Prática - Sockets

Sistemas Distribuídos e Mobile

Prof. Me. Gustavo Torres Custódio
gustavo.custodio@ulife.com.br

Introdução

Protocolo TCP

Sockets

Sockets em Java

Protocolo UDP

Exercícios

Introdução

- Atualmente a internet utiliza o protocolo TCP/IP.
  - Baseado no modelo OSI com 7 camadas de rede.
- IP:
  - protocolo de rede responsável pelos endereços IPs das máquinas e verificação de rotas disponíveis.
- TCP:
  - protocolo de transporte que utiliza o IP para integridade dos dados.

Pratica - Sockets

Protocolo TCP

TCP

- *Transmission Control Protocol* (TCP):
  - Protocolo orientado à conexão.
  - 3 fases para conexão:
    - Fase de conexão;
    - Fase de dados;
    - Fase de desconexão.

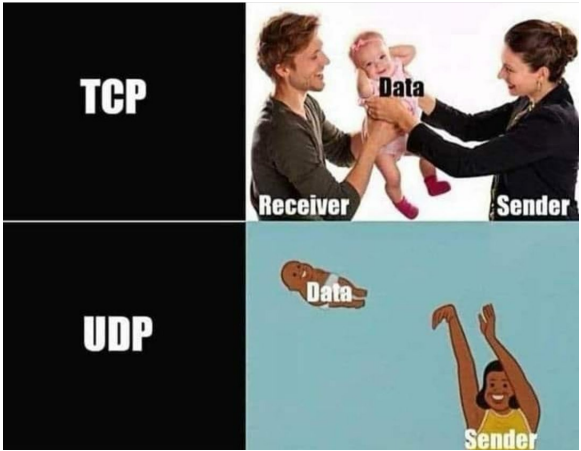
Protocolo UDP

- *User Datagram Protocol* (UDP):
  - Não é orientado a conexão;
  - Dados não são entregues com 100% de certeza;
  - Perdas de dados durante a transmissão acontecem;
  - Usado em serviços como VoIP (*Voice Over IP*).
    - A perda de dados resultaria em ruídos e falhas de vídeo.

TCP × UDP

- TCP:
  - Garante que o destinatário recebeu todos os pacotes;
  - Mais lento, pois checa todos os pacotes enviados e recebidos.
- UDP:
  - Sem garantia dos pacotes serem recebidos;
  - Mais rápido.

TCP × UDP



Sockets

Sockets

- As duas formas de comunicação (UDP e TCP) utilizam a abstração do **socket**.
- Um Socket é um ponto final (endpoint) de um canal bidirecional de comunicação entre dois programas rodando em uma rede;
  - Cada Socket tem os seguintes endereços de endpoint:
    - **Endereço local** (número da porta) que refere-se ao endereço da porta de comunicação para camada de transporte;
    - **Endereço global** (nome host) que refere-se ao endereço do computador (host) na rede.

Sockets

- Como a conexão acontece?
  - O servidor fica em espera, observando o *socket*, esperando um pedido de conexão do cliente;
  - O cliente sabe o endereço de IP e a porta associada à aplicação.
  - Assim que o servidor confirma a conexão, ele cria um novo Socket e pode ficar esperando novas conexões no Socket original, enquanto atende às requisições do cliente pelo novo Socket.

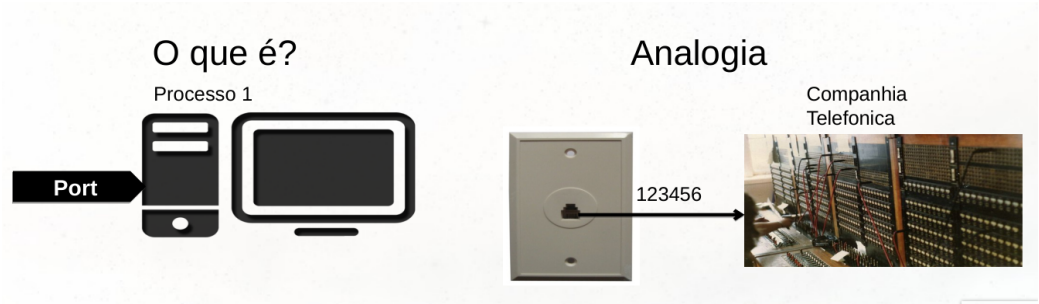
Como funciona? Analogia

- 1) Criação do Socket.

