Sistemi Operativi

PROVA IN ITINERE – 5 maggio 2014

Esercizio 1. (punti 12)

Un insieme di thread è caratterizzato dai tempi di CPU burst e I/O burst, dai tempi di arrivo in coda di ready e dalle priorità riportati di seguito (a valori più piccoli corrispondono priorità maggiori):

Thread	**T_{arrivo}**	**Priorità**	**Sequenza CPU/IO**
T_1	0	2	CPU(2) / IO(2) / CPU(2)
T_2	1	3	CPU(3) / IO(2) / CPU(5)
T_3	2	1	CPU(1) / IO(2) / CPU(1)
T_4	3	2	CPU(3) / IO(2) / CPU(4)

I thread sono organizzati in code secondo la loro priorità. Esistono tre code, rispettivamente per thread con priorità 1, 2 e 3. Tra le code esiste *priorità senza prelazione*. All'interno di ogni coda i thread sono schedulati con un algoritmo *round-robin* con quanti q=2, 3 e 4, rispettivamente, per le code con priorità 1, 2, 3 (il quanto è associato alla coda). Applicando la suddetta politica di scheduling si determinino:

- il diagramma di **Gantt**;
- il tempo di **attesa** medio;
- il tempo di **ritorno** medio;
- il tempo di **risposta** medio;
- il numero di **cambi di contesto**.

Esercizio 2. (punti 18)

Scrivere il programma Java che opera nel modo seguente:

- all'interno del metodo `main` di una classe `Principale` è istanziato un array di `N` oggetti di tipo `Worker` (i singoli elementi dell'array non sono però istanziati) e legge da tastiera un valore intero `value` in `[0,N);`
- successivamente, il metodo `main` istanzia ed avvia un thread `Server` a cui è passato l'array di `Worker` e `value`;
- A sua volta, la classe `Server` ha il compito di istanziare ed avviare i thread `Worker` a partire da quello di indice `value`. A parte il thread di indice `value`, prima di avviare un thread di indice `i+1` è necessario attendere il termine del thread di indice `i`;
- il metodo `main` deve infine stampare il risultato della elaborazione di ciascun thread `Worker` chiamando il metodo `printResult` della classe `Worker`.

Al fine di coordinare l'esecuzione dei thread è possibile usare il metodo `join` della classe `Thread`.

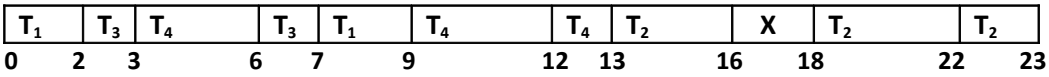
NOTA:
`N` è dato come costante della classe `Principale`.
La classe `Worker` è supposta data come classe di libreria. Estende la classe `java.lang.Thread` ed ha i seguenti metodi:

```
public Worker(int id);
public void printResult();
```

Soluzione

Esercizio 1. (punti 12)

Diagramma di *Gantt*:



Tempo di attesa medio:

$\mathbf{T_A} = (t_A(T_1) + t_A(T_2) + t_A(T_3) + t_A(T_4))/4 = (3 + 12 + 1 + 1)/4 = 17/4 = 4$**.25 ms**

Tempo di ritorno (turnaround) medio (media delle differenze tra tempo di fine esecuzione e tempo di arrivo):

$\mathbf{T_T} = (t_T(T_1) + t_T(T_2) + t_T(T_3) + t_T(T_4))/4 = (9 + 22 + 5 + 10)/4 = 46/4 =$ **11.5 ms**

Tempo di risposta medio (media delle differenze tra tempo di inizio esecuzione e tempo di arrivo):

$\mathbf{T_R} = (t_R(T_1) + t_R(T_2) + t_R(T_3) + t_R(T_4))/4 = (0 + 12 + 0 + 0)/4 = 12/4 =$ **3 ms**

Numero **cambi di contesto**: $\mathbf{N_{cs} = 9}$

Esercizio 2. (punti 18)

```
package compito_20140505;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Principale {
	public static final int max_thread = 10;

	public static void main( String args[] ) {
		int value = 0;
		Worker wk[] = new Worker[max_thread];

		InputStreamReader isr = new InputStreamReader( System.in );
		BufferedReader br = new BufferedReader( isr );
		try {
			System.out.println( "Please, insert the index of the starting thread
(>=0, <" + Principale.max_thread + ")" );
			String line = br.readLine();
			value = Integer.parseInt( line );
			System.out.println( "The starting index is: " + value );
		}
		catch (IOException e) {
			System.out.println( "error while reading" );
			System.exit(1);
		}

		Server rd = new Server(wk, value);
		rd.start();

		try {
			rd.join();
		}
		catch( InterruptedException ie) {}
```

```
			for( int i=0; i<max_thread; i++ ) {
				wk[i].printResult();
			}
		}
}

public class Server extends Thread {
		private Worker wk[];
		private int value;

		public Server( Worker wk[], int value ) {
			this.wk = wk;
			this.value = value;
		}

		public void run() {
			for (int i=0; i<Principale.max_thread; i++) {
				int j = (i+value)%Principale.max_thread;
				wk[j] = new Worker(j);
				wk[j].start();
				try {
					wk[j].join();
				}
				catch( InterruptedException ie ) {}
			}
		}
}

// non richiesta nello svolgimento del compito. Riportata per completezza
public class Worker extends Thread {
		private int id;

		public Worker( int id ) {
			this.id = id;
			System.out.println( "Created thread " + id );
			System.out.flush();
		}

		public void run() {
			int t = (int)(Math.random()*2000);
			try {
				System.out.println( "Thread " + id + " start working" );
				System.out.flush();
				Thread.sleep( t );
				System.out.println( "Thread " + id + " end working" );
				System.out.flush();
			}
			catch( InterruptedException ie ) {}
		}

		public void printResult() {
			System.out.println( "risultato thread " + id );
			System.out.flush();
		}
}
```