



TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TLAXIACO

Arquitectura De Computadoras

Alumnos:

Charly Joshua Sandoval Hernández

Edgar Axel Sandoval Hhernández

Tema:

Unidad Funcionamiento de una CPU - Simulación de administración de procesos/tareas en la CPU

Docente:

Ing. Osorio Salinas Edward.

Carrera:

Ingeniera en Sistemas Computacionales

Grupo: 5BS

Tlaxiaco, Oaxaca., A 28 de Octubre de 2024.

"Educación, Ciencia y Tecnología, Progresos día con día"®

Boulevard Tecnológico Km. 2.5, Llano Yosovee C.P. 69800. Tlaxiaco. Oax. México.

Tels. Dir. (953) 55 20788, (953) 55 21322, (953) 55 20405 e-mail:

dir_tlaxiaco@tecnm.mx | www.tlaxiaco.tecnm.mx



INTRODUCCIÓN:

En la era digital, entender el funcionamiento de una Unidad Central de Procesamiento (CPU) es fundamental para apreciar cómo los dispositivos computacionales ejecutan tareas. La CPU es el cerebro de la computadora, responsable de interpretar y ejecutar instrucciones de los programas. La administración de procesos y tareas en la CPU es una simulación que nos permite visualizar cómo se gestionan las operaciones en un entorno multitarea. Este proceso involucra la planificación de tareas, asignación de recursos y manejo de interrupciones, garantizando que el sistema operativo pueda ejecutar múltiples aplicaciones de manera eficiente y sin conflictos.

Durante la simulación de la administración de procesos, se pueden observar varios algoritmos de planificación como First Come First Serve (FCFS), Round Robin (RR), y Shortest Job Next (SJN), cada uno con sus ventajas y desventajas en términos de eficiencia y tiempo de respuesta. Esta práctica no solo ofrece una visión detallada de cómo se distribuyen las tareas dentro de la CPU, sino que también destaca la importancia de la sincronización y la comunicación entre procesos.

Además, la simulación permite comprender mejor conceptos esenciales como el cambio de contexto, donde la CPU guarda el estado de un proceso y carga el estado de otro, y las colas de espera, donde los procesos aguardan su turno para ser ejecutados. Aprender sobre la administración de procesos en la CPU proporciona una base sólida para cualquier persona interesada en la informática, ya que revela los mecanismos internos que permiten a nuestras computadoras funcionar de manera efectiva y eficiente en un mundo cada vez más demandante.

OBJETIVO:

El objetivo de esta práctica es que el alumno simule la administración de procesos/tareas en la CPU. Para ello, deberá investigar el funcionamiento de un sistema operativo y realizar un resumen de los pasos que se llevan a cabo en la administración de procesos/tareas en la CPU, estas extrategias son conocidas como scheduling y pueden ser:

First Come First Served (FCFS): Los procesos son ejecutados en el orden en el que llegan a la CPU.

Shortest Job Next (SJN): Los procesos son ejecutados en orden ascendente de acuerdo a su tiempo de ejecución.

Round Robin (RR): Los procesos son ejecutados en un orden circular, asignando un quantum de tiempo a cada proceso.

Priority Scheduling: Los procesos son ejecutados de acuerdo a su prioridad.

Multilevel Queue Scheduling: Los procesos son asignados a diferentes colas de acuerdo a su prioridad.

Multilevel Feedback Queue Scheduling: Los procesos son asignados a diferentes colas de acuerdo a su prioridad y se les asigna un quantum de tiempo.

Real Time Scheduling: Los procesos son ejecutados de acuerdo a su tiempo de ejecución.

Multicore Scheduling: Los procesos son asignados a diferentes núcleos de la CPU.

Multitasking: Los procesos son ejecutados de manera concurrente y se les asigna un quantum de tiempo.

RESUMEN DE LOS PASOS QUE SE LLEVAN A CABO EN LA ADMINISTRACIÓN DE PROCESOS/TAREAS EN LA CPU.

La administración de procesos en la CPU es un componente crítico del funcionamiento de los sistemas operativos. Aquí tienes un resumen de los pasos involucrados:

Creación del Proceso: El sistema operativo crea un nuevo proceso cuando se ejecuta una nueva tarea o aplicación, asignándole un identificador único y espacio en la memoria.

Planificación de Procesos: Se determina cuál de los procesos en espera obtendrá acceso a la CPU. Esto se realiza a través de algoritmos como FCFS, Round Robin, y SJN.

Carga en la CPU: El proceso seleccionado se carga en la CPU, copiando su estado desde la memoria.

Ejecución: La CPU ejecuta las instrucciones del proceso. Esto puede involucrar cálculos, acceso a memoria o dispositivos de E/S.

Interrupciones: Si ocurre una interrupción, la CPU detiene temporalmente el proceso en ejecución para atender el evento. Esto puede incluir interrupciones de hardware o software.

Cambio de Contexto: Si la CPU necesita cambiar de un proceso a otro, guarda el estado actual del proceso (incluyendo registros y contador de programa) y carga el estado del nuevo proceso.

Finalización: Cuando un proceso termina su ejecución, el sistema operativo libera los recursos asignados (memoria, archivos abiertos, etc.) y actualiza las estructuras de datos del sistema.

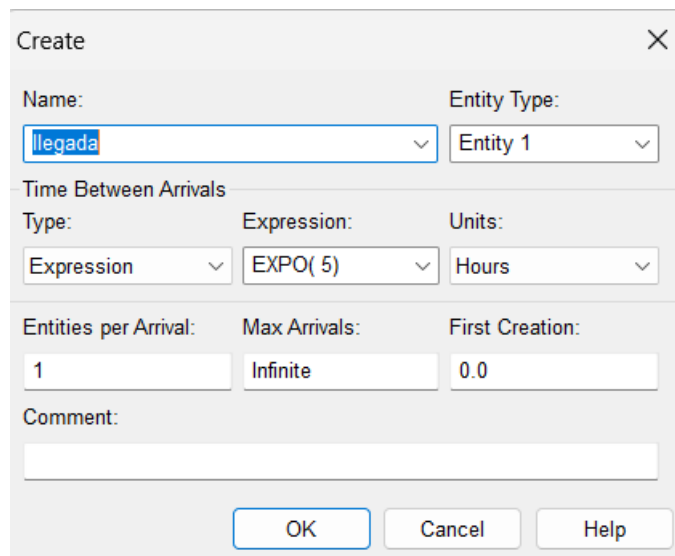
Gestión de Cola de Procesos: Los procesos se gestionan en colas, como la cola de procesos listos y la cola de procesos en espera, lo que facilita la organización y el acceso a los procesos en diferentes estados.

