Universidade da Beira Interior

Sistemas Distribuídos

Licenciatura em Engenharia Informática

Frequência

2015/05/20

Duração: 2 horas

(13 valores)

I

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

1 – Pretende-se uma aplicação cliente-servidor em que o processo cliente envia ao servidor um pedido que consiste em enviar uma String e receber como resultado essa String encriptada. A encriptação é feita num servidor multi-threaded onde para cada cliente é criada uma thread que recebe o pedido do cliente e o processa através de uma função cuja assinatura é "String enigma (String s)".

Pretende-se que o servidor nunca processe simultaneamente mais do que 5 pedidos.

a) Construa o código do processo servidor;

b) Construa o código da thread criada pelo servidor sempre que um cliente se liga. A Thread recebe a String do cliente, chama a função enigma e devolve a String encriptada ao cliente.

2 – Suponha uma aplicação cliente-servidor para uma biblioteca gerir o seu serviço de empréstimos. Cada livro é identificado pelo seu título (uma String) e os livros disponíveis para empréstimo estão armazenados num objecto do tipo Vector na memória do servidor. A comunicação cliente/servidor é feita através de Java RMI e as funcionalidades do processo servidor, implementadas por métodos remotos, são as seguintes:

i) Consultar livros disponíveis;

ii) Requisitar um livro. Se o livro existir no Vector de livros disponíveis, é eliminado; Se não existir, quando o livro for devolvido o processo cliente que tentou requisitar o livro receberá uma notificação a informar que o livro está disponível. Ignore a situação de o cliente já não estar ligado;

iii) Devolver um livro. Quando um cliente executa a operação "Devolver", o livro é inserido no Vector. Se previamente ocorreu alguma operação de "Requisitar" esse livro, o cliente (ou os clientes) interessados no livro receberão uma notificação.

- Construa as classes necessárias para implementar a aplicação.

II

1 – Defina o que é a arquitectura de um sistema distribuído e represente graficamente a arquitectura das aplicações dos exercícios 1 e 2 do grupo I.

2 – O que é e para que serve a serialização de uma estrutura de dados. Considerando os exercícios 1 e 2 do grupo I, em que situações é necessária a serialização de dados.

3 – Qual a diferença entre as semânticas perante falhas “at-least-once” e “at-most-once”.

4 – No contexto da tecnologia Java Enterprise Edition, explique a diferença entre um “stateful session bean” e um “stateless session bean”.

5 – Explique como funcionam os relógios lógicos de Lamport para atribuir um timestamp a uma sequência de eventos num sistema distribuído.

6 – Explique as principais diferenças um sistema de base de dados SQL e um sistema de bases de dados NoSQL?

7 – Quais as principais características de um sistema de Cloud Computing?

8 – Qual a(s) diferença(s) entre invocar um método remoto e invocar um web service?

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.writeObject(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 remota: java.rmi.Remote

Exceção remota: java.rmi.RemoteException

Objeto 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.