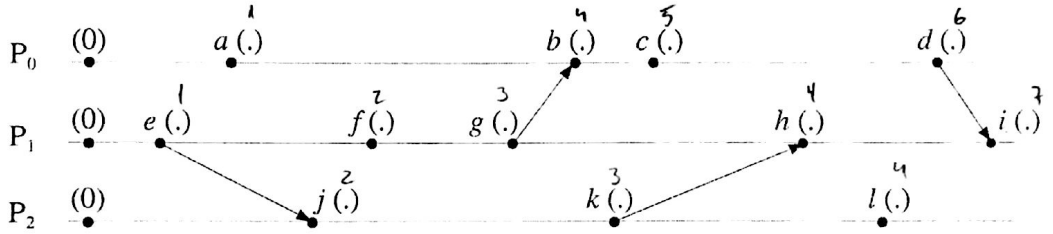
Sistemas Distribuídos
Teste 1 *17 de Junho de 2013*

Parte A (12 valores)

1. A *tolerância a falhas* é uma propriedade fundamental num sistema distribuído. Caracterize o conceito e descreva, usando exemplos esclarecedores, os níveis de tolerância que são habitualmente considerados.
2. Considere a *invocação de métodos sobre objectos remotos* como o paradigma de comunicação que foi escolhido para se estabelecer um modelo cliente-servidor entre processos residentes em diferentes nós de processamento de um sistema distribuído e admita que foi usado na sua implementação o Java RMI. Apresente um diagrama esquemático que descreva a interacção funcional entre os diferentes componentes de um tal sistema, quer no lado do servidor, quer no lado do cliente, identificando cada um dos componentes introduzido.
3. A figura abaixo descreve a evolução temporal de três processos cujos relógios locais são relógios lógicos escalares sincronizados segundo o algoritmo de acerto de Lamport..



   i. Atribua aos diferentes acontecimentos, especificados por letras do alfabeto minúsculo (*a* ... *l*), que ocorrem nos três processos, o valor do *time stamp* que lhes está associado.
   ii. Indique para os seguintes pares de acontecimentos, *a-f*, *f-h*, *b-i*, *i-i*, *j-i* e *a-l*, se se trata de acontecimentos concorrentes (||) ou de acontecimentos ligados por um nexo de causalidade (->).
   iii. Explique, apresentando exemplos do diagrama, porque é que o *time stamp* associado a cada acontecimento não pode ser em geral usado para a ordenação causal de acontecimentos. Em que situação especial tal é, porém, possível?
4. Considere um modelo de comunicação entre pares, formado por 6 processos, que formam um anel lógico e acedem a uma região partilhada quando na posse de um bastião (*token*).
   i. Descreva as operações de acesso à região crítica para manipulação da região partilhada: *entradaRC* e *saídaRC*.
   ii. Quando ocorre perda de bastião, deixa de ser possível o acesso à região partilhada. Descreva detalhadamente um algoritmo que permite a recuperação do bastião. Assuma que não há perda de mensagens e que a razão que conduziu à perda do bastião foi o sistema computacional onde um dado processo executa ter sido desligado.

