Universidad Distrital Francisco José de Caldas

Ingeniería de Sistemas

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Proyecto Final Implementación de Algoritmos de Planificación de Procesos

Grupo

Daniel Felipe Velandia Jerez – 20191020140
Johan Sebastian Fontecha Soler – 20191578026
Yeison Alexander Farfán Peralta – 20201020138
Daniel Andres Marquez Torres - 20222020238

Bogotá, febrero 2025

1. Resumen

Se presenta la implementación de cuatro algoritmos de planificación de procesos a corto plazo de la parte de gestión de procesos en un sistema operativo: **FCFS (First Come, First Served)**, **SJF (Shortest Job First), SRTF (Shortest Remaining Time First) y RR (Round Robin)**. Estos algoritmos son fundamentales para la gestión de procesos en sistemas multiprogramados, ya que determinan el orden en que los procesos acceden al CPU, optimizando el uso de recursos y mejorando el rendimiento del sistema.

El objetivo principal de este proyecto es demostrar cómo estos algoritmos funcionan en la práctica, utilizando JavaScript para simular su comportamiento y calcular métricas clave como el tiempo de retorno, tiempo de espera, tiempo perdido, penalidad y tiempo de respuesta.

2.Introducción

Cuando una computadora tiene un sistema operativo multiprogramado, con frecuencia tiene varios procesos o hilos que compiten por la CPU al mismo tiempo. Esta situación ocurre cada vez que dos o más de estos procesos se encuentran al mismo tiempo en el estado listo. Si sólo hay una CPU disponible, hay que decidir cuál proceso se va a ejecutar a continuación. La parte del sistema operativo que realiza esa decisión se conoce como planificador de procesos y el algoritmo que utiliza se conoce como algoritmo de planificación.

Una cuestión clave relacionada con la planificación es saber cuándo tomar decisiones de planificación. Resulta ser que hay una variedad de situaciones en las que se necesita la planificación. En primer lugar, cuando se crea un nuevo proceso se debe tomar una decisión en cuanto a si se debe ejecutar el proceso padre o el proceso hijo. En segundo lugar, se debe tomar una decisión de planificación cuando un proceso termina. En tercer lugar, cuando un proceso se bloquea por esperar una operación de E/S, un semáforo o por alguna otra razón, hay que elegir otro proceso para ejecutarlo. En cuarto lugar, cuando ocurre una interrupción de E/S tal vez haya que tomar una decisión de planificación. Casi todos los procesos alternan ráfagas de cálculos con peticiones de E/S (de disco), la E/S es cuando un proceso entra al estado bloqueado en espera de que un dispositivo externo complete su trabajo.

Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj. Un algoritmo de programación no apropiativo (nonpreemptive) selecciona un proceso para ejecutarlo y después sólo deja que se ejecute hasta que el mismo se bloquea o hasta que libera la CPU en forma voluntaria. Por el contrario, un algoritmo de planificación apropiativa selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo

3. Marco Teórico

Algoritmo FCFS

(FCFS, First-Come, First-Served) es no apropiativo. Con este algoritmo, la CPU se asigna a los procesos en el orden en el que la solicitan. En esencia hay una sola cola de procesos listos. A medida que van entrando otros trabajos, se colocan al final de la cola. Si el proceso en ejecución se bloquea, el primer proceso en la cola se

ejecuta a continuación. Cuando un proceso bloqueado pasa al estado listo, al igual que un trabajo recién llegado, se coloca al final de la cola.

Con este algoritmo, una sola lista ligada lleva la cuenta de todos los procesos listos. Para elegir un proceso a ejecutar sólo se requiere eliminar uno de la parte frontal de la cola. Para agregar un nuevo trabajo o desbloquear un proceso sólo hay que adjuntarlo a la parte final de la cola

Algoritmo SJF

Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero (SJF, Shortest Job First). El primer trabajo más corto es sólo óptimo cuando todos los trabajos están disponibles al mismo tiempo.

