Universidade da Beira Interior

Sistemas Distribuídos

Licenciatura em Engenharia Informática

Exame – 2013/06/21

Duração: 1 hora e 30 minutos
(10 valores)

1 – Diga o que é o protocolo pedido-resposta (request-replay), e descreva as operações que o implementam.

2 – Na comunicação entre dois processos considere as seguintes semânticas perante falhas: "pelo menos uma vez" (at-least-once) e "no máximo uma vez" (at-most-once). Explique a diferença na implementação e funcionamento de cada uma delas.

3 – Compare os tipos de sincronização das três formas de comunicação por mensagens: sockets UDP, sockets TCP e invocação remota de procedimentos em Java RMI:

4 – Diga o que é a serialização de dados, e dê 3 exemplos em que é utilizada.

5 – Suponha que a associação académica da UBI decidiu abrir uma campanha de angariação de fundos para a construção de uma pista de desportos radicais. Para dar suporte à gestão da campanha vai ser construída uma aplicação cliente / servidor através da qual quem quiser aderir pode inscrever-se e indicar qual o seu donativo. O donativo será feito através de um depósito numa conta bancária aberta para a campanha.

A aplicação, em Java RMI, deve instanciar um objeto remoto, no processo servidor, com dois métodos:
- Um método remoto chamado ***donativo,*** que deve ter como parâmetros o número do bilhete de identidade (BI) de quem faz o donativo, o valor do donativo e a referência remota de um objeto cliente para o qual possa ser feito um callback quando a organização detetar que o donativo foi transferido para a conta bancária. O método deve guardar os dados do donativo numa estrutura de dados em memória e devolver o valor total doado até ao momento.
- Um método remoto chamado ***transferenciaConfirmada,*** que deve ter como parâmetro o BI de um dador, significando que o donativo desse dador foi transferido para a conta bancária. Este método deverá invocar um callback no objeto remoto correspondente ao BI do dador, onde enviará uma mensagem a confirmar a receção do donativo e um agradecimento.

a) Construa as classes necessárias para o processo servidor: servidor, objeto remoto com os dois métodos descritos, e interface remota.

b) Construa o processo cliente a ser executado pelos dadores, isto é, o processo que irá invocar o método ***donativo.*** Construa ainda o objeto remoto do cliente para implementação do callback descrito e a respetiva interface remota.

Nota 1: Não é pedido o processo cliente que será executado pelo administrador e onde será invocado o método ***transferenciaConfirmada.*** Também não se pede que no processo servidor seja guardada informação sobre o facto de a transferência já estar confirmada.
Nota 2: Pode omitir o tratamento de exceções.

6 – Suponha a classes Exemplo, T1 e T2 listadas abaixo.

```
public class Exemplo {
 private int a;
 private int b;
 public synchronized void m1(){ a=b+1;}
 public synchronized void m2(){ b=a+1;}
public String toString(){return ("a= " + a + " b= " + b);
}
public class T1 extends Thread {
      Exemplo e;
      public T1(Exemplo e){
            super(); this.e=e; start();
      }
      public void run(){
        int pausa=(int)(Math.random()*1000);
        try {
            sleep(pausa);
       } catch (InterruptedException e1) {
            System.out.println(e1.getMessage());
       }
        e.m1();
      }
}
public class T2 extends Thread {
          Exemplo e;
          public T2(Exemplo e){
               super(); this.e=e; start();
          }
          public void run(){
            int pausa=(int)(Math.random()*1000);
            try {
                sleep(pausa);
            } catch (InterruptedException e1) {
                  System.out.println(e1.getMessage());
              }
              e.m2();
            }
}
```

a) Indique, **justificando**, quais os outputs possíveis do programa da classe Teste seguinte:

```
public class Teste {
      public static void main(String[] args) {
            Exemplo e = new Exemplo();
            T1 t1 = new T1(e);
            T2 t2 = new T2(e);
            try {
                  t1.join();
                  t2.join();
            }
            catch (InterruptedException x){
                  System.out.println ("Erro");
            }
            System.out.println ((e.toString()));
      }
}
```

b) Se as instruções "`t1.join(); t2.join();`" fossem retiradas o que acontecia de diferente? Poderia haver resultados diferentes do(s) anterior(es)? Se sim, quais?

Notas auxiliares:

Gerar um número aleatório:

Math.random(). Este método devolve um double pertencente ao intervalo [0, 1[

socket do cliente:

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

Socket meuCliente = null;
try {
        meuCliente = new Socket ("host", portNumber);
        }
catch (IOException e){
        System.out.println( e.getMessage());
}
```

socket do servidor:

```
ServerSocket  meuServidor = null;
    try {
        meuServidor = new ServerSocket (portNumber);
    } catch (IOException e){ System.out.println( e.getMessage());}

  Socket sServidor = null
 try {
        sServidor = meuServidor.accept();
} catch (IOException e){ System.out.println( e.getMessage());
```

Exemplo de utilização de ObjectStreams:

Com sockets:

```
ObjectOutputStream os = new ObjectOutputStream (meuCliente.getOutputStream());
os.writeObject("XPTO");

ObjectInputStream is = new ObjectInputStream (meuCliente.getInputStream());
String txt =(String) is.readObject();
```

Com ficheiros:

```
ObjectOutputStream os = new ObjectOutputStream ( new FileOutputStream ("XP.dat"));
```

RMI:

Interface remote:

java.rmi.Remote

Excepção remota:

java.rmi.RemoteException

Objecto remoto :

java.rmi.server.UnicastRemoteObject;

Sintaxe do nome que o objecto remoto tem no RMIregistry:

[rmi:] [//] [nomeMaquina] [:port] [/nomeObjecto]

Instalar um gestor de segurança:

System.setSecurityManager (new RMISecurityManager());

Métodos da classe java.rmi.Naming:

void rebind (String nomeObjecto, Remote objecto);

Remote lookup (String nomeObjecto)

Ficheiros:

```
os = new ObjectOutputStream ( new FileOutputStream ("XP.dat"));
os.writeObject( Object   )
…
is = new ObjectInputStream ( new FileInputStream ("XP.dat"));
Object o = is.readObject ();
```

API da classe java.util.Vector:

Vector()// construtor vector vazio, dimensão inicial zero.

Vector(int capacidInicial) // construtor vector vazio, com dimensão inicial.

void addElement(Object elemento) // adiciona o elemento especificado ao final do vector.

void insertElementAt(Object obj, int indice)// insere o elemento especificado na posição indice.

void removeElementAt(int indice) // remove o elemento na posição indice.

void setElementAt(Object obj, int indice) // substitui o elemento da posição indice pelo objecto dado.

Object elementAt(int indice)// devolve o componente presente no indice.

void clear() // remove todos os elementos do vector.

Object clone() // devolve uma cópia do vector.

boolean contains(Object elemento) // verifica se o objecto especificado é um componente deste vector

Object firstElement() // devolve o primeiro componente (indice 0) do vector.

Object lastElement() // devolve o último componente do vector.

int indexOf(Object elemento) // procura o índice da 1ª ocorrência de elemento (utiliza o método equals)

int indexOf(Object elemento, int indice) // inicia a procura anterior na posição indice.

boolean isEmpty() // verifica se o vector não tem componentes

int size() // devolve a dimensão actual.

...