Sincronización de Procesos: Asegurar que los procesos se ejecuten de manera sincronizada, especialmente cuando comparten recursos, para evitar condiciones de carrera y otros problemas de concurrencia.

Manejo de Deadlocks: Monitorear y resolver situaciones donde los procesos se bloquean mutuamente esperando recursos.

Cada uno de estos pasos es crucial para mantener el sistema operativo funcionando de manera eficiente, asegurando que los recursos de la CPU se utilicen de manera óptima y que las aplicaciones se ejecuten sin problemas.

DESARROLLO:

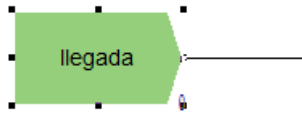


The image shows a 'Create' dialog box with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Name:** A text box containing the word 'llegada'.
- Entity Type:** A dropdown menu showing 'Entity 1'.
- Time Between Arrivals:** A section containing three sub-fields:
 - Type:** A dropdown menu showing 'Expression'.
 - Expression:** A text box containing 'EXPO(5)'.
 - Units:** A dropdown menu showing 'Hours'.
- Entities per Arrival:** A text box containing '1'.
- Max Arrivals:** A text box containing 'Infinite'.
- First Creation:** A text box containing '0.0'.
- Comment:** An empty text box.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons at the bottom.

En este proceso el primero es el ultimo en salir en los procesos de llegada del CPU.

En este apartado al elemento creat lo definomos como la entrada de los proceos.

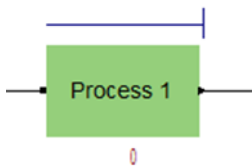


En el apartado de proces es donde resibira los elementos de entrada.

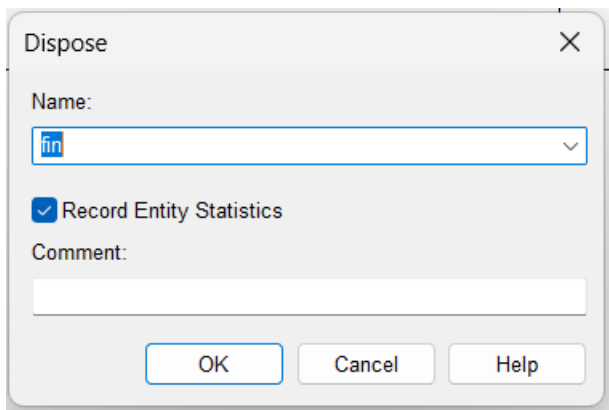
The 'Process' dialog box is shown with the following settings:

- Name: Process 1
- Type: Standard
- Logic: Action: Seize Delay, Priority: Medium(2)
- Resources: Resource, Resource 1, 1; <End of list>
- Delay Type: Triangular, Units: Hours, Allocation: Value Added
- Minimum: .5, Value(Most): 8, Maximum: 12
- Report Statistics: ☒
- Comment: (empty text box)

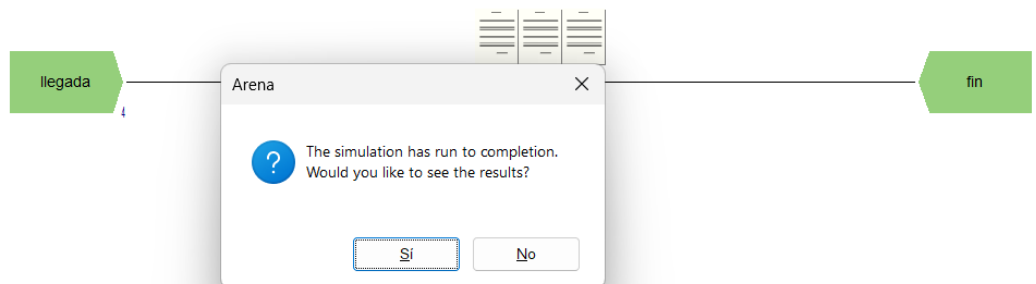
Buttons at the bottom: OK, Cancel, Help.



En el apartado de fin es donde el primero es el ultimo en salir.



En este apartado observamos el proceso de la simulación del CPU.



First Come First Served (FCFS): Los procesos son ejecutados en el orden en el que llegan a la CPU.

Estrategia 1: Primero en llegar, primero en ser atendido (FCFS)

En FCFS , los procesos se ejecutan en el orden en que llegan. No hay cambios en la ordenación de los procesos; el primero en entrar es el primero en salir. Vamos a crear una clase SimulacionFCFS para implementar esta estrategia.

```
import java.util.LinkedList;
```

```
import java.util.Queue;
```

```
class Proceso {
```

```
    int id;
```

```
    int tiempoEjecucion;
```

```
    public Proceso(int id, int tiempoEjecucion) {
```

```
        this.id = id;
```

```
        this.tiempoEjecucion = tiempoEjecucion;
```

```
    }
```

```
}
```

```
public class SimulacionFCFS {
```

```
    private Queue<Proceso> colaProcesos;
```

```
    public SimulacionFCFS() {
```

```
        this.colaProcesos = new LinkedList<>();
```

```
    }
```

```
    public void agregarProceso(Proceso proceso) {
```

```
        colaProcesos.add(proceso);
```

```
    }
```

```
    public void ejecutarProcesos() {
```



```

while (!colaProcesos.isEmpty()) {
    Proceso procesoActual = colaProcesos.poll();

    System.out.println("Ejecutando proceso ID: " + procesoActual.id + " con
tiempo de ejecución de " + procesoActual.tiempoEjecucion + " unidades de
tiempo.");

    // Simulación de tiempo de ejecución
    for (int i = 1; i <= procesoActual.tiempoEjecucion; i++) {
        System.out.println("Proceso ID " + procesoActual.id + " ejecutando...
Tiempo transcurrido: " + i + " de " + procesoActual.tiempoEjecucion);

        try {
            Thread.sleep(1000); // Pausa de 1 segundo para simular el paso del
tiempo
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.out.println("La ejecución fue interrumpida.");
        }
    }

    System.out.println("Proceso ID " + procesoActual.id + " completado.");
    System.out.println("-----");
}

}

public static void main(String[] args) {
    SimulacionFCFS simulacion = new SimulacionFCFS();

    // Agregar procesos con tiempos de ejecución
    simulacion.agregarProceso(new Proceso(1, 5));

