PROYECTO INDIVIDUAL “SIMULADOR DE ASIGNACION DE TIEMPO Y MEMORIA A PROCESOS”
SISTEMAS OPERATIVOS

Rosas Hernández Christian Guillermo
S22028194

Contenido

Introducción

El proyecto desarrollado en Java busca emular el funcionamiento de una Unidad Central de Procesamiento (CPU) al simular la asignación de tiempo y memoria a través del algoritmo de planificación de procesos Shortest Job Next (SJN). Este algoritmo, fundamental en la gestión de recursos en sistemas informáticos, prioriza la ejecución de los procesos más cortos en términos de tiempo de CPU, optimizando así el rendimiento y la utilización de recursos.

La simulación planteada se centra en la asignación eficiente de recursos computacionales a múltiples procesos concurrentes. El proyecto representa un entorno simulado donde cada proceso individual tiene requisitos de tiempo de ejecución y memoria, y la CPU debe gestionar su ejecución y el uso adecuado de la memoria disponible. Esto refleja la importancia de los algoritmos de planificación de procesos en entornos informáticos, donde la optimización de recursos es fundamental para el rendimiento del sistema.

El objetivo principal de esta simulación es proporcionar una representación interactiva y visual del comportamiento de un algoritmo de planificación de procesos, brindando una comprensión práctica de cómo la CPU asigna y administra eficientemente el tiempo y la memoria a los procesos en un entorno simulado. Esta experiencia práctica fortalece la comprensión de la gestión de recursos en sistemas informáticos, ofreciendo una base sólida para entender la importancia de los algoritmos de planificación en el rendimiento de los sistemas.

Desarrollo

El proyecto Java simula la asignación de tiempo y memoria en una Unidad Central de Procesamiento (CPU) a través del algoritmo de planificación de procesos SJN. Este algoritmo selecciona el próximo proceso a ejecutar basándose en su tamaño, permitiendo la ejecución del proceso más corto disponible en un momento dado.

Clase Process

La clase Process representa cada proceso individual y su comportamiento dentro del sistema. Cada proceso se identifica mediante un número único y requiere un cierto tamaño de memoria para ejecutarse. La implementación de la interfaz Runnable permite la ejecución de estos procesos de manera simultánea en hilos separados (Threads), emulando su ejecución paralela.

```
class Process implements Runnable {
    private final int id;
    private final CPU cpu;
    private final int memorySize;

    public Process(int id, CPU cpu, int memorySize) {
        this.id = id;
        this.cpu = cpu;
        this.memorySize = memorySize;
    }

    public int getId() {
        return id;
    }

    public int getMemorySize() {
        return memorySize;
    }
```

```
    @Override
    public void run() {
        try {
            System.out.println("Proceso " + id + " ha llegado.");

            // Simular el tiempo que tarda el proceso en ejecutarse
            TimeUnit.SECONDS.sleep(10);

            System.out.println("Proceso " + id + " ha terminado.");
            cpu.releaseMemory(this);
            cpu.AgregarProcesosFina(this);

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

Clase CPU

La clase CPU actúa como el administrador central de la CPU. Gestiona la lista de procesos en ejecución, aquellos que están en espera y los procesos finalizados. Además, controla la asignación de memoria a los procesos en ejecución y la liberación de la misma una vez finalizado un proceso. El método executeSJN() implementa la lógica del algoritmo SJN, seleccionando y ejecutando los procesos más cortos disponibles.

```
class CPU implements Runnable {
    private final List<Process> procesos;
    private final List<Process> cola;
```

```
private  final List<Process> finalizado;
private final int maxMemoria;

private int usedMemoria;

public CPU(int maxMemoria) {
    this.maxMemoria = maxMemoria;
    this.procesos = new ArrayList<>();
    this.cola = new ArrayList<>();
    this.finalizado = new ArrayList<>();
    this.usedMemoria = 0;
}

public void AgregarProcesos(Process process) {
    if (usedMemoria + process.getMemorySize() <= maxMemoria) {
        procesos.add(process);
        usedMemoria += process.getMemorySize();
        new Thread(process).start();
    } else {
        cola.add(process);
    }
}
public void AgregarProcesosFina(Process process) {
    finalizado.add(process);
}



public void SJN() {
    while (true) {
```

```
        if (!cola.isEmpty()) {
            Process smallestProcess = null;
            for (Process process : cola) {
                if (usedMemoria + process.getMemorySize() <= maxMemoria) {
                    if (smallestProcess == null || process.getMemorySize() <
smallestProcess.getMemorySize()) {
                        smallestProcess = process;
                    }
                }
            }

            if (smallestProcess != null) {
                cola.remove(smallestProcess);
                procesos.add(smallestProcess);
                usedMemoria += smallestProcess.getMemorySize();
                new Thread(smallestProcess).start();
            }
        }
    }
}

public List<Process> getProcesos() {
    return procesos;
}

public List<Process> getCola() {
    return cola;
}
```

```
public List<Process> getFinalizado() {
    return finalizado;
}
public synchronized void releaseMemory(Process process) {
    usedMemoria -= process.getMemorySize(); // Liberar la memoria utilizada por el proceso
    procesos.remove(process); // Eliminar el proceso de la lista de procesos en ejecución

