UNICAL - **Esame di Sistemi Operativi** - 20 settembre 2002 - Durata della prova: 2 ore e 15 minuti

Prova 1

Si definisca una possibile soluzione in pseudo-codice al problema dei cinque filosofi che risolva il problema dello stallo (deadlock) dei filosofi nell'accesso alle due bacchette per mangiare. Si usi uno qualsiasi dei modelli di sincronizzazione studiati.

Prova 2

Si implementi un programma Java per la simulazione di un sistema costituito da ***K*** Processi indipendenti (identificati da un numero intero compreso tra *0* e *K-1*) che utilizzano le risorse di calcolo di un Elaboratore centrale.

L'Elaboratore dispone di ***N*** CPU indipendenti (identificate da un intero compreso tra *0* ed *N-1*), ciascuna delle quali può essere assegnata ad un solo Processo per volta. Ciascun Processo chiede periodicamente all'Elaboratore l'assegnazione di una o più CPU. L'Elaboratore assegna insiemi disgiunti di CPU a Processi differenti (ad esempio, in un certo istante, *CPU0* e *CPU1* possono essere assegnate al Processo *P0* e *CPU2*, *CPU3* e *CPU4* al Processo *P1*).

Ciascun Processo (da implementare come Thread Java) effettua ciclicamente le seguenti operazioni:

- attende per un tempo casuale compreso tra 100 e 1000 msec;
- richiede all'Elaboratore l'assegnazione di ***M*** CPU (***M*** è un numero casuale compreso tra *1* ed ***N***);
- quando gli vengono assegnate le CPU richieste, le utilizza per un tempo casuale compreso tra 50 e 500 msec, e quindi le rilascia. L'uso delle CPU deve essere simulato con una stampa su video.

L'Elaboratore dovrà adottare la seguente politica: se un processo ***Pj*** richiede l'assegnazione di ***M*** CPU, queste vengono assegnate a ***Pj*** solo quando risultano disponibili almeno ***M*** CPU e non esiste alcun processo ***Pi***, con $i < j$, già in attesa di un numero di CPU minore o uguale di **M**. Ad esempio, se *P3* richiede 5 CPU, e *P2* è in attesa di ottenere 3 CPU, allora la richiesta di *P3* sarà soddisfatta solo dopo quella di *P2*.

K ed ***N*** sono costanti definite nel programma. Si faccia uso delle primitive di sincronizzazione e mutua esclusione fornite dal linguaggio.

Proposta di soluzione per la Prova 2

```
class Processo extends Thread
{
  private int ID;
  private Elaboratore elaboratore;

  public Processo (int ID, Elaboratore elaboratore)
  {
    this.ID=ID;
    this.elaboratore = elaboratore;
  }

  private void attesa (int min, int max)
  {
    try
    {
      sleep ( (int)(Math.random()*(max-min)+min) );
    } catch (InterruptedException e){ System.out.println (e); }
  }
```

```
  public void run ()
  {
    while (true)
    {
      attesa (100,1000);
      int CPURichieste = (int)(Math.random()*elaboratore.getNumeroCPU())+1;
      elaboratore.richiediProcessori(ID, CPURichieste);
      attesa (50,500);
      elaboratore.rilasciaProcessori(ID);
    }
  }

}


class Elaboratore
{
  private int numeroCPU;
  private int numeroProcessi;
  private int CPULibere;

  private boolean processoInAttesa[];
    // processoInAttesa[i] vale true se il processo i-esimo è in attesa
    // di ottenere l'assegnazione di CPU

  private int richiestaCPU[];
    // richiestaCPU[i] contiene il numero di CPU di cui il processo i-esimo
    // è in attesa di assegnazione

  private int assegnazioneCPU[];
    // assegnazioneCPU[i] contiene l'ID del processo a cui è assegnata
    // la CPU i-esima

  public Elaboratore (int numeroCPU, int numeroProcessi)
  {
    this.numeroCPU = numeroCPU;
    this.numeroProcessi = numeroProcessi;
    CPULibere = numeroCPU;

    processoInAttesa = new boolean[numeroProcessi];
    for (int i = 0; i < numeroProcessi; i++)
      processoInAttesa[i] = false;

    richiestaCPU = new int[numeroProcessi];

    assegnazioneCPU = new int[numeroCPU];
      for (int i = 0; i < numeroCPU; i++)
        assegnazioneCPU[i] = -1;
  }

  private boolean mioTurno (int ID)
  {
    if (richiestaCPU[ID] > CPULibere)
      return false;

    for (int i = 0; i < ID; i++)
      if (processoInAttesa[i] && richiestaCPU[i] <= richiestaCPU[ID])
        return false;

    return true;
  }
```

```
public int getNumeroCPU ()
{
  return numeroCPU;
}

public void stampaStatoElaboratore ()
{
  System.out.println ("---------------");

  System.out.println ("CPU libere = "+CPULibere+" / "+numeroCPU);

  System.out.print ("Richieste [id-proc:num-cpu] = ");
  for (int i = 0; i < numeroProcessi; i++)
    if (richiestaCPU[i] != -1)
      System.out.print (i+":"+richiestaCPU[i]+"  ");

  System.out.print ("\nAssegnazioni [id-cpu:id-proc] = ");
  for (int i = 0; i < numeroCPU; i++)
    if (assegnazioneCPU[i] == -1)
      System.out.print (i+":#"+"  ");
    else
      System.out.print (i+":"+assegnazioneCPU[i]+"  ");

  System.out.println ("\n---------------");
}

public synchronized void richiediProcessori (int ID, int CPURichieste)
{
  richiestaCPU[ID] = CPURichieste;
  System.out.println ("Il processo "+ID+" richiede "+CPURichieste+" CPU");
  processoInAttesa[ID] = true;

  while (!mioTurno(ID))
  {
    try
    {
      wait();
    }
    catch (InterruptedException e)
    {
      System.out.println (e);
    }
  }

  System.out.println ("Il processo "+ID+" ottiene "+CPURichieste+" CPU");
  processoInAttesa[ID] = false;
  CPULibere -= CPURichieste;

  for (int i = 0; i < numeroCPU && richiestaCPU[ID] > 0; i++)
    if (assegnazioneCPU[i] == -1)
    {
      assegnazioneCPU[i] = ID;
      richiestaCPU[ID]--;
    }

  stampaStatoElaboratore ();
}
```

```
  public synchronized void rilasciaProcessori (int ID)
  {
    int CPURilasciate = 0;

    for (int i = 0; i < numeroCPU; i++)
      if (assegnazioneCPU[i] == ID)
      {
        assegnazioneCPU[i] = -1;
        CPURilasciate++;
      }

    System.out.println ("Il processo "+ID+" rilascia "+CPURilasciate+" CPU");
    CPULibere += CPURilasciate;
    notifyAll ();
    stampaStatoElaboratore ();
  }
}


public class Prova2
{
  public static void main (String args[])
  {
    final int N = 9; // numero di processori
    final int K = 3; // numero di processi

    Elaboratore elaboratore = new Elaboratore (N, K);

    Processo processi[] = new Processo[K];

    System.out.println ("CPU: "+N);
    System.out.println ("Processi: "+K);
    System.out.println ("---------------");

    for (int i = 0; i < K; i++)
      processi[i] = new Processo (i,elaboratore);

    for (int i = 0; i < K; i++)
      processi[i].start();
  }
}
```