```

```

simulacion.agregarProceso(new Proceso(2, 3));
simulacion.agregarProceso(new Proceso(3, 8));

// Ejecutar la simulación
simulacion.ejecutarProcesos();
}
}

```

```

-----< com.mycompany:mavenproject3 >-----
Building mavenproject3 1.0-SNAPSHOT
from pom.xml
-----[ jar ]-----

```

```

--- resources:3.3.1:resources (default-resources) @ mavenproject3 ---
skip non existing resourceDirectory
C:\Users\sando\OneDrive\Documentos\NetBeansProjects\mavenproject3\src\main\
resources

```

```

--- compiler:3.11.0:compile (default-compile) @ mavenproject3 ---
Changes detected - recompiling the module! :source
Compiling 5 source files with javac [debug target 22] to target\classes

```

```

--- exec:3.1.0:exec (default-cli) @ mavenproject3 ---
Ejecutando proceso ID: 1 con tiempo de ejecución de 5 unidades de tiempo.
Proceso ID 1 ejecutando... Tiempo transcurrido: 1 de 5
Proceso ID 1 ejecutando... Tiempo transcurrido: 2 de 5
Proceso ID 1 ejecutando... Tiempo transcurrido: 3 de 5
Proceso ID 1 ejecutando... Tiempo transcurrido: 4 de 5
Proceso ID 1 ejecutando... Tiempo transcurrido: 5 de 5

```

Proceso ID 1 completado.

Ejecutando proceso ID: 2 con tiempo de ejecución de 3 unidades de tiempo.

Proceso ID 2 ejecutando... Tiempo transcurrido: 1 de 3

Proceso ID 2 ejecutando... Tiempo transcurrido: 2 de 3

Proceso ID 2 ejecutando... Tiempo transcurrido: 3 de 3

Proceso ID 2 completado.

Ejecutando proceso ID: 3 con tiempo de ejecución de 8 unidades de tiempo.

Proceso ID 3 ejecutando... Tiempo transcurrido: 1 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 2 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 3 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 4 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 5 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 6 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 7 de 8

Proceso ID 3 ejecutando... Tiempo transcurrido: 8 de 8

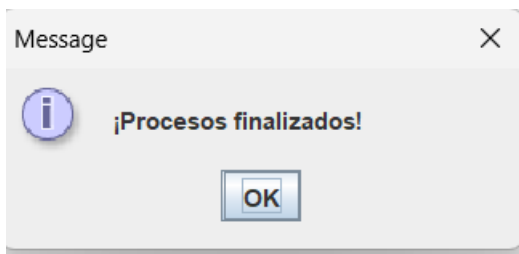
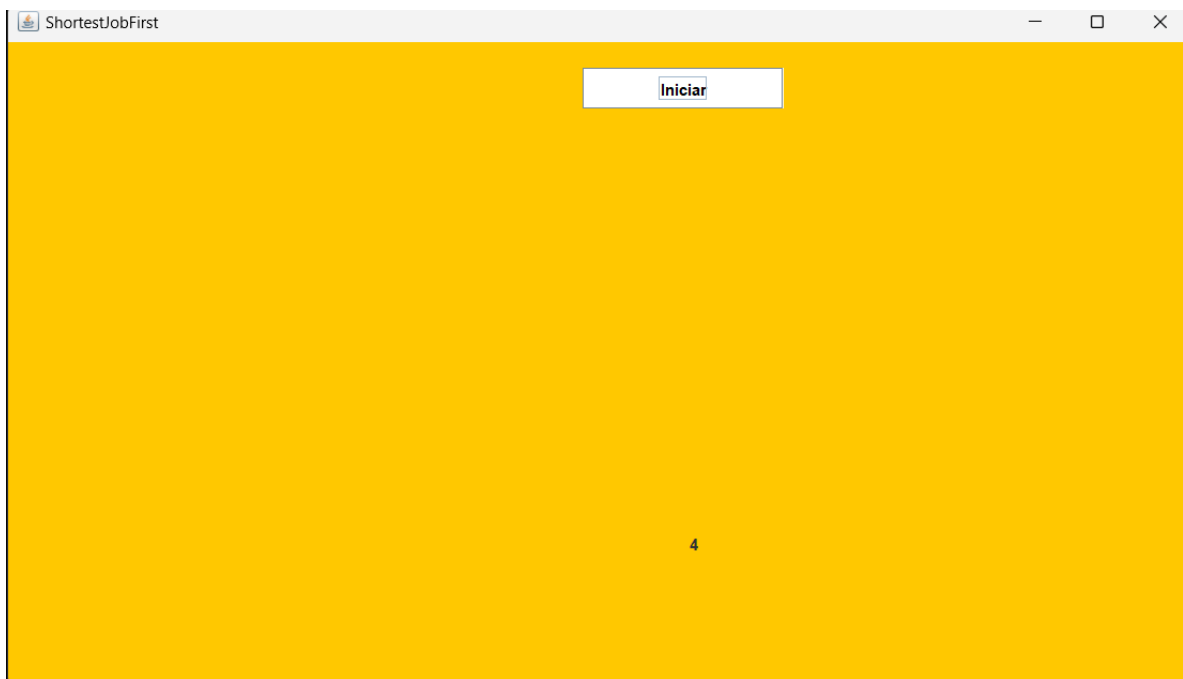
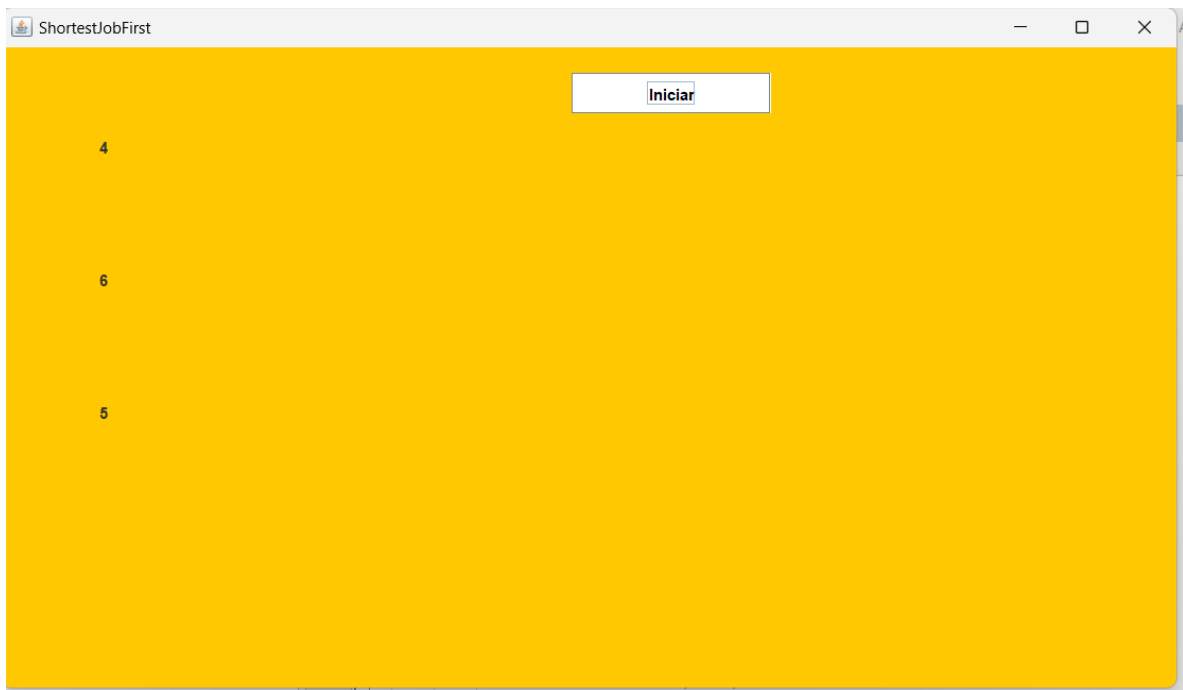
Proceso ID 3 completado.