Parte B (8 valores)

```
public class Semaphore
{
   private int val = 0;
   private int numbBlockThreads = 0;
   public synchronized void down ()
   {
      if (val == 0)
         { numbBlockThreads += 1;
           try
           { wait ();
           }
           catch (InterruptedException e) { }
         }
         else val -= 1;
   }
   public synchronized void up ()
   {
      if (numbBlockThreads != 0)
         { numbBlockThreads -= 1;
           notify ();
         }
         else val += 1;
   }
}

public class GenRegion
{
   private int n = 0;
   private int [] valSet = {20, 11, 30, 4, 16, 26, 7, 3, 9};
   public synchronized int produceVal ()
   {
      int val = 0;
      if (n < valSet.length)
         { val = valSet[n];
           n += 1;
         }
      return val;
   }
}

public class StoreRegion
{
   private int mem = 0;
   private int stat = 0;
   private Semaphore [] statL;
   public StoreRegion ()
   {
      statL = new Semaphore[3];
      for (int i = 0; i < 3; i++)
        statL[i] = new Semaphore ();
      statL[0].up ();
      statL[1].up ();
   }
   public void putVal (int val)
   {
      statL[1].down ();
      statL[0].down ();
      mem = val;
      statL[0].up ();
      statL[2].up ();
   }
   public int getVal ()
   {
      int val;
      if (stat == 0) statL[2].down ();
      statL[0].down ();
      val = mem;
      statL[0].up ();
      if (stat == 0) statL[1].up ();
      return val;
   }
```

```
    public void noVal ()
    {
       statL[1].down ();
       statL[0].down ();
       mem = 0;
       stat = 1;
       statL[0].up ();
       statL[2].up ();
    }
}

public class ThreadType1 extends Thread
{
   private int id;
   private GenRegion gen = null;
   private StoreRegion storeU = null,
                       storeD = null;
   public ThreadType1 (int id, GenRegion gen, StoreRegion storeU, StoreRegion storeD)
   {
      this.id = id;
      this.gen = gen;
      this.storeU = storeU;
      this.storeD = storeD;
   }
   public void run ()
   {
      int val;
      do
      { try
        { sleep ((int) (1 + 20 * Math.random ()));
        }
        catch (InterruptedException e) {};
        val = gen.produceVal ();
        if (val != 0)
           { if ((val % 2) == 0)
                storeU.putVal (100 * id + val);
                else storeD.putVal (100 * id + val);
           }
           else { storeU.noVal ();
                  storeD.noVal ();
                }
        try
        { sleep ((int) (1 + 20 * Math.random ()));
        }
        catch (InterruptedException e) {};
      } while (val != 0);
   }
}

public class ThreadType2 extends Thread
{
   private int id;
   private StoreRegion storeL = null,
                       storeR = null;
   public ThreadType2 (int id, StoreRegion storeL, StoreRegion storeR)
   {
      this.id = id;
      this.storeL = storeL;
      this.storeR = storeR;
   }
   public void run ()
   {
      int valL, valR;
      do
      { try
        { sleep ((int) (1 + 20 * Math.random ()));
        }
        catch (InterruptedException e) {};
        valL = storeL.getVal ();
        if (valL != 0)
           System.out.println ("O valor produzido pelo thread de tipo 1, " +
                       valL/100 + ", e impresso pelo thread de tipo 2, " +
                       id + ", foi " + valL%100 + "!");
```

```
            try
            { sleep ((int) (1 + 20 * Math.random ()));
            }
            catch (InterruptedException e) {};
            valR = storeR.getVal ();
            if (valR != 0)
               System.out.println ("O valor produzido pelo thread de tipo 1, " +
                           valR/100 + ", e impresso pelo thread de tipo 2, " +
                           id + ", foi " + valR%100 + "!");
         } while ((valL != 0) || (valR != 0));
      }
   }

   public class SimulSituation
   {
      public static void main (String [] args)
      {
         StoreRegion [] store = new StoreRegion [4];
         for (int i = 0; i < 4; i++)
           store[i] = new StoreRegion ();
         GenRegion gen = new GenRegion ();
         ThreadType1 [] thr1 = new ThreadType1[2];
         ThreadType2 [] thr2 = new ThreadType2[2];
         for (int i = 0; i < 2; i++)
           if (i == 0)
              { thr1[i] = new ThreadType1 (i+1, gen, store[0], store[3]);
                thr2[i] = new ThreadType2 (i+3, store[0], store[1]);
              }
              else { thr1[i] = new ThreadType1 (i+1, gen, store[1], store[2]);
                     thr2[i] = new ThreadType2 (i+3, store[3], store[2]);
                   }
         for (int i = 0; i < 2; i++)
           thr2[i].start();
         for (int i = 0; i < 2; i++)
           thr1[i].start ();
      }
   }
```

1. Representando as entidades activas por círculos e as entidades passivas por rectângulos, faça um diagrama ilustrativo da interacção em presença e indique por palavras simples qual é o papel desempenhado pelos *threads* de cada tipo (não mais do que uma frase).
2. O tipo de dados `StoreRegion` opera de acordo com um diagrama de estados bem-definido. Identifique cada um dos estados e descreva o papel atribuído a cada um dos elementos do *array* de semáforos `statL` e à variável `stat` na interacção.
3. Apresente o texto que é impresso por uma execução do programa. Tenha em atenção que, face a aleatoridade introduzida, os valores impressos não são únicos.
4. Modifique o programa de modo a que opere sempre que possível sobre produtos de pares de valores em vez de valores individuais, como actualmente sucede.