UNIVERSIDAD DE LAS FUERZAS ARMADAS

Computación Paralela

Departamento de Ciencias de la Computación

Laboratorio 1 Parcial 2 "Semáforos"

Estudiantes:
Josué Merino, Angelo Sánchez, Justin Villarroel

Docente:
Ing. Carlos Andrés Pillajo Bolagay

1. Objetivos

1.1. Objetivo General

Desarrollar los conocimientos de hilos y semáforos a través de la implementación de un programa que muestre los avances de un proceso para obtener tarjetas de crédito de una entidad bancaria. Usar semáforos para el ingreso a las regiones críticas que podría tener el programa.

1.2. Objetivos Específicos

- Investigar la teoría básica de hilos y semáforos para comprender los conceptos de sincronización y exclusión mutua en la programación concurrente.
- Garantizar que los semáforos prevengan condiciones de carrera y aseguren la exclusión mutua.

2. Resultados de Aprendizaje

1. **Análisis de ingeniería.** La capacidad de identificar, formular y resolver problemas de ingeniería en su especialidad; elegir y aplicar de forma adecuada métodos analíticos, de cálculo y experimentales ya establecidos; reconocer la importancia de las restricciones sociales, de salud y de seguridad, ambientales, económicas e industriales.
2. **Proyectos de ingeniería.** Capacidad para proyectar, diseñar y desarrollar productos complejos (piezas, componentes, productos acabados, etc.) procesos y sistemas de su especialidad, que cumplen con los requisitos establecidos, incluyendo tener conciencia de los aspectos sociales, de salud y seguridad, ambientales, económicos e industriales; así como seleccionar y aplicar métodos de proyectos apropiados.
3. **Aplicación práctica de la ingeniería.** Comprensión de las técnicas aplicables y métodos de análisis, proyecto e investigación y sus lineamientos en el ámbito de su especialidad.
4. **Comunicación y trabajo en equipo.** Capacidad para comunicar eficazmente información, ideas, problemas y soluciones en el ámbito de ingeniería y con la sociedad en general.

3. Herramientas Utilizadas

- Java
- Netbeans IDE
- Laptop
- Internet

4. Desarrollo

Implementar la clase Subproceso con la siguiente codificación:

```
1  import java.util.concurrent.Semaphore;
2
3  public class Subproceso extends Thread {
4      private int tiempo; // en segundos
5      private int aporte; // en porcentaje
6      private Semaphore semaforo;
7      javax.swing.JProgressBar barraProgreso;
8      javax.swing.JLabel label;
9      public int aporteTotal;
10
11
12
13
14     public Subproceso( int tiempo, int aporte, Semaphore semaforo,
           javax.swing.JProgressBar barraProgreso, javax.swing.JLabel
           label, int aporteTotal) {
15         this.tiempo = tiempo;
16         this.aporte = aporte;
17         this.semaforo = semaforo;
18         this.barraProgreso = barraProgreso;
19         this.label = label;
20         this.aporteTotal = aporteTotal;
21     }
22
23     @Override
24     public void run() {
25         try {
26
27                 semaforo.acquire();
28                 for (int i = 0; i <= 100; i++) {
29                     barraProgreso.setValue(i);
30                     label.setText(i+"%");
31                     aporteTotal +=i;
32                     Thread.sleep(tiempo * 50);
```

```
33                  }
34                  semaforo.release();
35
36
37          } catch (InterruptedException e) {
38              e.printStackTrace();
39          }
40      }
41
42      public int getAporte() {
43          return aporte;
44      }
45
46      public int getAporteTotal() {
47          return aporteTotal;
48      }
49  }
```

El código en Java define la clase Subproceso, que extiende Thread para gestionar la ejecución concurrente mediante hilos. Esta clase utiliza semáforos (Semaphore) para controlar el acceso a regiones críticas, asegurando la sincronización adecuada entre hilos. La clase tiene varios atributos, incluyendo el tiempo de espera (tiempo), el aporte porcentual (aporte), una barra de progreso (barraProgreso) y una etiqueta (label) de Swing, y un acumulador de aporte total (aporteTotal). El constructor inicializa estos atributos con los valores proporcionados. En el método run, se adquiere el semáforo para bloquear el acceso a la región crítica, luego se actualizan la barra de progreso y la etiqueta en un bucle que simula el progreso, incrementando el aporteTotal y durmiendo el hilo según el tiempo especificado. Finalmente, el semáforo se libera para permitir que otros hilos accedan a la región crítica. Los métodos getAporte y getAporteTotal proporcionan acceso a los valores de aporte y aporteTotal, respectivamente.

