

Institution of Technology

School of Computing

Department of Information Technology and computer science

Course Name: Introduction to Distributed System (ITec4102)

Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](mailto:yimer@wku.edu.et)

Lab 1: Connection oriented and Connection less communication

Objectives: After completing this lab. you will be able to:

• This Lab is an introduction and review of the basics of client and server communication.

• Connection-less and connection-oriented communication using Java.

• Sockets as a means of achieving a communication between the client and server in a connection-oriented manner.

• Datagram as a means of achieving a communication between the client and server in a connection-less manner.

• The followings are the primary objectives of this lab session:

– Understanding the basics of client and server communication.

– Discussion of the basics of clients and servers.

– Differentiate between sockets and datagrams.

– Implementing simple client and server communication using connection-oriented communication.

– Implementing simple client and server communication using connection-less communication.

1.1 Connection-Oriented communication using TCP/IP

Aim:Write a program for implementing Client Server communication model using both connection-oriented and connection-less communication.

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

Practical 1:

Implement a simple client server TCP based chatting application (connection-oriented) using sockets.

Procedures:

• Implementing the server

1. Enable the server to listen connections on a specific port.

2. Enable the server to accept sockets coming on that port.

1-1

3. Create streams for writing and reading data to and from the socket. Communicate with the client through the socket using the I/O streams.

4. Close the socket and the I/O streams when the communication is over.

• Implementing the client

1. Create a socket from the client to the server using the address and listening port of the server.

2. Create streams for writing and reading data to and from the socket.

3. Communicate with the server through the socket using the I/O streams.

4. Close the socket and the I/O streams when the communication is over.

Implementing the server:

• Step 1: Create a ServerSocket object.

ServerSocket server = new ServerSocket( portNumber);

• Step 2: Wait for a Connection

Socket connection = server.accept();

• Step 3: Get the Socket’s I/O Streams

PrintWriter out = new PrintWriter(connection.getOutputStream(),true);

BufferedReader in = new BufferedReader( new InputStreamReader(connection.getInputStream()));

• Step 4: Perform the Processing

Step 4 is the processing phase, in which the server and the client communicate via the OutputStream and

InputStream objects.

• Step 5: Close the Connection

In Step 5, when the transmission is complete, the server closes the connection by invoking the close method on the streams and on the Socket.

Code Implementation of Server side

package ds\_lab;

import java.net.\*;

import java.io.\*;

class ChatServer {

public static void main(String args[]) {

try {

ServerSocket server = new ServerSocket(8000); System.out.println("Waiting for client to connect.."); Socket connection = server.accept();

System.out.println("Connection created with the client....");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); PrintWriter out = new PrintWriter(connection.getOutputStream(), true); BufferedReader in = new BufferedReader(new

InputStreamReader(connection.getInputStream())); String receive, send;

do {

receive = in.readLine(); System.out.println("Client Says: " + receive); if (receive.equals("STOP")) {

break;

}

System.out.print("Server Says : "); send = br.readLine(); out.println(send);

} while (true); br.close(); in.close(); out.close(); connection.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

Implementing the Client:

• Step 1: Create a Socket to Connect to the Server.

Socket connection = new Socket( serverAddress, port );

• Step 2: Get the Socket’s I/O Streams

PrintWriter out = new PrintWriter(connection.getOutputStream(),true);

BufferedReader in = new BufferedReader( new InputStreamReader(connection.getInputStream()));

• Step 3: Perform the Processing

Step 3 is the processing phase, in which the server and the client communicate via the OutputStream and

InputStream objects.

• Step 4: Close the Connection

In Step 4, when the transmission is complete, the client closes the connection by invoking the close method on the streams and on the Socket.

Code Implementation of the client

import java.net.\*;

import java.io.\*;

class ChatClient {

public static void main(String args[]) {

try {

Socket connection = new Socket("Localhost", 8000);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); PrintWriter out = new PrintWriter(connection.getOutputStream(), true); BufferedReader in = new BufferedReader(new

InputStreamReader(connection.getInputStream())); String msg;

System.out.println("To stop chatting with server type STOP");

do {

System.out.print("Client Says: "); msg = br.readLine(); out.println(msg);

if (msg.equals("STOP")) {

break;

}

String response = in.readLine(); System.out.println("Server Says : " + response);

} while (true); br.close(); in.close(); out.close(); connection.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

Exercise:

1. Implement a server that can perform the four arithmetic operations of addition, subtraction, division and multiplication. Enable the clients to connect with the server, supply two numbers and get the result of the four operations.