Algoritmo SRTF

Una versión apropiativa del algoritmo tipo el trabajo más corto primero es el menor tiempo restante a continuación (SRTN, Shortest Remaining Time Next) o SRTF. Con este algoritmo, el planificador siempre selecciona el proceso cuyo tiempo restante de ejecución sea el más corto. De nuevo, se debe conocer el tiempo de ejecución de antemano. Cuando llega un nuevo trabajo, su tiempo total se compara con el tiempo restante del proceso actual. Si el nuevo trabajo necesita menos tiempo para terminar que el proceso actual, éste se suspende y el nuevo trabajo se inicia. Ese esquema permite que los trabajos cortos nuevos obtengan un buen servicio.

Algoritmo Round Robin

Uno de los algoritmos más antiguos, simples, equitativos y de mayor uso es el de turno circular (round-robin). A cada proceso se le asigna un intervalo de tiempo, conocido como quántum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del cuanto, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quántum, la conmutación de la CPU se realiza cuando el proceso se bloquea, desde luego. Cuando el proceso utiliza su quántum, se coloca al final de la lista.

4. Desarrollo

Estructura del Proyecto

El proyecto consta de tres archivos HTML, cada uno dedicado a un algoritmo de planificación:

1. **FCFS.html**: Implementa el algoritmo **First Come, First Served**.
2. **SJF.html**: Implementa el algoritmo **Shortest Job First**.
3. **SRTF.html**: Implementa el algoritmo **Shortest Remaining Time First**.
4. **RR.html:** Implementa el algoritmo **Round Robin**

Cada archivo contiene:

- Una interfaz gráfica para ingresar procesos (nombre, tiempo de llegada y tiempo de ráfaga).
- Un botón para calcular el planificador correspondiente.
- Una tabla que muestra los resultados detallados de cada proceso.
- Un diagrama de Gantt que visualiza la ejecución de los procesos.

Implementación de los Algoritmos

FCFS (First Come, First Served)

El algoritmo FCFS es el más simple de los tres. Los procesos se ejecutan en el orden en que llegan, sin priorizar el tiempo de ráfaga. La implementación en JavaScript sigue estos pasos:

1. **Entrada de datos**: El usuario ingresa los procesos con su nombre, tiempo de llegada y tiempo de ráfaga.
2. **Ordenamiento**: Los procesos se ordenan por tiempo de llegada.
3. **Ejecución**: Los procesos se ejecutan en orden, calculando el tiempo de inicio, finalización, retorno, espera, tiempo perdido, penalidad y tiempo de respuesta.
4. **Resultados**: Se muestran en una tabla y se genera un diagrama de Gantt.

Ejemplo en el codigo

```
function calculateFCFS() {
    let processQueue = JSON.parse(JSON.stringify(processes));
    let currentTime = 0;
    let completed = [];
    let timeline = [];

    processQueue.sort((a, b) => a.arrivalTime - b.arrivalTime);

    while (processQueue.length > 0) {
        let selectedProcess = processQueue.shift();
        if (selectedProcess.arrivalTime > currentTime) {
            timeline.push({ name: "IDLE", start: currentTime, duration: selectedProcess.arrivalT
ime - currentTime, state: "idle" });
            currentTime = selectedProcess.arrivalTime;
        }
        timeline.push({ name: selectedProcess.name, start: currentTime, duration: selectedProces
s.burstTime, state: "running" });
        selectedProcess.startTime = currentTime;
        selectedProcess.completionTime = currentTime + selectedProcess.burstTime;
        selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.arriva
lTime;
        selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burstTim
e;
        selectedProcess.responseTime = selectedProcess.startTime - selectedProcess.arrivalTime;
        selectedProcess.penalty = selectedProcess.turnaroundTime / selectedProcess.burstTime;
        selectedProcess.lostTime = selectedProcess.startTime - selectedProcess.arrivalTime;
        currentTime = selectedProcess.completionTime;
        completed.push(selectedProcess);
    }
    displayResults(completed);
    drawGanttChart(timeline);
}
```

