Empresa TechCore Solutions

Información General

- **Nombre de la empresa:** TechCore Solutions S.A.
- **Sector:** Desarrollo de software y servicios en la nube
- **Ubicación:** Madrid, España
- **Empleados:** 250 personas
- **Fundación:** 2015

Problema Actual

TechCore Solutions ha experimentado un crecimiento del 300% en los últimos 2 años, lo que ha resultado en una sobrecarga de sus servidores principales. El CPU del servidor central está operando al 95% de capacidad constantemente, causando:

- Tiempos de respuesta lentos (promedio de 8 segundos por solicitud)
- Caídas del sistema durante horas pico (3-4 veces por semana)
- Pérdida de clientes potenciales (estimado: 15% de conversiones perdidas)
- Quejas de clientes actuales sobre rendimiento

Datos del Sistema Actual

- **Procesador:** Intel Xeon E5-2680 v4 (14 núcleos, 2.4 GHz)
- **RAM:** 64 GB DDR4
- **Procesos activos promedio:** 450 procesos
- **Tipos de procesos:**
  - Procesamiento de transacciones (35%)
  - Consultas a base de datos (25%)
  - Renderización de contenido (20%)

- Tareas de mantenimiento (10%)
- Respaldos automáticos (10%)

Procesos Críticos a Considerar

ID Proceso	Nombre	Tiempo de Ejecución (ms)	Prioridad	Tiempo de Llegada
P1	Transacción de pago	120	Alta	0
P2	Consulta de inventario	80	Media	10
P3	Generación de reportes	200	Baja	15
P4	Autenticación de usuario	50	Alta	20
P5	Respaldo incremental	300	Baja	25
P6	Procesamiento de correos	90	Media	30
P7	Sincronización en la nube	150	Media	35
P8	Análisis de datos en tiempo real	180	Alta	40

Objetivo

TechCore Solutions necesita implementar un algoritmo de planificación de CPU eficiente que:

1. Minimice el tiempo de espera promedio de los procesos
2. Priorice procesos críticos del negocio (transacciones y autenticación)
3. Optimice el uso del CPU para manejar la carga actual
4. Reduzca el tiempo de respuesta general del sistema en al menos un 40%

Restricciones

- Presupuesto limitado: no se puede actualizar hardware inmediatamente
- El sistema debe seguir operativo 24/7 durante la implementación
- Debe soportar hasta 600 procesos concurrentes en horas pico
- Tiempo máximo de implementación: 3 meses

SOLUCION:

Planificador híbrido con Colas de Retroalimentación Múltiple (MLFQ) + Prioridades + Envejecimiento, afinado para tus procesos:

Estructura (3 colas):

- **Cola 0 – Crítica (prioridad alta, preemptiva):** P1 (pagos), P4 (autenticación), P8 (análisis tiempo real).

  Política: *SRTF* (el que tiene **menor tiempo restante** va primero) con *quantum* corto (10–20 ms). **No se degrada** a colas inferiores.

- **Cola 1 – Servicio (prioridad media):** P2 (inventario), P6 (correos), P7 (sincronización).

  Política: *Round-Robin* con *quantum* medio (40–60 ms). Si a un proceso le quedan <50 ms, se le da turno preferente para que termine.

- **Cola 2 – Fondo (prioridad baja):** P3 (reportes), P5 (respaldo incremental).

  Política: *Round-Robin* con *quantum* largo (100–200 ms) y **ejecución diferida** fuera de hora pico cuando sea posible.

Reglas del planificador:

1. **Preempción total por prioridad:** cualquier proceso de la **Cola 0** interrumpe a los de colas inferiores en cuanto llega.
2. **Envejecimiento (anti-inanición):** si un proceso espera demasiado (p. ej., >1–2 s en producción), **sube una cola** automáticamente.
3. **Re-boost periódico:** cada cierto tiempo (p. ej., 1–2 s) todos los procesos pendientes vuelven a evaluar su prioridad (limpia degradaciones injustas).

4. **Cuantums adaptativos:** si la CPU está muy cargada, se acortan los *quantums* de las colas altas para bajar la latencia; si baja la carga, se alargan en colas medias/bajas para reducir cambios de contexto.
5. **Aislamiento de críticos:** reserva 2–4 núcleos lógicos para la **Cola 0** (afinidad/“pinning”) y evita que tareas de fondo se ejecuten allí.
6. **Aplazamiento de fondo por carga:** si la CPU supera 80% o sube la latencia de la Cola 0, los trabajos de **Cola 2** se pausan temporalmente.