- Processo solicita um socket. Como se pedisse ao uma tomada de telefone.

Como funciona? Analogia

- 2) Binding



- Deve-se associar um endereço ao *socket* (o mesmo que associar um número de telefone à tomada).

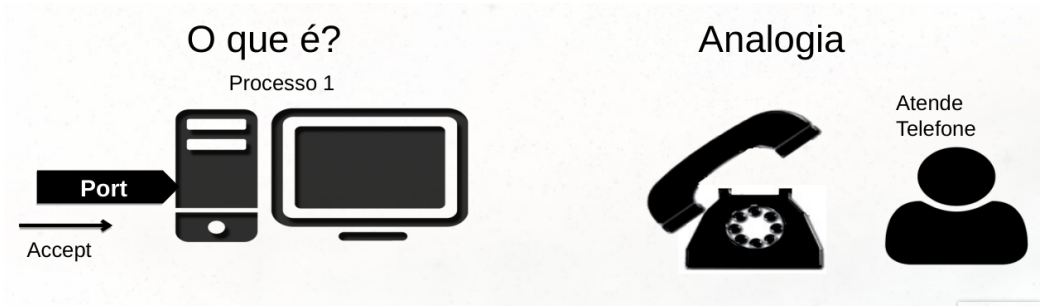
Como funciona? Analogia

- 3) Listening

- Processo fica aguardando por pedidos de conexão (esperando para ver se o telefone toca).

Como funciona? Analogia

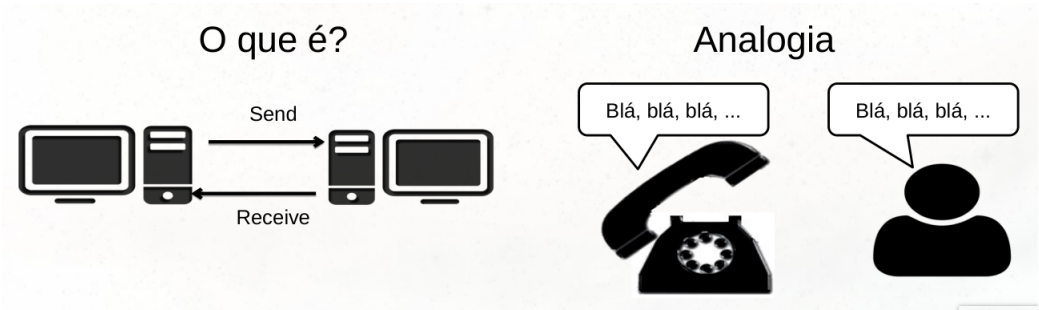
- 4) Accept



- Processo aceita pedidos de conexão (como se o telefone fosse atendido)

Como funciona? Analogia

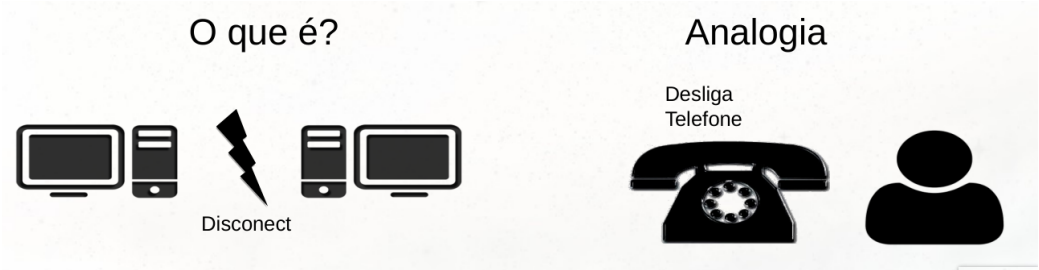
- 5) Send / Receive



- A conversação é realizada.

