Universidad Veracruzana
Facultad Estadística e Informática
Xalapa, Veracruz

Licenciatura:
Ingeniería de Software

Experiencia Educativa:
Sistemas Operativos

Docente:
Juan Luis López Herrera

Proyecto individual

Nombre del Alumno
Karol Quinto Guadalupe

Fecha:
Jueves 14 de diciembre de 2023

Introducción

En este proyecto se tiene la tarea de simular un mecanismo de planificación de procesos conocido como FCFS (First Come First Served), que se traduce como "primero en llegar, primero en ser atendido". Este enfoque representa una política no preventiva en la programación de procesos, donde la asignación de recursos se basa únicamente en el orden de llegada de los trabajos, por lo que es un proceso no expropiativo.

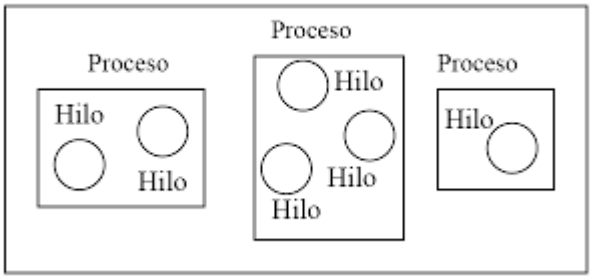
La simulación de este mecanismo se llevará a cabo mediante el uso de hilos, una herramienta esencial en la programación concurrente. Un hilo se encargará de simular la generación de procesos, reflejando la llegada secuencial de tareas al sistema. Mientras tanto, otro hilo se dedicará a determinar el tiempo que cada proceso pasará en la CPU, siguiendo la lógica de la política FCFS. Esta aproximación permitirá visualizar y comprender de manera efectiva cómo se comporta este algoritmo en un entorno simulado, brindando una visión práctica de su funcionamiento y sus implicaciones en la gestión de recursos computacionales.

Procesos e hilos

Un proceso se define como una instancia de ejecución de un programa que es identificable y controlable por el sistema operativo. Cada proceso encapsula la información necesaria para su ejecución, como el código, los datos, el estado del registro y el espacio de direcciones. Es la unidad básica de ejecución, y el sistema operativo asigna y administra recursos para cada proceso en función de sus necesidades y prioridades.

Un hilo representa una parte independiente de un proceso que puede ejecutarse de forma autónoma. Mientras que un proceso puede tener uno o varios hilos, estos comparten el mismo espacio de direcciones y recursos, facilitando la comunicación y el intercambio de datos entre ellos. En sistemas multiproceso, la presencia de varios subprocesos ejecutándose simultáneamente ofrece la posibilidad de una mayor concurrencia, ya que estos pueden trabajar de manera independiente con las mismas o diferentes prioridades.

Los procesos e hilos son componentes fundamentales en la arquitectura de sistemas operativos modernos, ofreciendo una estructura robusta para la ejecución de programas y la gestión eficiente de recursos.



Código

```
class Proceso extends Thread {
    private String nombre;
    private TurnoControl turnoControl;
    private JLabel etiqueta;
    private Animacion aux;
    private volatile boolean ejecutar = true;
    private JLabel mostrarP;
    private JLabel mostrarT;

    public Proceso(String nombre, TurnoControl turnoControl, JLabel etiqueta, Animacion aux, JLabel mostrarP, JLabel mostrarT) {
        this.nombre = nombre;
        this.turnoControl = turnoControl;
        this.etiqueta = etiqueta;
        this.aux = aux;
        this.mostrarP=mostrarP;
        this.mostrarT=mostrarT;
    }

    @Override
    public void run() {
        int autoA = 0;

        try {
            while (ejecutar) {
                autoA = aux.getCarro().getLocation().x;

                if (autoA < aux.getProcesador().getLocation().x-15) {
                    moverEtiqueta(50);
                    aux.repaint();
                }

                int turnoActual = turnoControl.obtenerTurno();

                Thread.sleep(1000);

                if (nombre.endsWith(Integer.toString(turnoActual))) {
                    mostrarP.setText(nombre + " iniciado.");

                    TiempoDespierto tiempoDespierto = new TiempoDespierto();
                    tiempoDespierto.start();

                    tiempoDespierto.join();
                    Thread.sleep(5000);

                    mostrarT.setText("Tiempo activo: "+tiempoDespierto.getTiempoDespierto()+" ms");

                    if(autoA < aux.getTope().getLocation().x-20) {
                        moverEtiqueta(80);
                        aux.repaint();
                    }

                    turnoControl.siguienteTurno();


                    }else{
                        Thread.sleep(3000);
                    }

            }
        } catch (InterruptedException e) {
            System.out.println("Error en los turnos: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private void moverEtiqueta(int distancia) {
        SwingUtilities.invokeLater(() -> etiqueta.setLocation(etiqueta.getLocation().x + distancia, etiqueta.getLocation().y));
    }

}
```

