UNIVERSIDAD VERACRUZANA

FACULTAD DE ESTADÍSTICA E INFORMATICA

Licenciatura en ingeniería de Software

Sistemas Operativos

Simulación de planificación de procesos mediante hilos en Java

Jesus Lorenzo Tlapa Hernández

Xalapa, Ver. **13 de diciembre de 2023**

Contenido

Introducción

El presente documento es la documentación de un programa elaborado en lenguaje Java para el proyecto individual de la experiencia educativa de Sistemas Operativos.

El propósito del proyecto es simular la planificación de procesos mediante un algoritmo asignado por el profesor, en este caso se simula el algoritmo FCFS (First come first served) que es un algoritmo que utiliza una fila de procesos determinando el funcionamiento de cada proceso por el orden de llegada. Al llegar el proceso es puesto detrás del que llegó antes que él y se van atendiendo del primero que llegó hasta el último que llegó.

La simulación no solo tiene que verse reflejada mediante código, sino también con una interfaz gráfica que permita ver los cambios que va teniendo la cola de los procesos que se van atendiendo, tomando en cuenta dos factores principales: la memoria del procesador y el tiempo que cada proceso tarda con el procesador.

Descripción

A continuación, presento una breve descripción a modo de resumen de lo que hace cada clase de mi código.

Clase Principal:

- Contiene el método main.
- Se crean instancias de Ventana, Procesos, y SistemaOperativo.
- Se crean dos hilos (HiloCrearProcesos y HiloSistemaOperativo) que interactúan con los objetos mencionados.

Clase HiloCrearProcesos:

- Extiende Thread y se encarga de crear procesos y agregarlos a la cola de procesos mediante el método addProceso de la clase Procesos con cierta periodicidad.

Clase HiloSistemaOperativo:

- Extiende Thread y se encarga de manejar el sistema operativo y la asignación de memoria.
- Llama a métodos de la clase Procesos y SistemaOperativo para obtener, eliminar y asignar procesos y memoria, respectivamente.

Clase Proceso:

- Extiende Thread y representa un proceso con cierto tiempo de ejecución y espacio en memoria.

- El método run simula la ejecución del proceso y utiliza métodos de SistemaOperativo para indicar cuándo está utilizando y liberando la memoria.

Clase Procesos:

- Mantiene una cola de procesos y tiene métodos para agregar, obtener y eliminar procesos de la cola.
- Utiliza un semáforo (mutex1) para controlar el acceso concurrente a la cola de procesos, ya que constantemente se va a requerir tanto agregar como leer procesos de la cola misma.

Clase SistemaOperativo:

- Gestiona la asignación y liberación de memoria para los procesos.
- Utiliza un semáforo (mutex) para controlar el acceso concurrente a la memoria compartida.
- Utiliza un objeto Ventana para visualizar la asignación de memoria.

Clase Ventana:

- Extiende JFrame y representa la interfaz gráfica del simulador.
- Utiliza JPanel para representar bloques de memoria asignados.
- La interfaz muestra la asignación de memoria y la leyenda de proceso actual.

Ahora describiré algunos puntos o partes del código que son muy importantes de resaltar:

- La memoria que se le asigna al sistema operativo es dada en un inicio como parámetro y preferentemente debe ser mayor que 100 para asegurar que siempre cabe al menos un proceso.

```
int tiempo = 50;
int memoria = 400;

Ventana mv = new Ventana();

Procesos procesos= new Procesos();
SistemaOperativo SO = new SistemaOperativo(memoria,mv);


HiloCrearProcesos creadorProcesos = new HiloCrearProcesos(procesos,SO,tiempo);
HiloSistemaOperativo sistemaOpe = new HiloSistemaOperativo(procesos,SO,mv);
```

(en este caso la memoria es de 400)

- La memoria y el tiempo de cada proceso se asignan de manera aleatoria, para la memoria es un numero entre 30 y 100 y para el tiempo entre 1 y 100.

```java
public void addProceso(SistemaOperativo sisO, int esc) throws InterruptedException{
    mutex1.acquire();

    int ti=rand.nextInt(100)+1;    //1-100
    int esp = rand.nextInt(71)+30;  //30-100

    p1 = new Proceso(ti, esp, sisO, esc);
    //System.out.println("hola "+p1);
    cola.add(p1);
    mutex1.release();
}
```