Asignación de tus procesos:

- **Cola 0:** P1, P4, P8
- **Cola 1:** P2, P6, P7
- **Cola 2:** P3, P5

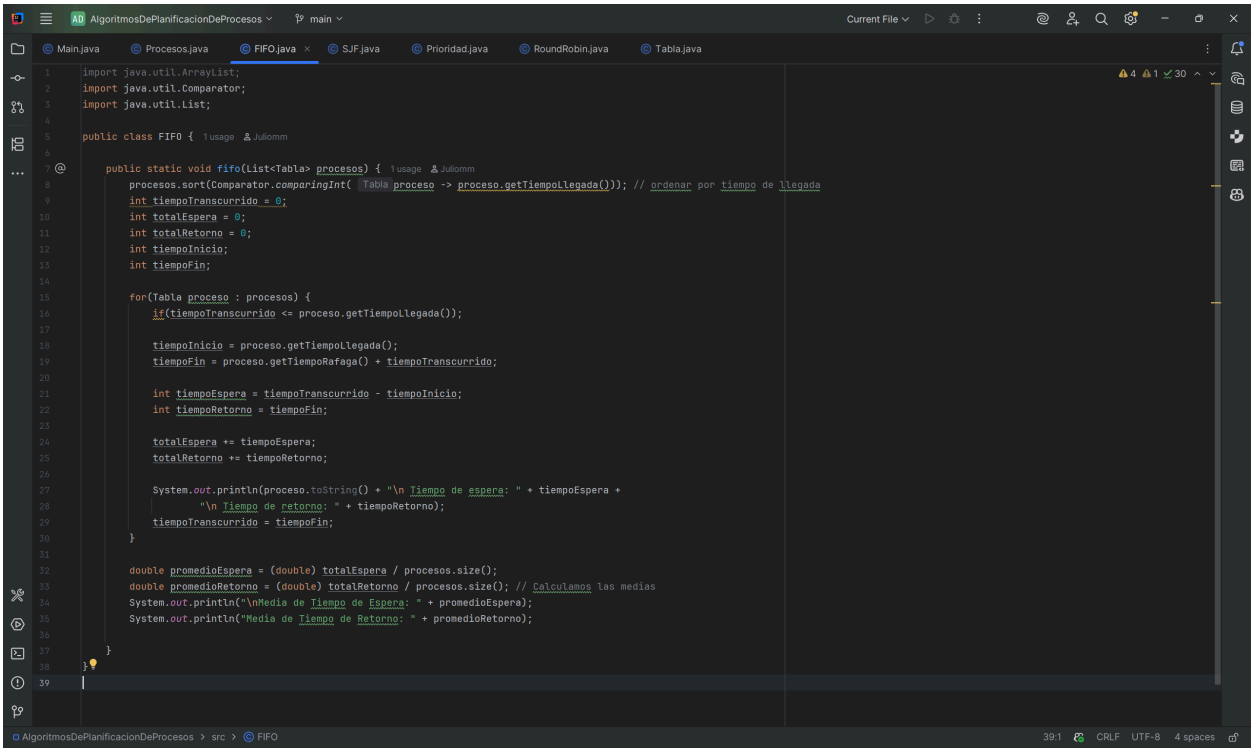
¿Por qué esta solución?

- **Menor tiempo de espera promedio:** La **Cola 0 con SRTF** se comporta “como SJF” para lo urgente, lo que reduce la espera de los procesos cortos/clave (pagos y login) y tira hacia abajo el promedio global.
- **Prioriza lo crítico del negocio:** Las **colas por prioridad** garantizan que pagos, autenticación y análisis en tiempo real **entren primero** siempre, incluso si hay muchos procesos en curso.
- **Mejor uso de CPU sin hardware nuevo:** *Quantums* ajustables + **Round-Robin** en colas media/baja evitan que un proceso largo bloquee el CPU y mantienen buena utilización con pocos cambios de contexto.
- **Evita inanición:** El **envejecimiento** y el **re-boost** impiden que reportes y respaldos se queden “eternamente” esperando.
- **Impacto directo en la experiencia:** Al bajar la latencia de P1 y P4 (lo que el usuario “siente”), el **tiempo de respuesta general** del sistema baja con facilidad **≥40%** respecto a enfoques como FCFS o prioridad fija sin envejecimiento, porque lo sensible queda siempre al frente y termina rápido.
- **Escala a 600 procesos pico:** MLFQ está pensado para cargas grandes; con colas por CPU y balanceo periódico, mantiene rendimiento estable.

