BC1513: Práticas 1 e 2

Para Quarta-feira, 12 de Abril de 2017

João H. Kleinschmidt

Rodrigo Martins de Oliveira

Introdução

Nestas práticas de laboratório de redes de computadores são exercitados os conceitos de comunicação entre aplicações através do protocolo de transporte não-orientado à conexão, UDP, e do protocolo de transporte orientado à conexão, TCP. Os conceitos de endereço IP, porta e socket são cobertos.

Prática 1

Exercício 1

a)

O servidor UDP abre um socket UDP em uma porta específica e "escuta" pacotes que chegam por esta porta, capturando-os e lendo informações nele contidas. Após receber um pacote o servidor capitaliza o conteúdo deste e o envia de volta para o endereço e porta de origem informados pelo pacote.
O cliente UDP abre um socket UDP em uma porta qualquer e envia um pacote para o endereço e porta esperados do servidor e espera pela resposta na mesma porta.

b)

Como não há quaisquer mecanismos mais complexos de verificação do status de rede do servidor e de confirmações de envio e recebimento de pacotes, o pacote enviado pelo cliente é perdido já que o servidor ainda não está online e escutando a porta designada, então o cliente fica "eternamente" aguardando a resposta do servidor. Mesmo que o servidor venha a ficar online posteriormente, o pacote enviado anteriormente pelo cliente já foi perdido e o servidor não saberá disso, portanto, continuará aguardando a chegada de um novo pacote.

c)

Se a porta para a qual o cliente envia os pacotes não for a mesma porta a qual o servidor está conectado qualquer pacote enviado pelo cliente será perdido (ou capturado por outra aplicação que detém controle sobre a porta) e o servidor nunca receberá os pacotes. O resultado é semelhante ao que foi visto no exercício 1.b)

Exercício 2

Desde que o endereço de IP e porta do servidor estejam corretamente configurados no cliente, a comunicação entre os dois processos acontece normalmente.

Exercício 3

Sim, o servidor receberá todas as mensagens. Pelo fato de ser uma comunicação não-orientada à conexão, todos os pacotes, de diferentes origens, são enfileirados pela camada de transporte e enviados sequencialmente para o servidor lê-los, portanto a concorrência de recebimento de múltiplos pacotes de diferentes endereços de origem é abstraída da aplicação.

Exercício 4

O servidor não precisa sofrer quaisquer alterações dada sua abstração sobre a origem dos pacotes, para o servidor, o endereço e porta de origem do pacote apenas são um parâmetro e não definem qualquer tratamento de sessão especial (como seria no caso de uma conexão TCP, em que uma conexão é estabelecida).

```
public class UDPClient {
    public static void main(String[] args) throws SocketException,
            UnknownHostException, IOException
    {
        BufferedReader inFromUser =
                new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("127.0.0.1");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence;
        while (true) {
            sentence = inFromUser.readLine();
            if ("sair".equals(sentence)) {
                break;
            }
            sendData = sentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length, IPAddress, 9876);
            clientSocket.send(sendPacket);
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
                receiveData.length);
            clientSocket.receive(receivePacket);
            String modifiedSentence = new String(receivePacket.getData());
            System.out.println("Do servidor:" + modifiedSentence);
        }
        clientSocket.close();
    }
}
```

Exercício 5

```
public class UDPServer {
    public static void main(String[] args) throws SocketException,
        IOException {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        List<String> blacklist = Arrays.asList("172.31.33.27",
            "172.31.33.29");
        while(true)
        {
```

