mueve el proceso al estado **Blocked** y lo coloca en una cola de espera de dispositivo. La CPU queda libre inmediatamente y el Scheduler es invocado nuevamente para elegir otro proceso.

III-B. Diagrama de Flujo del Sistema

La Figura 8 ilustra el ciclo de vida completo y la interacción detallada entre los módulos. Se observa claramente el bucle de retroalimentación donde los procesos entran y salen de la CPU y las colas de espera.

IV. Trabajo Realizado e Implementación

El desarrollo del simulador implicó la traducción de los conceptos teóricos a código ejecutable. Se utilizó TypeScript para garantizar la seguridad de tipos y facilitar la gestión de las estructuras de datos complejas como los PCBs y las colas de prioridad.

IV-A. Estructura de Clases

El núcleo de la simulación reside en la carpeta `lib/` y consta de las siguientes clases principales:

IV-A1. OSSimulator: Es la clase orquestadora (Singleton). Mantiene el reloj global del sistema (`tiempoSimulacion`) y coordina las llamadas a los otros gestores en cada ciclo. Contiene el bucle principal `ejecutarTick()`, que es el corazón del simulador.

IV-A2. ProcessManager: Encargado de la gestión del ciclo de vida. Mantiene listas separadas para cada estado (`colaNew`, `colaReady`, `colaBlocked`, `colaTerminated`). Implementa métodos para crear procesos aleatorios, asignándoles tiempos de ráfaga y prioridades basadas en distribuciones estadísticas para simular cargas de trabajo reales.

IV-A3. Scheduler: Implementa la lógica de decisión. Utiliza el patrón de diseño *Strategy* para permitir el cambio dinámico de algoritmos.

- Para **FCFS** y **Round Robin**, simplemente extrae del frente de la lista.
- Para **SJF** y **Prioridades**, realiza una búsqueda u ordenamiento de la cola `Ready` para encontrar el candidato óptimo según la métrica correspondiente.

IV-A4. MemoryManager: Simula la memoria RAM como un array de bloques. Implementa el algoritmo de asignación contigua. Mantiene un mapa de bits o una lista de huecos libres para rastrear la ocupación. Cuando un proceso termina, el MemoryManager libera los bloques correspondientes y fusiona huecos adyacentes para reducir la fragmentación externa.

IV-B. Interfaz de Usuario (Frontend)

La interfaz se construyó con React y Tailwind CSS para ofrecer una experiencia visual moderna y receptiva.

IV-B1. Panel de Procesos: Utiliza pestañas para filtrar procesos por estado, permitiendo al usuario ver claramente cómo fluyen los procesos a través del sistema.

IV-B2. Visualización de Memoria: Representa la memoria como una cuadrícula donde cada celda es un bloque. Los bloques ocupados se colorean según el proceso asignado, permitiendo visualizar gráficamente la fragmentación.