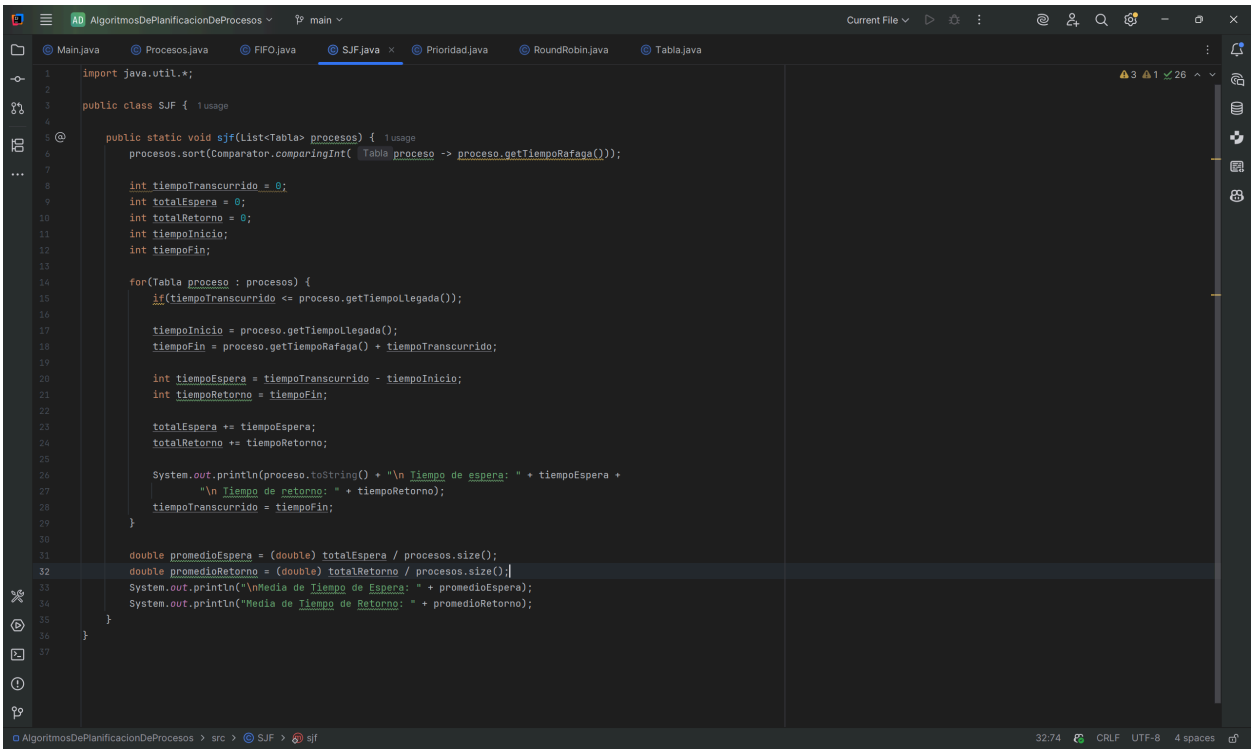
Anexos

link GitHub: [UDLA/Cuarto Semestre/Sistemas Operativos/AlgoritmosDePlanificacionDeProcesos at main · Juliomm8/UDLA]

