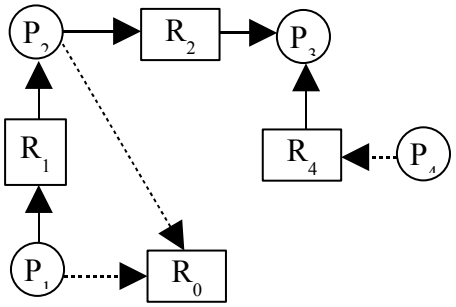
SISTEMI OPERATIVI IIN

seconda prova in itinere - 26.03.2007

Nome: _____ Cognome: _____

Esercizio 1. (punti 10)

Sia dato il grafo di allocazione delle risorse seguente al tempo t_0 (una istanza per ogni tipo di risorsa):



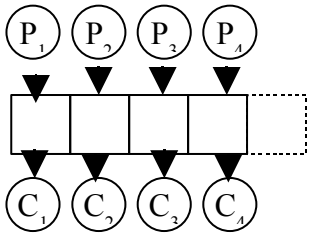
Supponendo che il grafo sia utilizzato per implementare l'*algoritmo del grafo di allocazione delle risorse* con l'obiettivo di impedire (*evitare)* lo stallo, si verifichi se:

a) a t_0 esiste uno stallo ed i processi eventualmente coinvolti;
b) lo stato dei processi è sicuro;
c) nel caso in cui lo stato non sia sicuro, la eventuale sequenza di richieste che può condurre a stallo ed i processi coinvolti (nel caso esistano più sequenze indicarle tutte).

Esercizio 2. (punti 20)

Un array di dimensione *N* di oggetti di tipo `ArrayElement`, è utilizzato da un insieme di *N* thread produttori e di *N* thread consumatori per comunicare. Il numero *N* è letto da linea di comando. La comunicazione avviene tra coppie di thread produttore (P_i) e consumatore (C_i) contraddistinti dallo stesso indice *i.* Pertanto, P_i e C_i comunicano attraverso l'elemento *i* dell'array nel modo seguente:

- il thread produttore P_i produce un oggetto modificando l'elemento *i* dell'array. Per modificare un elemento dell'array, P_i chiama il metodo `setValue(Object o)` della classe `ArrayElement`; a cui passa l'oggetto prodotto;
- il produttore P_i rimane sospeso fino a quando l'elemento *i* dell'array è pieno;
- il thread consumatore C_i rimane sospeso fino a quando l'elemento *i* dell'array è vuoto;
- La verifica che un elemento *i* dell'array sia pieno/vuoto è fatta attraverso la chiamata al metodo `boolean isFull();` della classe `ArrayElement` che ritorna `true/false` a seconda che l'elemento dell'array sia pieno/vuoto;
- il thread consumatore C_i preleva l'elemento *i* dell'array appena è disponibile chiamando il metodo `Object getValue()` della classe `ArrayElement`.



La soluzione dovrà utilizzare i meccanismi di sincronizzazione *Java* (metodi *synchronized, wait()* e *notity()/notifyAll()*). Si ipotizzi inoltre che la classe `ArrayElement` ed i suoi metodi siano dati.

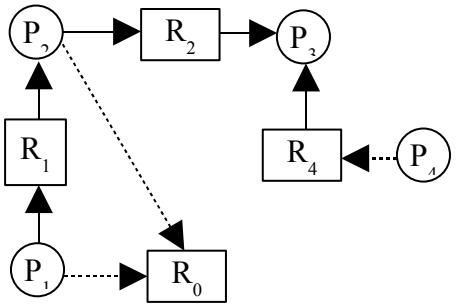
SISTEMI OPERATIVI IIN

seconda prova in itinere - 26.03.2007

Name: _____ Surname: _____

Exercise 1. (points 10)

The following resource allocation graph at t_0 (one instance for type of resource) is given:



In the hypothesis that the graph be used to implement the algorithm of the resource allocation graph in order to avoid the deadlock, it is require to verify if:

d) a deadlock exists at t_0 and, in the case, the processes that are involved in the deadlock;

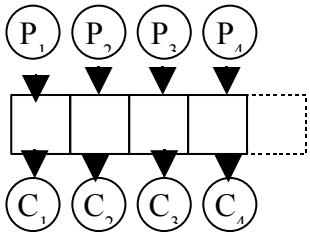
e) the state of resource allocation is safe;

f) in the case the state is not safe, determine the sequence of requests that can conduce to a deadlock and the processes involved in it (in the case more sequences exist, all of them must be indicated).