En la aplicación web

Agregar Proceso

Nombre del proceso | Tiempo de llegada | Tiempo de ráfaga | Agregar Proceso

Calcular FCFS

Procesos Ingresados

Proceso: A Llegada: 0 Ráfaga: 6 Eliminar

Proceso: B Llegada: 1 Ráfaga: 8 Eliminar

Proceso: C Llegada: 2 Ráfaga: 7 Eliminar

Proceso: D Llegada: 4 Ráfaga: 3 Eliminar

Proceso: E Llegada: 6 Ráfaga: 9 Eliminar

Proceso: F Llegada: 6 Ráfaga: 2 Eliminar

Resultados Detallados

Tiempo de Retorno Promedio	Tiempo de Espera Promedio	Tiempo Perdido Promedio	Penalidad Promedio	Tiempo de Respuesta Promedio
19.00	13.17	13.17	4.92	13.17

Proceso	Llegada	Ráfaga	Comienzo	Final	Retorno	Espera	T. Perdido	Penalidad	Respuesta
A	0	6	0	6	6	0	0	1.00	0
B	1	8	6	14	13	5	5	1.63	5
C	2	7	14	21	19	12	12	2.71	12
D	4	3	21	24	20	17	17	6.67	17
E	6	9	24	33	27	18	18	3.00	18
F	6	2	33	35	29	27	27	14.50	27

SJF (Shortest Job First)

En el partado de SJF tenemos que se prioriza los procesos con el menor tiempo de ráfaga. La implementación incluye:

1. **Entrada de datos**: Similar a FCFS.
2. **Ordenamiento**: Los procesos se ordenan por tiempo de llegada.
3. **Ejecución**: Se selecciona el proceso con el menor tiempo de ráfaga en cada paso.
4. **Resultados**: Se muestran en una tabla y se genera un diagrama de Gantt.

En la aplicación web

```
function calculateSJF() {
    let processQueue = JSON.parse(JSON.stringify(processes));
    let currentTime = 0;
    let completed = [];
    let timeline = [];
    let waitingProcesses = [];

    processQueue.sort((a, b) => a.arrivalTime - b.arrivalTime);

    while (processQueue.length > 0 || waitingProcesses.length > 0) {
        while (processQueue.length > 0 && processQueue[0].arrivalTime <= currentTime) {
            waitingProcesses.push(processQueue.shift());
        }
        if (waitingProcesses.length === 0) {
            timeline.push({ name: 'BLOCKED', start: currentTime, duration: processQueue[0].arriv
alTime - currentTime, state: 'blocked' });
            currentTime = processQueue[0].arrivalTime;
            continue;
        }
        waitingProcesses.sort((a, b) => a.burstTime - b.burstTime);
        let selectedProcess = waitingProcesses.shift();
        timeline.push({ name: selectedProcess.name, start: currentTime, duration: selectedProces
s.burstTime, state: 'running' });
        selectedProcess.startTime = currentTime;
        selectedProcess.completionTime = currentTime + selectedProcess.burstTime;
        selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.arriva
lTime;
        selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burstTim
e;
        selectedProcess.lostTime = selectedProcess.startTime - selectedProcess.arrivalTime;
        selectedProcess.penalty = selectedProcess.turnaroundTime / selectedProcess.burstTime;
        selectedProcess.responseTime = selectedProcess.startTime - selectedProcess.arrivalTime;
        currentTime = selectedProcess.completionTime;
        completed.push(selectedProcess);
    }
    displayResults(completed);
    drawGanttChart(timeline);
```

Planificador SJF (Shortest Job First)

