



SHORTEST REMAINING TIME

Sistemas operativos

Ingeniería de software
Universidad Veracruzana

Martinez Ramirez Fernando
Zs22013690@estudiantes.uv.mx

Introducción.

En este proyecto se simula el algoritmo de asignación de procesador SHORTEST REMAINING TIME (SRT) en java, la clase proceso simula ser un proceso que tiene un tiempo de ejecución y un tiempo restante. Con ayuda de la interfaz grafica se puede observar de manera mas sencilla como trabaja el algoritmo.

Desarrollo

Primeramente, se crea una clase proceso, con el fin de simular ser un proceso real.

```
public class Proceso {
    String nombre;
    int tiempoTotal;
    int tiempoRestante;

    public Proceso(String nombre, int tiempoTotal) {
        this.nombre = nombre;
        this.tiempoTotal = tiempoTotal;
        this.tiempoRestante = tiempoTotal;
    }
}
```

Posteriormente hacemos uso de una de una clase que nos permite crear varios procesos aleatorios, esto con el fin de mas adelante y crear procesos con un hilo.

```
/**
 *
 * @author FerRMZ
 */
import java.util.Random;

public class GeneradorProcesos {
    public static Proceso generarProcesoAleatorio() {
        Random random = new Random();
        String nombre = "Proceso" + (random.nextInt(100) + 1);
        int tiempo = random.nextInt(10) + 1;
        return new Proceso(nombre, tiempo);
    }
}
```

Creación de la interfaz gráfica

En esta parte se procede a crear la interfaz gráfica, se inicializan los componentes.

Primero, definimos la ventana principal llamada "AlgoritmoSRT". Le ponemos un título llamado "Algoritmo SRT" para identificar fácilmente de qué se trata. Además, le decimos qué hacer cuando se cierre la ventana, en este caso, que termine toda la aplicación. Además, le damos un tamaño inicial de 400 por 400 píxeles para que tenga un aspecto adecuado.

Dentro de esta ventana, necesitamos algunos elementos para mostrar la información de los procesos. Entonces, creamos una tabla (processTable) que nos ayudará a visualizar esta información. Para manejar la información dentro de la tabla, utilizamos un modelo de tabla predeterminado (DefaultTableModel). Este modelo nos permite organizar los datos en columnas, y en este caso, creamos columnas para el "Nombre del Proceso" y el "Tiempo Restante".

Además, queremos hacer que sea fácil agregar nuevos procesos, así que creamos un botón con el texto "Agregar Proceso Aleatorio" (btnAgregarProceso). También agregamos una barra de progreso (progressBar) para representar visualmente algún tipo de progreso o estado.

Para organizar todo esto en la ventana, colocamos la tabla en un JScrollPane para que podamos desplazarnos por la lista de procesos si se vuelve muy larga. Luego, colocamos el botón en la parte inferior y la barra de progreso en la parte superior para que todo tenga un aspecto ordenado.

Ahora, queremos que algo suceda cuando presionamos el botón. Para eso, agregamos un ActionListener al botón. Cuando hacemos clic en él, se ejecutará el método agregarProcesoAleatorio().

Y finalmente, para mantener las cosas interesantes, creamos un hilo en segundo plano que se ejecutará continuamente. Este hilo espera 3 segundos en cada iteración y luego utiliza SwingUtilities.invokeLater para agregar un nuevo proceso aleatorio a la lista y actualizar la interfaz gráfica. Esto podría representar la llegada de nuevos procesos al sistema cada cierto tiempo. Y así, nuestra interfaz gráfica está lista para funcionar.

```

public AlgoritmoSRT() {
    setTitle("Algoritmo SRT");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);

    tableModel = new DefaultTableModel();
    tableModel.addColumn("Nombre del Proceso");
    tableModel.addColumn("Tiempo Restante");

    processTable = new JTable(tableModel);
    btnAgregarProceso = new JButton("Agregar Proceso Aleatorio");
    progressBar = new JProgressBar();

    add(new JScrollPane(processTable), BorderLayout.CENTER);
    add(btnAgregarProceso, BorderLayout.SOUTH);
    add(progressBar, BorderLayout.NORTH);

    btnAgregarProceso.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            agregarProcesoAleatorio();
        }
    });

    procesos = new ArrayList<>();
    mostrarProcesosEnTabla();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            while (true) {
                try {
                    Thread.sleep(3000); // Esperar 3 segundos
                    SwingUtilities.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                            agregarProcesoAleatorio();
                        }
                    });
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    });
    thread.start();
}