BUILD SUCCESS

Total time: 18.593 s

Finished at: 2024-10-28T09:37:52-06:00

Shortest Job Next (SJN): Los procesos son ejecutados en orden ascendente de acuerdo a su tiempo de ejecución.



```
package ShortestJobFirst;
```

```
import java.awt.Color;
```

```
import java.awt.Image;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.util.ArrayList;
```

```
import java.util.Collections;
```

```
import java.util.List;
```

```
import java.util.Random;
```

```
import java.util.concurrent.ExecutorService;
```

```
import java.util.concurrent.Executors;
```

```
import javax.swing.ImageIcon;
```

```
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
```

```
import javax.swing.JOptionPane;
```

```
import javax.swing.SwingUtilities;
```

```
public class ShortestJobFirst extends JFrame {
```

```
    private Random random = new Random();
```

```
    private List<JLabel> procesos = new ArrayList<>();
```

```
    private int finishLine = 0;
```

```
    private ExecutorService executor;
```

```
    private List<JLabel> startOrder;
```

```
    private JButton btnStart;
```

```
public ShortestJobFirst() {  
    setTitle("ShortestJobFirst");  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    getContentPane().setBackground(Color.ORANGE);  
    setLayout(null);  
    setSize(900, 520);  
  
    // Añadir procesos a la pantalla  
    for (int i = 0; i < 5; i++) {  
        ImageIcon procesolcon = new ImageIcon("carpeta.gif");  
        JLabel proceso = new JLabel(procesolcon);  
        proceso.setBounds(160, 100 * i, 150, 150);  
        procesos.add(proceso);  
        add(proceso);  
    }  
  
    ImageIcon runnerlcon = new ImageIcon("cpu2.gif");  
    JLabel runner = new JLabel(runnerlcon);  
    runner.setBounds(695, 30, 170, 450);  
    add(runner);  
  
    // Crear las líneas de referencia  
    ImageIcon horizontallcon = new ImageIcon("Imagen1.png");  
    Image image = horizontallcon.getImage();  
    Image newImage = image.getScaledInstance(690, 9,  
Image.SCALE_SMOOTH);  
    horizontallcon = new ImageIcon(newImage);  
  
    JLabel horizontal = new JLabel(horizontallcon);
```

```
horizontal.setBounds(15, 60, 690, 90);  
add(horizontal);
```

```
ImageIcon linealcon = new ImageIcon("Imagen2.png");  
Image imagen2 = linealcon.getImage();  
Image newImagen = imagen2.getScaledInstance(690, 9,  
Image.SCALE_SMOOTH);  
linealcon = new ImageIcon(newImagen);
```

```
JLabel linea = new JLabel(linealcon);  
linea.setBounds(15, 150, 690, 90);  
add(linea);
```

```
ImageIcon linea2Icon = new ImageIcon("Imagen3.png");  
Image imagen3 = linea2Icon.getImage();  
Image newImagen3 = imagen3.getScaledInstance(690, 9,  
Image.SCALE_SMOOTH);  
linea2Icon = new ImageIcon(newImagen3);
```

```
JLabel linea2 = new JLabel(linea2Icon);  
linea2.setBounds(15, 250, 690, 90);  
add(linea2);
```

```
ImageIcon linea3Icon = new ImageIcon("Imagen4.png");  
Image imagen4 = linea3Icon.getImage();  
Image newImagen4 = imagen4.getScaledInstance(690, 9,  
Image.SCALE_SMOOTH);  
linea3Icon = new ImageIcon(newImagen4);
```

```

JLabel linea3 = new JLabel(linea3Icon);
linea3.setBounds(15, 350, 690, 90);
add(linea3);

// Botón para iniciar los procesos
btnStart = new JButton("Iniciar");
btnStart.setBackground(Color.white);
btnStart.setForeground(Color.black);
btnStart.setBounds(430, 20, 150, 30);
add(btnStart);

btnStart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        startRace();
    }
});
}

private void assignRandomNumbers() {
    List<Integer> numerosAsignados = new ArrayList<>();
    for (int i = 0; i <= 7; i++) {
        numerosAsignados.add(i);
    }
    Collections.shuffle(numerosAsignados);

    for (int i = 0; i < procesos.size(); i++) {
        JLabel proceso = procesos.get(i);
        int numero = numerosAsignados.get(i);