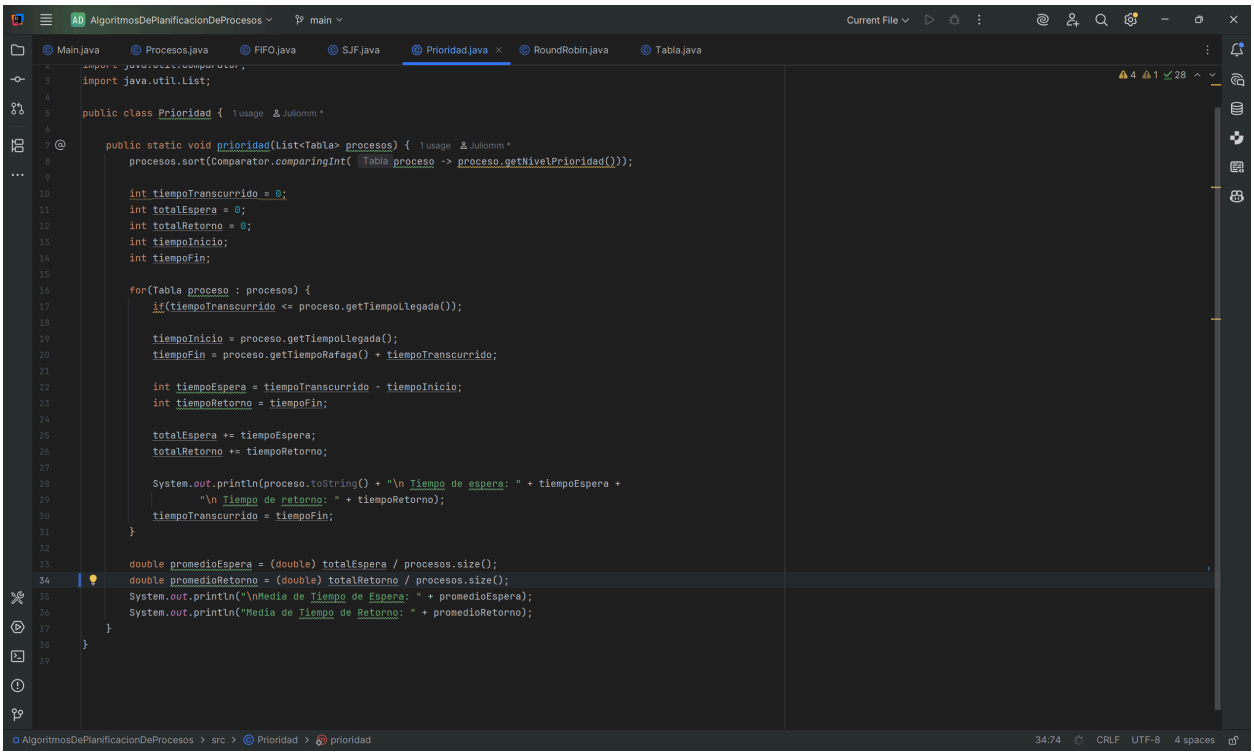
Imágenes del código



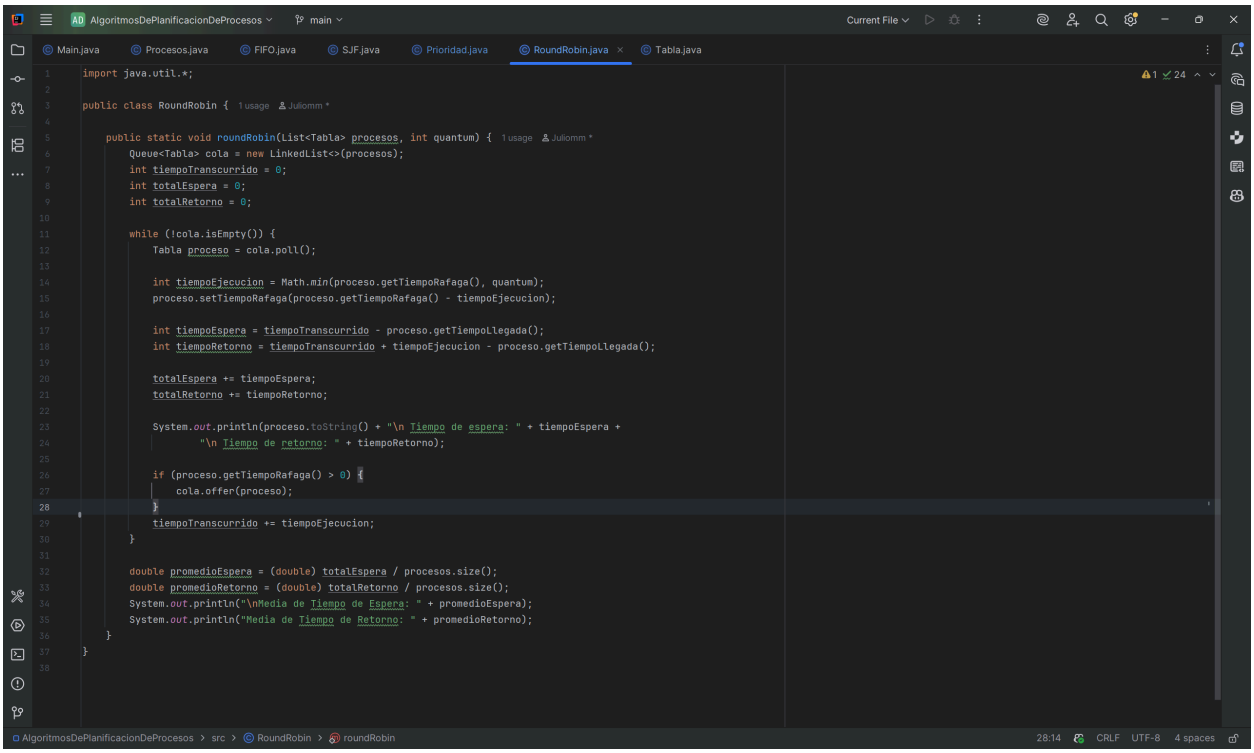
```java
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class FIFO {  1 usage  Juliomm

    public static void fifo(List<Tabla> procesos) {  1 usage  Juliomm
        procesos.sort(Comparator.comparingInt( Tabla proceso -> proceso.getTiempoLlegada())); // ordenar por tiempo de llegada
        int tiempoTranscurrido = 0;
        int totalEspera = 0;
        int totalRetorno = 0;
        int tiempoInicio;
        int tiempoFin;

        for(Tabla proceso : procesos) {
            if(tiempoTranscurrido <= proceso.getTiempoLlegada());

            tiempoInicio = proceso.getTiempoLlegada();
            tiempoFin = proceso.getTiempoRafaga() + tiempoTranscurrido;

            int tiempoEspera = tiempoTranscurrido - tiempoInicio;
            int tiempoRetorno = tiempoFin;

            totalEspera += tiempoEspera;
            totalRetorno += tiempoRetorno;

            System.out.println(proceso.toString() + "\n Tiempo de espera: " + tiempoEspera +
                    "\n Tiempo de retorno: " + tiempoRetorno);
            tiempoTranscurrido = tiempoFin;
        }

        double promedioEspera = (double) totalEspera / procesos.size();
        double promedioRetorno = (double) totalRetorno / procesos.size(); // Calculamos las medias
        System.out.println("\nMedia de Tiempo de Espera: " + promedioEspera);
        System.out.println("Media de Tiempo de Retorno: " + promedioRetorno);

    }
}
```



