UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

Departamento de Ciencias de la Computación

Carrera de Ingeniería de software

Computación Paralela **– NRC:** 14561

Laboratorio 1 – Unidad 2

Tema: Aplicación de Semáforos

Grupo #9

Miembros:

Casignia Ruiz Diego Alejandro

Pabón González Elkin Andrés

Sanmartín Galán José Miguel

Docente: Ing. Carlos Pillajo

Fecha: jueves 27 de junio de 2024

1. **Objetivo General**

Desarrollar un programa que utilice semáforos para coordinar el acceso a regiones críticas en la simulación del proceso de obtención de una tarjeta de crédito en una entidad bancaria, asegurando una ejecución ordenada y segura de los subprocesos involucrados.

Objetivos Específicos

- Implementar semáforos para sincronizar y coordinar la ejecución de los subprocesos de información personal, ingresos, egresos, referencias y formas de pago, garantizando que cada subproceso acceda a las regiones críticas de manera controlada y evitando condiciones de carrera.
- Desarrollar una interfaz gráfica que muestre el progreso de cada subproceso y del proceso general, actualizando las barras de progreso en tiempo real. Esto permitirá observar cómo los semáforos regulan el acceso a las regiones críticas y facilitan una ejecución paralela ordenada y eficiente.
- Utilizar semáforos para prevenir interferencias y conflictos entre subprocesos concurrentes, asegurando que las operaciones críticas se realicen de manera atómica y sin interrupciones, manteniendo la integridad y coherencia de los datos durante la simulación.

2. **Herramientas utilizadas**

Lenguaje de Programación: Java
IDE: Apache NetBeans 21
Sistema Operativo: Windows 11
Procesador: Intel Core i5 (o equivalente), con 4 núcleos.
Bibliotecas Utilizadas: Ninguna adicional al estándar de Java

3. **Desarrollo (Explicación del código)**

Subproceso.java

```
package labsemaforo;

import java.util.logging.Level;
import java.util.logging.Logger;

import java.util.concurrent.Semaphore;
import javax.swing.JProgressBar;

public class Subproceso extends Thread{
```

La clase Subproceso tiene varios atributos: aporte, que representa el porcentaje de contribución del subproceso al proceso general; tiempo, que indica el tiempo total que debe tomar el subproceso en completarse; semaforo, un objeto de tipo Semaphore que controla el acceso a las regiones críticas; jBarSubproceso, una barra de progreso

individual para el subproceso; y jBarGeneral, una barra de progreso que muestra el avance del proceso general. El constructor de la clase inicializa estos atributos con los valores proporcionados al crear una nueva instancia de Subproceso.

```
    private final int aporte;
    private final int tiempo;
    Semaphore semaforo;
    JProgressBar jBarSubproceso;
    JProgressBar jBarGeneral;

    public Subproceso(int aporte, int tiempo, Semaphore semaforo,
JProgressBar jBarSubproceso, JProgressBar jBarGeneral){
        this.aporte = aporte;
        this.tiempo = tiempo;
        this.semaforo = semaforo;
        this.jBarSubproceso = jBarSubproceso;
        this.jBarGeneral = jBarGeneral;
    }
```

El método run es el que se ejecuta cuando el hilo se inicia. Este método adquiere un permiso del semáforo mediante semaforo.acquire(), lo que asegura que solo un número controlado de hilos pueda acceder a la región crítica al mismo tiempo. Luego, en un bucle for que va de 1 a 100, se simula el progreso del subproceso "durmiendo" el hilo por una fracción del tiempo total. En cada iteración del bucle, la barra de progreso del subproceso se actualiza con jBarSubproceso.setValue(i). Una vez que el bucle termina, se actualiza la barra de progreso general del proceso sumando el aporte del subproceso completado.

```
    @Override
    public void run(){
        try {
            semaforo.acquire();
            for(int i = 1; i <= 100; i++){
                Thread.sleep(tiempo/100);
                jBarSubproceso.setValue(i);
            }
            jBarGeneral.setValue(aporte + jBarGeneral.getValue());
        } catch (InterruptedException ex) {
            Logger.getLogger(Subproceso.class.getName()).log(Level.SEVERE,
null, ex);
        } finally {
            semaforo.release();
        }
    }
}
```

Si el hilo es interrumpido, se captura una excepción, que se maneja registrando el error. Finalmente, el método run libera el permiso del semáforo, permitiendo que otros hilos puedan acceder a la región crítica. Esto asegura una ejecución ordenada y

segura de los subprocesos concurrentes, evitando condiciones de carrera y asegurando que las operaciones críticas se realicen de manera atómica.