2. Implement a time server that can return the current time, whenever a client requests for it.

1.2 Connection-less Communication using UDP

Practical 2:

A client server based program using datagrams to find if the number entered is even or odd. (Connection-less)

Procedures: Implementing the server

• Step 1: Create a DatagramSocket object.

DatagramSocket datagramSocket = new DatagramSocket( 2000);

• Step 2: Arrange a datagram packet to accept any incoming message in the given port from clients.

byte b[] = new byte[1024];

DatagramPacket dp = new DatagramPacket(b,b.length);

• Step 3: Receive an incoming message using the datagram packet

datagramSocket.receive(dp);

• Step 4: Perform the Processing on the message. Step 4 is the processing phase, in which the server reads the message from the datagram packet sent from the client, performs the required actions and arranges the response in the form of a datagram packet to be sent for the client.

byte b1[] = new byte[1024];

b1 = response.getBytes();

DatagramPacket dp1 = new DatagramPacket(b1,b1.length,InetAddress.getLocalHost(),1000);

• Step 5: Send the datagram packet to the client using the datagram socket

datagramSocket.send(dp1);

Code Implementation of the server

package ds\_lab;

import java.io.\*;

import java.net.\*;

public class UDP\_Server {

public static void main(String args[]) {

try {

DatagramSocket datagramSocket = new DatagramSocket(2000);

byte b[] = new byte[1024];

DatagramPacket dp = new DatagramPacket(b, b.length);

datagramSocket.receive(dp);

String str = new String(dp.getData(), 0, dp.getLength()); System.out.println(str);

int a = Integer.parseInt(str); String response = new String(); if (a % 2 == 0) {

response = "Number is even";

} else {

response = "Number is odd";

}

byte b1[] = new byte[1024];

b1 = response.getBytes();

DatagramPacket dp1 = new DatagramPacket(b1, b1.length, InetAddress.getLocalHost(),

1000);

datagramSocket.send(dp1);

} catch (Exception e) {

e.printStackTrace();

}

}

}

Procedures: Implementing the Client

• Step 1: Create a DatagramSocket object on a specific port.

DatagramSocket datagramSocket = new DatagramSocket( 1000);

• Step 2: Arrange a datagram packet to be sent to the server using the server port.

b=num.getBytes();

DatagramPacket dp = new DatagramPacket(b,b.length,InetAddress.getLocalHost(),2000);

• Step 3: Send the datagram packet using the datagram socket

datagramSocket.send(dp);

• Step 4: Arrange a datagram packet to accept the server’s response.

byte b1[] = new byte[1024];

DatagramPacket dp1 = new DatagramPacket(b1,b1.length);

• Step 5: Receive the servers response using the datagram packet

datagramSocket.receive(dp1);

Code Implementation of the client

package ds\_lab;

import java.io.\*;

import java.net.\*;

public class UDP\_Client {

public static void main(String args[]) {

try {

DatagramSocket datagramSocket = new DatagramSocket(1000);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); System.out.println("Enter a number : ");

String num = br.readLine();

byte b[] = new byte[1024];

b = num.getBytes();

DatagramPacket dp = new DatagramPacket(b, b.length, InetAddress.getLocalHost(), 2000);

datagramSocket.send(dp);

byte b1[] = new byte[1024];

DatagramPacket dp1 = new DatagramPacket(b1, b1.length);

datagramSocket.receive(dp1);

String str = new String(dp1.getData(), 0, dp1.getLength()); System.out.println(str);

} catch (Exception e) {

e.printStackTrace();

}

}

}

Assignment: Socket programming

Develop an application for exchanging text files in a local network. Your application consists the following:

• A server program that waits for a connection on a TCP port.

• A client program when started does the following:

– Accepts the username of the user

– Accepts the IP address and TCP port of the destination

– Accepts which file to send

– Opens TCP connection.

– Performs handshake where it exchanges username.

– Sends the text file.

– Gracefully terminates the session.

• The server program accepts the file, stores the file in a default directory and notifies the user of arriving files.

• You are expected to define and explain in detail your own messaging format including message headers and special characters for example for indicating start and end of header.

• You are expected to design and explain in detail your own protocol for handshake and session control.

• Test the program and include screenshots in your report.

• Trace the test file exchange using Wireshark and include the result of trace analysis in your report. The trace should indicate the messaging format and the session control in action.

• Optional tasks:

– Try to implement a simple encryption on the message body such as: Adding a constant shift on the ASCI

code, perform some sort of complement.

– Use threads to initiate both server and client on the same program.

– Develop GUI

– Include a desktop notification for incoming messages.

• You may use Java or Python programming languages