Sistemas Distribuídos
Teste 1 *20 de Junho de 2008*

Parte A (12 valores)

1. Explique qual é a importância de se garantir a tolerância a falhas no âmbito dos sistemas distribuídos e indique quais são as estratégias mais comuns usadas no tratamento das falhas. Dê um exemplo de cada uma delas.
2. Descreva as operações principais que têm que ser efectuadas para comunicação, na perspectiva do programador, na implementação de um modelo cliente-servidor, com replicação do servidor, usando o protocolo TCP, quer do lado do servidor, quer do lado do cliente.
3. Indique quais são as diferenças fundamentais que existem em Java, em termos de transparência, no acesso a objectos locais e a objectos remotos.
4. No âmbito da comunicação entre pares, descreva detalhadamente um algoritmo que permita de uma forma distribuída e dinâmica fazer-se a sincronização dos relógios locais. Que tipo de ordenação de acontecimentos é que resulta da sua aplicação. Mostre como é possível garantir a consistência de dados entre diferentes cópias localizadas em locais geograficamente separados quando ele é usado.

Parte B (8 valores)

```
public class GenRegion
{
   private int n = 0;
   private int [] valSet = {5, 12, 3, 6, 4, 2, 1, 18, 15, 16, 9, 7};
   public synchronized int produceVal ()
   {
      if (n == valSet.length)
         return 0;
         else { int val = valSet[n];
                n += 1;
                return val;
              }
   }
}

public class StoreRegion
{
   private int mem = 0;
   private int stat = 0;
   public synchronized void putVal (int val)
   {
      while (stat != 0)
      { try
        { wait ();
        }
        catch (InterruptedException e) {}
      }
      stat = 1;
      mem = val;
      notifyAll ();
      while (stat == 1)
      { try
        { wait ();
        }
        catch (InterruptedException e) {}
      }
   }
   public synchronized int getVal ()
   {
      while ((stat != 1) && (stat != 2))
      { try
        { wait ();
        }
        catch (InterruptedException e) {}
      }
      if (stat == 1) stat = 0;
      int val = mem;
      mem = 0;
      notifyAll ();
      return val;
   }
   public synchronized void noWait ()
   {
      stat = 2;
      notifyAll ();
   }
}
}
```

```
public class Resource
{
   private int n;
   private StoreRegion [] store = null;
   public Resource (int n, StoreRegion [] store)
   {
      this.n = n;
      this.store = store;
   }
   public synchronized boolean printVal (int id, int val)
   {
      if (n >= 1)
         { System.out.println ("O valor processado por " + (val/100) + " e por " +
                               id + " foi " + (val%100) + ".");
           n -= 1;
           if (n == 0)
              for (int i = 0; i < 2; i++)
                store[i].noWait ();
         }
      return (n == 0);
   }
}

public class ThreadType1 extends Thread
{
   private int id;
   private GenRegion gen = null;
   private StoreRegion [] store = null;
   public ThreadType1 (int id, GenRegion gen, StoreRegion [] store)
   {
      this.id = id;
      this.gen = gen;
      this.store = store;
   }
   public void run ()
   {
      int val;
      do
      { try
        { sleep ((int) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        val = gen.produceVal ();
        try
        { sleep ((int) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        if (val != 0)
           switch (val % 3)
           { case 0: store[0].putVal (100*id+2*val);
                     break;
             case 1:
             case 2: store[1].putVal (100*id+val);
           }
      } while (val != 0);
   }
}

public class ThreadType2 extends Thread
{
   private int id;
   private StoreRegion [] store = null;
   private Resource writer = null;
   public ThreadType2 (int id, StoreRegion [] store, Resource writer)
   {
      this.id = id;
      this.store = store;
      this.writer = writer;
   }
```

```
   public void run ()
   {
      int val;
      boolean end = false;
      while (!end)
      { val = store[(id-1)/2].getVal ();
        try
        { sleep ((int) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        end = writer.printVal (id, val);
      }
   }
}

public class SimulSituation
{
   public static void main (String [] args)
   {
      StoreRegion [] store = new StoreRegion [2];
      for (int i = 0; i < 2; i++)
        store[i] = new StoreRegion ();
      GenRegion gen = new GenRegion ();
      Resource writer = new Resource (12, store);
      ThreadType1 [] thr1 = new ThreadType1[4];
      for (int i = 0; i < 4; i++)
        thr1[i] = new ThreadType1 (i+1, gen, store);
      ThreadType2 [] thr2 = new ThreadType2[4];
      for (int i = 0; i < 4; i++)
        thr2[i] = new ThreadType2 (i+1, store, writer);
      for (int i = 0; i < 4; i++)
        thr2[i].start ();
      for (int i = 0; i < 4; 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. Assuma que, quando o programa é executado, se obtém o resultado seguinte

   ```
   O valor processado por 1 e por 2 foi 24.
   O valor processado por 3 e por 1 foi 3.
   O valor processado por 1 e por 2 foi 12.
   O valor processado por 2 e por 4 foi 5.
   O valor processado por 4 e por 3 foi 4.
   O valor processado por 4 e por 4 foi 11.
   O valor processado por 3 e por 2 foi 36.
   O valor processado por 1 e por 3 foi 2.
   O valor processado por 4 e por 3 foi 7.
   O valor processado por 2 e por 1 foi 30.
   O valor processado por 4 e por 4 foi 16.
   O valor processado por 1 e por 1 foi 18.
   ```

   Tenha em atenção que, face à aleatoridade introduzida, este não é o único resultado possível. De facto, nem sequer está correcto. Existem três erros, respectivamente, nas linhas 2, 6 e 9. Identifique-os. Justifique cuidadosamente a sua resposta.
3. Explique como é sempre garantida a terminação do programa.
4. Altere o programa de modo a ser possível processar pares (somas) de valores. Comece por explicar que tipo de alterações tem que fazer face à organização do programa que lhe é apresentado.