Prova 3

Si realizzi una applicazione Java nella quale *N* oggetti Thread comunicano mediante socket secondo lo schema di seguito descritto.

Ciascun Thread è caratterizzato da un ***ID*** (compreso tra 0 ed N-1), e dalla porta sulla quale è in ascolto il relativo Socket (si usi il valore 1024+ID, ovvero: il Thread T0 ascolta sulla porta 1024, T1 sulla 1025 e così via).

T0 legge ciclicamente da tastiera una stringa S e la invia tramite socket a T1. T1 riceve S e la invia a T2, T2 trasmette S a T3, e così via. Infine, TN-1 riceverà S e la stamperà su video.

Gli oggetti *T0..TN-1* dovranno essere istanza di una classe Repeater (che estende Thread). Oltre alla classe Repeater si implementi un main che inizializza opportunamente i Thread e li esegue. ***N*** è una costante definita nel programma.

Proposta di soluzione per la Prova 3

```
import java.io.*;
import java.net.*;

class Repeater extends Thread
{
  private int ID;
  private int N;
  private int port;
  private ServerSocket server;

  public Repeater (int ID, int N)
  {
    this.ID=ID;
    this.N = N;
    this.port=1024+ID;
    if (ID != 0) // apro un ServerSocket su tutti i Thread (tranne sul primo)
    {
      try
      {
        server = new ServerSocket (port);
      }
      catch(IOException e) { System.out.println(e); }
    }
  }

  public void run ()
  {
    String S = "";

    if (ID == 0) // Primo Thread
      while (true)
      {
        // legge la stringa da input
        System.out.print ("Stringa da trasmettere: ");
        try
        {
          BufferedReader input = new BufferedReader(new
                                 InputStreamReader(System.in));
          S = input.readLine ();
        }
        catch (IOException e) { System.out.println(e); }

        // invia la stringa al Thread a destra
        try
        {
          // apre un socket verso il Thread alla sua destra
          Socket dx = new Socket ("127.0.0.1", port+1);
          PrintWriter out=new PrintWriter(dx.getOutputStream(),true);
          out.println(S);
          dx.close();
        }
        catch(IOException e) { System.out.println(e);}
      }
```

```
    else if (ID == N-1) // Ultimo Thread
      while (true)
      {
        // riceve la stringa dal Thread a sinistra
        try
        {
          // accetta l'apertura di un socket dal Thread alla sua sinistra
          Socket sx = server.accept();
          BufferedReader in=new BufferedReader (new
                            InputStreamReader(sx.getInputStream()));
          S=in.readLine();
          sx.close();
        }
        catch(IOException e) { System.out.println(e);}

        // scrive la stringa su output
        System.out.println ("Stringa ricevuta: "+S);
      }

    else // Thread intermedio
      while (true)
      {
        // riceve la stringa dal Thread a sinistra
        try
        {
          // accetta l'apertura di un socket dal Thread alla sua sinistra
          Socket sx = server.accept();
          BufferedReader in=new BufferedReader
                           (new InputStreamReader(sx.getInputStream()));
          S=in.readLine();
          sx.close();
        }
        catch(IOException e) { System.out.println(e);}

        // invia la stringa al Thread a destra
        try
        {
          // apre un socket verso il Thread alla sua destra
          Socket dx = new Socket ("127.0.0.1", port+1);
          PrintWriter out=new PrintWriter(dx.getOutputStream(),true);
          out.println(S);
          dx.close();
        }
        catch(IOException e) { System.out.println(e);}
      }
  }
}


public class Prova3
{
  public static void main (String args[])
  {
    final int N = 10;
    Repeater repeaters[] = new Repeater[N];
    for (int i=0; i<N; i++)
      repeaters[i] = new Repeater(i,N);
    for (int i=0; i<N; i++)
      repeaters[i].start();
  }
}
```