DOCUMENTACIÓN

UNIVERSIDAD VERACRUZANA

14 DE DICIEMBRE DE 2023

SISTEMAS OPERATIVOS

Facultad de Estadística e Informática
Mtro. Juan Luis López Herrera
Alumno:
Juan Pablo Torres Ortiz

DOCUMENTACION

Objetivo:

El objetivo de este código es simular un sistema de planificación de procesos utilizando el algoritmo FCFS (First-Come, First-Served). En este contexto, se crean procesos de manera aleatoria con tiempos de ejecución y requerimientos de memoria, y estos procesos son atendidos por un hilo FCFS en el orden en el que llegan.

Funcionamiento:

- Se crean dos hilos: uno para crear procesos (CrearHilo) y otro para atenderlos utilizando el algoritmo FCFS (FCFS).
- El hilo CrearHilo genera procesos aleatorios con tiempos de ejecución y requerimientos de memoria.
- Se verifica si la capacidad de memoria disponible permite la creación de un nuevo proceso. En caso afirmativo, se añade a la cola de espera y se notifica al hilo FCFS.
- El hilo FCFS espera a que haya procesos en la cola. Cuando hay un proceso disponible, lo atiende, simulando su ejecución y liberando la memoria utilizada.

Ventajas:

Simplicidad: El algoritmo FCFS es simple y fácil de entender. Los procesos se ejecutan en el orden en que llegan, lo que facilita la implementación y comprensión del sistema.

Desventajas:

Tiempo de Espera: Puede haber un tiempo de espera significativo para procesos de alta duración que llegan antes. Otros procesos más cortos deben esperar hasta que los procesos más largos hayan completado su ejecución.

Ineficiencia en Utilización de Recursos: Si un proceso largo llega primero, los procesos más cortos que podrían haberse ejecutado rápidamente deben esperar, lo que puede resultar en una utilización ineficiente de los recursos del sistema.

Sugerencias de Mejora:

Implementación de Prioridades: Podría considerarse la implementación de un sistema de prioridades para mejorar la eficiencia y dar preferencia a ciertos tipos de procesos.

Manejo de Cola de Espera: La cola de espera actual es sencilla, pero en entornos reales podría ser beneficioso implementar estructuras de datos más avanzadas para optimizar el manejo de la cola.

En resumen, este código proporciona una implementación básica de un sistema de planificación de procesos utilizando el algoritmo FCFS. Su simplicidad es una ventaja para

entender los conceptos básicos, pero en aplicaciones del mundo real, podrían requerirse estrategias más avanzadas para optimizar el rendimiento y la eficiencia del sistema.

Funcionamiento de la Interfaz y Simulación:

- Cola de Espera y Creación de Procesos:

La interfaz utiliza JList para mostrar una lista de procesos en espera. Inicialmente, esta lista está vacía.

Cuando presionas el botón "Iniciar Sistema", se inician dos hilos (CrearHilo y FCFS).

CrearHilo es responsable de generar procesos aleatorios con tiempos de ejecución y requerimientos de memoria. Estos procesos se agregan a una cola de espera (colaEspera).

El hilo CrearHilo notifica al hilo FCFS cuando hay un nuevo proceso disponible en la cola.

- Algoritmo FCFS:

El hilo FCFS está constantemente esperando en un bucle.

Cuando CrearHilo notifica que hay un nuevo proceso en la cola, FCFS extrae el primer proceso de la cola y lo atiende.

La información sobre el proceso atendido se imprime en la consola y se elimina de la lista visual en la interfaz.

Se simula la ejecución del proceso utilizando Thread.sleep para representar la duración del proceso.

Después de que el proceso ha sido atendido, el hilo FCFS vuelve al bucle de espera para el próximo proceso.

- Visualización en la Interfaz Gráfica:

La lista (JList) en la interfaz se actualiza dinámicamente cada vez que se agrega o elimina un proceso en la cola de espera.

El botón "Iniciar Sistema" inicia la simulación al comenzar la creación y atención de procesos.

- Consideraciones sobre la Simulación:

La simulación utiliza procesos generados aleatoriamente con tiempos de ejecución y requerimientos de memoria.

La memoria total del sistema está limitada a la constante CapacidadMemoria.

Los procesos generados aleatoriamente se agregan a la cola de espera si hay suficiente memoria disponible.

Notas Adicionales:

El uso de la palabra clave synchronized en partes del código indica que ciertas operaciones críticas están sincronizadas para evitar problemas de concurrencia entre hilos.