```



```
        proceso.setText(String.valueOf(numero)); // Número asignado internamente
```

```
    }  
}
```

```
private void startRace() {
```

```
    // Asignar los números antes de comenzar la carrera
```

```
    assignRandomNumbers();
```

```
    // Reiniciar las posiciones de los corredores
```

```
    for (JLabel runner : procesos) {
```

```
        runner.setLocation(0, runner.getY());
```

```
    }
```

```
    finishLine = getWidth() - 25;
```

```
    executor = Executors.newFixedThreadPool(1);
```

```
    startOrder = new ArrayList<>(procesos);
```

```
    // Ordenar los procesos en función de los números asignados
```

```
    Collections.sort(startOrder, (a, b) -> {
```

```
        int numA = Integer.parseInt(a.getText());
```

```
        int numB = Integer.parseInt(b.getText());
```

```
        return Integer.compare(numA, numB);
```

```
    });
```

```
    startNextProceso();
```

```
}
```

```
private void startNextProceso() {
```

```

if (startOrder.isEmpty()) {
    executor.shutdown();
    JOptionPane.showMessageDialog(null, "¡Procesos finalizados!");

    btnStart.setEnabled(true);
} else {
    JLabel proceso = startOrder.remove(0);

    // Mostrar el proceso antes de iniciar la animación
    proceso.setVisible(true);

    // Introducir un retraso para que el proceso aparezca
    SwingUtilities.invokeLater(() -> {
        try {
            Thread.sleep(500); // Retraso en milisegundos
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        executor.submit(() -> {
            moveProceso(proceso);
            startNextProceso();
        });
    });
}

private void moveProceso(JLabel proceso) {
    while (proceso.getX() < finishLine) {

```

```

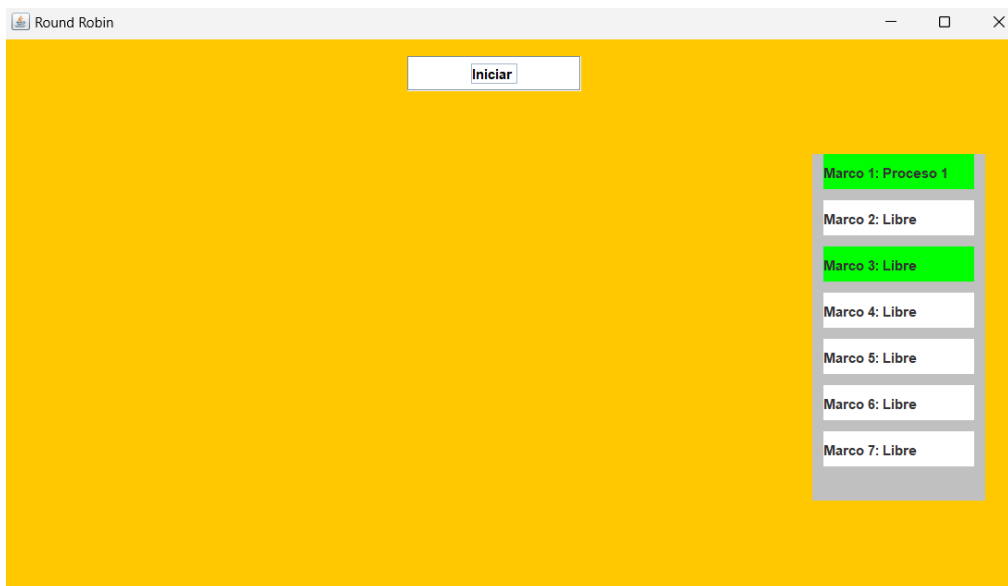
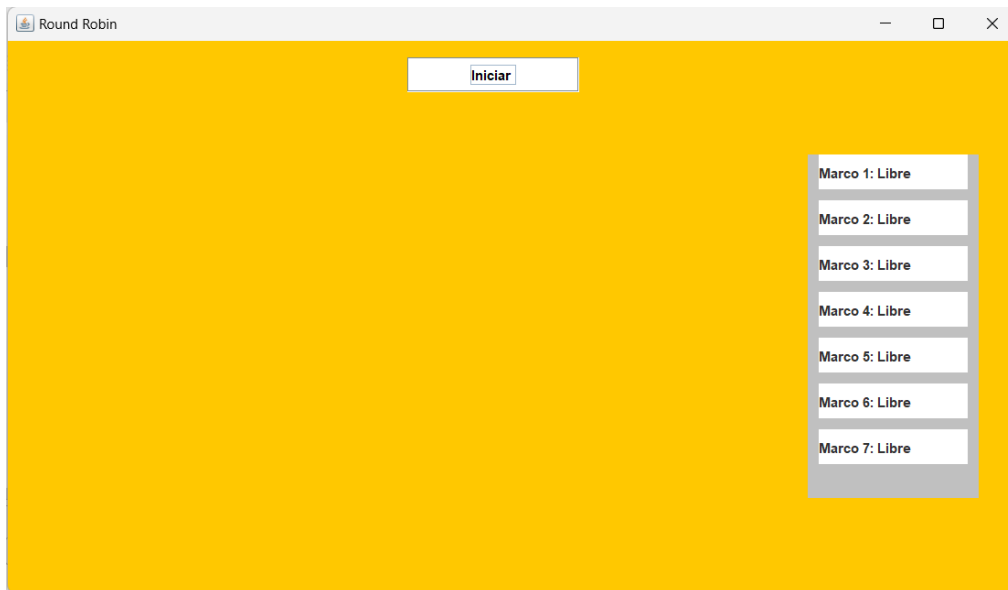
int move = random.nextInt(20) + 5;
try {
    Thread.sleep(50);
} catch (InterruptedException e) {
    e.printStackTrace();
}
proceso.setLocation(proceso.getX() + move, proceso.getY());
}

// Hacer que el proceso desaparezca una vez que haya terminado
proceso.setVisible(false);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        ShortestJobFirst form = new ShortestJobFirst();
        form.setVisible(true);
    });
}
}