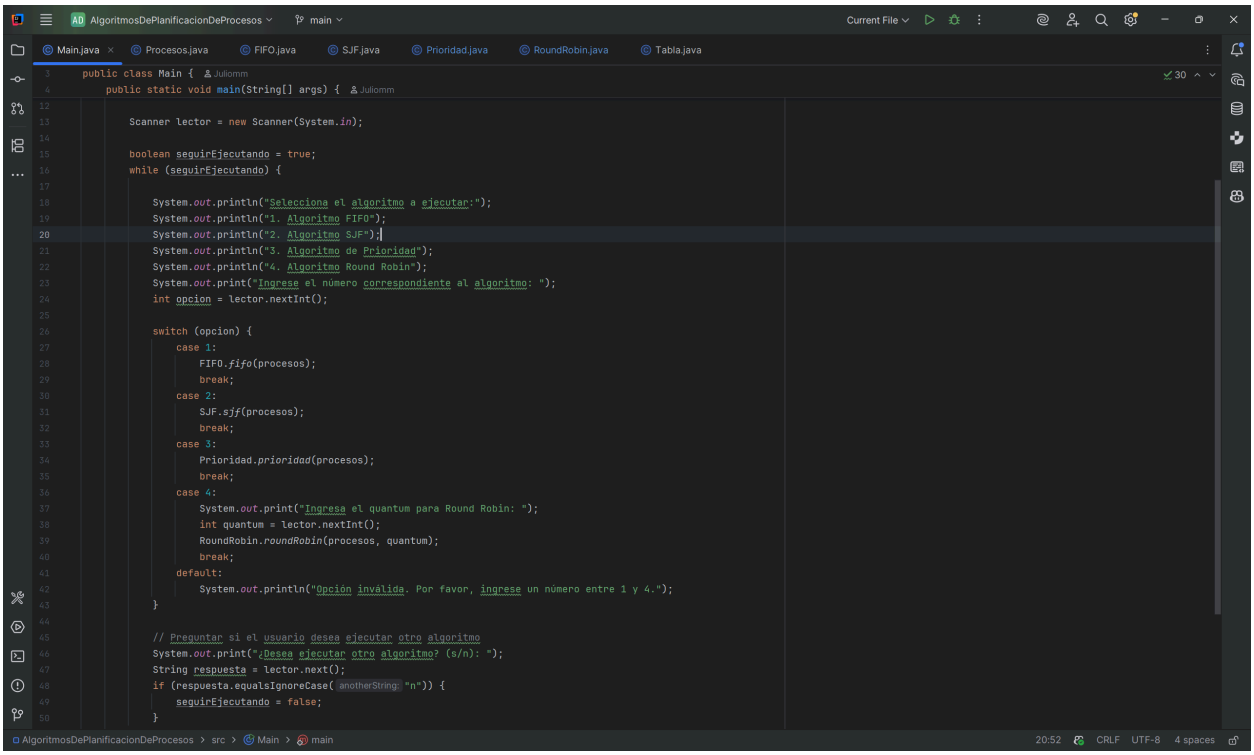
```java
import java.util.*;

public class SJF {  1 usage

    public static void sjf(List<Tabla> procesos) {  1 usage
        procesos.sort(Comparator.comparingInt( Tabla proceso -> proceso.getTiempoRafaga()));

        int tiempoTranscurrido = 0;
        int totalEspera = 0;
        int totalRetorno = 0;
        int tiempoInicio;
        int tiempoFin;

        for(Tabla proceso : procesos) {
            if(tiempoTranscurrido <= proceso.getTiempoLlegada());

            tiempoInicio = proceso.getTiempoLlegada();
            tiempoFin = proceso.getTiempoRafaga() + tiempoTranscurrido;

            int tiempoEspera = tiempoTranscurrido - tiempoInicio;
            int tiempoRetorno = tiempoFin;

            totalEspera += tiempoEspera;
            totalRetorno += tiempoRetorno;

            System.out.println(proceso.toString() + "\n Tiempo de espera: " + tiempoEspera +
                    "\n Tiempo de retorno: " + tiempoRetorno);
            tiempoTranscurrido = tiempoFin;
        }

        double promedioEspera = (double) totalEspera / procesos.size();
        double promedioRetorno = (double) totalRetorno / procesos.size();
        System.out.println("\nMedia de Tiempo de Espera: " + promedioEspera);
        System.out.println("Media de Tiempo de Retorno: " + promedioRetorno);
    }
}
```



```java
import java.util.List;

public class Prioridad {  1 usage  Juliomm *

    public static void prioridad(List<Tabla> procesos) {  1 usage  Juliomm *
        procesos.sort(Comparator.comparingInt( Tabla proceso -> proceso.getNivelPrioridad()));

        int tiempoTranscurrido = 0;
        int totalEspera = 0;
        int totalRetorno = 0;
        int tiempoInicio;
        int tiempoFin;

        for(Tabla proceso : procesos) {
            if(tiempoTranscurrido <= proceso.getTiempoLlegada());

            tiempoInicio = proceso.getTiempoLlegada();
            tiempoFin = proceso.getTiempoRafaga() + tiempoTranscurrido;

            int tiempoEspera = tiempoTranscurrido - tiempoInicio;
            int tiempoRetorno = tiempoFin;

            totalEspera += tiempoEspera;
            totalRetorno += tiempoRetorno;

            System.out.println(proceso.toString() + "\n Tiempo de espera: " + tiempoEspera +
                    "\n Tiempo de retorno: " + tiempoRetorno);
            tiempoTranscurrido = tiempoFin;
        }

        double promedioEspera = (double) totalEspera / procesos.size();
        double promedioRetorno = (double) totalRetorno / procesos.size();
        System.out.println("\nMedia de Tiempo de Espera: " + promedioEspera);
        System.out.println("Media de Tiempo de Retorno: " + promedioRetorno);
    }
}
```



