Programación de Sistemas Concurrentes y Distribuidos 1ª Convocatoria curso 13/14

5 de febrero de 2014

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Ejercicio 1 (2.0 ptos.)

Considérese el código que se muestra en el Anexo I, en el que se lanzan dos procesos concurrentes que se coordinan mediante semáforos.

El ejercicio pide:

1. Construir un modelo red de Petri para el programa considerado
2. A partir del modelo, construir su grafo de estados alcanzables
3. Analizando el grafo, determinar si el programa tiene problemas de bloqueo o de falta de equidad
4. Demostrar que se cumple el siguiente invariante: $(x \geq 0) \wedge (y \leq 0) \wedge (x + y \geq 0)$.

Ejercicio 2 (2.0 ptos.)

Considérese el código que se muestra a continuación, en el que se lanzan M procesos concurrentes con el objetivo de que entre todos encuentren el valor máximo de un vector. Una vez todos han acabado, el proceso `informador` muestra el valor máximo del vector. Se pide completar el código de acuerdo a la especificación, utilizando semáforos como elementos de sincronización donde se considere oportuno. No se permite definir nuevos procesos. Es un objetivo que el programa se ejecute de la manera más eficiente posible.

```
constant integer N := ... //lo que sea
                 M := ... //lo que sea
                 NM := N*M
integer array[1..NM] val := ... //lo que sea, ya inicializado
integer max

// código a completar

process P(i: 1..M)::
    // código a completar

    operation maxTrozo(integer array[1..NM] v, int i1,i2): integer
    //Pre: 1<=i1<=NM, 1<=i2<=NM, i1<=i2
    //Post: devuelve el valor máximo de las componentes
    //      v[i1],v[i1+1],...,v[i2]

    // código a completar

    end operation

   // código a completar
   // al terminar todos los procesos, "max" contiene el
   // valor máximo del vector "val"
end process

process informador::
    // código a completar

    write('Max=␣')
    write(max)
end process
```

Ejercicio 3 (2.0 ptos.)

El fondo bibliográfico de una librería está formado por 100 libros distintos. De cada libro se conoce su código ISBN (representado por un valor entero que actúa como identificador único de la publicación) y el número de unidades disponibles para su venta.

El librero desea mejorar sus oportunidades de negocio por medio de un sistema automático de venta de libros. Para ello, ha enviado por correo postal a sus 50 mejores clientes un catálogo en papel con todos los títulos disponibles. El comportamiento de estos clientes es simple. Cada cliente decide cuántos libros va a comprar (un máximo de 20 por cliente) y qué títulos concretos selecciona. Para cada libro enviará una orden de compra al sistema de venta automática, especificando el ISBN de la publicación que desea adquirir. Dependiendo de la disponibilidad de unidades, el sistema le indicará si ha

tenido éxito o no su petición de compra. En caso de que ésta se haya efectuado correctamente, el cliente tendrá la opción de decidir aleatoriamente si se queda definitivamente el libro o si lo devuelve. En ambos casos, el sistema de venta le notificará la recepción de su decisión final.

Se pide diseñar, utilizando la notación algorítmica vista en clase, un programa que simule el sistema anteriormente descrito. El programa consta de dos tipos de procesos: el proceso que implementa el sistema de venta automática y los procesos clientes. La comunicación y coordinación de los procesos para garantizar las condiciones del problema debe ser programada utilizando Linda.

Por simplicidad podéis asumir que las siguientes dos funciones están disponibles y pueden ser usadas para programar el sistema:

- Para la programación del proceso sistema de venta, la función *cargarCatálogo()* inicializa el contenido de la estructura de datos donde almacenéis la información del fondo bibliográfico.
- Para la programación de los procesos cliente, la función *seleccionaLibro()* devuelve el ISBN de un libro cualquiera del catálogo en papel recibido por un cliente. Podéis asumir también que en el contexto de un proceso, cada vez que se invoca la función, ésta devuelve un ISBN diferente.

Ejercicio 4 (2.0 ptos.)