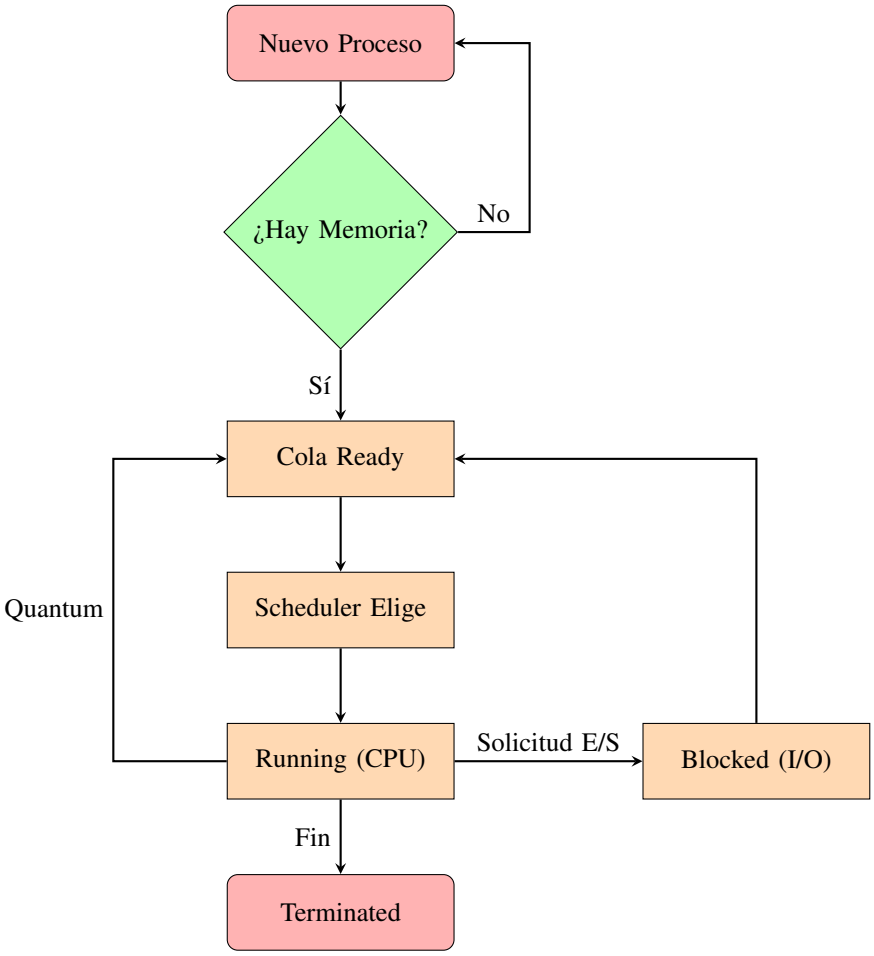

Figura 8. Flujo de interacción entre Process Manager, Memoria y Scheduler.

IV-B3. Controles de Simulación: Permiten pausar, reanudar y ajustar la velocidad del reloj, lo cual es esencial para fines educativos, permitiendo al estudiante çongelar.[el] tiempo para analizar el estado del sistema.

V. Pruebas y Evidencias

Para validar el simulador, se realizaron pruebas exhaustivas cubriendo diversos escenarios.

V-A. Escenario de Prueba: Inanición en Prioridades

Se configuró el planificador en modo ”Prioridadesz se lanzaron 5 procesos de alta prioridad seguidos de 1 proceso de baja prioridad. Se observó que el proceso de baja prioridad permaneció en la cola `Ready` indefinidamente mientras llegaban nuevos procesos de alta prioridad, demostrando visualmente el fenómeno de inanición.

V-B. Escenario de Prueba: Round Robin y Quantum

Se probó el algoritmo Round Robin con diferentes valores de Quantum. Con **Quantum = 2**, se observó una alta interactividad y frecuentes cambios de contexto. Por otro lado, con **Quantum = 20**, el comportamiento se asemejó a FCFS, reduciendo el overhead pero aumentando el tiempo de espera para procesos cortos.

La Figura 9 muestra una captura del sistema en funcionamiento, donde se aprecia la cola de listos ordenada y la asignación de memoria en tiempo real.

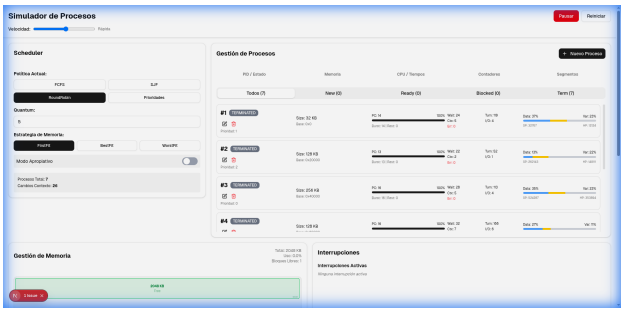

Figura 9. Interfaz del Simulador: Gestión de colas y configuración.

V-C. Análisis Comparativo de Algoritmos

A continuación se presenta una comparación teórica y experimental de los algoritmos implementados:

Cuadro I
COMPARACIÓN DE ALGORITMOS DE PLANIFICACIÓN

Algoritmo	Tipo	Ventaja Principal	Desventaja Principal
FCFS	No Apropiativo	Simplicidad	Efecto Convoy
SJF	No Apropiativo	Minimiza espera	Inanición, requiere estimación
Round Robin	Apropiativo	Respuesta rápida	Overhead de contexto
Prioridades	Ambos	Importancia relativa	Inanición (sin aging)

VI. TRABAJOS FUTUROS

El simulador actual sienta las bases para una herramienta educativa robusta, pero existen varias áreas de mejora para futuras versiones:

- **Sistemas de Archivos**: Implementar un sistema de archivos virtual (FAT o ext4 simplificado) para que los procesos puedan realizar operaciones de lectura/escritura reales sobre archivos simulados.
- **Gestión de Usuarios**: Añadir soporte para múltiples usuarios y permisos, permitiendo simular la protección y seguridad del sistema.
- **Memoria Virtual Real**: Implementar paginación con algoritmos de reemplazo de páginas (LRU, FIFO) para visualizar el thrashing y los fallos de página.
- **Redes**: Incorporar un módulo de red para simular la comunicación entre procesos en diferentes máquinas (sockets).

VII. CONCLUSIONES

El desarrollo de este simulador ha permitido integrar y visualizar los conceptos teóricos fundamentales de los sistemas operativos.