JFrmCredito.java

```
package labsemaforo;

import java.util.concurrent.Semaphore;

public class JFrmCredito extends javax.swing.JFrame {

    public JFrmCredito() {
        initComponents();
    }
```

En el siguiente proceso se inicializa los métodos utilizados para el correcto funcionamiento de este, como veremos a continuación para el proceso mencionado se utilizará dos métodos, los mismo descritos en la parte superior de este documento

```
   private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        inicializarProgressBox();
        simularTarjetaCredito();
    }
```

En esta parte del código, se inicializarán nuestro progressBox, es decir todas las barras que nos permiten ver el progreso parcial del proceso hasta su culminación y como se desarrollan las actividades parciales de los mismo, todas inicializaran en 0

```
    private void inicializarProgressBox(){
        jBarInformacionPersonal.setValue(0);
        jBarIngresos.setValue(0);
        jBarEgresos.setValue(0);
        jBarReferencias.setValue(0);
        jBarFormasDePago.setValue(0);
        jBarGeneral.setValue(0);
    }
```

En esta parte del código, se establece los semáforos y sub procesos a realizar, se inicializan los mismo con los parámetros correspondientes de la siguiente manera Subproceso(porcentaje , tiempo en ms , inicializar semáforo, progresos en barra del proceso, progreso en barra general), se inicializan de igual manera los hilos

```
    private void simularTarjetaCredito(){
        Semaphore semaforo = new Semaphore(1);
        Subproceso informacionPersonal = new Subproceso(40, 5000,
         semaforo, jBarInformacionPersonal, jBarGeneral);
        Subproceso ingresos = new Subproceso(10, 10000, semaforo,
jBarIngresos, jBarGeneral);
        Subproceso egresos = new Subproceso(10, 10000, semaforo,
jBarEgresos, jBarGeneral);
        Subproceso referencias = new Subproceso(5, 5000, semaforo,
jBarReferencias, jBarGeneral);
        Subproceso formasDePago = new Subproceso(35, 10000, semaforo,
jBarFormasDePago, jBarGeneral);

        informacionPersonal.start();
```

```
        ingresos.start();
        egresos.start();
        referencias.start();
        formasDePago.start();
    }
```

En esta parte se llama el main, la misma inicia los procesos dentro del jFrame.

```
public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new JFrmCredito().setVisible(true);
            }
        });
    }
```

Se analizará los resultados obtenidos a continuación en el siguiente apartado

4. **Análisis de resultados:**

El principal propósito del laboratorio resultado era observar el comportamiento de los semáforos dentro de un programa realizado en Java, por lo mismo se realizó una simulación de un proceso en el cual es necesario que los procesos se realicen uno por uno.

Se realizo la misma simulación, pero sin el uso de semáforos, es decir nuestro código en ese momento, tenía la siguiente entrada

```
@Override
public void run(){
    try {

        for(int i = 1; i <= 100; i++){
            Thread.sleep(tiempo/100);
            jBarSubproceso.setValue( n: i);
        }
        jBarGeneral.setValue(aporte + jBarGeneral.getValue());
    } catch (InterruptedException ex) {
        Logger.getLogger( name: Subproceso.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
    } finally {

    }
}
```

Figura 1: Código sin la utilización de semáforos

De esta manera al realizar la simulación se obtuvo el siguiente resultado

De esta manera al realizar la simulación se obtuvo el siguiente resultado: Si bien se obtuvo un tiempo un poco más rápido que al realizarse con semáforos, sin embargo, no existe un orden en los subprocesos, si bien el resultado general sigue siendo el mismo, hay que recordar que este es un proceso que se busca se realice en orden entonces no es viable esta solución. Entonces le agregamos los semáforos para analizar los cambios.