Agregar Proceso

Nombre del proceso | Tiempo de llegada | Tiempo de ráfaga | Agregar Proceso

Calcular SJF

Procesos Ingresados

Proceso: A Llegada: 0 Ráfaga: 6 Eliminar

Proceso: B Llegada: 1 Ráfaga: 8 Eliminar

Proceso: C Llegada: 2 Ráfaga: 7 Eliminar

Proceso: D Llegada: 4 Ráfaga: 3 Eliminar

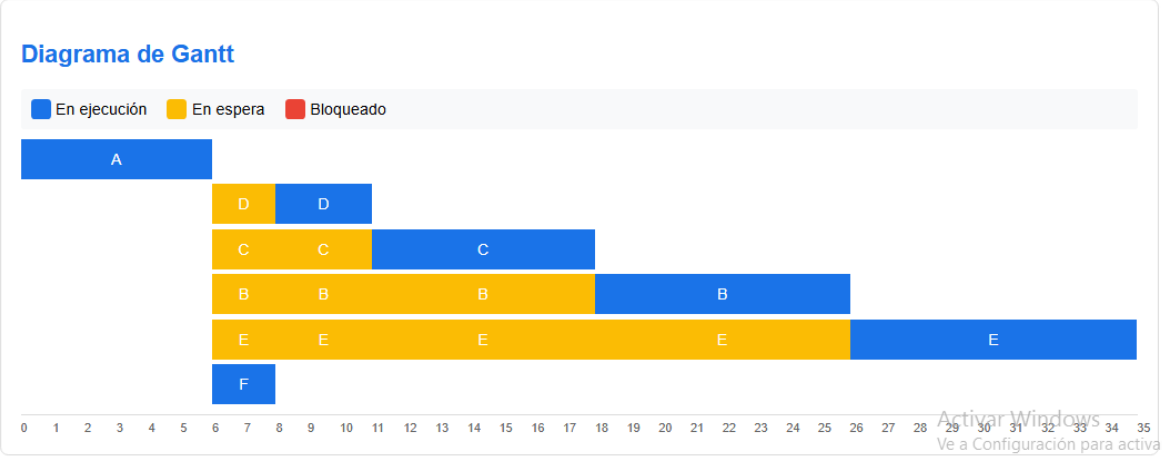
Proceso: E Llegada: 6 Ráfaga: 9 Eliminar

Proceso: F Llegada: 6 Ráfaga: 2 Eliminar

Resultados Detallados

Tiempo de Retorno Promedio	Tiempo de Espera Promedio	Tiempo Perdido Promedio	Penalidad Promedio	Tiempo de Respuesta Promedio
14.17	8.33	8.33	2.16	8.33

Proceso	Llegada	Ráfaga	Comienzo	Final	Retorno	Espera	T. Perdido	Penalidad	Respuesta
A	0	6	0	6	6	0	0	1.00	0
F	6	2	6	8	2	0	0	1.00	0
D	4	3	8	11	7	4	4	2.33	4
C	2	7	11	18	16	9	9	2.29	9
B	1	8	18	26	25	17	17	3.13	17
E	6	9	26	35	29	20	20	3.22	20



SRTF (Shortest Remaining Time First)

El algoritmo SRTF es una versión apropiativa de SJF. En cada paso, se selecciona el proceso con el menor tiempo de ráfaga restante. La implementación incluye:

1. **Entrada de datos**: Similar a los anteriores.
2. **Ordenamiento**: Los procesos se ordenan por tiempo de llegada y ráfaga restante.
3. **Ejecución**: En cada unidad de tiempo, se selecciona el proceso con el menor tiempo restante.
4. **Resultados**: Se muestran en una tabla y se genera un diagrama de Gantt.

```
function calculateSRTF() {
    let processQueue = JSON.parse(JSON.stringify(processes));
    let currentTime = 0;
    let completed = [];
    let timeline = [];
    let waitingProcesses = [];
    let remainingTime = {};

    processQueue.sort((a, b) => a.arrivalTime - b.arrivalTime);
    processQueue.forEach((p) => { remainingTime[p.name] = p.burstTime; });
```

```
    while (processQueue.length > 0 || waitingProcesses.length > 0) {
        while (processQueue.length > 0 && processQueue[0].arrivalTime <= currentTime) {
            waitingProcesses.push(processQueue.shift());
        }
        if (waitingProcesses.length === 0) {
            timeline.push({ name: "BLOCKED", start: currentTime, duration: processQueue[0].arriv
alTime - currentTime, state: "blocked" });
            currentTime = processQueue[0].arrivalTime;
            continue;
        }
        waitingProcesses.sort((a, b) => remainingTime[a.name] - remainingTime[b.name]);
        let selectedProcess = waitingProcesses[0];
        timeline.push({ name: selectedProcess.name, start: currentTime, duration: 1, state: "run
ning" });
        remainingTime[selectedProcess.name] -= 1;
        currentTime += 1;
        if (remainingTime[selectedProcess.name] === 0) {
            selectedProcess.completionTime = currentTime;
            selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.ar
rivalTime;
            selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burst
Time;
            selectedProcess.responseTime = selectedProcess.startTime - selectedProcess.arrivalTi
me;
            selectedProcess.penalty = selectedProcess.turnaroundTime / selectedProcess.burstTim
e;
            selectedProcess.lostTime = selectedProcess.startTime - selectedProcess.arrivalTime;
            completed.push(selectedProcess);
            waitingProcesses.shift();
        }
    }
    displayResults(completed);
    drawGanttChart(timeline);
```

En la aplicación web

Planificador SRTF (Short Remaining Time First)

Agregar Proceso

Nombre del proceso | Tiempo de llegad | Tiempo de ráfaga | Agregar Proceso

Calcular SRTF

Procesos Ingresados

Proceso: A Llegada: 0 Ráfaga: 6 Eliminar

Proceso: B Llegada: 1 Ráfaga: 8 Eliminar

Proceso: C Llegada: 2 Ráfaga: 7 Eliminar

Proceso: D Llegada: 4 Ráfaga: 3 Eliminar

Proceso: E Llegada: 6 Ráfaga: 9 Eliminar

Proceso: F Llegada: 6 Ráfaga: 2 Eliminar

Resultados Detallados

Tiempo de Retorno Promedio
14.17

Tiempo de Espera Promedio
8.33

Tiempo Perdido Promedio
8.33

Penalidad Promedio
2.16

Tiempo de Respuesta Promedio
8.33

Proceso	Llegada	Ráfaga	Comienzo	Final	Retorno	Espera	T. Perdido	Penalidad	Respuesta
A	0	6	0	6	6	0	0	1.00	0
F	6	2	6	8	2	0	0	1.00	0
D	4	3	8	11	7	4	4	2.33	4
C	2	7	11	18	16	9	9	2.29	9
B	1	8	18	26	25	17	17	3.13	17
E	6	9	26	35	29	20	20	3.22	20

RR (Round Robin)