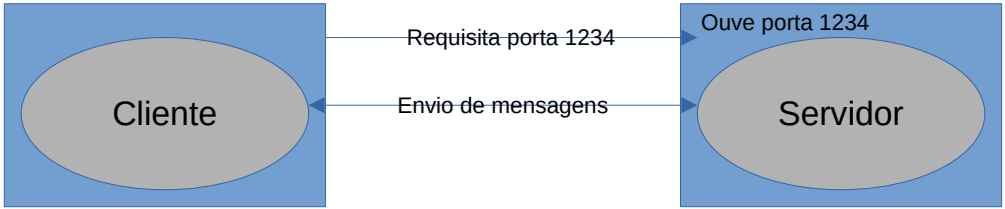
Como funciona? Analogia

- 6) Desconexão



- Fim da comunicação. O canal é fechado.

Sockets


Cliente
Requisita porta 1234
Envio de mensagens
Ouve porta 1234
Servidor

Portas

- A porta é a solução para identificar diversas aplicações em uma máquina.
  - 2 bytes (0 a 65636).
  - Se uma porta estiver ocupada, não é possível se comunicar a ela.

Sockets em Java

Sockets em Java

- Pacote `java.net` utiliza sockets.
- `Socket` e `ServerSocket` utilizam o protocolo TCP.
  - Orientado à conexão (servidor precisa aceitar o pedido de conexão).
- `DatagramSocket`:
  - Utiliza o protocolo UDP.

Leitura / Escrita em Sockets

- Sockets enviam e recebem dados na **forma de bytes**.
  - Cada conjunto de bytes é chamado stream.
  - Utilizam os seguintes métodos:
    - `getInputStream` (*read*).
    - `getOutputStream` (*write*).

Exemplo - Conexão TCP

- Conexao.java:

```
public static String receber(Socket socket) throws IOException {
    InputStream in = socket.getInputStream();
    byte infoBytes[] = new byte[100];
    int bytesLidos = in.read(infoBytes);

    if (bytesLidos > 0) {
        return new String(infoBytes);
    }else {
        return "";
    }
}

public static void enviar(Socket socket, String textoRequisicao) throws
     IOException {
    OutputStream out = socket.getOutputStream();
    out.write(textoRequisicao.getBytes());
}
```

Exemplo - Cliente TCP

- Cliente.java:

```
Socket socket;

public void comunicarComServidor() throws Exception {
    String textoRequisicao = "Conexao estabelecida";
    String textoRecebido = "";
    socket = new Socket("localhost", 9600);

    Scanner input = new Scanner(System.in);
    System.out.print("\nDigite a sua mensagem: ");
    textoRequisicao = input.nextLine();

    // Enviar mensagem para o servidor
    Conexao.enviar(socket, textoRequisicao);

    // Receber mensagem do servidor
    textoRecebido = Conexao.receber(socket);

    System.out.println("Servidor enviou: " + textoRecebido);
}
```

Exemplo - Cliente

- Cliente.java:

```
public static void main(String[] args) {
    try {
        Client cliente = new Client();
        cliente.comunicarServidor();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Exemplo - Servidor TCP

- Servidor.java:

```
Socket socketClient;
ServerSocket serversocket;

public boolean connect() {
    try {
        socketClient = serversocket.accept(); // fase de conexao
        return true;
    }catch (IOException e) {
        System.err.println("Nao fez conexao" + e.getMessage());
        return false;
    }
}

public static void main(String[] args) {
    try {
        Server servidor = new Server();
        servidor.rodarServidor();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Exemplo - Servidor TCP