```

```

procesos = new ArrayList();
mostrarProcesosEnTabla();

Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(3000); // Esperar 3 segundos
                SwingUtilities.invokeLater(new Runnable() {
                    @Override
                    public void run() {
                        agregarProcesoAleatorio();
                    }
                });
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
});
thread.start();

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

private void agregarProcesoAleatorio() {

```

```

private void agregarProcesoAleatorio() {
    Proceso nuevoProceso = GeneradorProcesos.generarProcesoAleatorio();
    procesos.add(nuevoProceso);

    tableModel.addRow(new Object[]{nuevoProceso.nombre, nuevoProceso.tiempoRestante});

    progressBar.setMaximum(nuevoProceso.tiempoTotal);
    progressBar.setValue(nuevoProceso.tiempoRestante);
    progressBar.setForeground(Color.GREEN);
}

```

Este método se encarga de la incorporación de un nuevo proceso de manera aleatoria al sistema. En primer lugar, se genera un proceso utilizando un generador específico (GeneradorProcesos). Posteriormente, el proceso recién creado se añade tanto a la lista de procesos como a la representación visual de los mismos en una tabla. Además, se ajusta la barra de progreso para reflejar el tiempo total y restante del nuevo proceso, y se establece el color de la barra en verde para indicar visualmente su estado.

```

private void ejecutarProceso() {
    if (!procesos.isEmpty()) {
        Proceso procesoActual = Collections.min(procesos, (p1, p2) -> Integer.compare(p1.tiempoRestante, p2.tiempoRestante));
        procesoActual.tiempoRestante--;

        progressBar.setValue(procesoActual.tiempoRestante);

        int rowIndex = obtenerIndiceFilaPorNombre(procesoActual.nombre);
        tableModel.setValueAt(procesoActual.tiempoRestante, rowIndex, 1);

        if (procesoActual.tiempoRestante <= 0) {
            procesos.remove(procesoActual);

            int rowIndexToRemove = obtenerIndiceFilaPorNombre(procesoActual.nombre);
            tableModel.removeRow(rowIndexToRemove);

            progressBar.setValue(0);
        }
    }
}