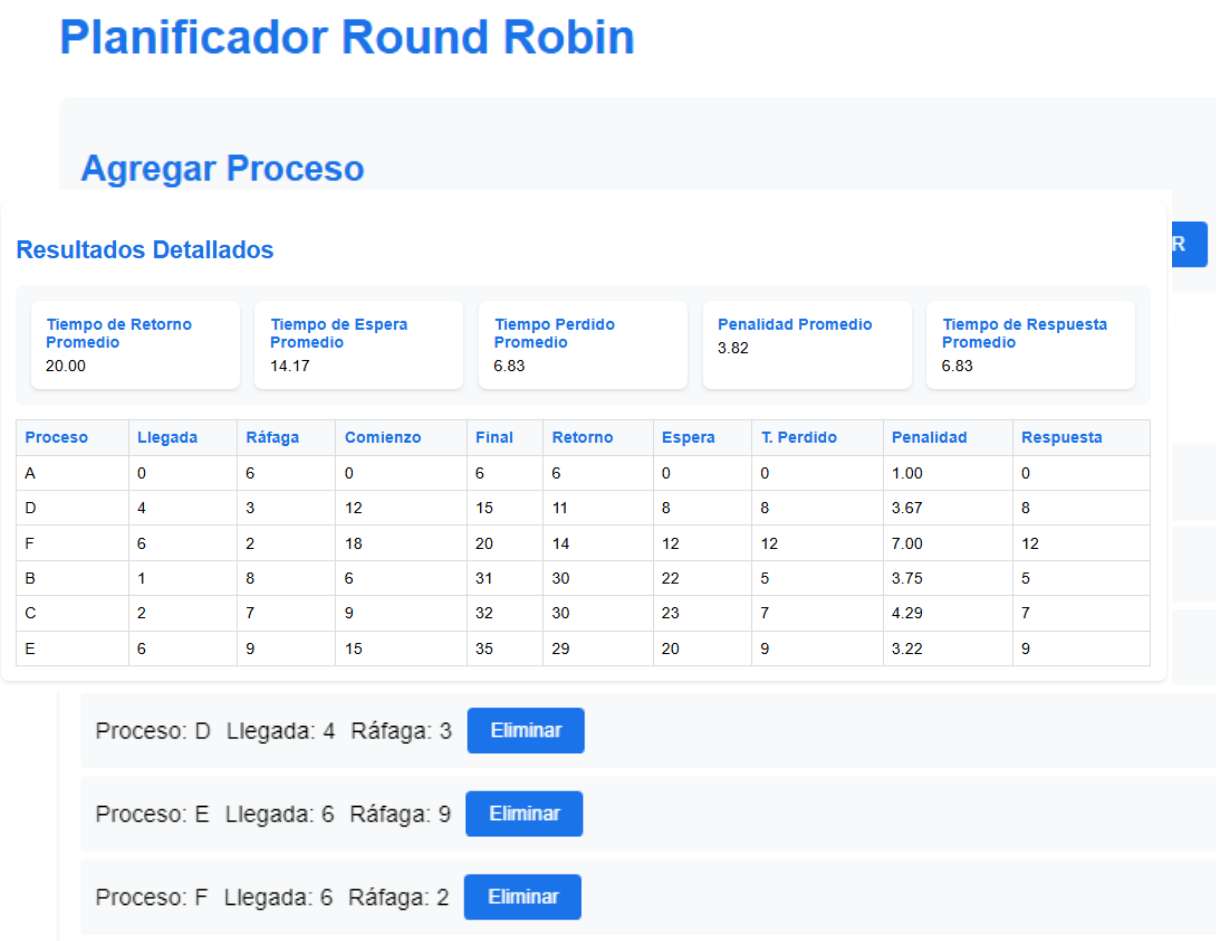
Round Robin o turno rotatorio utiliza una cola circular para ejecutar los procesos, cada proceso utiliza una unidad de tiempo llamada quantum **(q)**, cuando finaliza el quantumel proceso actual regresa a la cola de procesos listos para ejecutarse, situándose al final de la cola. La implementación incluye:

1. **Entrada de datos**: Similar a los anteriores.
2. **Ordenamiento**: Los procesos se ordenan por tiempo de llegada.
3. **Ejecución**: Los procesos se ejecutan hasta agotar su quantum asignado, luego se sitúan a la cola de procesos listos, cuando un proceso termina de ejecutarse no vuelve a ingresar a la cola de procesos listos.
4. **Resultados**: Se muestran en una tabla y se genera un diagrama de Gantt.

```
function calculateRR() {
    setQuantum(); // Configura el quantum

    if (processes.length === 0) {
        alert('Agregue al menos un proceso');
        return;
    }

    if (quantum <= 0) {
        alert('Por favor, configure un quantum válido');
        return;
    }

    let processQueue = JSON.parse(JSON.stringify(processes));
    let currentTime = 0;
    let completed = [];
    let timeline = [];
    let readyQueue = [];

    processQueue.sort((a, b) => a.arrivalTime - b.arrivalTime);

    while (processQueue.length > 0 || readyQueue.length > 0) {
        // Mover procesos que han llegado a la cola de listos
        while (processQueue.length > 0 && processQueue[0].arrivalTime <= currentTime) {
            readyQueue.push(processQueue.shift());
        }

        if (readyQueue.length === 0) {
            // Si no hay procesos listos, avanzar el tiempo
            if (processQueue.length > 0) {
                currentTime = processQueue[0].arrivalTime;
            }
            continue;
        }

        // Tomar el primer proceso de la cola
        let selectedProcess = readyQueue.shift();
```

```
        // Verificar si es la primera vez que el proceso se ejecuta
        if (selectedProcess.startTime === undefined) {
            selectedProcess.startTime = currentTime;
        }

        // Ejecutar el proceso por el quantum o el tiempo restante, lo que sea menor
        let executionTime = Math.min(selectedProcess.remainingTime, quantum);
        timeline.push({
            name: selectedProcess.name,
            start: currentTime,
            duration: executionTime,
            state: 'running'
        });

        selectedProcess.remainingTime -= executionTime;
        currentTime += executionTime;

        // Si el proceso no ha terminado, volver a agregarlo a la cola
        if (selectedProcess.remainingTime > 0) {
            readyQueue.push(selectedProcess);
        } else {
            // Calcular métricas del proceso completado
            selectedProcess.completionTime = currentTime;
            selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.ar
rivalTime;
            selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burst
Time;
            selectedProcess.lostTime = selectedProcess.startTime - selectedProcess.arrivalTime;
            selectedProcess.penalty = selectedProcess.turnaroundTime / selectedProcess.burstTim
e;
            selectedProcess.responseTime = selectedProcess.startTime - selectedProcess.arrivalTi
me;

            completed.push(selectedProcess);
        }
    }

    displayResults(completed);
    drawGanttChart(timeline);
```

5.



Resultados

En cada algoritmo, se calcularon las siguientes métricas para cada proceso:

- **Tiempo de retorno**: Tiempo total desde la llegada hasta la finalización.
- **Tiempo de espera**: Tiempo que el proceso estuvo en la cola de espera.
- **Tiempo perdido**: Tiempo desde la llegada hasta el inicio de la ejecución.
- **Penalidad**: Relación entre el tiempo de retorno y el tiempo de ráfaga.
- **Tiempo de respuesta**: Tiempo desde la llegada hasta la primera ejecución.

Además, se generó un diagrama de Gantt para visualizar la ejecución de los procesos.

6. Conclusión

La implementación de estos algoritmos demostró cómo la planificación de procesos puede afectar el rendimiento de un sistema. FCFS es simple pero puede generar largos tiempos de espera. SJF y SRTF optimizan el tiempo de espera, pero requieren más complejidad en la implementación. Este proyecto permitió

comprender la importancia de la gestión de procesos en sistemas operativos y cómo los algoritmos de planificación pueden mejorar la eficiencia del sistema.

7. Referencias

Tanenbaum, A. S. (2008). *Modern operating systems*. Prentice Hall.

https://keepcoding.io/blog/la-logica-round-robin-en-programacion/

https://en.wikipedia.org/wiki/Round-robin_scheduling

https://www.geeksforgeeks.org/first-come-first-serve-cpu-scheduling-non-preemptive/

https://www.geeksforgeeks.org/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/

https://www.tpointtech.com/os-srtf-scheduling-algorithm

https://webplusvalencia.es/que-es-el-algoritmo-srtf-y-como-funciona-guia-completa/