La programación de la interfaz de usuario es la siguiente, en donde el usuario ve las barras de progreso:

```
1  import java.awt.Color;
2  import java.util.Arrays;
3  import java.util.Collections;
4  import javax.swing.*;
5
6  import java.util.concurrent.Semaphore;
7
8  public class SuperInterface extends javax.swing.JFrame {
9
10     /**
11      * Creates new form SuperInterface
12      */
```

```
13      public SuperInterface() {
14          initComponents();
15
16      }
17
18
19      Subproceso[] subprocesos = new Subproceso[5];
20      private void jButton1ActionPerformed(java.awt.event.ActionEvent
            evt) {
21          int aporteTotal=0;
22          if(Thread.activeCount()==2){
23              Semaphore semaforo;
24              semaforo = new Semaphore(1);
25
26              subprocesos[0] = new
                    Subproceso(1,40,semaforo,jProgressBar5,jLabel9,aporteTotal);
27              subprocesos[1]= new
                    Subproceso(2,10,semaforo,jProgressBar1,
                    jLabel10,aporteTotal);
28              subprocesos[2] = new
                    Subproceso(2,10,semaforo,jProgressBar3,
                    jLabel11,aporteTotal);
29              subprocesos[3] = new
                    Subproceso(1,5,semaforo,jProgressBar7,
                    jLabel12,aporteTotal);
30              subprocesos[4] = new
                    Subproceso(2,35,semaforo,jProgressBar4,
                    jLabel13,aporteTotal);
31
32              Collections.shuffle(Arrays.asList(subprocesos));
33
34              for (Subproceso subproceso : subprocesos) {
35                  subproceso.start();
36              }
37
38              new Thread(() -> {
39                  try {
40                      while (true) {
41                          int progresoTotal = 0;
42                          for (Subproceso subproceso : subprocesos) {
43                              progresoTotal += subproceso.getAporte() *
                                    subproceso.barraProgreso.getValue() /
                                    100;
44                          }
45
46                          System.out.println(aporteTotal);
47
48                          jProgressBar2.setValue(progresoTotal);
49                          jLabel8.setText(progresoTotal + "%");
```

```
50                          Thread.sleep(100);
51                          if(jProgressBar2.getValue()==100){
52                              JOptionPane.showMessageDialog(rootPane,
                                    "Pago Realizado Correctamente");
53                              break;
54                          }
55                      }
56                  } catch (InterruptedException e) {
57                      e.printStackTrace();
58                  }
59              }).start();
60          }else{
61              JOptionPane.showMessageDialog(rootPane, "Los procesos se
                    encuentran en ejecucion");
62          }
63      }
64
65      private void jButton2ActionPerformed(java.awt.event.ActionEvent
            evt) {
66          System.out.println(Thread.activeCount());
67          if(Thread.activeCount()>2){
68              JOptionPane.showMessageDialog(rootPane, "Proceso No
                    Realizado");
69          }else{
70              jProgressBar1.setValue(0);
71              jProgressBar2.setValue(0);
72              jProgressBar3.setValue(0);
73              jProgressBar4.setValue(0);
74              jProgressBar5.setValue(0);
75              jProgressBar6.setValue(0);
76              jProgressBar7.setValue(0);
77              jLabel8.setText("%");
78              jLabel9.setText("%");
79              jLabel10.setText("%");
80              jLabel11.setText("%");
81              jLabel12.setText("%");
82              jLabel13.setText("%");
83          }
84      }
85
86      private void jProgressBar2MouseDragged(java.awt.event.MouseEvent
            evt) {
87          jProgressBar1.setForeground(Color.green);
88      }
89
90
91      public static void main(String args[]) {
92          java.awt.EventQueue.invokeLater(new Runnable() {
93              public void run() {
```

```
94                  new SuperInterface().setVisible(true);
95              }
96          });
97      }
98      private javax.swing.JButton jButton1;
99      private javax.swing.JButton jButton2;
100     private javax.swing.JLabel jLabel1;
101     private javax.swing.JLabel jLabel10;
102     private javax.swing.JLabel jLabel11;
103     private javax.swing.JLabel jLabel12;
104     private javax.swing.JLabel jLabel13;
105     private javax.swing.JLabel jLabel2;
106     private javax.swing.JLabel jLabel3;
107     private javax.swing.JLabel jLabel4;
108     private javax.swing.JLabel jLabel5;
109     private javax.swing.JLabel jLabel6;
110     private javax.swing.JLabel jLabel7;
111     private javax.swing.JLabel jLabel8;
112     private javax.swing.JLabel jLabel9;
113     private javax.swing.JProgressBar jProgressBar1;
114     private javax.swing.JProgressBar jProgressBar2;
115     private javax.swing.JProgressBar jProgressBar3;
116     private javax.swing.JProgressBar jProgressBar4;
117     private javax.swing.JProgressBar jProgressBar5;
118     private javax.swing.JProgressBar jProgressBar6;
119     private javax.swing.JProgressBar jProgressBar7;
120
121 }
```

Se tiene la interfaz en donde al dar clic en el botón ïniciar"se ejecutan los procesos.