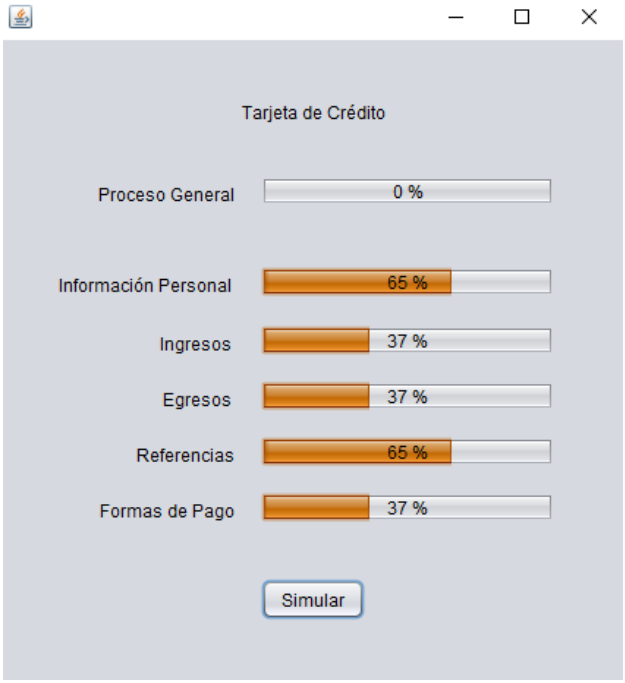

Figura 2: Simulación sin la utilización de semáforos

Después agregamos semáforos al momento de realizar el proceso, utilizando el código proporcionado en el apartado 3, con esto se obtuvo los siguientes resultados, el proceso esta vez tuvo un pequeño retraso, sin embargo, se realizó en un orden determinado, como se observa en la imagen, con ayuda de los semáforos, se espera a que termine un proceso antes de continuar con el siguiente

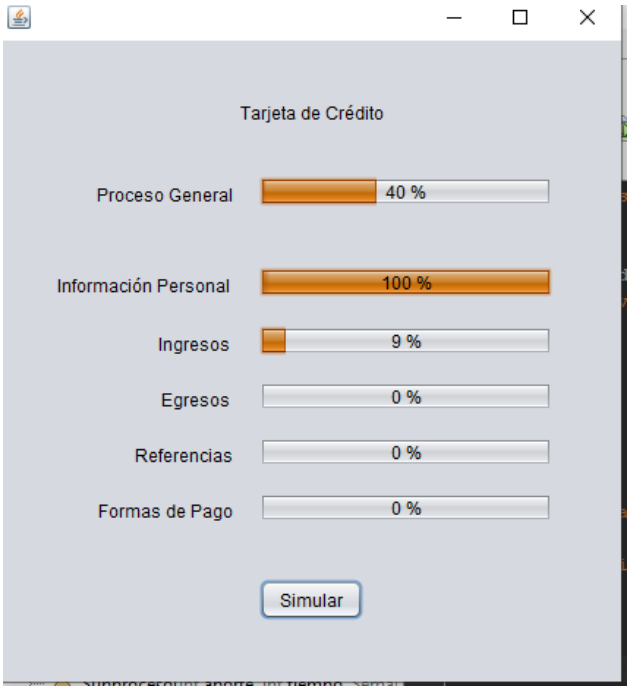

Figura 3: Simulación con la utilización de semáforos

También fue más visible el progreso general, y su aporte para el mismo, esto se debe a que se espera que un subproceso termine para continuar con el siguiente, para todo integrarse en un resultado final.