```

Round Robin (RR): Los procesos son ejecutados en un orden circular, asignando un quantum de tiempo a cada proceso.



```
package RoundRobinn;
```

```
import java.awt.Color;
```

```
import java.awt.Image;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Random;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class RoundRobin extends JFrame {

    private Random random = new Random();
    private List<JLabel> processes = new ArrayList<>();
    private List<JLabel> pageFrames = new ArrayList<>(); // Páginas
    private int quantum = 100; // Tiempo de quantum en milisegundos
    private int finishLine = 0;
    private Thread[] threads;
    private boolean[] finished;
    private int[] positions;

    public RoundRobin() {

        setTitle("Round Robin");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setBackground(Color.ORANGE);
        setLayout(null);
```

```
setSize(900, 520);
```

```
// Añadir los procesos en la pantalla, pero ocultarlos inicialmente
```

```
for (int i = 0; i < 5; i++) {
```

```
    ImageIcon processIcon = new ImageIcon("Carpeta.gif");
```

```
    JLabel process = new JLabel(processIcon);
```

```
    process.setBounds(10, 100 * i, 150, 150);
```

```
    process.setVisible(false); // Ocultar los procesos al principio
```

```
    processes.add(process);
```

```
    add(process);
```

```
}
```

```
ImageIcon cpuIcon = new ImageIcon("cpu2.gif");
```

```
JLabel cpu = new JLabel(cpuIcon);
```

```
cpu.setBounds(495, 30, 170, 450);
```

```
add(cpu);
```

```
// Botón para iniciar los procesos
```

```
JButton btnStart = new JButton("Iniciar ");
```

```
btnStart.setBackground(Color.white);
```

```
btnStart.setForeground(Color.black);
```

```
btnStart.setBounds(350, 15, 150, 30);
```

```
add(btnStart);
```

```
btnStart.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        startSimulation();
```

```
    }
```

```

});

// Crear panel para los marcos de página
JPanel pagePanel = new JPanel();
pagePanel.setBounds(700, 100, 150, 300);
pagePanel.setLayout(null);
pagePanel.setBackground(Color.LIGHT_GRAY);
add(pagePanel);

// Añadir marcos de página
for (int i = 0; i < 7; i++) {
    JLabel pageFrame = new JLabel("Marco " + (i + 1) + ": Libre");
    pageFrame.setBounds(10, 40 * i, 130, 30);
    pageFrame.setOpaque(true);
    pageFrame.setBackground(Color.WHITE);
    pageFrames.add(pageFrame);
    pagePanel.add(pageFrame);
}

// Líneas visuales de referencia
createReferenceLines();
}

private void createReferenceLines() {
    for (int i = 0; i < 4; i++) {
        ImageIcon linelcon = new ImageIcon("Imagen" + (i + 1) + ".png"); //
        Asegúrate de que estas imágenes existen
        Image image = linelcon.getImage();
    }
}

```

```
        Image newImage = image.getScaledInstance(690, 9,
Image.SCALE_SMOOTH);
```

```
        lineIcon = new ImageIcon(newImage);
```

```
        JLabel line = new JLabel(lineIcon);
```

```
        line.setBounds(15, 60 + (100 * i), 690, 90);
```

```
        add(line);
```

```
    }
```

```
}
```

```
private void startSimulation() {
```

```
    // Reiniciar las posiciones de los procesos y ocultarlos
```

```
    for (JLabel process : processes) {
```

```
        process.setLocation(0, process.getY());
```

```
        process.setVisible(false); // Ocultar todos los procesos al inicio
```

```
    }
```

```
finishLine = getWidth() - 30; // Definir la línea de meta a la derecha del panel
```

```
threads = new Thread[processes.size()];
```

```
finished = new boolean[processes.size()];
```

```
positions = new int[processes.size()];
```

```
// Iniciar cada proceso en su propio hilo
```

```
for (int i = 0; i < processes.size(); i++) {
```

```
    final int index = i;
```

```
    threads[index] = new Thread(() -> moveProcess(index));
```

```
    threads[index].start();
```

```
}
```

```
}
```



```

private void moveProcess(int index) {
    JLabel process = processes.get(index);

    // Mostrar el proceso antes de que comience a moverse
    SwingUtilities.invokeLater(() -> process.setVisible(true));

    while (positions[index] < finishLine) {
        int moveSize = random.nextInt(6) + 1; // Tamaño variable entre 1 y 6
        try {
            Thread.sleep(quantum); // Simula el quantum

            positions[index] += moveSize * 10; // Mueve el proceso según el tamaño
            de la página

            SwingUtilities.invokeLater(() -> process.setLocation(positions[index],
            process.getY()));

            // Actualizar marcos de página
            updatePageFrames(index);

            // Si el proceso alcanza o supera la meta
            if (positions[index] >= finishLine) {
                finished[index] = true;

                // Hacer que el proceso desaparezca
                SwingUtilities.invokeLater(() -> process.setVisible(false));
                break;
            }
        }
    }
}

```

```

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

// Verificar si todos los procesos han terminado
if (allFinished()) {
    JOptionPane.showMessageDialog(this, "¡Procesos completados!");
}
}

private void updatePageFrames(int index) {
    for (int j = 0; j < pageFrames.size(); j++) {
        JLabel frame = pageFrames.get(j);
        if (j == index && !finished[j]) {
            frame.setText("Marco " + (j + 1) + ": Proceso " + (index + 1));
            frame.setBackground(Color.GREEN); // Verde si está ocupado
        } else if (!finished[j]) {
            frame.setText("Marco " + (j + 1) + ": Libre");
            frame.setBackground(Color.WHITE); // Libre si no está ocupado
        }

        if (finished[j]) {
            frame.setText("Marco " + (j + 1) + ": Libre");
            frame.setBackground(Color.WHITE); // Libre si ya ha terminado
        }
    }
}
}

```

```

private boolean allFinished() {
    for (boolean finish : finished) {
        if (!finish) {
            return false;
        }
    }
    return true;
}

```

```

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        RoundRobin form = new RoundRobin();
        form.setVisible(true);
    });
}
}

```

```

-----
package com.mycompany.mavenproject3;

```

```

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

```

```

public class SimulacionPriorityScheduling {
    private List<Proceso> listaProcesos;

    public SimulacionPriorityScheduling() {

```

```

        this.listaProcesos = new ArrayList<>();
    }

    public void agregarProceso(Proceso proceso) {
        listaProcesos.add(proceso);
    }

    public void ejecutarProcesos() {
        // Ordenar la lista de procesos por prioridad (menor prioridad primero)
        listaProcesos.sort(Comparator.comparingInt(p -> p.prioridad));

        for (Proceso procesoActual : listaProcesos) {
            System.out.println("Ejecutando proceso ID: " + procesoActual.id + " con
prioridad " + procesoActual.prioridad + " y tiempo de ejecución de " +
procesoActual.tiempoEjecucion + " unidades de tiempo.");

            // Simulación de tiempo de ejecución
            for (int i = 1; i <= procesoActual.tiempoEjecucion; i++) {
                System.out.println("Proceso ID " + procesoActual.id + " ejecutando...
Tiempo transcurrido: " + i + " de " + procesoActual.tiempoEjecucion);

                try {
                    Thread.sleep(1000); // Pausa de 1 segundo para simular el paso del
tiempo
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    System.out.println("La ejecución fue interrumpida.");
                }
            }
        }
    }