Una piscina municipal dispone de 8 calles. La dirección ha decidido que cada calle pueda ser simultáneamente utilizada por dos persona como máximo. En la actualidad, la instalación puede ser usada por 100 usuarios distintos.

El comportamiento de un usuario es siempre el mismo. Durante un número aleatorio de veces acude a la piscina a nadar. En cada ocasión, cuando llega lo primero que hace es solicitar permiso. El usuario puede especificar la calle concreta en la que quiere nadar o dejar que se le asigne una cualquiera. Cuando se dan las condiciones para que pueda comenzar su actividad se le comunica, indicándole la calle asignada si fuera necesario. Después de un determinado tiempo el usuario decide finalizar, y de esta forma libera la calle que ocupaba.

Por otro lado, en la instalación trabaja una persona encargada del mantenimiento de la piscina. En orden secuencial, va limpiando una calle tras otra. Para ello la calle objeto del mantenimiento debe estar vacía. Una vez ha limpiado todas las calles, comienza de nuevo por la primera de ellas.

Se pide:

- Escribir un programa concurrente, utilizando la notación algorítmica presentada

en clase, que simule el funcionamiento del sistema descrito. En primer lugar se deberá definir un monitor que controle el uso de las calles conforme las condiciones descritas y la sincronización necesaria entre los usuarios y el responsable del mantenimiento. Desde el punto de vista del monitor, todos los procesos del sistema tienen la misma prioridad a la hora de usar las instalaciones. Posteriormente, se deberán diseñar un proceso usuario y un proceso personal de mantenimiento, conforme a las condiciones previamente descritas.

- Razonar sobre si la versión previamente programada presenta problemas de inanición y bloqueo de procesos. En caso afirmativo, esbozar una solución alternativa donde los problemas identificados sean resueltos.

Ejercicio 5 (2.0 ptos.)

Dado el siguiente algoritmo, implementad en Java un programa concurrente que se comporte de una manera equivalente. Para los aspectos de sincronización se puede usar el mecanismo que se crea más conveniente.

```
constant integer N := 23
integer x,y,z

process Main::
    read(x,y,z)
    for i := 1..N
        new instance of Modificador();
end process

process Modificador::
    integer miX

    read(miX)
    <await  x > y
        x := 2*x
        y := y+1
    >
    <z := z+3+miX>
    <await x > 0
        y := 2*x+y
    >
end process
```

Anexo I

```
//fichero: Ejercicio.java

import java.util.concurrent.Semaphore;

public class Ejercicio{
    static Semaphore S1 = new Semaphore(1);
    static Semaphore S2 = new Semaphore(0);

    public static void main(String[] args){
        Datos d = new Datos(0,0);
        Thread proc1 = new Thread(new ProcExamen(d,S1,S2,1));
        Thread proc2 = new Thread(new ProcExamen(d,S1,S2,2));

        proc1.start();
        proc2.start();
    }
}

//fichero: ProcExamen.java
import java.util.concurrent.Semaphore;

class ProcExamen implements Runnable{
    private Datos losDatos;
    private static Semaphore s1,s2;
    private int tipo;

    public ProcExamen(Datos d, Semaphore S1,
                      Semaphore S2, int Tipo){
        this.losDatos = d;
        this.s1 = S1;
        this.s2 = S2;
        this.tipo = Tipo;
    }

    public void run(){
        while(true){
            if ( 0 != (tipo%2)){//impares
                try{
                    s1.acquire();
                    losDatos.incX(1);
                    losDatos.incY(-1);
                    s2.release();
                }catch(InterruptedException ex){
                    ex.printStackTrace();
                }
            }else{ //pares y 0
                try{
                    s2.acquire();
                    losDatos.incY(1);
                    losDatos.incX(-1);
```

```
                    s1.release();
                }catch(InterruptedException ex){
                    ex.printStackTrace();
                }
            }
        }
    }
}

//fichero: ProcExamen.java
class Datos{
    private int x,y;

    Datos(int xx, int yy){
        x = xx;
        y = yy;
    }

    public synchronized void incX(int v){
        x = x+v;
    }

    public synchronized void incY(int v){
        y = y+v;
    }
}
```