La interfaz gráfica se construye utilizando la biblioteca Swing de Java, proporcionando una ventana (JFrame) con una lista (JList) para mostrar los procesos en espera.

La simulación es infinita, ya que los hilos CrearHilo y FCFS se ejecutan en bucles infinitos.

En resumen, la interfaz gráfica proporciona una representación visual de la cola de espera y el proceso de planificación de procesos utilizando el algoritmo FCFS. La simulación utiliza hilos para crear y atender procesos, y la interfaz se actualiza en tiempo real para reflejar los cambios en la cola de espera.

CODIGO:

```
class CrearHilo extends Thread {
    private final Queue<Proceso> colaEspera;
    private final int CapacidadMemoria;
    private final DefaultListModel<String> listModel;

    public CrearHilo(Queue<Proceso> colaEspera, DefaultListModel<String> listModel, int CapacidadMemoria) {
        this.colaEspera = colaEspera;
        this.listModel = listModel;
        this.CapacidadMemoria = CapacidadMemoria;
    }

    public void run() {
        Random random = new Random();
        int i = 1;

        while (true) {
            int te = random.nextInt(10) + 1;
            int me = random.nextInt(10) + 1;

            // Verifica si el nuevo proceso excede la capacidad de memoria
            synchronized (colaEspera) {
                int memoriaActual = colaEspera.stream().mapToInt(Proceso::getMemoria).sum();
                if (memoriaActual + me <= CapacidadMemoria) {
                    Proceso nuevoProceso = new Proceso(te, me);
                    System.out.println("Nuevo proceso: " + i + " - Tiempo de ejecución: " + te + " - Memoria: " + me);

                    // Añade el nuevo proceso a la cola
                    colaEspera.offer(nuevoProceso);
                    listModel.addElement("Nuevo proceso: " + i + " - Memoria: " + me);

                    colaEspera.notify(); // Notifica al hilo FCFS que hay un nuevo proceso
                }
            }

            try {
                // Tiempo aleatorio de espera antes de crear el próximo proceso
                Thread.sleep(random.nextInt(500) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            i++;
        }
    }
}
```

```
class FCFS extends Thread {
    private final Queue<Proceso> colaEspera;
    private final DefaultListModel<String> listModel;

    public FCFS(Queue<Proceso> colaEspera, DefaultListModel<String> listModel) {
        this.colaEspera = colaEspera;
        this.listModel = listModel;
    }

    public void run() {
        while (true) {
            Proceso procesoActual;
            // Espera a que haya un proceso en la cola
            synchronized (colaEspera) {
                while (colaEspera.isEmpty()) {
                    try {
                        colaEspera.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                procesoActual = colaEspera.poll();
                listModel.remove(0);
            }

            System.out.println("Atendiendo proceso - Tiempo de ejecución: " +
                    procesoActual.getTiempoEjecucion() + " - Memoria requerida: " + procesoActual.getMemoria());

            // Simula la ejecución del proceso
            try {
                Thread.sleep((procesoActual.getTiempoEjecucion() * 1000) / 2); // Multiplicado por 1000 para convertir a milisegundos y dividir entre 2 para que no sea tanto tiempo de espera
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Proceso { //clase procesp
    private int tiempoEjecucion;
    private int memoria;

    public Proceso(int te, int m) {
        this.tiempoEjecucion = te;
        this.memoria = m;
    }

    public int getTiempoEjecucion() {
        return tiempoEjecucion;
    }

    public int getMemoria() {
        return memoria;
    }
}
```

```
public class ProyectoFCFSInterfaz {
    public static final int CapacidadMemoria = 50;

    public static void main(String[] args) {
        Queue<Proceso> colaEspera = new LinkedList<>();
        DefaultListModel<String> listModel = new DefaultListModel<>();

        CrearHilo hiloCrear = new CrearHilo(colaEspera, listModel, CapacidadMemoria);
        FCFS fcfs = new FCFS(colaEspera, listModel);

        // Crear y configurar la ventana
        JFrame frame = new JFrame("Simulador de Procesos FCFS");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        // Crear y configurar la lista
        JList<String> processList = new JList<>(listModel);
        JScrollPane scrollPane = new JScrollPane(processList);
        frame.getContentPane().add(scrollPane, BorderLayout.CENTER);

        // Crear botón para iniciar el sistema
        JButton startButton = new JButton("Iniciar Sistema");
        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                hiloCrear.start();
                fcfs.start();
            }
        });
        frame.getContentPane().add(startButton, BorderLayout.SOUTH);

        // Mostrar la ventana
        frame.setVisible(true);
    }
}
```