Name: K Srikanth Registration Number: 17ETCS002124

Laboratory 2

Title of the Laboratory Exercise: Basic Client server Programs

1. Introduction and Purpose of Experiment

   A basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection.

   Aim and Objectives

   Aim

   - To do socket programming with java

2. Experimental Procedure

   i. Analyse the problem statement

   ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

   iii. Implement the algorithm in C language

   iv. Compile the C program

   v. Test the implemented program

   vi. Document the Results

   vii. Analyse and discuss the outcomes of your experiment

1. **Questions**

   Implement the following using Java

   - Basic Client Server Communication using UDP
   - Basic Client Server Communication using TCP

4. Calculations/Computations/Algorithms

Server TCP (Algorithm)

1. Start
2. Class server_TCP
3. Initialize all the sockets and input streams
4. Create a constructor with port
    a. Try Block
        1. Create a Server Object with Port Number
        2. Accept the Connection
        3. Get the input buffer
        4. While Loop Until (Over Keyword is passed on)
            1. Try Block
                a. Read the Line using UTF
            2. Catch Block
                a. Catch the exception if there is
        5. Close the connection

    b. Catch Block

        1. Catch the exception if there

5. Main Function

    a. Create an Object of the Class (server_TCP) and listen at port 5000

6. Stop

Client TCP (Algorithm)

1. Start
2. Class Client_TCP
3. Initialize all the sockets and input and Output streams
4. Create a constructor with port and IP Address
    a. Try Block
        1. Create a Server Object with Port Number and IP Address
        2. Connected
        3. Input the buffer steam
        4. Output the buffer steam

    b. Catch Block

        1. Catch the Exceptions if the host is not known

    c. Catch Block

        1. Catch the Exceptions if the IO is not proper

        5. While Loop Until (Over Keyword is passed on)

            1. Try Block
                a. Input the Line using UTF
            2. Catch Block
                a. Catch the exception if there is

        1. Close the connection

    b. Catch Block

        1. Catch the exception if there

5. Main Function

    a. Create an Object of the Class (Client_TCP) and listen at port 5000

6. Stop

Server UDP (Algorithm)

1. Start
2. Class Server_UDP
3. Initialize the port at 8080
4. Main Function
    a. Try Block
        1. Create Datagram Socket Object with the Port Number
        2. Take the incoming Data Buffer
        3. Receive the Input Buffer
        4. Get the data from the input Buffer
        5. Obtain the IP Address and Port from the client
        6. Create a UDP packet to send to client
        7. Send the created packet
        8. Close the connection
    b. Catch Block
        1. Catch the exception if there

4. Stop

Client UDP (Algorithm)

1. Start
2. Class Client_UDP
3. Initialize the port at 8080
4. Main Function
    a. Try Block
        1. Create Datagram Socket Object with the Port Number
        2. Get the IP Address
        3. Receive the Input Buffer
        4. Get the data from the input Buffer and Send it to that Port
        5. Receive the data if there is from the server
        6. Close the connection
    b. Catch Block
        1. Catch the exception if there

4. Stop

5. Presentation of Results

Server TCP Code

```java
package TCP;
import java.net.*;
import java.io.*;// 17ETCS002124 K Srikanth
You, seconds ago | 1 author (You)
public class server_TCP {
    // initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;
    // constructor with port
    public server_TCP(int port) {
        // starts server and waits for a connection
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");
            System.out.println("Waiting for a client ...");
            socket = server.accept();
            System.out.println("Client accepted");
            // takes input from the client socket
            in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
            String line = "";
            // reads message from client until "Over" is sent
            while (!line.equals("Over")) {
                try {
                    line = in.readUTF();
                    System.out.println(line);
                } catch (IOException i) {
                    System.out.println(i);}}
            System.out.println("Closing connection");
            // close connection
            socket.close();
            in.close();
        } catch (IOException i) {
            System.out.println(i);}}
    Run | Debug
    public static void main(String args[]) {
        server_TCP server = new server_TCP(5000);}}    The value of the local variable server is not used
```