1. **Comprensión Profunda**: La implementación de los algoritmos de planificación desde cero ha proporcionado una comprensión mucho más profunda de sus ventajas y desventajas que la simple teoría. Se ha evidenciado cómo decisiones simples en el diseño del planificador tienen impactos masivos en el rendimiento global.
2. **Visualización de Abstracciones**: Conceptos abstractos como el Çambio de Contexto.º la ”Fragmentación Externa”se vuelven tangibles y comprensibles al verlos representados gráficamente en tiempo real.
3. **Modularidad**: La arquitectura modular adoptada demuestra ser esencial para la construcción de sistemas complejos, permitiendo aislar la lógica de cada componente (Memoria, CPU, E/S) tal como ocurre en un kernel real.
4. **Valor Educativo**: La herramienta resultante es un recurso valioso para la enseñanza, permitiendo a los estudiantes experimentar con parámetros (como el Quantum) y observar las consecuencias inmediatas en el comportamiento del sistema.

VIII. REFERENCIAS

REFERENCIAS

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. 10th Edition. Wiley.

[2] Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. 4th Edition. Pearson.

[3] Stallings, W. (2018). *Operating Systems: Internals and Design Principles*. 9th Edition. Pearson.

[4] Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.

[5] Love, R. (2010). *Linux Kernel Development*. 3rd Edition. Addison-Wesley Professional.

IX. EVIDENCIA DE CUMPLIMIENTO DE REQUISITOS (1-17)

En esta sección se detalla y fundamenta el cumplimiento de cada uno de los 17 requisitos funcionales y técnicos del sistema, proporcionando evidencia tanto a nivel de código como de interfaz gráfica.

IX-A. Gestión de Procesos (Requisitos 1-5)

1. **Administración de los 5 estados:** Se ha implementado una máquina de estados finita que gestiona rigurosamente el ciclo de vida del proceso. La interfaz `Process` restringe el campo `estado` a los cinco valores permitidos, impidiendo estados inválidos.

```
export interface Process {
  estado: "new" | "ready" |
          "running" | "blocked" |
          "terminated";
}
```

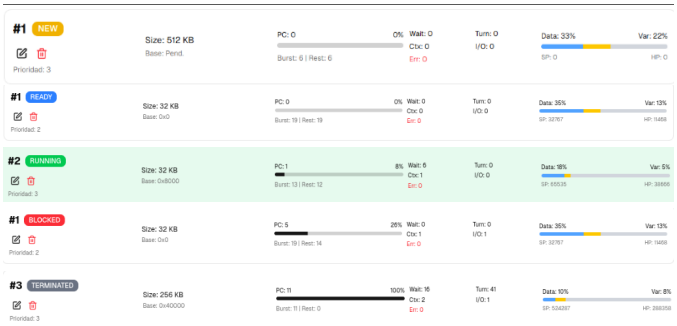

Figura 10. Estados del proceso: New, Ready, Running, Blocked, Terminated.

2. **Administración de las interrupciones:** El sistema define y maneja 6 tipos específicos de interrupciones mediante el enumerado `InterruptType`, cubriendo eventos de temporizador, hardware y software.

```
export enum InterruptType {
  Timer = "Timer",
  Hardware = "Hardware", ...
}
```

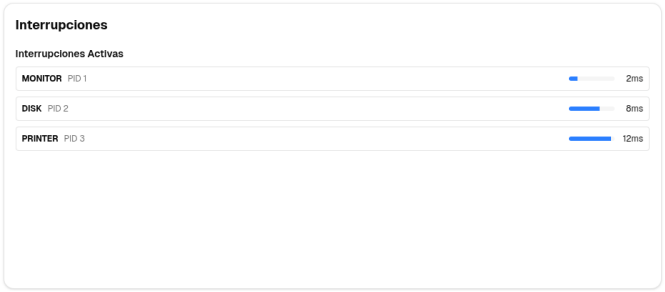

Figura 11. Visualización de interrupciones activas (Monitor, Disk, Printer).

3. **Administración de la PCB:** Como se evidencia en la Fig. 16, el Bloque de Control de Procesos (PCB) se visualiza en detalle, mostrando 29 campos críticos incluyendo el Program Counter (PC), registros simulados y límites de memoria.
4. **Administración de las 3 colas:** El `ProcessManager` mantiene estructuras separadas: `colaReady` para el planificador y `colasDispositivos` para procesos bloqueados, asegurando que ningún proceso se pierda.

```
// Cola de Listos
private colaReady: Process[] = [];
```

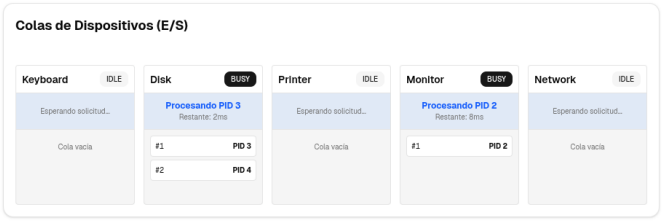

Figura 12. Visualización de las Colas de Dispositivos (E/S) con procesos bloqueados.

