SJN

Simulador de SJN

DICIEMBRE 15, 2023

MARLA JASEL AGUILAR AGUILAR

Sistemas operativos

Introducción:

En este proyecto, exploraremos el algoritmo SJN (Shortest Job Next) para la planificación de procesos en sistemas informáticos. Implementaremos un simulador interactivo en Java con interfaz gráfica, permitiendo a los usuarios experimentar con los principios fundamentales de SJN. A través de esta herramienta, buscamos ofrecer una comprensión práctica de la optimización de procesos, facilitando tanto a principiantes como a usuarios más avanzados el análisis del impacto de SJN en la eficiencia de la planificación de procesos. Este proyecto sirve como puente entre la teoría y la aplicación práctica de los conceptos fundamentales de la informática.

```java
class HiloEjecucion {
    2 usages
    private final String nombre;
    5 usages
    private int tiempoEjecucion;

    1 usage
    public HiloEjecucion(String nombre, int tiempoEjecucion) {
        this.nombre = nombre;
        this.tiempoEjecucion = tiempoEjecucion;
    }

    no usages
    public String getNombre() {
        return nombre;
    }

    4 usages
    public int getTiempoEjecucion() {
        return tiempoEjecucion;
    }
```

Se crea el hilo con los atributos nombre y tiempo de ejecución.

```java
    1 usage
    public void decrementarTiempoEjecucion(int cantidad) {
        tiempoEjecucion -= cantidad;
        if (tiempoEjecucion < 0) {
            tiempoEjecucion = 0;
        }
    }
}
```

Esta función decrementa el tiempo de ejecución de acuerdo a una cantidad dada y a la vez se asegura que no sea menor que cero.

```
private void ejecutarSJN(ActionEvent e) {
    int reduccion = (int) (Math.random() * 40) + 1;
    labels[hiloActual].setSize(Math.max(labels[hiloActual].getWidth() - reduccion, 0), labels[hiloActual].getHeight());
    hilos[hiloActual].decrementarTiempoEjecucion( cantidad: 1000);

    if (labels[hiloActual].getWidth() <= 0) {
        hiloActual++;

        if (hiloActual < labels.length) {
            remove(labels[hiloActual - 1]);
            Arrays.sort(hilos, (h1, h2) -> Integer.compare(h1.getTiempoEjecucion(), h2.getTiempoEjecucion()));
            add(labels[hiloActual]);

            int newWidth = Math.min(labels[hiloActual].getWidth(), MAX_LABEL_WIDTH);
            labels[hiloActual].setSize(newWidth, labels[hiloActual].getHeight());

            revalidate();
            repaint();
        } else {
            ((Timer) e.getSource()).stop();
        }
    }
}
```

En este apartado del código se simula el algoritmo SJN, la variable reducción establece la cantidad que se va a reducir el label, la segunda línea reduce el tamaño del label de acuerdo a el valor de reducción, en decrementar se va reduciendo el tiempo de ejecución de cada hilo 1 segundo

```
public SimulacionSJN() {
    setTitle("Simulación SJN");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new GridLayout( rows: 1,  cols: 1));

    int numHilos = (int) (Math.random() * (MAX_NUM_HILOS - MIN_NUM_HILOS + 1)) + MIN_NUM_HILOS;

    labels = new JLabel[numHilos];
    hilos = new HiloEjecucion[numHilos];
    hiloActual = 0;

    for (int i = 0; i < numHilos; i++) {
        int tiempo = (int) (Math.random() * 5000) + 1000;
        hilos[i] = new HiloEjecucion( nombre: "Hilo" + (i + 1), tiempo);

        labels[i] = new JLabel();
        labels[i].setOpaque(true);
        labels[i].setBackground(getRandomColor());
    }

    Arrays.sort(hilos, (h1, h2) -> Integer.compare(h1.getTiempoEjecucion(), h2.getTiempoEjecucion()));

    add(labels[hiloActual]);
```

```
    setSize( width: 300,  height: 300);
    setLocationRelativeTo(null);
    setVisible(true);

    Timer timer = new Timer( delay: 1000, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            ejecutarSJN(e);
        }
    });
    timer.start();
}
```

Se crea la ventana y se crean n cantidad de hilos con distintos tiempos de ejecución y a la vez se crea la etiqueta para representar le ejecución de cada hilo, antes de que se ejecuten se ordenan de menor a mayor para seguir el algoritmo SJN. Se crea un temporizador para que se ejecute junto al método ejecutarSJN, con un tiempo de 1 segundo.

Conclusión.

La indagación del algoritmo SJN mediante la construcción de un simulador interactivo en Java con interfaz gráfica ha proporcionado una apreciación substancial de los principios subyacentes en la planificación de procesos. La estrategia de SJN para privilegiar procesos de menor duración en aras de minimizar el tiempo total de procesamiento se manifestó claramente, y la interfaz gráfica facilitó la interacción y experimentación de los usuarios con los parámetros del

algoritmo. Este proyecto destaca la importancia de combinar algoritmos eficientes con presentaciones visuales, mejorando así la comprensión y aplicación de conceptos fundamentales en informática.