- Se utiliza un parámetro llamado 'tiempo' que es una escala que se multiplica por el tiempo de cada proceso para determinar el total de milisegundos que este proceso va a utilizar el procesador.

```java
int tiempo = 50;
int memoria = 400;

Ventana mv = new Ventana();

Procesos procesos= new Procesos();
SistemaOperativo SO = new SistemaOperativo(memoria,mv);


HiloCrearProcesos creadorProcesos = new HiloCrearProcesos(procesos,SO,tiempo);
HiloSistemaOperativo sistemaOpe = new HiloSistemaOperativo(procesos,SO,mv);
```

```java
@Override
public void run(){
    try{
        so.usar(this);

        Thread.sleep(tiempo*escala);
        so.dejarDeUsar(this);
    }catch(Exception e){

    }
}
```

(Método de la clase Proceso)

- Los procesos (instancia de la clase Proceso) se almacenan en una cola que es una estructura FIFO y en ese orden se van atendiendo.

```java
class Procesos{
    private Queue<Proceso> cola;
    private Semaphore mutex1;
    private Random rand;
    private Proceso p1;

    public Procesos(){
        cola = new LinkedList<>();
        mutex1 = new Semaphore(1);
        rand = new Random();
    }
    public void addProceso(SistemaOperativo sisO, int esc) throws InterruptedException{
        mutex1.acquire();

        int ti=rand.nextInt(100)+1;   //1-100
        int esp = rand.nextInt(71)+30;  //30-100

        p1 = new Proceso(ti, esp, sisO, esc);
        //System.out.println("hola "+p1);
        cola.add(p1);
        mutex1.release();
    }
    public Proceso getProceso() throws InterruptedException{
        mutex1.acquire();
        Proceso p1 = cola.peek();
        mutex1.release();
        return p1;
    }
```

- El tamaño y el número pequeño que tiene cada proceso en la interfaz gráfica es dado de acuerdo con la memoria que ocupan.
- Los procesos que están en la cola (los que caben de acuerdo con la memoria) tienen un color naranja.

```java
public void agregarBloque(int me){
    bloque = new JPanel();
    bloque.add(new JLabel(Integer.toString(me)));
    bloque.setBackground(Color.orange);
    bloque.setPreferredSize(new Dimension(me, 50));
    panel.add(bloque);
    bloques.add(bloque);
}
```

