Università degli Studi di Firenze, Scuola di Ingegneria
Corso di Laurea in Ingegneria Informatica – Prof. Stefano Berretti

Sistemi Operativi

PROVA SCRITTA IN ITINERE – 29 giugno 2015

Esercizio 1. (punti 20)

Un sistema comprende un *pool* di `N` thread *Worker* i quali sono avviati tutti insieme e inizialmente posti in attesa di una richiesta. Le richieste arrivano ad un *Server* attraverso la chiamata al metodo `void request(Object)` da parte di thread *Client* che eseguono un loop infinito in cui generano degli oggetti e li inviano al *Server*. Il *Server* gestisce le richieste nel metodo `request` aggiungendo l'oggetto fornito dal *Client* in coda ad una lista di tipo `java.util.ArrayList()` (con metodi `void add(Object)`, `Object remove(int)`). Ad ogni nuova richiesta il *Server* sveglia un *Worker* secondo una politica FIFO. Il *Worker* rimuove un oggetto dalla testa della lista e lo elabora con la chiamata al metodo `doWork(Object)`. Al termine del servizio il thread *Worker* torna in attesa.

Il numero `N` deve essere letto nella lista degli argomenti data dall'utente all'avvio del programma. Il metodo `doWork(Object)` è dato ed appartiene alla classe `Worker;` il numero di *Client* è fissato ed uguale a 100.

Scrivere il codice Java che modella il sistema, utilizzando i **semafori Java** come strumento per gestire i problemi di sincronizzazione tra thread concorrenti.

Esercizio 2. (punti 10)

In un sistema sono presenti quattro tipi di risorse A, B, C, D con, rispettivamente, 5, 4, 2, 3 istanze. In un certo istante T_0 sono in esecuzione 4 processi, P_1, P_2, P_3, P_4, caratterizzati dal seguente stato di allocazione delle risorse:

	Allocation			
Processi	**A**	**B**	**C**	**D**
P_1	1	2	0	1
P_2	2	0	0	1
P_3	1	1	1	0
P_4	0	1	0	1

e dall'assenza di richiesta di risorse da parte dei processi (la richiesta di risorse per ogni processo è nulla a T_0).

Utilizzando l'***algoritmo di rilevazione dello stallo*** verificare se:

- esiste uno stallo nel sistema e gli eventuali processi coinvolti all'istante T_0;
- la richiesta in T_1 da parte del processo P_1 di 2 ulteriori istanze della risorsa di tipo D, req_1 = (0,0,0,2), conduce o meno ad una stallo e gli eventuali processi coinvolti.

Soluzione

Esercizio 1.

```
package compito_20150629;

import java.util.ArrayList;
import java.util.concurrent.Semaphore;

public class Server {
	public static int nClient = 100;

	private int nwk;
	private int turn;
	private ArrayList array;
	private Semaphore mutex = new Semaphore( 1 );
	private Semaphore sempool[];

	public Server(int n, Semaphore s[]) {
		turn = 0;
		nwk = n;
		array = new ArrayList();
		mutex = new Semaphore( 1 );
		sempool = s;
	}

	public void request(Object obj) {
		try {
			mutex.acquire();
			array.add(obj);
			sempool[turn].release();
			turn = (turn+1)%nwk;
			mutex.release();
		}
		catch( InterruptedException ie ) {
		}
	}

	public Object getArrayObject() {
		Object obj = null;
		try {
			mutex.acquire();
			obj = array.remove(0);
			mutex.release();
		}
		catch( InterruptedException ie ) {
		}
		return obj;
	}

	public static void main( String[] args ) {
		if( args.length < 1 ) {
			System.out.println( "Usage: Server <n_worker>" );
			System.exit( 1 );
		}
```

```
		int N = Integer.parseInt( args[0] );
		if ( N < 0 ) {
			System.out.println( "The number of worker must be > 0" );
			System.exit( 1 );
		}

		Semaphore sempool[] = new Semaphore[N];
		for (int i=0; i<N; i++) {
			sempool[i] = new Semaphore(0);
		}

		// create the Server object
		Server sv = new Server(N,sempool);

		// create and start the Worker
		for (int i=0; i<N; i++) {
			Worker wk = new Worker(i,sv,sempool);
			wk.start();
		}

		// create and start the Client
		for (int i=0; i<nClient; i++) {
			Client cl = new Client(i,sv);
			cl.start();
		}
	}
}

public class Worker extends Thread {
	private int id;
	private Server sv;
	private Semaphore sempool[];

	public Worker(int i, Server s, Semaphore sempool[]) {
		id = i;
		sv = s;
		this.sempool = sempool;
	}

	private void doWork(Object obj) {
		try {
			Thread.sleep((long)(Math.random()*2000)+1000);
			System.out.println("Worker " + id + " dowork");
		}
		catch( InterruptedException ie ){
		}
	}

	public void run() {
		while( true ) {
			try {
				sempool[id].acquire();
				Object obj = sv.getArrayObject();
				doWork(obj);
			}
			catch( InterruptedException ie ) {
```

```
                }
            }
        }
}

public class Client extends Thread {
        private int id;
        private Server sv;

        public Client(int i, Server s) {
                id = i;
                sv = s;
        }

        public void run() {
                while( true ) {
                        Object obj = new Object();
                        System.out.println("Client " + id + " request");
                        System.out.flush();
                        sv.request(obj);
                        try {
                                Thread.sleep((long)(Math.random()*2000)+2000);
                        }
                        catch( InterruptedException ie ){
                        }
                }
        }
}
```

Esercizio 2.

All'istante T_0 lo stato di allocazione delle risorse è il seguente (tenuto conto che le istanze per risorsa sono A = 5, B = 4, C = 2, D = 3):

	Allocation				Available			
Processi	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P_1	1	2	0	1	1	0	1	0
P_2	2	0	0	1				
P_3	1	1	1	0				
P_4	0	1	0	1				

- Lo stato T_0 non è di stallo dato che in base alle informazioni date, nessuno dei processi è in uno stato di attesa (non essendoci richiesta non può esserci attesa) ed una delle condizioni necessarie allo stallo è perciò non verificata.
- La richiesta di P_1, req_2 = (0,0,0,2) può essere accettata. Infatti è possibile trovare la sequenza di esecuzione dei processi: **P_2** che libera le risorse allocate in modo che le disponibili diventano uguali a (3,0,1,1); poi **P_3** che può eseguire e libera le risorse in modo che le disponibili diventano (4,1,2,1); a questo punto può eseguire **P_4** con disponibilità aumentata a (4,2,2,2); A questo punto può eseguire anche **P_1** e tutti i processi possono perciò terminare senza stallo. La sequenza di esecuzione risulta: <**P_2,P_3,P_4,P_1**>.