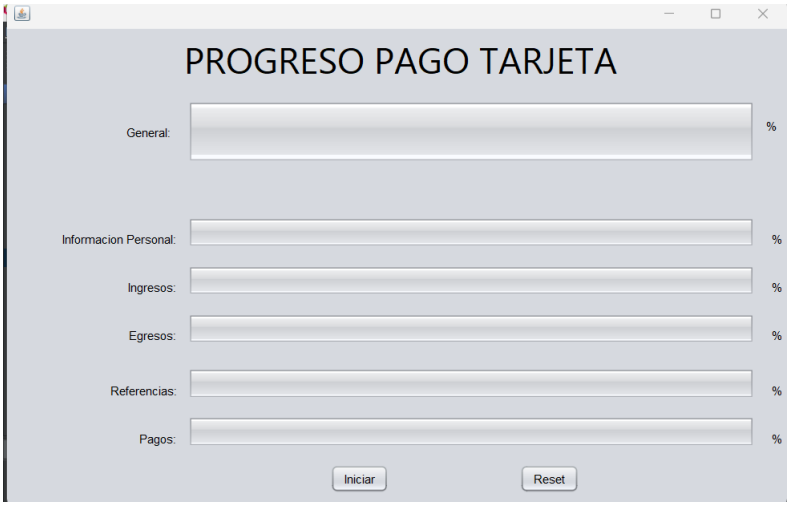

Figura 1: Inicio del Programa

Empieza la corrida del programa como se puede observar empieza a cargar las referencias.

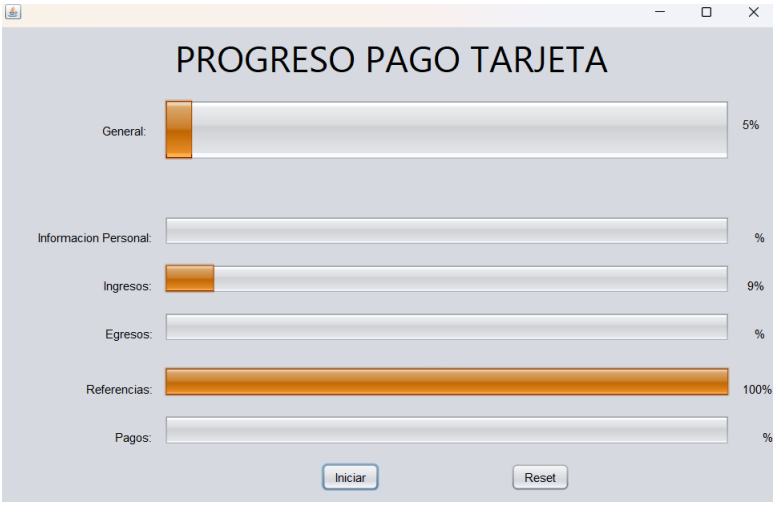

Figura 2: Ejecución

Al terminar el proceso aparece el siguiente mensaje:

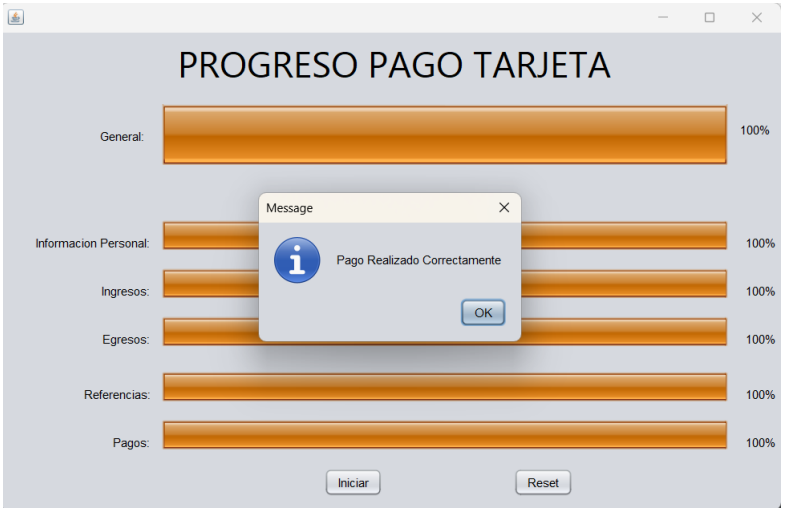

Figura 3: Resultado

Se resetea el programa para empezar desde cero

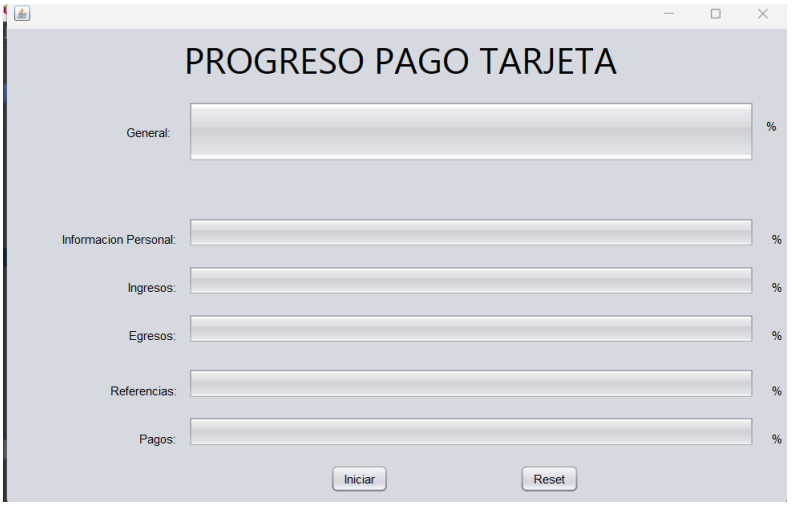

Figura 4: Reset

Ahora en el aplicativo se ejecuta primero la información personal.

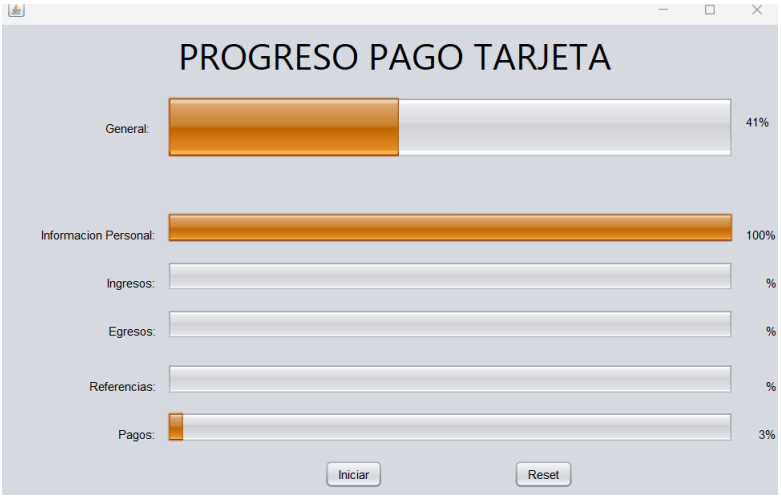

Figura 5: Ejecución 2

5. Conclusiones

- La implementación del programa demostró cómo los semáforos pueden ser utilizados eficazmente para sincronizar múltiples hilos y garantizar que los subprocesos críticos sean gestionados correctamente sin condiciones de carrera o inconsistencias en los datos.
- Las barras de progreso proporcionaron una representación visual clara y comprensible del avance del proceso general y de cada subproceso, lo que facilitó el seguimiento del estado de la tarea de obtención de tarjetas de crédito.
- La estructura del código y la utilización de semáforos demostraron ser efectivos para manejar la concurrencia de manera simple y clara, asegurando un flujo de trabajo sin conflictos y con un buen rendimiento.

6. Recomendaciones

- Considerar la optimización del código para mejorar la eficiencia y escalabilidad del programa. Esto podría incluir la revisión y ajuste del uso de semáforos para minimizar tiempos de espera y aumentar el rendimiento general.
- Realizar pruebas exhaustivas para asegurar que el programa funciona correctamente bajo diferentes escenarios y cargas de trabajo. Identificar y corregir cualquier posible punto de falla o error que pueda surgir durante el uso del programa.

- Investigar y considerar el uso de otras técnicas de sincronización y gestión de concurrencia, como monitores, bloqueos de lectura/escritura o estructuras de datos concurrentes, que puedan ofrecer ventajas en términos de simplicidad o rendimiento en ciertos contextos.