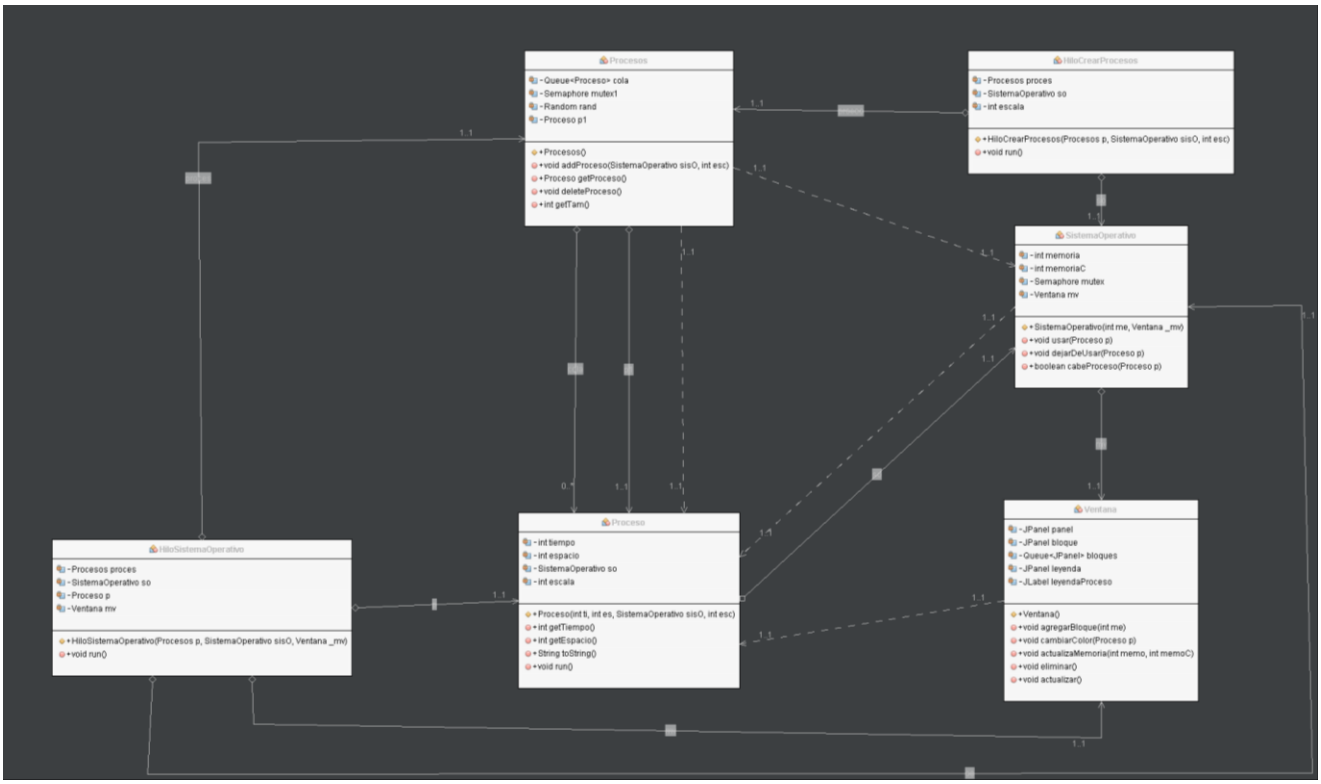
- El proceso que se está ejecutando (el primero de la cola) es de color verde para diferenciarlo

```
public void cambiarColor(Proceso p){
    bloque = bloques.peek();
    bloque.setBackground(Color.green);

    leyendaProceso.setText("<html>Proceso actual:<br>Tiempo: "+p.getTiempo()+"<br>Espacio: "+p.getEspacio()+" </html>");
    }
public void actualizaMemoria(int memo, int memoC){
    leyendaMemoria.setText("<html>Memoria: <br>Disponible: "+Integer.toString(memo)+"<br>Ocupada: "+Integer.toString(memoC-memo)+"</html>");

}
```

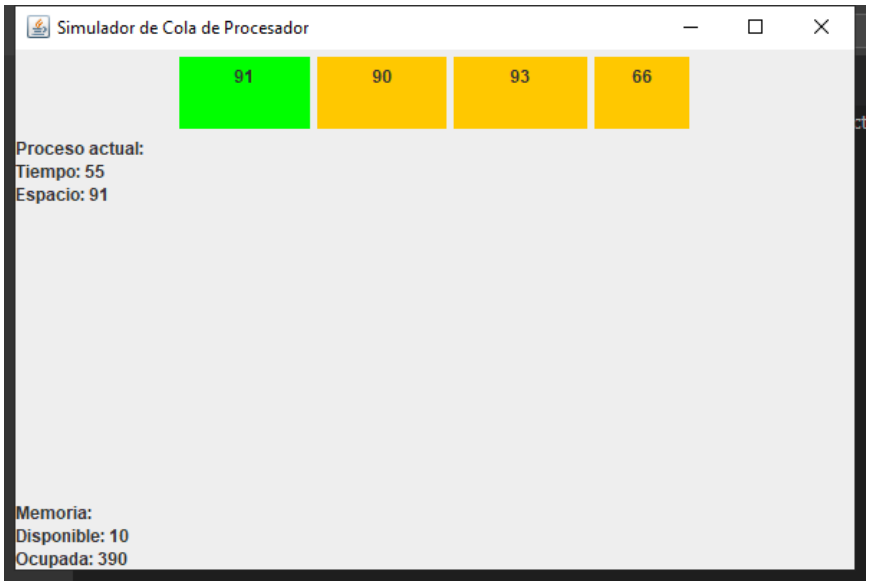
Diagrama UML



Funcionamiento de la interfaz gráfica

Ejemplo con una memoria de 400 y donde un tiempo es igual a 50ms.