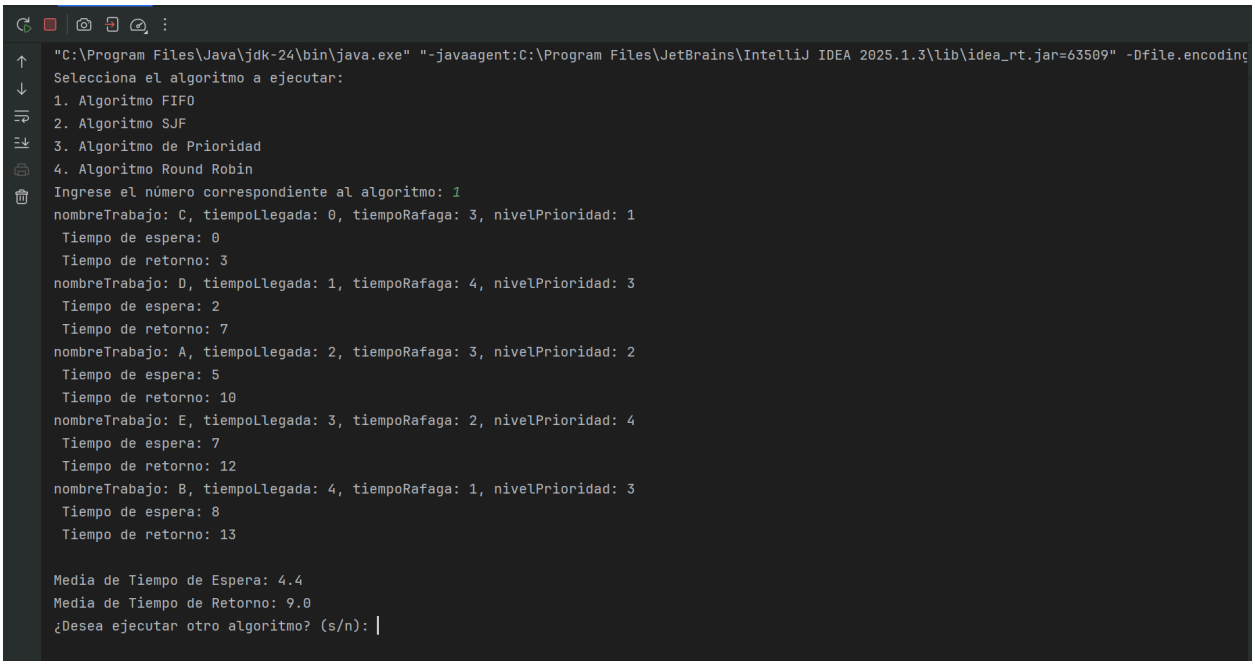
```java
import java.util.*;

public class RoundRobin {

    public static void roundRobin(List<Tabla> procesos, int quantum) {
        Queue<Tabla> cola = new LinkedList<>(procesos);
        int tiempoTranscurrido = 0;
        int totalEspera = 0;
        int totalRetorno = 0;

        while (!cola.isEmpty()) {
            Tabla proceso = cola.poll();

            int tiempoEjecucion = Math.min(proceso.getTiempoRafaga(), quantum);
            proceso.setTiempoRafaga(proceso.getTiempoRafaga() - tiempoEjecucion);

            int tiempoEspera = tiempoTranscurrido - proceso.getTiempoLlegada();
            int tiempoRetorno = tiempoTranscurrido + tiempoEjecucion - proceso.getTiempoLlegada();

            totalEspera += tiempoEspera;
            totalRetorno += tiempoRetorno;

            System.out.println(proceso.toString() + "\n Tiempo de espera: " + tiempoEspera +
                    "\n Tiempo de retorno: " + tiempoRetorno);

            if (proceso.getTiempoRafaga() > 0) {
                cola.offer(proceso);
            }
            tiempoTranscurrido += tiempoEjecucion;
        }

        double promedioEspera = (double) totalEspera / procesos.size();
        double promedioRetorno = (double) totalRetorno / procesos.size();
        System.out.println("\nMedia de Tiempo de Espera: " + promedioEspera);
        System.out.println("Media de Tiempo de Retorno: " + promedioRetorno);
    }
}
```