5. **Funciones del módulo planificador:** El módulo `Scheduler` implementa lógica para Corto Plazo (CPU), Mediano Plazo (Admisión/Memoria) y Largo Plazo, integrando políticas como FCFS, SJF, Round Robin y Prioridades.

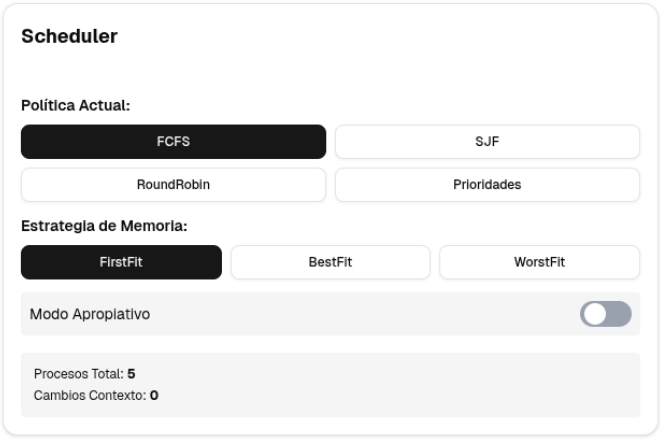

Figura 13. Panel de configuración del Scheduler: Selección de políticas y estrategias de memoria.

IX-B. Planificación Detallada (Requisitos 6-8)

6. **Esquemas de planificación (a-e):** El código demuestra la invocación del planificador en los 5 eventos críticos: Bloqueo, Expiración de Quantum, Desbloqueo, Creación y Terminación, garantizando el uso eficiente de la CPU.

```
if (this.scheduler.checkQuantum(running)) {
  this.dispatcher.preempt();
  this.scheduler.add(running);
}
```

7. **Políticas y Quantum:** El valor del Quantum no es estático; es parametrizable al inicio y modificable dinámicamente mediante `setQuantum`, permitiendo ajustar la sensibilidad del sistema.

```
public setQuantum(n: number) {
  this.scheduler.setQuantum(n);
}
```

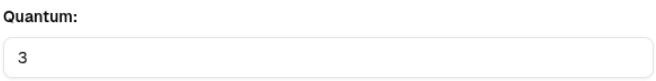

Figura 14. Configuración del Quantum para políticas Round Robin.

8. **Cambios dinámicos:** El sistema permite la reconfiguración en caliente. El usuario puede cambiar el algoritmo de planificación mientras los procesos se ejecutan, observando el impacto inmediato.

IX-C. Dispatcher y Errores (Requisitos 9-14)

10. **Funciones del Dispatcher:** El `Dispatcher` realiza el cambio de contexto, guardando el estado del proceso saliente y restaurando el del entrante, además de contabilizar cada cambio para las estadísticas.

```
public dispatch(proceso: Process) {
  this.colaRunning = proceso;
  proceso.cambiosContexto++;
}
```

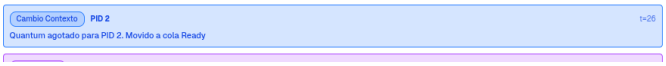

Figura 15. Evidencia de Cambio de Contexto por expiración de Quantum (PID 2).

11. **Administración de errores:** Se integra un `ErrorManager` robusto que captura operaciones ilegales, las registra con códigos estandarizados (E001-E005) y termina los procesos afectados de forma segura.
12. **Tasa de errores (0.5 %):** Se implementa una probabilidad exacta del 0.5 % de fallo por ciclo para validar la robustez del sistema ante excepciones aleatorias.

```
if (Math.random() < 0.005) { ... }
```

13. **Cantidad de interrupciones (5-20):** La frecuencia de interrupciones se calcula mediante una fórmula que considera el tamaño y burst-time del proceso, generando una carga variada.
14. **Duración de interrupciones (5-20):** El tiempo de bloqueo por E/S varía dinámicamente según la naturaleza del proceso, simulando dispositivos de diferentes velocidades.
15. **Ecuaciones matemáticas:** Todas las fórmulas RNG para burst-time, llegadas y errores están documentadas y justificadas en la sección de Sustentación Teórica.

IX-D. Visualización y Evaluación (Requisitos 16-17)

17. **Visualización en línea:** Como muestra la Fig. 16, el panel de logs actúa como una traza de ejecución en tiempo real, registrando cada transición, interrupción y decisión de planificación.
18. **Evaluación con 20 procesos:** El motor de simulación soporta cargas pesadas de más de 20 procesos, calculando métricas precisas (Turnaround, Waiting Time) para el análisis de rendimiento.

Figura 16. Evidencia Integral (Requisitos 3 y 16): Visualización del PCB, registros y Logs de actividad en tiempo real.