Exercício 5 continued on next page...

```
            DatagramPacket receivePacket
                    = new DatagramPacket(receiveData, receiveData.length);
            System.out.println("Servidor aguardando..." );
            serverSocket.receive(receivePacket);
            InetAddress IPAddress = receivePacket.getAddress();

            if (blacklist.contains(IPAddress.toString())) {
                continue;
            }

            String sentence = new String( receivePacket.getData());
            System.out.println("Mensagem recebida: " + sentence);
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket
                    = new DatagramPacket(sendData, sendData.length,
                        IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

Exercício 6

patients.xds

Exercício 7

A aplicação do Whatsapp foi criada contendo: (i) um servidor principal que é responsável por receber as mensagens de clientes e garantir que sejam recebidas aos respectivos clientes de destino; e (ii) aplicações cliente que enviam mensagens endereçadas a outros clientes para o servidor e recebem mensagens do servidor remetidas por outros clientes.
Cada aplicação cliente possui um identificador único obtido informado pelo servidor e pode enviar mensagens endereçando-as a outros clientes através de seus respectivos identificadores. O cliente espera receber informações sobre o recebimento da mensagem pelo destinatário e também sobre mensagens recebidas.
O servidor espera que cada cliente envie mensagens formatadas segundo um determinado padrão e envia respostas de confirmação de recebimento para os clientes e espera que eles também enviem respostas de confirmação de recebimento de mensagens para poder informar os remetentes das mensagens sobre sua entrega.
Cada cliente possui um mecanismo de *retry* para o envio de mensagens para garantir que elas sejam entregues ao servidor.
O servidor guarda representações de conversas entre clientes, nas quais enfileira mensagens a serem entregues. Quando as mensagens são entregues aos destinatários o servidor apaga suas cópias locais.

Prática 2

Exercício 1

a)

O servidor TCP abre um socket TCP em uma porta específica para aceitar conexões externas e espera uma conexão ser estabelecida nesta porta. Após estabelecer a conexão com um cliente, o servidor lê o fluxo de dados escrito na porta da conexão como uma String, capitaliza o conteúdo dessa String e a escreve no buffer de saída da porta da conexão.
O cliente TCP tenta conectar-se, através de um socket TCP qualquer, ao endereço e porta esperados do servidor e, obtendo sucesso na conexão, escreve no buffer de saída uma String para ser lida pelo servidor e espera ler do buffer de entrada a resposta do servidor.
Diferente do servidor UDP, este servidor espera por uma *conexão* e não lê um pacote, mas sim um buffer de entrada. Já o cliente TCP, diferentemente do UDP, tenta estabelecer uma conexão com o servidor, para então escrever em um buffer de saída dentro desta conexão e não enviar um pacote para o endereço e porta do servidor.

b)

Como não há quaisquer mecanismos mais complexos de verificação do status de rede do servidor o cliente tenta estabelecer conexão no endereço e porta esperados do servidor, mas tem sua conexão recusada, pois, como o servidor não está executando, a porta especificada no endereço requerido pelo cliente não estará aberta e irá rejeitar quaisquer conexões. O cliente falhará sua execução pela falta de mecanismos para lidar com a exceção lançada ao ter a conexão recusada.

c)

Se a porta para a qual o cliente envia os pacotes não for a mesma porta a qual o servidor está conectado a conexão do cliente será recusada (ou estabelecida por outra aplicação que detém controle sobre a porta) e o servidor nunca estabelecerá conexão com o cliente. O resultado é semelhante ao que foi visto no exercício 1.b)

Exercício 2

Desde que o endereço de IP e porta do servidor estejam corretamente configurados no cliente, a comunicação entre os dois processos acontece normalmente.

Exercício 3

Sim, o servidor receberá todas as mensagens. Pelo fato de haver um controle de acesso concorrente à porta aberta pelo servidor por parte do Java ServerSocket, cada requisição de conexão é enfileirada e espera a

porta estar livre para um nova conexão para estabelecer a conexão. Como cada cliente espera o retorno de sua requisição de conexão para escrever sua mensagem, todos clientes entregarão suas mensagens quando estabelecerem sua conexão com o servidor.

Exercício 4

Servidor TCP:

```
class TCPServer {

    public static void main (String args[]) throws Exception {

        ServerSocket serverSocket = new ServerSocket(9000);
        // waits for a new connection. Accepts connetion from multiple clients
        while (true)
        {
            System.out.println("Esperando conexao na porta 9000");
            Socket s = serverSocket.accept();
            System.out.println("Conexao estabelecida de " +
                s.getInetAddress());

            // create a BufferedReader object to read strings from
            // the socket. (read strings FROM CLIENT)
            BufferedReader br = new BufferedReader(new
                InputStreamReader(s.getInputStream()));
            String input;

            double weight, height, imc;

            input = br.readLine();
            weight = Double.parseDouble(input);

            input = br.readLine();
            height = Double.parseDouble(input);

            imc = weight/(height*height);

            //create output stream to write to/send TO CLINET
            DataOutputStream output = new
                DataOutputStream(s.getOutputStream());

            output.writeBytes(String.valueOf(imc) + "\n");
            // close current connection
            s.close();
        }
    }
}
```

Cliente TCP:

```
public class TCPClient {
```

```
    public static void main (String args[]) throws Exception {

        // Connect to the server process running at host
        Socket s = new Socket("127.0.0.1", 9000);

        // The next 2 lines create a output stream we can
        // write to.  (To write TO SERVER)
        OutputStream os= s.getOutputStream();
        DataOutputStream serverWriter = new DataOutputStream(os);

        // The next 2 lines create a buffer reader that
        // reads from the standard input. (to read stream FROM SERVER)
        InputStreamReader isrServer = new
            InputStreamReader(s.getInputStream());
        BufferedReader serverReader = new BufferedReader(isrServer);


        //create buffer reader to read input from user. Read the user input
            to string 'sentence'
        BufferedReader inFromUser = new BufferedReader(new
            InputStreamReader(System.in));
        String input;

        input = inFromUser.readLine();

        // Send a user input to server
        serverWriter.writeBytes(input +"\n");

        input = inFromUser.readLine();

