UNIVERSIDADE LUSÓFONA

Redes de Computadores

Sockets

Universidade Lusófona de Humanidades e Tecnologias

Licenciatura em Engenharia Informática / Engenharia Informática, Redes e Telecomunicações
Docente: Miguel Tavares, Pedro Sá da Costa

1. Objetivo do trabalho

Pretende-se que os alunos tomem conhecimento com a interface de programação (API) socket da linguagem Java (package java.net) para desenvolver aplicações de rede. Os exercícios apresentados podem ser realizados em qualquer sistema operativo, utilizando ambientes de desenvolvimento da linguagem Java. O Javadoc relativo ao package java.net deve ser o documento base de referência. É aconselhado iniciar o trabalho em Linux.

2. Programação de Sockets

O package java.net dispõe de diversas classes para manipular e processar informação em rede. Estas classes possibilitam comunicações baseadas em sockets:

- Permitem manipular I/O de rede como I/O de ficheiros;
- Os sockets são tratados como streams alto-nível o que possibilita a leitura/escrita de/para sockets como se fosse para ficheiros;

Uma socket é um mecanismo que permite que programas troquem pacotes de bytes entre si. Quando uma socket envia um pacote, este é acompanhado por duas componentes de informação:

- Um endereço de rede que especifica o destinatário do pacote;
- Um número de porto que indica ao destinatário qual a socket a utilizar para enviar informação;

3. Realização do trabalho

Os sockets normalmente funcionam em pares: um cliente e um servidor (figura 1). Estes exercícios exploram este paradigma utilizando o protocolo TCP (Transfer Control Protocol).

Uma socket cliente estabelece uma ligação para a socket servidor, assim que é criado; Os pacotes são trocados de forma fiável;

A programação Java de sockets via protocolo TCP é suportado pelas classes Socket (socket de dados) e ServerSocket (socket do servidor), podendo resumir-se no seguinte pseudo-código:

Para um servidor;

1. Criar um objeto ServerSocket para aceitar ligações
2. Repetidamente:
    a. Invocar o método accept() do ServerSocket para receber a conexão do próximo cliente. Quando uma ServerSocket aceita uma ligação, cria um objeto Socket que encapsula a ligação;
    b. A socket deve criar objectos InputStream e OutputStream para ler e escrever bytes através da ligação;
    c. A ServerSocket pode opcionalmente criar uma nova thread para cada ligação, por forma a que o servidor possa aceitar novas ligações enquanto comunica com os clientes;
    d. Quando terminado, fechar a conexão utilizando o método close() da socket

Para um cliente:

1. Criar um objecto Socket que abre a ligação com o servidor, e utilizá-lo para comunicar com o servidor;
2. A socket deve criar objectos InputStream e OutputStream para ler e escrever bytes através da ligação;
3. Terminar a conexão utilizando o método close() da socket.

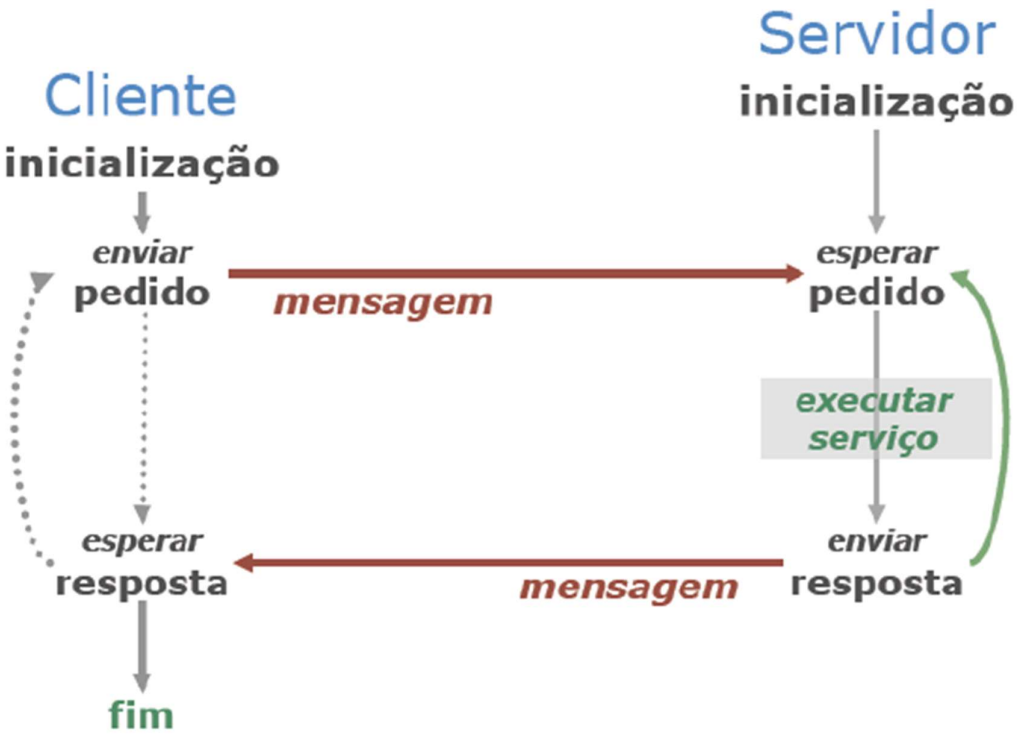

Figura 1: Cliente/Servidor diagrama de execução

Exercício 1

Estude e implemente o seguinte exemplo de um Cliente/Servidor (ECHO):