```

```

        System.out.println("Proceso ID " + procesoActual.id + " completado.");
        System.out.println("-----");
    }
}

public static void main(String[] args) {
    SimulacionPriorityScheduling simulacion = new
SimulacionPriorityScheduling();

    // Agregar procesos con tiempos de ejecución y prioridad
    simulacion.agregarProceso(new Proceso(1, 4, 2)); // Proceso ID 1 con
prioridad 2
    simulacion.agregarProceso(new Proceso(2, 3, 1)); // Proceso ID 2 con
prioridad 1 (más alta)
    simulacion.agregarProceso(new Proceso(3, 5, 3)); // Proceso ID 3 con
prioridad 3 (más baja)

    // Ejecutar la simulación
    simulacion.ejecutarProcesos();
}
}

```

-----< com.mycompany:mavenproject3 >-----

Building mavenproject3 1.0-SNAPSHOT

from pom.xml

-----[jar]-----

--- resources:3.3.1:resources (default-resources) @ mavenproject3 ---

skip non existing resourceDirectory
C:\Users\sando\OneDrive\Documentos\NetBeansProjects\mavenproject3\src\main\resources

--- compiler:3.11.0:compile (default-compile) @ mavenproject3 ---

Changes detected - recompiling the module! :source

Compiling 7 source files with javac [debug target 22] to target\classes

--- exec:3.1.0:exec (default-cli) @ mavenproject3 ---

Ejecutando proceso ID: null con prioridad null y tiempo de ejecución de 0 unidades de tiempo.

Proceso ID null completado.

Ejecutando proceso ID: null con prioridad null y tiempo de ejecución de 0 unidades de tiempo.

Proceso ID null completado.

Ejecutando proceso ID: null con prioridad null y tiempo de ejecución de 0 unidades de tiempo.

Proceso ID null completado.

BUILD SUCCESS

Total time: 2.496 s

Finished at: 2024-10-28T10:17:15-06:00

Priority Scheduling: Los procesos son ejecutados de acuerdo a su prioridad

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

class Proceso {
    int id;
    int tiempoEjecucion;
    int prioridad;

    public Proceso(int id, int tiempoEjecucion, int prioridad) {
        this.id = id;
        this.tiempoEjecucion = tiempoEjecucion;
        this.prioridad = prioridad;
    }
}

public class SimulacionPriorityScheduling {
    private List<Proceso> listaProcesos;

    public SimulacionPriorityScheduling() {
        this.listaProcesos = new ArrayList<>();
    }

    public void agregarProceso(Proceso proceso) {
        listaProcesos.add(proceso);
    }
}
```

```

public void ejecutarProcesos() {
    // Ordenar la lista de procesos por prioridad (menor prioridad primero)
    listaProcesos.sort(Comparator.comparingInt(p -> p.prioridad));

    for (Proceso procesoActual : listaProcesos) {
        System.out.println("Ejecutando proceso ID: " + procesoActual.id + " con
prioridad " + procesoActual.prioridad + " y tiempo de ejecución de " +
procesoActual.tiempoEjecucion + " unidades de tiempo.");

        // Simulación de tiempo de ejecución
        for (int i = 1; i <= procesoActual.tiempoEjecucion; i++) {
            System.out.println("Proceso ID " + procesoActual.id + " ejecutando...
Tiempo transcurrido: " + i + " de " + procesoActual.tiempoEjecucion);

            try {
                Thread.sleep(1000); // Pausa de 1 segundo para simular el paso del
tiempo
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("La ejecución fue interrumpida.");
            }
        }

        System.out.println("Proceso ID " + procesoActual.id + " completado.");
        System.out.println("-----");
    }
}

public static void main(String[] args) {

```



```

        SimulacionPriorityScheduling simulacion = new
SimulacionPriorityScheduling();

        // Agregar procesos con tiempos de ejecución y prioridad

        simulacion.agregarProceso(new Proceso(1, 4, 2)); // Proceso ID 1 con
prioridad 2

        simulacion.agregarProceso(new Proceso(2, 3, 1)); // Proceso ID 2 con
prioridad 1 (más alta)

        simulacion.agregarProceso(new Proceso(3, 5, 3)); // Proceso ID 3 con
prioridad 3 (más baja)


        // Ejecutar la simulación

        simulacion.ejecutarProcesos();

    }
}

```

Resultados:

```

-----< com.mycompany:mavenproject5 >-----

Building mavenproject5 1.0-SNAPSHOT
from pom.xml

-----[ jar ]-----

--- resources:3.3.1:resources (default-resources) @ mavenproject5 ---
skip non existing resourceDirectory
C:\Users\sando\OneDrive\Documentos\NetBeansProjects\mavenproject5\src\main\
resources

--- compiler:3.11.0:compile (default-compile) @ mavenproject5 ---
Changes detected - recompiling the module! :source
Compiling 1 source file with javac [debug target 22] to target\classes

```

--- exec:3.1.0:exec (default-cli) @ mavenproject5 ---

Ejecutando proceso ID: 2 con prioridad 1 y tiempo de ejecución de 3 unidades de tiempo.

Proceso ID 2 ejecutando... Tiempo transcurrido: 1 de 3

Proceso ID 2 ejecutando... Tiempo transcurrido: 2 de 3

Proceso ID 2 ejecutando... Tiempo transcurrido: 3 de 3

Proceso ID 2 completado.

Ejecutando proceso ID: 1 con prioridad 2 y tiempo de ejecución de 4 unidades de tiempo.

Proceso ID 1 ejecutando... Tiempo transcurrido: 1 de 4

Proceso ID 1 ejecutando... Tiempo transcurrido: 2 de 4

Proceso ID 1 ejecutando... Tiempo transcurrido: 3 de 4

Proceso ID 1 ejecutando... Tiempo transcurrido: 4 de 4

Proceso ID 1 completado.

Ejecutando proceso ID: 3 con prioridad 3 y tiempo de ejecución de 5 unidades de tiempo.

Proceso ID 3 ejecutando... Tiempo transcurrido: 1 de 5

Proceso ID 3 ejecutando... Tiempo transcurrido: 2 de 5

Proceso ID 3 ejecutando... Tiempo transcurrido: 3 de 5

Proceso ID 3 ejecutando... Tiempo transcurrido: 4 de 5

Proceso ID 3 ejecutando... Tiempo transcurrido: 5 de 5

Proceso ID 3 completado.

Multilevel Queue Scheduling: Los procesos son asignados a diferentes colas de acuerdo a su prioridad.