La clase proceso (que es un hilo) genera la simulación de procesos y los sincroniza con el hilo que genera el tiempo que pasa cada proceso en la CPU. Al mismo tiempo, se encarga de darle movimiento a la interfaz gráfica y mostrando mensajes, como el nombre de los procesos y el tiempo que pasa cada uno en el procesador. Para

finalizar con el cambio de turno del proceso. Todo dentro de un try – catch para evitar o señalar cualquier error que pueda surgir mientras todo se lleva a cabo.

```
class TurnoControl {
    private int turno = 1;

    public int obtenerTurno() {
        return turno;
    }

    public void siguienteTurno() {
        turno = (turno % 2) + 1;
    }
}
```

Esta clase solo nos devuelve el turno que le corresponde a cada proceso o pasar al siguiente dependiendo del numero de procesos que se desee generar.

```
class TiempoDespierto extends Thread {
    private int tiempoDespierto;

    @Override
    public void run() {
        tiempoDespierto = (int) (Math.random() * 1000 + 1);

        try {
            Thread.sleep(tiempoDespierto);
        } catch (InterruptedException e) {
            System.out.println("Error en generar tiempo");
            e.printStackTrace();
        }

    }

    public int getTiempoDespierto() {
        return tiempoDespierto;
    }
}
```

Finalmente, la clase TiempoDespierto que es el hilo que simula el tiempo que pasa cada proceso en la CPU. Generando el tiempo de manera aleatoria, duerme el proceso el tiempo que se genera y finalizando con un método que devuelve el tiempo generado.

