Sistemi Operativi

Prima prova in itinere 16-05-2016 - **Compito B**

Esercizio 1 (12 punti)

Un sistema operativo adotta la politica di scheduling dei thread a code multiple con *prelazione* tra le code. Sono presenti due code, una priorità con priorità 1 (alta priorità) con scheduling round-robin con quanto q=2ms e una coda a priorità 2 (bassa priorità) con scheduling FCFS. Si consideri che nel caso un thread nella coda FCFS sia prelazionato questo rimane in testa alla coda ready. Inoltre quando un thread nella coda FCFS viene prelazionato questo rimane in testa alla coda ready.

Il sistema deve schedulare i seguenti thread (presenti al tempo 0 ed arrivati in ordine T_1, T_2, T_3, T_4) con priorià e uso CPU/IO indicati:

T_1 pri=1 CPU(3ms)/IO(5ms)/CPU(2ms)
T_2 pri=1 CPU(1ms)/IO(5ms)/CPU(3ms)
T_3 pri=2 CPU(3ms)/IO(3ms)/CPU(3ms)
T_4 pri=2 CPU(2ms)/IO(2ms)/CPU(2ms)

Si determini:

1. il **diagramma di Gantt**,
2. il **tempo di attesa** medio,
3. il **tempo di ritorno** medio,
4. il **tempo di risposta** medio e
5. il numero di cambi di contesto

Esercizio 2 (18 punti)

Data la seguente classe:

```
abstract class MatrixWorkerThread extends Thread {
        protected int[][] result;
        protected MatrixWorkerThread(int n) {
                result=new int[n][n];
        }
        public int[][] getResult() { return result; }
}
```

Definire la classe *MatrixRndGeneratorThread* derivata da *MatrixWorkerThread* che dato intero N inizializza su thread parallelo la matrice *result* di dimensione NxN con numeri random tra -5 e 5 (compresi).

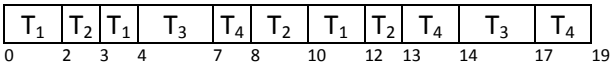
Definire la classe *MatrixMulThread* derivata da *MatrixWorkerThread* che dati due *MatrixWorkerThread* attende (su thread separato) che entrambi siano terminati e calcola nella matrice *result* il prodotto dei risultati ottenuti da questi due thread.

Nel programma principale nella classe *Compito2* chiedere in input il valore N>0 quindi creare quattro *MatrixRndGeneratorThread* (di dimensione N) quindi usare la classe *MatrixMulThread* per fare prodotto tra le quattro matrici cercando di parallelizzare al massimo le operazioni. Il programma principale deve attendere che il thread che calcola il risulato finale termini e stampi il risultato.

SOLUZIONE Compito B

Esercizio 1

Diagramma Gantt



Tempo attesa medio T_a = (2+4+8+13)/4 = 27/4 = 6,75ms
Tempo di ritorno medio T_r = (12+13+17+19)/4 = 61/4 = 15,25ms
Tempo di risposta medio T_{rs} = (0+2+4+7)/4 = 13/4 = 3,25ms
Numero cambi contesto N_{cs} = 10

Esercizio 2

```
import java.util.Scanner;

abstract class MatrixWorkerThread extends Thread {
     protected int[][] result;
     protected MatrixWorkerThread(int n) {
          result=new int[n][n];
     }
     public int[][] getResult() { return result; }
}

class MatrixRndGenerator extends MatrixWorkerThread {
   public MatrixRndGenerator(int n) {
      super(n);
   }
   public void run() {
      String s="";
      for(int i=0; i<result.length; i++) {
         for(int j=0; j<result[0].length; j++) {
            result[i][j]= -5+(int)(Math.random()*11);
            s+=result[i][j]+" ";
         }
         s+="\n";
      }
      System.out.println(getName()+"\n"+s);
   }
}

class MatrixMulThread extends MatrixWorkerThread {
   MatrixWorkerThread w1,w2;
   public MatrixMulThread(MatrixWorkerThread w1, MatrixWorkerThread w2) {
      super(w1.getResult().length);
      this.w1=w1;
      this.w2=w2;
   }

   public void run() {
      try {
         w1.join();
         w2.join();
         int[][] r1=w1.getResult();
```

```
            int[][] r2=w2.getResult();
            for(int i=0;i<r1.length; i++)
               for(int j=0; j<r2[0].length; j++) {
                  result[i][j]=0;
                  for(int k=0;k<r2.length; k++)
                     result[i][j]+=r1[i][k]*r2[k][j];
               }
         } catch(InterruptedException e) {
         }
      }
   }

public class Compito2 {
   public static void main(String[] args) {
      int N;
      Scanner s=new Scanner(System.in);
      do {
         System.out.print("N (N>0) ? ");
         N=s.nextInt();
      } while(N<=0);
      MatrixRndGenerator g[] = new MatrixRndGenerator[4];
      for(int i=0; i<g.length; i++) {
         g[i]=new MatrixRndGenerator(N);
         g[i].start();
      }
      MatrixMulThread m1=new MatrixMulThread(g[0], g[1]);
      m1.start();
      MatrixMulThread m2=new MatrixMulThread(g[2], g[3]);
      m2.start();
      MatrixMulThread m3=new MatrixMulThread(m1, m2);
      m3.start();
      try {
         m3.join();
         int[][] r=m3.getResult();
         for(int i=0; i<r.length; i++) {
            for(int j=0; j<r[0].length; j++)
               System.out.print(r[i][j]+" ");
            System.out.println();
         }
      } catch(InterruptedException e) {
      }
   }
}
```