Universidade da Beira Interior

Sistemas Distribuídos
Licenciatura em Engenharia Informática

Frequência
2014/06/02

Duração: 2 horas

(14 valores)

I

Nota: Nos exercícios 1 e 2, pode omitir o tratamento de exceções.

1 – Suponha uma entidade que organiza passeios turísticos. A organização propõe um passeio e espera que um certo número mínimo de pessoas se inscreva (suponha que 30 pessoas). Quando é atingido esse número de pessoas, é comunicado a todas elas que o passeio se pode realizar.
Construa para esta entidade uma aplicação cliente-servidor que deverá possuir no servidor um objeto remoto (em java RMI) com as seguintes operações:
1- Inscrição no passeio, 2 – Consultar inscritos. Em cada instante só existe um passeio possível. Quando é atingido o número mínimo de inscritos, o servidor enviará a todos os inscritos um callback a avisar que o passeio se pode realizar.
Deve construir todas as classes e interfaces necessárias para implementar a aplicação.

2 – Construa uma solução para o problema anterior usando uma aplicação cliente-servidor em que os processos comunicam por Sockets. O servidor deverá criar uma Thread para cada cliente que acede ao servidor. Em cada ligação do cliente ao servidor, o cliente deverá poder escolher várias opções entre elas a opção de terminar a ligação. Nesta implementação, o cliente, quando se inscreve, recebe do servidor uma mensagem informando qual o número de inscritos até ao momento. No caso de o número de inscritos atingir o mínimo exigido o cliente deve receber a informação de que o passeio se vai realizar.

II

1 – Defina sistema distribuído. Dê exemplos.

2 - O que é a serialização de uma estrutura de dados? Dê exemplos em que seja utilizada.

3 – Se num sistema de objetos remotos quiser implementar a semântica perante falhas: at-most-once (no máximo-uma-vez) explique o que precisa de implementar?

4 – No contexto da linguagem java, explique a diferença entre um lock de classe e um lock de objeto.

5 - Diga o que entende por arquitetura de um sistema distribuído.

6 – Represente esquematicamente a arquitetura das aplicações descritas nos exercícios 1 e 2 do grupo I.

7 – Analise os modelos de comunicação por mensagens relativamente ao tipo de sincronização e relativamente à forma como são identificados os vários intervenientes na comunicação para cada uma dos sistemas:

i. Comunicação por Sockets TCP
ii. Comunicação por Sockts UDP
iii. Comunicação em Java RMI
iv. Comunicação na linguagem Linda

Notas auxiliares:

socket do cliente:

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

Socket meuCliente = null;
try { meuCliente = new Socket ("host", portNumber); }
catch (IOException e){ … }
```

Obter as Streams do socket e associar objectStrems:

```
ObjectOutputStream os = new ObjectOutputStream (
                                    meuCliente.getOutputStream());
ObjectInputStream is = new ObjectInputStream (
                                    meuCliente.getInputStream());
```

```
String s = "exemplo";
os.writeStream(s);
s  = (String) is.readObject();
```

socket do servidor:

```
 ServerSocket  meuServidor = null;
 try { meuServidor = new ServerSocket (portNumber); }
catch (IOException e){ …}

Socket sServidor = null
try { sServidor = meuServidor.accept(); }
catch (IOException e){ … }
```

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());

Iniciar o registry: java.rmi.registry.LocateRegistry.createRegistry(1099);

Métodos da classe java.rmi.Naming:

void rebind (String nomeObjecto, Remote objecto);

Remote lookup (String nomeObjecto)

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 na posição índice.

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

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

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

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

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.