```

Este método simula la ejecución de un proceso en el sistema. Primero, verifica que existan procesos en la lista. Luego, identifica el proceso con el tiempo restante más corto utilizando la función `Collections.min`. El tiempo restante de este proceso se reduce en uno para simular su ejecución. A continuación, se actualizan la barra de progreso y la fila correspondiente en la tabla para reflejar los cambios. Si el tiempo restante del proceso llega a cero, se elimina de la lista y se actualiza nuevamente la tabla y la barra de progreso.

```

private void mostrarProcesosEnTabla() {
    tableModel.setRowCount(0);

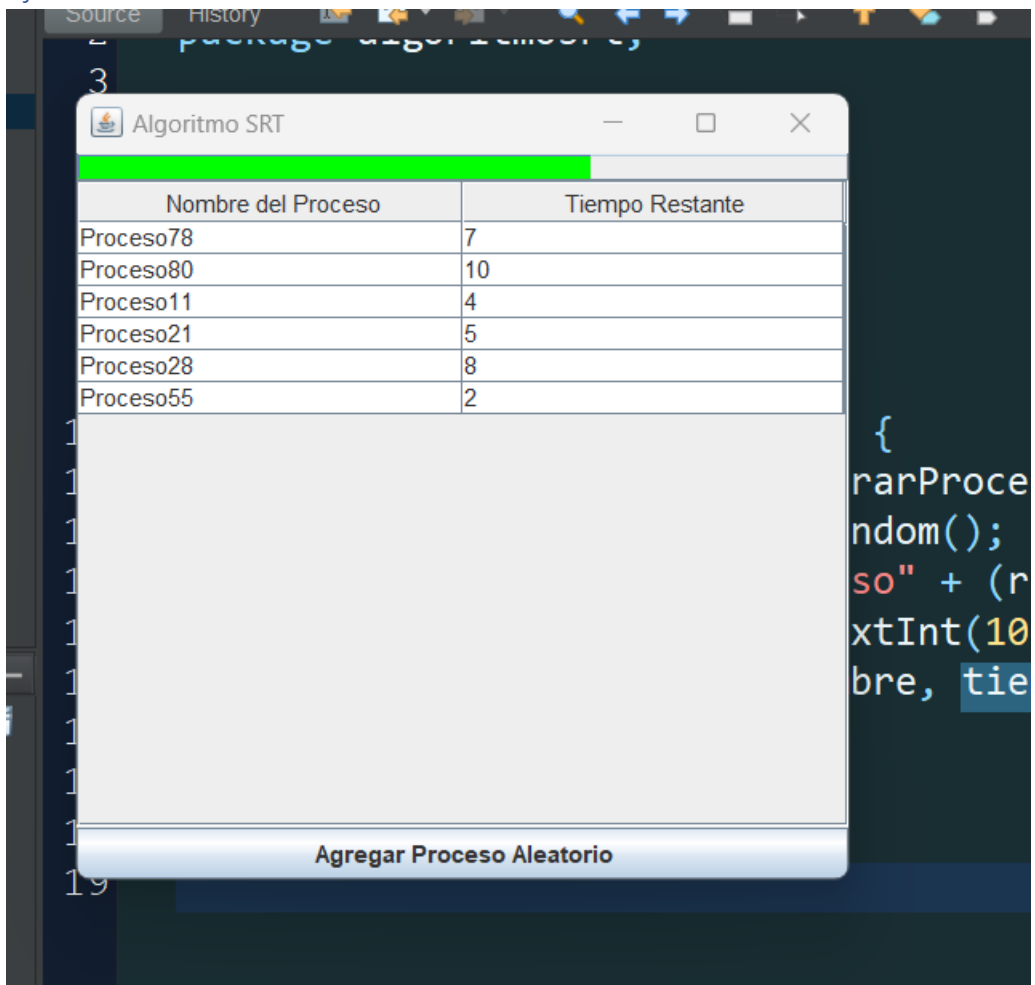
    for (Proceso proceso : procesos) {
        tableModel.addRow(new Object[]{proceso.nombre, proceso.tiempoRestante});
    }
}

private int obtenerIndiceFilaPorNombre(String nombre) {
    for (int i = 0; i < tableModel.getRowCount(); i++) {
        if (tableModel.getValueAt(i, 0).equals(nombre)) {
            return i;
        }
    }
    return -1;
}

```

Este método facilita la búsqueda del índice de una fila en la tabla basándose en el nombre de un proceso. Recorre cada fila de la tabla comparando el nombre del proceso con el contenido de cada celda en la columna correspondiente. Si encuentra una coincidencia, devuelve el índice de esa fila; en caso contrario, retorna `-1`, indicando que no se ha encontrado una correspondencia.

Ejecución



The screenshot shows a Java IDE with a window titled "Algoritmo SRT" in the foreground. The window contains a table with two columns: "Nombre del Proceso" and "Tiempo Restante". The table lists six processes with their respective remaining times. Below the table is a large empty text area and a button labeled "Agregar Proceso Aleatorio". The background shows Java code for the SRT algorithm.