    // Verificar si hay procesos en la cola y espacio en memoria para ejecutarlos
    while (!cola.isEmpty() && usedMemoria < maxMemoria) {
        Process nextProcess = cola.remove(0);
        if (usedMemoria + nextProcess.getMemorySize() <= maxMemoria) {
            procesos.add(nextProcess);
            usedMemoria += nextProcess.getMemorySize();
            new Thread(nextProcess).start();
        } else {
            cola.add(nextProcess);
            break; // No hay suficiente memoria para agregar más procesos en este momento
        }
    }
}




@Override
public void run() {
    SJN();



}
```

```
}
```

Clase GUI

Para proporcionar una visualización dinámica del estado de los procesos, la clase GUIThread se encarga de la interfaz gráfica. Actualiza periódicamente áreas dedicadas a los procesos en ejecución, en espera y finalizados, permitiendo una visualización en tiempo real del comportamiento del algoritmo SJN.

```
class GUI implements Runnable {
    private final CPU cpu;
    private final JTextArea ejecutando;
    private final JTextArea esperando;
    private final JTextArea finalizando;

    public GUI(CPU cpu, JTextArea ejecutando, JTextArea esperando, JTextArea finalizando) {
        this.cpu = cpu;
        this.ejecutando = ejecutando;
        this.esperando = esperando;
        this.finalizando = finalizando;
    }

    @Override
    public void run() {
        JFrame frame = new JFrame("CPU Scheduler");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);
        frame.setLayout(new GridLayout(1, 3));

        JScrollPane ejeScrollPane = new JScrollPane(ejecutando);
        JScrollPane espeScrollPane = new JScrollPane(esperando);
```

```
    JScrollPane finaScrollPane = new JScrollPane(finalizando);

    frame.add(ejeScrollPane);
    frame.add(espeScrollPane);
    frame.add(finaScrollPane);

    frame.setVisible(true);

    while (true) {
        ActualizarGUI();
        try {
            Thread.sleep(1000); // Actualiza la GUI cada segundo
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

private void ActualizarGUI() {
    ejecutando.setText("Procesos en ejecucion:\n");
    for (Process p : cpu.getProcesos()) {
        ejecutando.append("Proceso " + p.getId() + "\n");
    }

    esperando.setText("Procesos en espera:\n");
    for (Process p : cpu.getCola()) {
        esperando.append("Proceso " + p.getId() + "\n");
    }
```

```
        finalizando.setText("Procesos Finalizados:\n");
        for (Process p : cpu.getFinalizado()){
            finalizando.append("Proceso "+ p.getId()+" finalizado" + "\n");
        }
    }
}
```

Clase CPUManager

El punto de entrada del programa, CPUManager, inicializa la CPU, la interfaz gráfica y simula la llegada de procesos cada 5 segundos. Estos procesos se asignan con un tamaño de memoria aleatorio y se envían a la CPU para su ejecución o para ser puestos en cola si la memoria disponible es insuficiente.

```
public class CPUManager {
    public static void main(String[] args) {
        CPU cpu = new CPU(10); // Máximo de procesos simultáneos en la CPU
        Thread cpuThread = new Thread(cpu);
        cpuThread.start();

        JTextArea ejecutando = new JTextArea();
        JTextArea esperando = new JTextArea();
        JTextArea finalizando = new JTextArea();

        GUI guiThread = new GUI(cpu, ejecutando, esperando, finalizando);
        Thread gui = new Thread(guiThread);
        gui.start();

        // Simulación de llegada de procesos cada 5 segundos
        int processId = 1;
```

```
        while (true) {
            int memorySize = (int) (Math.random() * 10) + 1;
            Process process = new Process(processId++,cpu,memorySize);
            cpu.AgregarProcesos(process);
            try {
                TimeUnit.SECONDS.sleep(5);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Conclusiones

El desarrollo de este proyecto individual representó un desafío técnico significativo al crear un simulador de asignación de tiempo y memoria en una CPU mediante el algoritmo de planificación de procesos Shortest Job Next (SJN).

La planificación y ejecución meticulosa de las clases Process, CPU, GUI y CPUManager requirieron una atención minuciosa a los detalles y una comprensión profunda de los conceptos subyacentes. La conceptualización de la lógica de asignación de memoria y tiempo de CPU, la coordinación de múltiples procesos y la presentación gráfica en tiempo real de la ejecución de procesos fueron logros destacados en este proyecto individual.

El esfuerzo solitario invertido en resolver desafíos técnicos, diseñar y programar cada componente del simulador refleja la dedicación y la disciplina necesarias en el ámbito de la informática.

En resumen, este proyecto individual no solo representó la aplicación práctica de algoritmos de planificación de procesos, sino que también significó un valioso viaje de aprendizaje y crecimiento profesional. La comprensión adquirida sobre la gestión de recursos en sistemas informáticos servirá como una base sólida para abordar desafíos futuros y seguir avanzando en el campo de la informática.