Si sumamos la memoria de cada uno de los procesos, podemos notar que en total nos da 390 y solo deja 10 disponibles.



En las siguientes 3 imágenes, se tiene una memoria de 280 y un tiempo es igual a 100ms, con el objetivo de que vaya mas lento.

Imagen 1: El proceso en ejecución es el de 74 y en total se ocupan 265 de la memoria, dejando solamente 15 libres. Esto significa que el proceso siguiente a entrar a la cola es mayor de 15 y por eso no cabe.

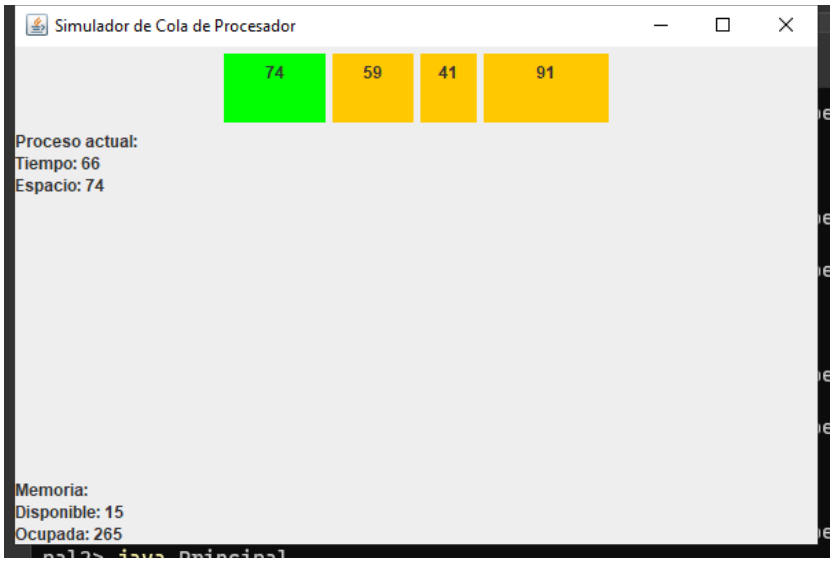


Imagen 2: Ahora podemos observar que se desocupó la memoria del proceso de 74 pero llegó uno nuevo de 66

Ocupada=265-74+66

Ocupada= 257

Por eso ahora la memoria ocupada es de 257 y hay solamente 23 libres

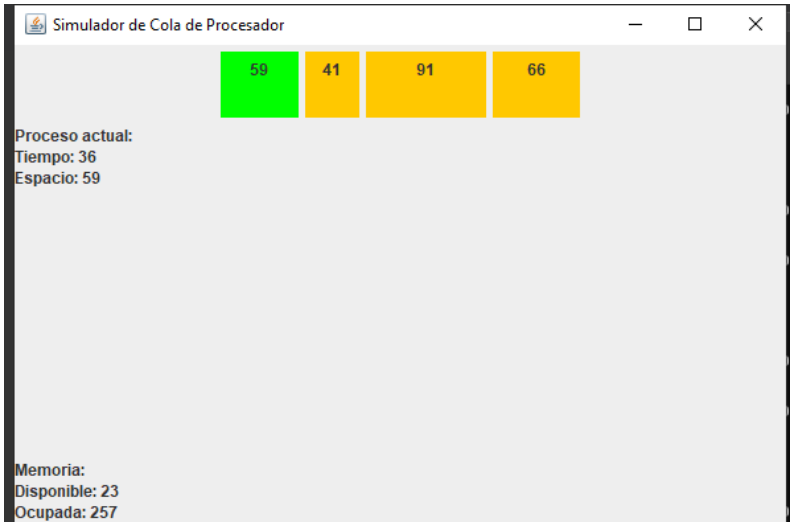


Imagen 3: Ahora podemos observar que se desocupó la memoria del proceso de 59 pero llegó uno nuevo de 68

Ocupada=257-59+68

Ocupada= 266

Por eso ahora la memoria ocupada es de 266 y hay solamente 14 libres