a) O objetivo do cliente é enviar uma mensagem de texto para o servidor (IP + porto).

```
import java.net.*;
import java.io.*;
// Connectar ao porto 6500 de um servidor especifico,
// envia uma mensagem e imprime resultado,

public class EchoClient {
	// usage: java EchoClient <servidor> <mensagem>
	public static void main(String args[]) throws Exception {
		Socket socket = new Socket(args[0], 6500);
		BufferedReader br = new BufferedReader(
			new InputStreamReader(socket.getInputStream()));
		PrintStream ps = new PrintStream(socket.getOutputStream());
		ps.println(args[1]); // escreve mensagem na socket
		// imprime resposta do servidor
		System.out.println("Recebido : " + br.readLine());
		// termina socket
		socket.close();
	}
}
```

b) O objetivo do servidor é receber a mensagem e devolvê-la à precedência

```
import java.net.*;
import java.io.*;
// Aguarda comunicação no porto 6500,
// recebe mensagens e devolve-as
public class EchoServer {
	public static void main(String args[]) throws Exception {
	//criar socket no porto 6500
	ServerSocket server = new ServerSocket(6500);
	System.out.println ("servidor iniciado no porto 6500");
	Socket socket = null;
	//aguarda mensagens
	while(true) {
		socket = server.accept();
		BufferedReader br = new BufferedReader(
			new InputStreamReader(socket.getInputStream()));
		PrintStream ps = new PrintStream(socket.getOutputStream());
		ps.println(br.readLine()); // Echo input para output
		//termina socket
		socket.close();
		}
	}
}
```

1. Execute o cliente e o servidor na mesma máquina.
2. Tente executar novamente o cliente com a máquina do seu colega. Para tal, o seu colega tem que inicializar o servidor antes de executar o cliente.
3. Modifique o programa do servidor para que imprima as mensagens recebidas do cliente.
4. Modifique o servidor de forma a que este permaneça em execução à espera da ligação de novos clientes utilizando *threads*.

5. Para testar o servidor, peça aos seus colegas que se conectem à sua máquina e que enviem mensagens.

Exercício 2

Pretende-se construir um servidor que a cada ligação de um cliente através de um Socket responda com uma frase aleatória. O servidor deverá possuir um array de Strings ou ficheiro de input com diferentes frases, por exemplo {"Bom dia", "Bem disposto?", "Ola", “Boas”, “Viva”}.

Sempre que um processo cliente envia uma mensagem ao servidor (independentemente do conteúdo dessa mensagem) o servidor responderá com uma das frases do array escolhida aleatoriamente. O servidor deve possuir um ciclo infinito onde aceita a ligação com o cliente, recebe a mensagem daquele e envia a sua frase aleatória.

Exercício 3

Utilize o exercício anterior e implemente um servidor que a cada ligação de um cliente através de um Socket responda de acordo com um comando. Implemente os seguintes comandos:

- horas – responde com a hora do sistema servidor;
- listar – lista todas as frases do servidor;
- frase – responde com uma frase aleatória
- tchau – termina a conexão com o servidor

Exercício 4

Para além do protocolo TCP, é também possível criar um cliente-servidor que comunica usando o protocolo UDP.

DatagramSockets é um tipo de *socket* que permite uma ligação sem conexão. Cada pacote enviado por esta *socket* é individualmente entregue. *Datagramsockets* fornece uma comunicação via UDP. Neste exemplo, temos um serviço de eco, em que o cliente envia mensagens via UDP para o servidor.

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

public class EchoClientUdp {
    private DatagramSocket socket;
    private InetAddress address;

    private byte[] buf;

    public EchoClientUdp(String address) throws SocketException,
UnknownHostException {
        socket = new DatagramSocket();
        this.address = InetAddress.getByName(address);
    }

    public String sendEcho(String msg) throws IOException {
        buf = msg.getBytes();
        DatagramPacket packet
```

```
          = new DatagramPacket(buf, buf.length, address, 4445);
        socket.send(packet);
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        String received = new String(
          packet.getData(), 0, packet.getLength());
        return received;
    }

    public void close() {
        socket.close();
    }


    public static void main(String[] args) {
        try {
            EchoClientUdp client = new EchoClientUdp(args[0]);
            System.out.println("Mensagem recebida: " + client.sendEcho(args[1]));
            client.sendEcho("end");
        } catch (SocketException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class EchoServerUdp extends Thread {
    private DatagramSocket socket;
    private boolean running;
    private byte[] buf = new byte[256];

    public EchoServerUdp() throws SocketException {
        socket = new DatagramSocket(4445);
    }

    public void run() {
        running = true;

        while (running) {
            try {
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                socket.receive(packet);

                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                packet = new DatagramPacket(buf, buf.length, address, port);
                String received
                        = new String(packet.getData(), 0, packet.getLength());

                if (received.equals("end")) {
                    running = false;
                    continue;
                }
                socket.send(packet);
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        }
        socket.close();
    }

    public static void main(String[] args) throws SocketException {
        EchoServerUdp serverUdp = new EchoServerUdp();
        serverUdp.run();
    }
}
```

1. Adapte a classe para implementar o exercício anterior.
2. Indique a mensagem e o IP do cliente quando é recebido uma mensagem.
3. Juntamente com os colegas, instancie um servidor e envia mensagens de diferentes clientes.