HILOS - Exclusión mutua

Objetivo

El objetivo de la guía es el de afianzar el conocimiento acerca de los hilos para resolver problemas reales con el uso del lenguaje de programación Java. Además le presentará las diferentes variantes de dicho mecanismo y los pros y contras de cada una.

Competencias

- Entender el concepto de Hilo
- Utilizar el lenguaje Java para implementar Hilos
- Exclusión mutua
- Resolver problemas reales mediante el uso de Hilos

Introducción

La exclusión mutua –o sección crítica– es un problema básico y fundamental de sincronización entre procesos con memoria compartida; se trata de asegurar el acceso ordenado a recursos compartidos para impedir errores e inconsistencias.

Un conjunto de procesos independientes que pueden ser considerados cíclicos ejecutan en cada ciclo una parte de código que accede y modifica recursos o zonas de memoria compartidas, la sección crítica. La intercalación de instrucciones en esas secciones críticas provocan condiciones de carrera que pueden generar resultados erróneos dependiendo de la secuencia de ejecución.
Debemos asegurar la exclusión mutua de la ejecución de esas secciones críticas; mientras se ejecuta una de ellas no se debe permitir la ejecución de las secciones críticas de otros procesos.

Exclusión mutua

Se debe asegurar que solo uno de los procesos ejecuta instrucciones de la sección crítica.

Progreso o libre de interbloqueos (deadlock free o lock-free)

Si varios procesos desean entrar a la sección crítica al menos uno de ellos debe poder hacerlo.

Espera limitada o libre de inanición (starvation free o wait-free)

Cualquier proceso debe poder entrar a la sección crítica en un tiempo finito. Esta condición es deseable pero no siempre se puede asegurar.

Problemas potenciales de control:

- Exclusión mutua (secciones críticas): Solo un programa a la vez puede ejecutar su sección crítica
- Deadlock (abrazo mortal)
- Starvation (inanición): Es posible que un proceso esperando por un recurso bloqueado nunca lo obtenga

Solución. Los algoritmos de Dekker y finalmente la solución de Peterson.

El código analizado muestra las aproximaciones de Dekker y la solución de Peterson al problema de la sección crítica para dos competidores. La idea es experimentar con los valores de tiempo de “secciones críticas y no críticas”, inclusive eliminando la aleatoriedad del mismo, a fin de tratar de obtener, para las aproximaciones de Dekker, estados indeseables de interbloqueo.

Examine el programa:

Listing 1. Implementación de algoritmos de exclusión mutua

```
/**
 * @(#) Hilo.java
 *     La implementación de hilos se realiza mediante
 *     la herencia de la clase Thread.
 *     debe implementar el método run que será el encargado
 *     de ejecutar el código del hilo.
 *     Un hilo entra al estado runnable (preparado),
 *     al momento de la llamada al método start
 *     (heredado de Thread), llamada que efectúa el padre del hilo.
 *     El método yield, por último, permite seleccionar
 *     otro hilo para su ejecución.
 *     El código analizado muestra las aproximaciones de
 *     Dekker y la solución de Peterson al
 *     problema de la sección crítica para dos competidores.
 *     La idea es experimentar con los valores de tiempo de
 *     “secciones críticas y no críticas”,
 *     inclusive eliminando la aleatoriedad del mismo,
```

```
 *     a fin de tratar de obtener, para las
 *     aproximaciones de Dekker, estados indeseables de interbloqueo.
 *
 * @author Carlos I. Buchart I
 * @version 1.00 2017/9/5
 */

public class Hilo extends Thread {
    private String name;
    private int m_iId;
    private ModeloExclusion m_oExc;

    public Hilo(String sName, int iId, ModeloExclusion oExc) {
       name = sName;
       m_iId = iId;
       m_oExc = oExc;
    }

    public void run() {
       while (true) {
             System.out.println(name + " desea entrar a la SC");
             m_oExc.entrarSC(m_iId);
             System.out.println(name + " entro a la SC");
             ModeloExclusion.SC();
             m_oExc.salirSC(m_iId);
             System.out.println(name + " salio de la SC");
             ModeloExclusion.noSC();
       }
    }
}

/**
 * @(#)ModeloExclusion.java
 *
 *
 * @author
 * @version 1.00 2017/9/5
 */
public abstract class ModeloExclusion {
    public static final int TURN_0 = 0;
    public static final int TURN_1 = 1;
    public static final int TIME = 2000;

    public static void SC() {
       try {
             Thread.sleep((int) (Math.random() * TIME + 1000));
       }
       catch (InterruptedException e) {
       }
    }
```

```
    public static void noSC() {
       try {
             Thread.sleep((int) (Math.random() * TIME));
       }
       catch (InterruptedException e) {}
    }
    public abstract void entrarSC(int iId);
    public abstract void salirSC(int iId);
}


/**
 * @(#)Dekker_1.java
 *     No satisface condición de progreso
 *     ya que requiere alternancia estricta.
 * @author
 * @version 1.00 2017/9/5
 */
public class Dekker_1 extends ModeloExclusion {
    private volatile int m_iTurn;
    public Dekker_1() {
       m_iTurn = TURN_0;
    }
    public void entrarSC(int iId) {
       while (m_iTurn != iId)
             Thread.yield();
    }
    public void salirSC(int iId) {
       m_iTurn = 1 - iId;
    }
}

/**
 * @(#)Dekker_2.java
 *     No satisface condición de progreso ya que ambos
 *     hilos podrían fijar la bandera
 *     respectiva  en verdadero.
 * @author
 * @version 1.00 2017/9/5
 */
public class Dekker_2 extends ModeloExclusion {
            private volatile boolean[] m_bFlag = new boolean[2];
            public Dekker_2() {
                        m_bFlag[0] = false;
                        m_bFlag[1] = false;
            }
            public void entrarSC(int iId) {
                        int other = 1 - iId;
                        m_bFlag[iId] = true;
            while (m_bFlag[other] == true)
                        Thread.yield();
```

```
            }
            public void salirSC(int iId) {
                        m_bFlag[iId] = false;
            }
}


/**
 * @(#)Peterson.java
 *     Solucion final
 * @author: Carlos I. Buchart I
 * @version 1.00 2017/9/5
 */
public class Peterson extends ModeloExclusion {
    private volatile int m_iTurn;
    private volatile boolean[] m_bFlag = new boolean[2];

    public Peterson() {
       m_iTurn = TURN_0;
       m_bFlag[0] = false;
       m_bFlag[1] = false;
    }
    public void entrarSC(int iId) {
       int other = 1 - iId;
       m_bFlag[iId] = true;
       m_iTurn = other;
       while (m_bFlag[other] == true && m_iTurn == other)
             Thread.yield();
    }
    public void salirSC(int iId) {
       m_bFlag[iId] = false;
    }
}


/**
 * @(#)TExclusionMutua.java
 *           Tarea
 *     Implemente una aplicacion con n procesos.
 *
 * @author
 * @version 1.00 2017/9/5
 */


public class TExclusionMutua {

    public static void main(String[] args) {
       ModeloExclusion alg = new Dekker_1(); // alternar entre los
algoritmos
```

```
        Hilo hilo1 = new Hilo("Hilo 1", 0, alg);
        Hilo hilo2 = new Hilo("Hilo 2", 1, alg);
        hilo1.start();
        hilo2.start();
    }
}
```