        // Send a user input to server
        serverWriter.writeBytes(input +"\n");

        // Server should convert to upper case and reply.
        // Read server's reply below and output to screen.
        String response = serverReader.readLine();
        System.out.println(response);
        s.close();
    }
}
```

Exercício 5

Servidor TCP:

```
class TCPServer {

    public static void main (String args[]) throws Exception {
```

```
        ServerSocket serverSocket = new ServerSocket(9000);
        // waits for a new connection. Accepts connetion from multiple clients
        while (true)
        {
            System.out.println("Esperando conexao na porta 9000");
            Socket s = serverSocket.accept();
            System.out.println("Conexao estabelecida de " +
               s.getInetAddress());

            // create a BufferedReader object to read strings from
            // the socket. (read strings FROM CLIENT)
            BufferedReader br = new BufferedReader(new
               InputStreamReader(s.getInputStream()));
            String input;

            double weight, height, imc;

            while (true) {
                input = br.readLine();

                if ("tchau".equals(input)) {
                    break;
                }

                weight = Double.parseDouble(input);

                input = br.readLine();
                height = Double.parseDouble(input);

                imc = weight / (height * height);

                //create output stream to write to/send TO CLINET
                DataOutputStream output = new
                   DataOutputStream(s.getOutputStream());

                output.writeBytes(String.valueOf(imc) + "\n");
            }
            // close current connection
            s.close();
        }
    }
}
```

Cliente TCP:

```
public class TCPClient {

    public static void main (String args[]) throws Exception {
```

```
        // Connect to the server process running at host
        Socket s = new Socket("127.0.0.1", 9000);

        // The next 2 lines create a output stream we can
        // write to.  (To write TO SERVER)
        OutputStream os= s.getOutputStream();
        DataOutputStream serverWriter = new DataOutputStream(os);

        // The next 2 lines create a buffer reader that
        // reads from the standard input. (to read stream FROM SERVER)
        InputStreamReader isrServer = new
            InputStreamReader(s.getInputStream());
        BufferedReader serverReader = new BufferedReader(isrServer);


        //create buffer reader to read input from user. Read the user input
            to string 'sentence'
        BufferedReader inFromUser = new BufferedReader(new
            InputStreamReader(System.in));
        String input;

        while (true) {
            input = inFromUser.readLine();

            // Send a user input to server
            serverWriter.writeBytes(input + "\n");

            if ("tchau".equals(input)) {
                break;
            }

            input = inFromUser.readLine();

            // Send a user input to server
            serverWriter.writeBytes(input + "\n");

            // Server should convert to upper case and reply.
            // Read server's reply below and output to screen.
            String response = serverReader.readLine();
            System.out.println(response);
        }
        s.close();
    }
}
```

Exercício 6

O cliente não sofre alterações. O servidor deve criar threads para aceitar múltiplas conexões:

```
public class TCPServer {
    public static void main (String[] args) {
        try {
            ServerSocket server = new ServerSocket(9000);
            while (true) {
                Socket client = server.accept();
                ServerThread handler = new ServerThread(client);
                handler.start();
            }
        }
        catch (Exception e) {
            System.err.println("Exception caught:" + e);
        }
    }
}

class ServerThread extends Thread {

    Socket client;
    ServerThread (Socket client) {
        this.client = client;
    }

    public void run () {
        try {
            // waits for a new connection. Accepts connetion from multiple
               clients
            while (true) {
                System.out.println("Esperando conexao na porta 9000");
                System.out.println("Conexao estabelecida de " +
                   client.getInetAddress());

                // create a BufferedReader object to read strings from
                // the socket. (read strings FROM CLIENT)
                BufferedReader br = new BufferedReader(new
                   InputStreamReader(client.getInputStream()));
                String input;

                double weight, height, imc;

                while (true) {
                    input = br.readLine();

                    if ("tchau".equals(input)) {
                        break;
                    }

                    weight = Double.parseDouble(input);
```

```
                    input = br.readLine();
                    height = Double.parseDouble(input);

                    imc = weight / (height * height);

                    //create output stream to write to/send TO CLIENT
                    DataOutputStream output = new
                        DataOutputStream(client.getOutputStream());

                    output.writeBytes(String.valueOf(imc) + "\n");
                }
                // close current connection
                client.close();
            }
        } catch (Exception e) {
            System.err.println("Exception caught: client disconnected.");
        } finally {
            try { client.close(); }
            catch (Exception e ){

            }
        }
    }
}
```

Conclusão

Nesta prática observamos que o protocolo UDP permite abstrair a origem dos pacotes de dados, já que não requer conexão, permitindo ao servidor lidar fácilmente com múltiplos clientes quando o tratamento *per* pacote é isolado e independente.
Também é notável que o protoloco UDP requer cuidados especiais para garantir o recebimento de pacotes.