- Servidor.java:

```
public void rodarServidor() throws Exception {
    String textoRecebido = "";
    String textoEnviado = "Olá, Cliente";
    Scanner input = new Scanner(System.in);

    serversocket = new ServerSocket(9600);
    System.out.println("Servidor iniciado!");

    while(true) {
        if (connect()) {
            textoRecebido = Conexao.receber(socketClient);

            System.out.println("Cliente enviou: " + textoRecebido);
            System.out.print("\nDigite a sua mensagem: ");// fase de dados
            textoEnviado = input.nextLine();
            Conexao.enviar(socketClient, textoEnviado);
            socketClient.close();
        }
    }
}
```

Servidor TCP

- Tente rodar os dois programas ao mesmo tempo, enviando mensagens um para o outro.
- Troque o “localhost” pelo IP da máquina de outra pessoa
  - que esteja rodando o servidor.

Pratica - Sockets

Protocolo UDP

UDP

- DatagramSocket
  - Pacote java.net;
  - Utiliza protocolo UDP;
  - Não é orientado à conexão;
- Envia os pacotes como bytes.
  - Utiliza o `DatagramPacket`.

UDP

- ClienteUdp.java:

```
public void comunicarServidor() throws Exception {
    DatagramSocket socket = new DatagramSocket();
    InetAddress address = InetAddress.getByName("localhost");
    int porta = 5252;

    String texto = "";
    Scanner entrada = new Scanner(System.in);

    while (!texto.trim().equalsIgnoreCase("fim")) {
        System.out.print("Digite uma mensagem: ");
        texto = entrada.nextLine();
        byte saida[];
        saida = texto.getBytes();

        DatagramPacket datagram = new DatagramPacket(saida, saida.length, address,
            porta);
        // connect() method
        socket.connect(address, porta);

        // send() method
        socket.send(datagram);
        System.out.println("Pacote enviado...");
    }
}
```

UDP

- ClienteUdp.java:

```
public static void main(String[] args) {
    try {
        Client cliente = new Client();
        cliente.comunicarServidor();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

UDP

- ServidorUdp.java:

```
public void rodarServidor() throws Exception {
    System.out.println("Servidor iniciado!");
    DatagramSocket socket = new DatagramSocket(5252);

    String mensagem = "";

    while(!mensagem.trim().equalsIgnoreCase("fim")) {
        byte[] buffer = new byte[200];

        DatagramPacket datagram = new DatagramPacket(buffer, 200);
        socket.receive(datagram);

        mensagem = new String(datagram.getData());

        System.out.println(mensagem);
    }
}
```

UDP

- ServidorUdp.java:

```
public static void main(String[] args) {
    try {
        Server servidor = new Server();
        servidor.rodarServidor();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

UDP

- O cliente envia uma mensagem para o servidor.
- O servidor devolve a mesma mensagem.

Pratica - Sockets

Exercícios

Exercício 1

- Vamos desenvolver um sistema cliente servidor de acordo com o protocolo definido na aula passada.

Exercício 2

- Criar uma aplicação utilizando UDP onde o cliente envia uma mensagem para o servidor.
  - A mensagem recebida pelo servidor deve ser exibida no console junto com o endereço do cliente que enviou a mensagem.

Exercício 3

- Faça um programa onde o usuário digita uma operação de inteiros, por exemplo (2 + 2) ou 2 * 2, que envie a operação para o servidor. O servidor irá processar a operação e retornará o resultado para o usuário.

Exercício 4

- Faça um programa onde o servidor armazene os dados de uma agenda, com nome e telefone.
  - O cliente deseja buscar um telefone na agenda passando o nome.

Exercício 5

- Crie um jogo da forca utilizando UDP ou TCP.
  - O servidor controla o jogo e o cliente fornece palpites das letras.
  - As mensagens devem conter a forca desenhada e as letras corretas até o momento.

Conteúdo

https://gustavotcustodio.github.io/sdmobile.html

Referências

- Sistemas Distribuídos: Conceitos e Projetos - Coulouris
  - Cap 4.

Obrigado

gustavo.custodio@ulife.com.br