THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Name: Tanvi Khasgiwal
Roll No: 42

Date: 12/04/21

Lab Assignment No 9

Aim: Socket programming using UDP

Lab Outcome Attained: To implement client-server socket programs.

Theory:

Socket Programming

UDP also allows two (or more) processes running on different hosts to communicate. However, UDP differs from TCP in many fundamental ways. First, UDP is a connectionless service – there isn't an initial handshaking phase during which a pipe is established between the two processes. Because UDP doesn't have a pipe, when a process wants to send a batch of bytes to another process, the sending process must exclude attach the destination process's address to the batch of bytes. And this must be done for each batch of bytes the sending process sends. Thus UDP is similar to a taxi service – each time a group of people get in a taxi, the group has to inform the driver of the destination address. As with TCP, the destination address is a tuple consisting of the IP address of the destination host and the port number of the destination process. We shall refer to the batch of information bytes along with the IP destination address and port number as the "packet".

After having created a packet, the sending process pushes the packet into the network through a socket. Continuing with our taxi analogy, at the other side of the socket, there is a taxi waiting for the packet. The taxi then drives the packet in the direction of the packet's destination address. However, the taxi does not guarantee that it will eventually get the datagram to its ultimate destination; the taxi could break down. In other terms, UDP provides an unreliable transport service to its communication processes – it makes no guarantees that a datagram will reach its ultimate destination.

The java code of UDP is different from the TCP code in many important ways. In particular, we shall see that there is (i) no initial handshaking between the two processes, and therefore no need for a welcoming socket, (ii) no streams are attached to the sockets, (iii) the sending hosts creates "packets" by attaching the IP destination address and port number to each batch of bytes it sends, and (iv) the receiving process must unravel to received packet to obtain the packet's information bytes.

UDPServer

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

The Java I/O package, a.k.a. java.io, provides a set of input streams and a set of output streams used to read and write data to files or other input and output sources.

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

java.net package is used for network related programming. In the above command we have imported some packets.

```
class UDPServer {
  public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(9876);
```

The above command creates datagram socket at port 9876.

```
    byte[] receiveData = new byte[1024];
```

The above command creates byte array receiveData to hold the the data sever receives.

```
    byte[] receiveData1 = new byte[1024];
```

The above command creates byte array receiveData to hold the the data server receives

```
    byte[] sendData = new byte[1024];
```

The above command creates byte array sendData to hold the the data server sends.

```
    int num1, num2 = 0, sub;
```

The above command is used to declare integer variables required for the program.

```
    while (true) {
```

The above command starts the while loop and the loop is working till server is on.

```
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
```

The above command creates space for received datagram.

```
      serverSocket.receive(receivePacket);
```

The above command is used to receive datagram.

DatagramPacket receivePacket1 = new DatagramPacket(receiveData1, receiveData1.length);

The above command creates space for received datagram .*

serverSocket.receive(receivePacket1);

**** The above command is used to receive datagram.****

String sentence = new String(receivePacket.getData());

The above command creates a string variable to hold the received data from the client.*

String sentence1 = new String(receivePacket1.getData());

The above command creates a string variable to hold the received data from the client.*

System.out.println("RECEIVED: " + sentence.trim());

The above command prints the received data and trim it. *

System.out.println("RECEIVED: " + sentence1.trim());

***The above command prints the received data and trim it*

num1 = Integer.parseInt(sentence.trim());

The above command converts the string variable into integer variable. *

num2 = Integer.parseInt(sentence1.trim());

The above command converts the string variable into integer variable. *

sub = num1 - num2;

The above command subtracts the two numbers. *

InetAddress IPAddress = receivePacket.getAddress();

The above command gets the IPAddress of the sender. *

int port = receivePacket.getPort();

The above command is used to get the port number of the sender. *

String capitalizedSentence = String.valueOf(sub);

The above command gets and converts the integer variable into string variable. *

sendData = capitalizedSentence.getBytes();

The above command converts the string variable into bytes. *

```
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
IPAddress, port);
```

The above command creates datagram to send to client. *

```
        serverSocket.send(sendPacket);
```

The above command writes out datagram to socket. *

```
    }
```

After this bracket the while loop ends and the loop back and wait fror another client connection. *

```
  }
}
```

UDPClient

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

The Java I/O package, a.k.a. java.io, provides a set of input streams and a set of output streams used to read and write data to files or other input and output sources.*

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

java.net package is used for network related programming. In the above command we have imported some packets.*

```
class UDPClient {
  public static void main(String args[]) throws Exception {
    BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
```

The above command creates an input stream.*

```
    DatagramSocket clientSocket = new DatagramSocket();
```

***The above command creates the object clientSocket of type DatagramSocket*

```
    InetAddress IPAddress = InetAddress.getByName("localhost");
```

The above command translates hostname to IP address using DNS. The line invokes a DNS look up that translates "hostname" to an IP address.*

byte[] sendData = new byte[1024];

The above command creates byte array sendData to hold the the data client sends .

byte[] sendData1 = new byte[1024];

The above command creates byte array sendData to hold the the data client sends.

byte[] receiveData = new byte[1024];

The above command creates byte array receiveData to hold rhe the data client receives.

System.out.println("Enter Two numbers: ");

String sentence = inFromUser.readLine();

The above command reads the string variable from the client

String sentence2 = inFromUser.readLine();

The above command reads the string variable from the client

String num1 = sentence;

The above command takes one string variable and stores in another

String num2 = sentence2;

The above command takes one string variable and stores in another

sendData = num1.getBytes();

The above command converts the string variable into bytes.

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

***The above command creates datagram with data-to-send, length, IP addr, port ***

clientSocket.send(sendPacket);

The abovecommand sends datagram to server.

sendData1 = num2.getBytes();

The above command converts the string variable into bytes.

DatagramPacket sendPacket1 = new DatagramPacket(sendData1, sendData1.length, IPAddress, 9876);

***The above command creates datagram with data-to-send, length, IP addr, port ***

```
    clientSocket.send(sendPacket1);
```

The above command sends datagram to server.

```
    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
```

The above command is used while waiting for the packet from the server the client creates a place holder for the packet.

```
    clientSocket.receive(receivePacket);
```

The above command reads datagram from the server.

```
    String modifiedSentence = new String(receivePacket.getData());
```

The above command stores the data received from server into a string variable.

```
    System.out.println("FROM SERVER: Subtraction : " + modifiedSentence.trim());
```

The above command displays the received data from the server.

```
    clientSocket.close();
```

The above command closes the socket connection.

```
  }
}
```

Name: Tanvi Khasgiwal
Roll No: 42

Screenshots:

Command Prompt - java UDPServer

```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ajayk>d:

D:\>cd java prog

D:\Java prog>javac UDPServer.java

D:\Java prog>java UDPServer
RECEIVED: 18
RECEIVED: 9
```

Command Prompt

```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ajayk>d:

D:\>cd java prog

D:\Java prog>javac UDPClient.java

D:\Java prog>java UDPClient
Enter two numbers:
18
9
FROM SERVER: Division of two numbers:2

D:\Java prog>
```

Conclusion:

Thus, we have successfully implemented socket programing using UDP.