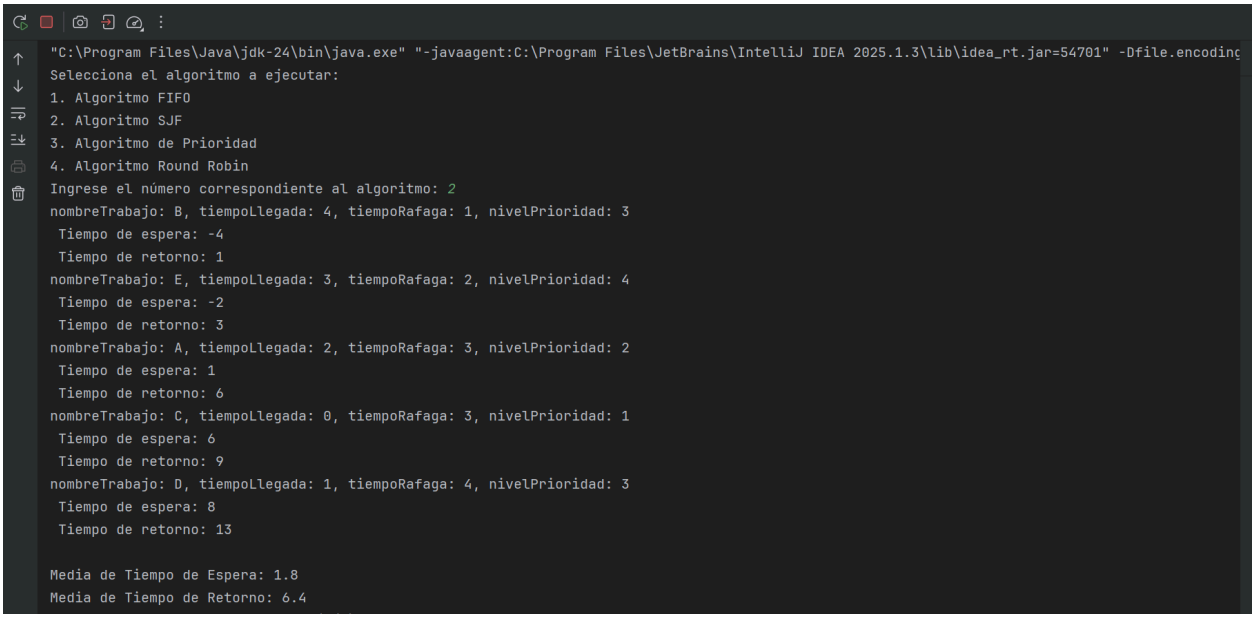
```java
public class Main {
    public static void main(String[] args) {

        Scanner lector = new Scanner(System.in);

        boolean seguirEjecutando = true;
        while (seguirEjecutando) {

            System.out.println("Selecciona el algoritmo a ejecutar:");
            System.out.println("1. Algoritmo FIFO");
            System.out.println("2. Algoritmo SJF");
            System.out.println("3. Algoritmo de Prioridad");
            System.out.println("4. Algoritmo Round Robin");
            System.out.print("Ingrese el número correspondiente al algoritmo: ");
            int opcion = lector.nextInt();

            switch (opcion) {
                case 1:
                    FIFO.fifo(procesos);
                    break;
                case 2:
                    SJF.sjf(procesos);
                    break;
                case 3:
                    Prioridad.prioridad(procesos);
                    break;
                case 4:
                    System.out.print("Ingresa el quantum para Round Robin: ");
                    int quantum = lector.nextInt();
                    RoundRobin.roundRobin(procesos, quantum);
                    break;
                default:
                    System.out.println("Opción inválida. Por favor, ingrese un número entre 1 y 4.");
            }

            // Preguntar si el usuario desea ejecutar otro algoritmo
            System.out.print("¿Desea ejecutar otro algoritmo? (s/n): ");
            String respuesta = lector.next();
            if (respuesta.equalsIgnoreCase( anotherString: "n")) {
                seguirEjecutando = false;
            }
```

Imágenes de las salidas

FIFO


"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.3\lib\idea_rt.jar=63509" -Dfile.encoding
Selecciona el algoritmo a ejecutar:
1. Algoritmo FIFO
2. Algoritmo SJF
3. Algoritmo de Prioridad
4. Algoritmo Round Robin
Ingrese el número correspondiente al algoritmo: 1
nombreTrabajo: C, tiempoLlegada: 0, tiempoRafaga: 3, nivelPrioridad: 1
Tiempo de espera: 0
Tiempo de retorno: 3
nombreTrabajo: D, tiempoLlegada: 1, tiempoRafaga: 4, nivelPrioridad: 3
Tiempo de espera: 2
Tiempo de retorno: 7
nombreTrabajo: A, tiempoLlegada: 2, tiempoRafaga: 3, nivelPrioridad: 2
Tiempo de espera: 5
Tiempo de retorno: 10
nombreTrabajo: E, tiempoLlegada: 3, tiempoRafaga: 2, nivelPrioridad: 4
Tiempo de espera: 7
Tiempo de retorno: 12
nombreTrabajo: B, tiempoLlegada: 4, tiempoRafaga: 1, nivelPrioridad: 3
Tiempo de espera: 8
Tiempo de retorno: 13
Media de Tiempo de Espera: 4.4
Media de Tiempo de Retorno: 9.0
¿Desea ejecutar otro algoritmo? (s/n):