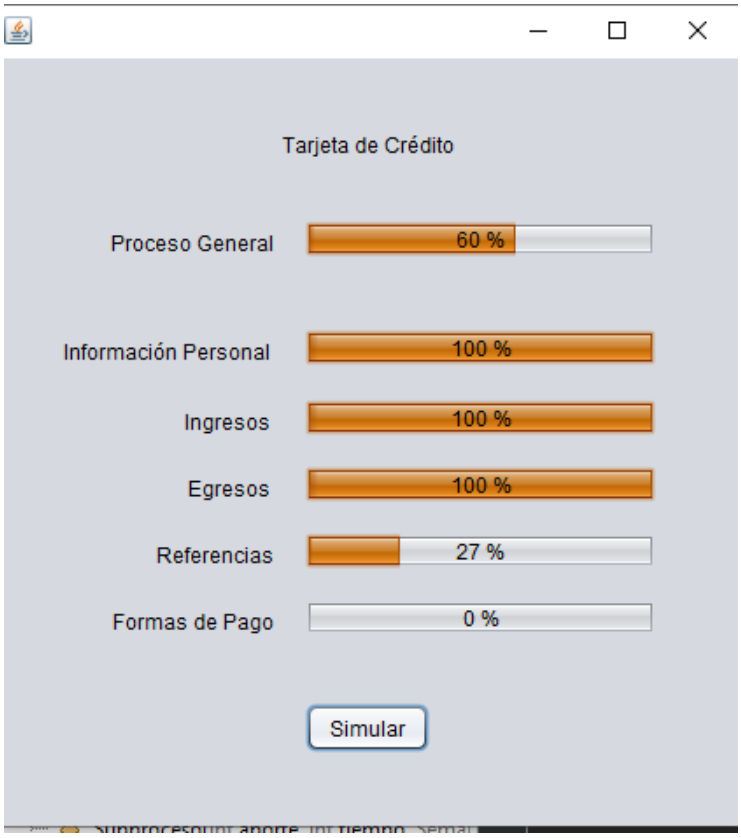

Figura 3: Simulación sin la utilización de semáforos, se observa cómo se llena paulatinamente la barra de progreso general

Como resultado se puede podría presentar en la siguiente tabla:

	Sin semáforos	Usando Semáforos
Tiempo Empleado	13000 ms	27000 ms
Subprocesos en simultaneo	5	1
Cantidad de hilos	5	5
Visualización de progreso en barra general	Lenta	Rápida

Tabla 1: Análisis de resultados importantes

Si bien pareciera que sin semáforos el proceso se realiza más rápido, el proceso fue más desordenado y poco apegado a la realidad, en un proceso que es necesario realizar paso a paso para tener un buen resultado al final de este

5. Conclusiones

- La implementación de semáforos en el programa asegura una ejecución ordenada y controlada de los subprocesos, evitando condiciones de carrera y conflictos entre

ellos. Esto garantiza que cada subproceso acceda a las regiones críticas de manera sincronizada, manteniendo la integridad y coherencia de los datos. La utilización de semáforos permite que múltiples hilos trabajen en paralelo sin interferir entre sí, mejorando la eficiencia general del sistema.

- La interfaz gráfica desarrollada permite visualizar en tiempo real el progreso de cada subproceso y del proceso general mediante barras de progreso. Esto facilita el monitoreo del estado y avance de cada tarea, proporcionando una retroalimentación visual inmediata. Además, el uso de semáforos asegura que las operaciones críticas se realizan atómicamente y sin interrupciones, evitando interferencias entre subprocesos concurrentes y manteniendo la integridad y coherencia de los datos en el proceso de simulación.

6. Recomendaciones

- Evaluar la posibilidad de ajustar el número de permisos del semáforo según la naturaleza y dependencia de los subprocesos. Si los subprocesos pueden ejecutarse simultáneamente sin conflicto, incrementar el número de permisos del semáforo puede mejorar la concurrencia y reducir el tiempo total de ejecución. Esto es útil si algunos subprocesos no dependen del acceso exclusivo a la región crítica.

- Implementar mecanismos para monitorear y registrar el uso del semáforo, como contadores de permisos adquiridos y liberados, tiempos de espera para adquirir permisos, y la frecuencia de acceso a la región crítica. Estos datos pueden proporcionar información valiosa sobre el comportamiento del sistema y ayudar a identificar posibles cuellos de botella o problemas de sincronización. Utilizar esta información para ajustar y optimizar la configuración del semáforo puede llevar a una ejecución más eficiente y ordenada de los subprocesos.

Referencias:

- Moreno Segura, G. J. (2022). PROGRAMACIÓN PARALELA Y DISTRIBUIDA. ARQUITECTURAS PARALELAS Y DISTRIBUIDAS. PROGRAMACIÓN DE APLICACIONES MULTIPROCESO. http://crea.ujaen.es/jspui/handle/10953.1/17697
- Giulianelli, D. A., Rodríguez, R. A., & Vera, P. M. (2006, octubre). Modelo de aplicaciones con procesos concurrentes y distribuidos. XII Congreso Argentino de Ciencias de la Computación. http://sedici.unlp.edu.ar/handle/10915/22231