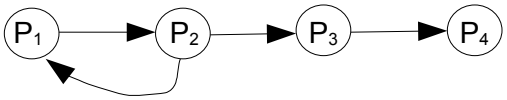
SISTEMI OPERATIVI IIN

seconda prova in itinere - 31.05.2010

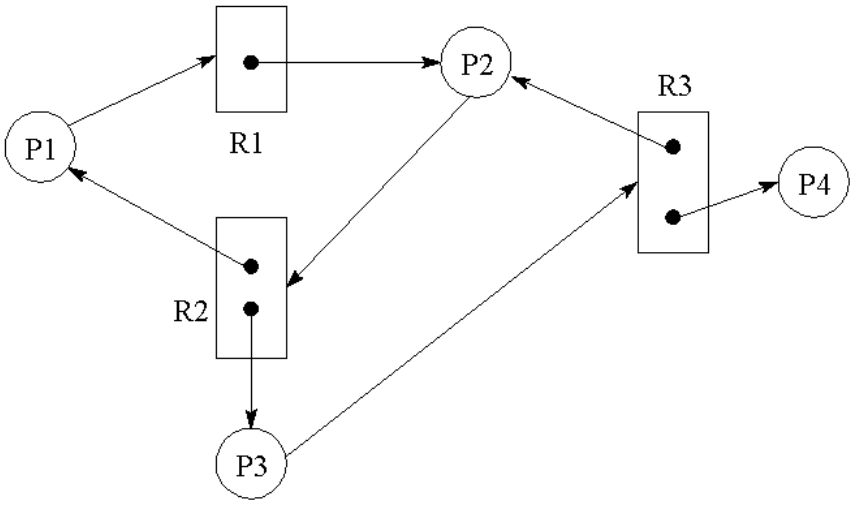
Nome: _____ Cognome: _____

Esercizio 1. (punti 10)

- Dato il seguente grafo di attesa, verificare se esiste stallo ed eventualmente quali sono i processi coinvolti:



- Dato il seguente grafo di allocazione delle risorse, verificare se esiste stallo ed eventualmente quali sono i processi coinvolti:



Esercizio 2. (punti 20)

In un sistema, un thread distributore (D), due thread produttori (A e B) ed un thread consumatore (C) cooperano nel modo seguente:

- all'interno di un ciclo infinito: il thread produttore A, produce un oggetto chiamando il metodo *Object produceObject()* (metodo dato, della stessa classe che definisce il thread A) e lo inserisce in un array di oggetti di dimensione 2 (condiviso con B). Una volta inserito l'oggetto, notifica il thread D e si pone in attesa di produrre un nuovo oggetto;
- all'interno di un ciclo infinito: il thread produttore B, produce un oggetto chiamando il metodo *Object produceObject()* (metodo dato, della stessa classe che definisce il thread B) e lo inserisce in un array di oggetti di dimensione 2 (condiviso con A). Una volta inserito l'oggetto notifica il thread D e si pone in attesa di produrre un nuovo oggetto;
- all'interno di un ciclo infinito: il thread consumatore C, consuma i due oggetti dell'array chiamando il metodo *void consumeObjects(Object a, Object b)* (metodo dato, della stessa classe che definisce il thread C). Una volta rimossi gli oggetti notifica il thread D e si pone in attesa di consumare nuovi oggetti;
- all'interno di un ciclo infinito: il thread distributore D dopo aver ricevuto le notifiche di A e B notifica a C la possibilità di consumare. Dopo aver ricevuto la notifica da C, notifica A e B di avviare una nuova produzione.

Scrivere il programma Java che realizza il funzionamento descritto sopra usando i semafori per risolvere i problemi di sincronizzazione.

Soluzione

Esercizio 1.

- Il grafo di attesa è definito per risorse con una singola istanza. In questo caso la presenza di un ciclo nel grafo è condizione *necessaria e sufficiente* alla presenza di uno stallo. Nel grafo dato nell'esercizio c'è un ciclo che include i processi P_1 e P_2 che sono perciò in stallo.
- Il grafo include istanze multiple per tipo di risorsa. In questo caso la presenza di uno o più cicli è condizione *necessaria* allo stallo, ma non *sufficiente*. Nel grafo dato nell'esercizio esistono 2 cicli: P_1->R_1->P_2->R_2->P_1 e P_2->R_2->P_3->R_3->P_2. Però, il processo P_4 non è incluso in un ciclo e non è in attesa. Può perciò eseguire, terminare e liberare una istanza della risorsa R_3. In questo modo il processo P_3 ha le risorse per eseguire, interrompendo uno dei due cicli. Quando P_3 rilascia una istanza della risorsa R_2, anche l'altro ciclo è interrotto e P_2 può eseguire.

Esercizio 2.

```
package compito_20100531;

import java.util.concurrent.Semaphore;

public class Principale {
	public static void main( String args[] ) {
		Object array[] = new Object[2];
		Semaphore sA = new Semaphore( 0 );
		Semaphore sB = new Semaphore( 0 );
		Semaphore sC = new Semaphore( 0 );
		Semaphore sD = new Semaphore( 0 );

		Produttore a = new Produttore( 0, array, sA, sD );
		a.start();
		Produttore b = new Produttore( 1, array, sB, sD );
		b.start();
		Consumatore c = new Consumatore( array, sC, sD );
		c.start();
		Distributore d = new Distributore( sA, sB, sC, sD );
		d.start();
	}
}

public class Produttore extends Thread {
	private int type;
	private Object[] array;
	private Semaphore s;
	private Semaphore sD;

	public Produttore( int t, Object[] a, Semaphore s, Semaphore sD ) {
		type = t;
		array = a;
		this.s = s;
		this.sD = sD;
	}

	public Object produceObject() { // implementazione data
	}

	public void run() {
		while( true ) {
			Object obj = produceObject();
			System.out.println( type + ": " + obj );
			array[ type ] = obj;
			sD.release();
```

```
                try {
                        s.acquire();
                }
                catch ( InterruptedException e ) {
                }
        }
    }
}

public class Consumatore extends Thread {
    private Object[] array;
    private Semaphore sC;
    private Semaphore sD;

    public Consumatore( Object[] a, Semaphore sC, Semaphore sD ) {
        array = a;
        this.sC = sC;
        this.sD = sD;
    }

    public void consumeObjects( Object obja, Object objb ) {
        // implementazione data
    }

    public void run() {
        while( true ) {
            try {
                sC.acquire();
                consumeObjects( array[0], array[1] );
            }
            catch ( InterruptedException e ) {
            }
            sD.release();
        }
    }
}

public class Distributore extends Thread {
    private Semaphore sA;
    private Semaphore sB;
    private Semaphore sC;
    private Semaphore sD;

    public Distributore( Semaphore sA, Semaphore sB, Semaphore sC, Semaphore
sD ) {
        this.sA = sA;
        this.sB = sB;
        this.sC = sC;
        this.sD = sD;
    }

    public void run() {
        while( true ) {
            try { // aspetta le notifiche di A e B
                sD.acquire( 2 );
            }
            catch ( InterruptedException e ) {
            }
            sC.release();
            try { // aspetta la notifica di C
                sD.acquire( 1 );
            }
```

```
                    catch ( InterruptedException e ) {
                    }
                    sA.release();
                    sB.release();
                }
            }
        }
```