SJF


"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.3\lib\idea_rt.jar=54701" -Dfile.encoding
Selecciona el algoritmo a ejecutar:
1. Algoritmo FIFO
2. Algoritmo SJF
3. Algoritmo de Prioridad
4. Algoritmo Round Robin
Ingrese el número correspondiente al algoritmo: 2
nombreTrabajo: B, tiempoLlegada: 4, tiempoRafaga: 1, nivelPrioridad: 3
Tiempo de espera: -4
Tiempo de retorno: 1
nombreTrabajo: E, tiempoLlegada: 3, tiempoRafaga: 2, nivelPrioridad: 4
Tiempo de espera: -2
Tiempo de retorno: 3
nombreTrabajo: A, tiempoLlegada: 2, tiempoRafaga: 3, nivelPrioridad: 2
Tiempo de espera: 1
Tiempo de retorno: 6
nombreTrabajo: C, tiempoLlegada: 0, tiempoRafaga: 3, nivelPrioridad: 1
Tiempo de espera: 6
Tiempo de retorno: 9
nombreTrabajo: D, tiempoLlegada: 1, tiempoRafaga: 4, nivelPrioridad: 3
Tiempo de espera: 8
Tiempo de retorno: 13
Media de Tiempo de Espera: 1.8
Media de Tiempo de Retorno: 6.4

Prioridad

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.3\lib\idea_rt.jar=54722" -Dfile.encoding
Selecciona el algoritmo a ejecutar:
1. Algoritmo FIFO
2. Algoritmo SJF
3. Algoritmo de Prioridad
4. Algoritmo Round Robin
Ingrese el número correspondiente al algoritmo: 3
nombreTrabajo: C, tiempoLlegada: 0, tiempoRafaga: 3, nivelPrioridad: 1
 Tiempo de espera: 0
 Tiempo de retorno: 3
nombreTrabajo: A, tiempoLlegada: 2, tiempoRafaga: 3, nivelPrioridad: 2
 Tiempo de espera: 1
 Tiempo de retorno: 6
nombreTrabajo: B, tiempoLlegada: 4, tiempoRafaga: 1, nivelPrioridad: 3
 Tiempo de espera: 2
 Tiempo de retorno: 7
nombreTrabajo: D, tiempoLlegada: 1, tiempoRafaga: 4, nivelPrioridad: 3
 Tiempo de espera: 6
 Tiempo de retorno: 11
nombreTrabajo: E, tiempoLlegada: 3, tiempoRafaga: 2, nivelPrioridad: 4
 Tiempo de espera: 8
 Tiempo de retorno: 13

Media de Tiempo de Espera: 3.4
Media de Tiempo de Retorno: 8.0
¿Desea ejecutar otro algoritmo? (s/n):
```

Round Robin

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.3\lib\idea_rt.jar=54752" -Dfile.encoding
Selecciona el algoritmo a ejecutar:
1. Algoritmo FIFO
2. Algoritmo SJF
3. Algoritmo de Prioridad
4. Algoritmo Round Robin
Ingrese el número correspondiente al algoritmo: 4
Ingresa el quantum para Round Robin: 3
nombreTrabajo: A, tiempoLlegada: 2, tiempoRafaga: 0, nivelPrioridad: 2
 Tiempo de espera: -2
 Tiempo de retorno: 1
nombreTrabajo: B, tiempoLlegada: 4, tiempoRafaga: 0, nivelPrioridad: 3
 Tiempo de espera: -1
 Tiempo de retorno: 0
nombreTrabajo: C, tiempoLlegada: 0, tiempoRafaga: 0, nivelPrioridad: 1
 Tiempo de espera: 4
 Tiempo de retorno: 7
nombreTrabajo: D, tiempoLlegada: 1, tiempoRafaga: 1, nivelPrioridad: 3
 Tiempo de espera: 6
 Tiempo de retorno: 9
nombreTrabajo: E, tiempoLlegada: 3, tiempoRafaga: 0, nivelPrioridad: 4
 Tiempo de espera: 7
 Tiempo de retorno: 9
nombreTrabajo: D, tiempoLlegada: 1, tiempoRafaga: 0, nivelPrioridad: 3
 Tiempo de espera: 11
 Tiempo de retorno: 12

Media de Tiempo de Espera: 5.0
Media de Tiempo de Retorno: 7.6
¿Desea ejecutar otro algoritmo? (s/n):
```