Nombre del Proceso	Tiempo Restante
Proceso78	7
Proceso80	10
Proceso11	4
Proceso21	5
Proceso28	8
Proceso55	2

Agregar Proceso Aleatorio

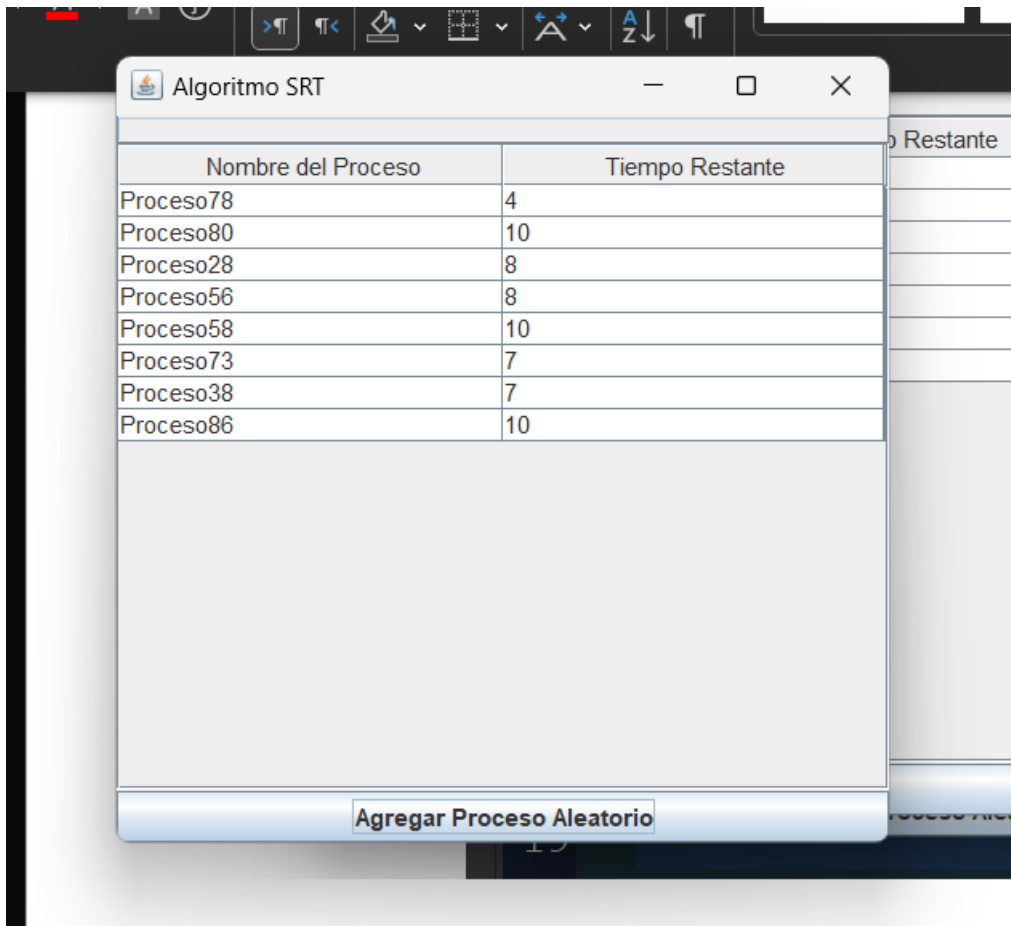
```
package algoritmosort;

3
{
    rarProce
    ndom();
    so" + (r
    xtInt(10
    bre, tie
```

Algorithm SRT

Nombre del Proceso	Tiempo Restante
Proceso78	7
Proceso80	10
Proceso28	8
Proceso21	2
Proceso96	6
Proceso56	8
Proceso58	10

Agregar Proceso Aleatorio



Conclusión

Crear una simulación del código permite entender y analizar como funciona este algoritmo, pero no solo eso, sino también, también las limitaciones que este puede llegar a presentar, también el manejo de la sincronización de los proceso fue un problema que presente al principio, ya que mi intención es era que todos los procesos se crearan a la vez e intentaran pedir memoria, gracias a esto tuve que generar de manera aleatoria y manejarlo de otro modo. Por otra parte, usar la interfaz grafica hace que resulte interesante observar cómo la barra de progreso se utiliza para visualizar de manera intuitiva el tiempo restante de ejecución de cada proceso. La elección de colorear la barra de progreso de verde proporciona una indicación visual clara y positiva del progreso del proceso.