Sistemi Operativi

PROVA SCRITTA IN ITINERE – 06 giugno 2013

Esercizio 1. (punti 18)

Un ambiente ad accesso controllato ammette la presenza contemporanea di un massimo di N persone al suo interno. Le persone che desiderano accedere all'ambiente verificano che non sia stato raggiunto il massimo numero consentito N prima di entrare. Se ottengono l'accesso spendono un certo tempo nella stanza e poi escono. Ogni persona che entra incrementa anche un contatore globale degli accessi all'ambiente.

Considerando le persone come thread distinti di esecuzione (NT persone, NT costante >> N) e supponendo che il tempo speso nella visita della stanza sia simulato dalla chiamata al metodo `visitRoom()`, scrivere il codice Java che modella il sistema, utilizzando i **semafori Java** come strumento per gestire i problemi di sincronizzazione tra thread concorrenti (**Nota**: il numero N deve essere letto nella lista degli argomenti data dall'utente all'avvio del programma. Il metodo `visitRoom()` può essere attribuito alla classe che si ritiene più opportuna secondo la logica di funzionamento realizzata).

Esercizio 2. (punti 12)

In un sistema sono presenti tre tipi di risorse A, B, C con, rispettivamente, 6, 3, 1 istanze. In un certo istante T_0 sono in esecuzione 4 processi, P_1, P_2, P_3, P_4, caratterizzati dalle richieste massime e dallo stato di allocazione delle risorse seguenti:

Processi	Maximum			Allocation		
	A	B	C	A	B	C
P_1	4	2	1	2	1	0
P_2	3	3	0	1	0	0
P_3	1	2	1	0	1	0
P_4	5	1	1	1	0	0

Utilizzando l'***algoritmo del banchiere*** verificare:

- se lo stato di allocazione delle risorse all'istante T_0 è sicuro;
- se la richiesta in T_0 da parte del processo P_2 di un'ulteriore istanza della risorsa di tipo A, $req_2 = (1,0,0)$, può essere soddisfatta.

Soluzione

Esercizio 1.

```
package compito_20130606;
import java.util.concurrent.Semaphore;

public class Principale {
	public static final int NT = 100;

	public static void main( String args[] ) {
		int n = 0;
		if ( args.length > 0 ) {
			n = Integer.parseInt( args[0] );
			if ( n <= 0 ) {
				System.out.println( "The number must be > 0!" );
				System.exit( 1 );
			}
			Room rm = new Room( n );
			for (int i=0; i<NT; i++ ) {
				Visitor vs = new Visitor( i, rm );
				vs.start();
			}
		}
	}
}

public class Visitor extends Thread {
	private Room rm;
	private int id;

	public Visitor( int i, Room r ) {
		id = i;
		rm = r;
	}

	public void run() {
		rm.enterRoom( id );
		rm.visitRoom();
		rm.exitRoom( id );
	}
}

public class Room {
	private final int size;
	private int totalNAccess;
	private Semaphore mutex;
	private Semaphore inVisitors;

	public Room( int n ) {
		size = n;
		totalNAccess = 0;
		mutex = new Semaphore( 1 );
		inVisitors = new Semaphore( size );
	}
```

```
	public void enterRoom( int id ) {
		try {
			inVisitors.acquire();
			System.out.println( "thread " + id + " enter the room!" );
			System.out.flush();
			mutex.acquire();
			totalNAccess ++;
			System.out.println( "total number of access " + totalNAccess );
			System.out.flush();
			mutex.release();
		}
		catch ( InterruptedException ie ) {
		}
	}

	public void exitRoom( int id ) {
		System.out.println( "thread " + id + " exit the room!" );
		System.out.flush();
		inVisitors.release();
	}

	public void visitRoom() {
		try {
			Thread.sleep( (long)(Math.random()*1000) );
		}
		catch ( InterruptedException ie ) {
		}
	}
}
```

La soluzione proposta utilizza un semaforo contatore per mantenere traccia del numero di persone presenti nell'ambiente. In modo alternativo era possibile gestire questo numero utilizzando una variabile condivisa, verificata e modificata all'interno di una sezione critica in modo simile alla variabile `totalNAccess`.

Esercizio 2.

All'istante T_0 lo stato di allocazione delle risorse è il seguente (tenuto conto che le istanze per risorsa sono A = 6, B = 3, C = 1):

Processi	Maximum			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	4	2	1	2	1	0	2	1	1	2	1	1
P_2	3	3	0	1	0	0	2	3	0			
P_3	1	2	1	0	1	0	1	1	1			
P_4	5	1	1	1	0	0	4	1	1			

- Lo stato è sicuro in quanto la sequenza di esecuzione $<P_1, P_3, P_4, P_2>$ è sicura (in alternativa è sicura anche la sequenza $<P_1, P_3, P_2, P_4>$).
- La richiesta di P_2, $req_2 = (1,0,0)$, soddisfa la condizione di essere minore della necessità del processo (1,0,0) < (2,3,0), e delle risorse disponibili (1,0,0) < (2,1,1). Tuttavia lo stato di allocazione delle risorse che si otterrebbe ammettendo tale richiesta risulta non sicuro come risulta dalla tabella seguente:

Processi	Maximum			Allocation			Need			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P_1	4	2	1	2	1	0	2	1	1	1	1	1
P_2	3	3	0	2	0	0	1	3	0			
P_3	1	2	1	0	1	0	1	1	1			
P_4	5	1	1	1	0	0	4	1	1			

Infatti è possibile eseguire il processo **P_3** che libera le risorse allocate in modo che le disponibili diventano uguali a (1,2,1). Tuttavia tale disponibilità è incompatibile con le necessità degli altri processi. Di conseguenza la richiesta di P_2 non può essere soddisfatta ed il processo deve attendere.