```
import java.util.LinkedList;
import java.util.Queue;

class Proceso {
    int id;
    int tiempoEjecucion;
    int prioridad;

    public Proceso(int id, int tiempoEjecucion, int prioridad) {
        this.id = id;
        this.tiempoEjecucion = tiempoEjecucion;
        this.prioridad = prioridad;
    }
}

public class SimulacionMultilevelQueueScheduling {
    // Tres colas para cada nivel de prioridad
    private Queue<Proceso> colaAltaPrioridad;
    private Queue<Proceso> colaMediaPrioridad;
    private Queue<Proceso> colaBajaPrioridad;

    public SimulacionMultilevelQueueScheduling() {
        this.colaAltaPrioridad = new LinkedList<>();
        this.colaMediaPrioridad = new LinkedList<>();
        this.colaBajaPrioridad = new LinkedList<>();
    }

    public void agregarProceso(Proceso proceso) {
```

```

// Asignar el proceso a la cola correspondiente según su prioridad
switch (proceso.prioridad) {
    case 1:
        colaAltaPrioridad.add(proceso);
        break;
    case 2:
        colaMediaPrioridad.add(proceso);
        break;
    case 3:
        colaBajaPrioridad.add(proceso);
        break;
    default:
        System.out.println("Prioridad no válida para el proceso ID: " +
proceso.id);
    }
}

public void ejecutarProcesos() {
    System.out.println("Ejecutando procesos en cola de alta prioridad...");
    ejecutarCola(colaAltaPrioridad);

    System.out.println("Ejecutando procesos en cola de media prioridad...");
    ejecutarCola(colaMediaPrioridad);

    System.out.println("Ejecutando procesos en cola de baja prioridad...");
    ejecutarCola(colaBajaPrioridad);
}

private void ejecutarCola(Queue<Proceso> cola) {

```

```

while (!cola.isEmpty()) {

    Proceso procesoActual = cola.poll();

    System.out.println("Ejecutando proceso ID: " + procesoActual.id + " con
prioridad " + procesoActual.prioridad + " y tiempo de ejecución de " +
procesoActual.tiempoEjecucion + " unidades de tiempo.");

    for (int i = 1; i <= procesoActual.tiempoEjecucion; i++) {

        System.out.println("Proceso ID " + procesoActual.id + " ejecutando...
Tiempo transcurrido: " + i + " de " + procesoActual.tiempoEjecucion);

        try {

            Thread.sleep(1000); // Pausa de 1 segundo para simular el paso del
tiempo

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

            System.out.println("La ejecución fue interrumpida.");

        }

    }

    System.out.println("Proceso ID " + procesoActual.id + " completado.");
    System.out.println("-----");

}

}

public static void main(String[] args) {

    SimulacionMultilevelQueueScheduling simulacion = new
SimulacionMultilevelQueueScheduling();

    // Agregar procesos con tiempos de ejecución y prioridad

```

```

        simulacion.agregarProceso(new Proceso(1, 4, 1)); // Prioridad 1 - Alta
prioridad

        simulacion.agregarProceso(new Proceso(2, 3, 2)); // Prioridad 2 - Media
prioridad

        simulacion.agregarProceso(new Proceso(3, 5, 3)); // Prioridad 3 - Baja
prioridad

        simulacion.agregarProceso(new Proceso(4, 2, 1)); // Prioridad 1 - Alta
prioridad


    // Ejecutar la simulación
    simulacion.ejecutarProcesos();
}
}

```

Resultado:

Ejecutando procesos en cola de alta prioridad...

Ejecutando proceso ID: 1 con prioridad 1 y tiempo de ejecución de 4 unidades de tiempo.

Proceso ID 1 ejecutando... Tiempo transcurrido: 1 de 4

...

Proceso ID 1 completado.

Ejecutando proceso ID: 4 con prioridad 1 y tiempo de ejecución de 2 unidades de tiempo.

...

Proceso ID 4 completado.

Ejecutando procesos en cola de media prioridad...

Ejecutando proceso ID: 2 con prioridad 2 y tiempo de ejecución de 3 unidades de tiempo.

...

Ejecutando procesos en cola de baja prioridad...

Ejecutando proceso ID: 3 con prioridad 3 y tiempo de ejecución de 5 unidades de tiempo.

...

CONCLUSIÓN:

La simulación de administración de procesos en una CPU permite entender cómo el procesador gestiona múltiples tareas, optimizando su tiempo y recursos para ejecutar instrucciones de manera eficiente. A través de algoritmos de planificación (como FIFO, Round Robin, y SJF), la CPU asigna tiempo a cada proceso de acuerdo a ciertas reglas, maximizando el rendimiento y minimizando el tiempo de espera ocioso.


Con estas simulaciones se concluye que una administración eficiente de procesos es fundamental para el rendimiento del sistema, ya que permite equilibrar la carga de trabajo, evitar cuellos de botella y reducir el tiempo de espera de los procesos. Así, la CPU puede aprovechar su capacidad total, proporcionando una experiencia fluida en el sistema y mejorando la respuesta en tiempo real para el usuario.

La conclusión es que no existe un único algoritmo de planificación óptimo para todos los casos; cada uno tiene ventajas y desventajas según el tipo de carga de trabajo. Esto sugiere que una estrategia adaptativa o combinada puede ser más eficiente en sistemas que manejan tareas diversas, permitiendo que la CPU responda con mayor flexibilidad y optimización en entornos variables.


BIBLIOGRAFÍAS:

 <https://www.xataka.com/basics/cpu-que-como-sirve>

 <https://concepto.de/cpu/>

 Teleki, N. (2022, 28 marzo). *CPU: ¿qué es, cómo funciona y para qué sirve?* -

ProximaHost. ProximaHost. <https://proximahost.es/blog/cpu-como-funciona/>

 <https://www.ppstech.mx/blog/cpu-que-es-y-cual-es-la-funcion-del-procesador>

 <https://www.derichebourgespana.com/que-es-y-para-que-sirve-el-microprocesador-o-cpu/>