JFRAME

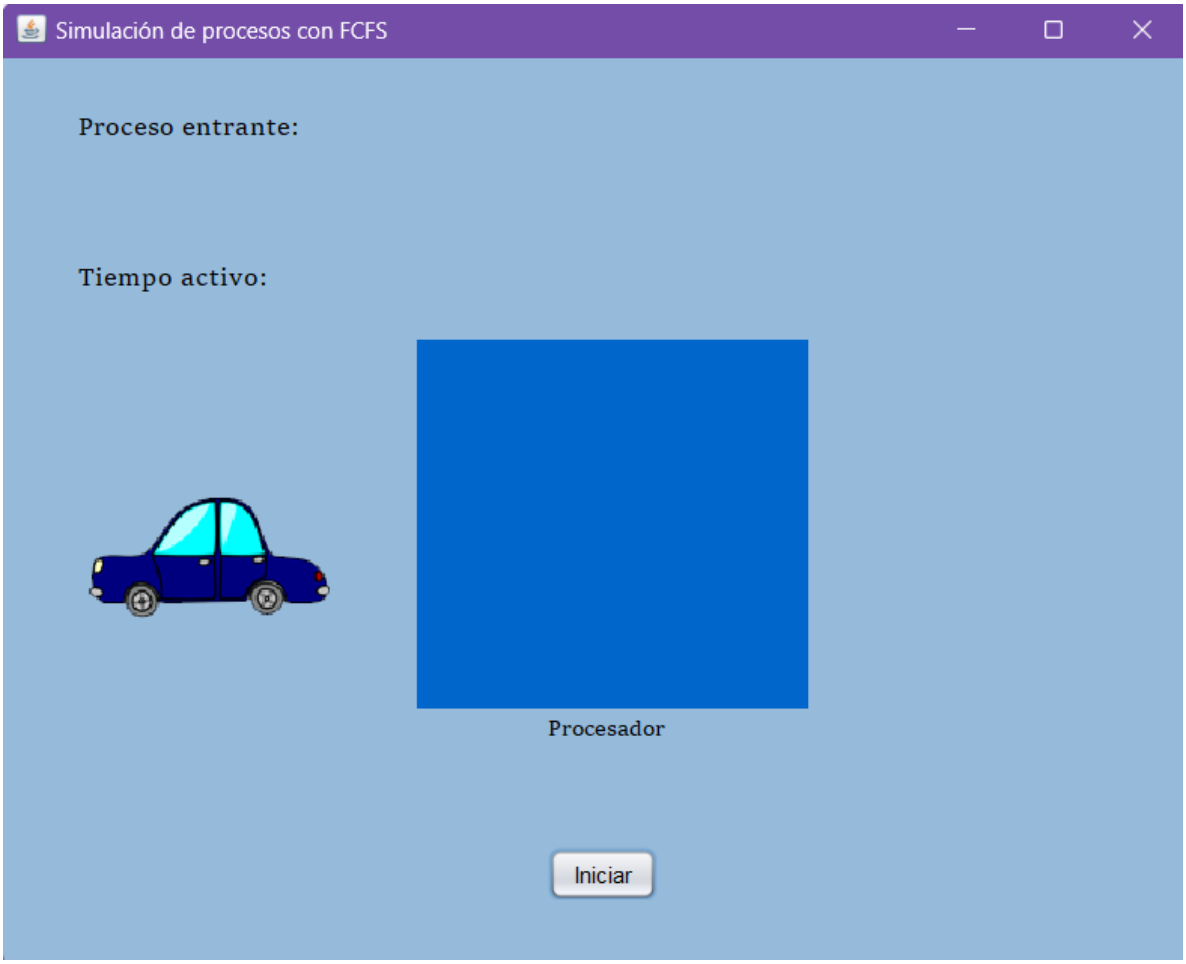
```
public class Animacion extends javax.swing.JFrame {

    /**
     * Creates new form Animacion
     */
    public Animacion() {
        initComponents();
        this.setLocationRelativeTo(null);
        this.setTitle("Simulación de procesos con FCFS");

    }

    public JLabel getCarro(){
        return carro;
    }

    public JLabel getProcesador(){
        return procesador;
    }


    public JLabel getMensaje(){
        return mensaje;
    }

    public JLabel getTope(){
        return tope;
    }
```

En esta parte del código, se trata de un Jframe, donde se muestra la interfaz con movimiento, en el constructor le damos nombre a la ventana y se centra en la pantalla. Posterior a eso, se tienen métodos para poder acceder a los elementos de la interfaz gráfica.

```
private void botonActionPerformed(java.awt.event.ActionEvent evt) {
    carro.setLocation(0, carro.getLocation().y);

    TurnoControl turnoControl  = new TurnoControl();

    for(int i=1;i<=2; i++){
        Proceso proceso = new Proceso("Proceso " + i, turnoControl, carro, this,mensaje,tiempo);
        proceso.start();
    }

}
```

Toda la simulación comienza cuando se da clic en el botón que se encuentra en la interfaz gráfica.

Resultado de la interfaz:



Conclusión

En conclusión, al explorar el algoritmo de planificación de procesos conocido como "First Come First Served" (FCFS), se destacan la simplicidad y facilidad de programación son innegables ventajas, brindando una implementación directa y comprensible. Sin embargo, como se ha señalado, lo que resulta justo en términos de orden de llegada no siempre se traduce en eficiencia operativa.

La desventaja más significativa de FCFS es su incapacidad para garantizar un rendimiento óptimo. Al asignar recursos basándose únicamente en el orden de llegada, este algoritmo puede llevar a situaciones en las que procesos largos y demandantes ocupan la CPU antes que procesos más cortos y rápidos, lo que afecta negativamente el tiempo de respuesta y la eficiencia global del sistema. En comparación con otros algoritmos de planificación más sofisticados, FCFS se clasifica como uno de los menos eficientes y no ofrece garantía alguna en términos de rendimiento.

Mientras que FCFS puede ser apropiado en ciertos contextos, es esencial evaluar críticamente sus ventajas y desventajas en relación con las necesidades particulares del sistema, buscando siempre el equilibrio entre simplicidad y eficiencia para lograr un rendimiento óptimo.