Exercise 2. (points 20)

An array of size *N* is constituted by objects of type `ArrayElement`. The array is used by a set of *N* producer threads and *N* consumer threads to communicate. The number *N* is read from the command line. The communication occurs between a pair of producer (P_i) and consumer (C_i) threads with the same index *i*. P_i and C_i communicate through the array element *i* in the following way:

- the produce thread P_i produces an object by modifying the element *i* of the array. To modify an element of the array, P_i calls the method `setValue(Object o)` of the class `ArrayElement`; to which the object is passed to;
- the producer P_i waits as long as the element *i* of the array is full;
- the consumer C_i waits as long as the element *i* of the array is empty;
- the test of an element *i* of the array to see if it is full or empty is done by calling the method `boolean isFull()`; of the class `ArrayElement` which returns `true/false` depending on the fact the element of the array is full or empty;
- the consumer C_i takes the element *i* of the array as soon as it is available by calling the method `Object getValue()` of the class `ArrayElement`.



The solution must use the Java synchronization mechanism (*synchronized* methods, *wait()* and *notity()/notifyAll()*). It can be assumed that the class `ArrayElement` and its methods are given.

Soluzione

Esercizio 1.

a) A t_0, non ci sono cicli nel grafo, quindi non esiste stallo.

b) Lo stato è sicuro. Esiste infatti una sequenza sicura di esecuzione.

c) (Non richiesto) Il sistema potrebbe passare ad uno stato insicuro se ad esempio la risorsa R_0 fosse assegnata al processo P_1 (arco di assegnamento R_0 -> P_1). In questo caso si avrebbe un ciclo nel grafo comprendente un *arco di reclamo*, che per l'algoritmo del grafo di allocazione delle risorse corrisponde ad uno stato non sicuro.

Esercizio 2.

```
import java.util.Date;

public class Principale {
	public static void main( String[] args ) {
		int N = Integer.parseInt( args[0] );
		Buffer buffer = new Buffer( N );
		// crea ed avvia i thread produttore e consumatore
		for (int i=0;i<N;i++) {
			Producer p = new Producer( i, buffer );
			p.start();
			Consumer c = new Consumer( i, buffer );
			c.start();
		}
	}
}

class Buffer {
	private static final int NAP_TIME = 3;
	private ArrayElement[] array;

	public Buffer( int size ) {
		array = new ArrayElement[size];
		for (int i=0;i<size;i++)
			array[i] = new ArrayElement();
	}

	public static void napping() {
		int time = (int)(Math.random()*NAP_TIME);
		try {
			Thread.sleep( time*1000 );
		}
		catch( InterruptedException ie ) {
			System.out.println( "Thread interrotto durante sleep" );
		}
	}

	public void insert( int id, Object obj ) {
		synchronized( array[id] ) {
			if ( array[id].isFull() ) {   // è sufficiente if
				try {
					array[id].wait();
				}
				catch( InterruptedException e ) {
					System.out.println( "Producer " + id + "
interrotto durante wait" );
				}
			}
```

```
                array[id].setValue( obj );
                array[id].notify();
            }
        }

        public Object remove( int id ) {
            synchronized( array[id] ) {
                if ( !array[id].isFull() ) {  // è sufficiente if
                    try {
                        array[id].wait();
                    }
                    catch( InterruptedException e ) {
                        System.out.println( "Consumer " + id + "
interrotto durante wait" );
                    }
                }
                array[id].notify();
                return array[id].getValue();
            }
        }
}


class Producer extends Thread {
        private int id;   // thread identificator
        private Buffer buffer;

        public Producer ( int id, Buffer buffer ) {
            this.id = id;
            this.buffer = buffer;
        }

        public void run () {
            while ( true ) {
                Date date = new Date();
                buffer.insert( id, date );
                Buffer.napping();
            }
        }
}


class Consumer extends Thread {
        private int id;   // thread identificator
        private Buffer buffer;

        public Consumer ( int id, Buffer buffer ) {
            this.id = id;
            this.buffer = buffer;
        }

        public void run () {
            while ( true ) {
                Date date = (Date)buffer.remove( id );
                Buffer.napping();
            }
        }
}
```