Figure 1 Java Program for TCP Server Side

Client TCP Code

```java
package TCP;
// A Java program for a ClientSide
import java.net.*;
import java.io.*;
You, seconds ago | 1 author (You)
// 17ETCS002124 K Srikanth
public class client_TCP {
    // initialize socket and input output streams
    private Socket socket = null;
    private DataInputStream input = null;
    private DataOutputStream out = null;
    // constructor to put ip address and port
    public client_TCP(String address, int port) {
        // establish a connection
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");
            // takes input from terminal
            input = new DataInputStream(System.in);
            // sends output to the socket
            out = new DataOutputStream(socket.getOutputStream());
        } catch (UnknownHostException u) {
            System.out.println(u);
        } catch (IOException i) {
            System.out.println(i);
        } // string to read message from input
        String line = "";
        // keep reading until "Over" is input
        while (!line.equals("Over")) {
            try {
                line = input.readLine();    The method readLine() from the type DataInputStream is deprecated
                out.writeUTF(line);} catch (IOException i) {
                System.out.println(i); } }
        // close the connection
        try {
            input.close();
            out.close();
            socket.close();} catch (IOException i) {
            System.out.println(i);  }}
    Run | Debug
    public static void main(String args[]) {
        client_TCP client = new client_TCP("127.0.0.1", 5000);}    The value of the local variable client is not used
```

Figure 2 Java Program for TCP Client Side

TCP Output for Server and Client

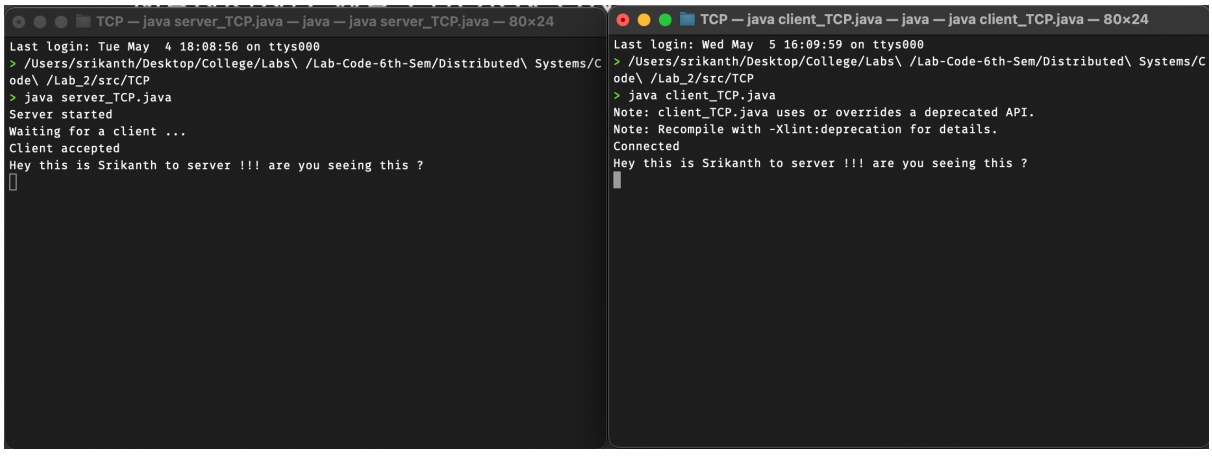

Figure 3 Java Program Output for TCP Connection

Server UDP Code

```java
package UDP;

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

You, seconds ago | 1 author (You)
// 17ETCS002124 K Srikanth
public class server_UDP {
    // Server UDP socket runs at this port
    public final static int SERVICE_PORT = 8080;

    Run | Debug
    public static void main(String[] args) throws IOException {
        try {
            // Instantiate a new DatagramSocket to receive responses from the client
            DatagramSocket serverSocket = new DatagramSocket(SERVICE_PORT);
            byte[] receivingDataBuffer = new byte[1024];
            byte[] sendingDataBuffer = new byte[1024];
            DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);
            System.out.println("Waiting for a client to connect...");
            // Receive data from the client and store in inputPacket
            serverSocket.receive(inputPacket);
            // Printing out the client sent data
            String receivedData = new String(inputPacket.getData());
            System.out.println("Sent from the client: " + receivedData);
            sendingDataBuffer = receivedData.toUpperCase().getBytes();
            // Obtain client's IP address and the port
            InetAddress senderAddress = inputPacket.getAddress();
            int senderPort = inputPacket.getPort();
            // Create new UDP packet with data to send to the client
            DatagramPacket outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress,
                    senderPort);
            // Send the created packet to client
            serverSocket.send(outputPacket);
            // Close the socket connection
            serverSocket.close();
        } catch (SocketException e) {
            e.printStackTrace();}}      You, seconds ago • Uncommitted changes
```

Figure 4 Java Program for UDP Server Side

Client UDP Code

```java
package UDP;

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

You, seconds ago | 1 author (You)
// 17ETCS002124 K Srikanth
public class client_UDP {
    /*
     * The server port to which the client socket is going to connect
     */
    public final static int SERVICE_PORT = 8080;

    Run | Debug
    public static void main(String[] args) throws IOException {
        try {
            DatagramSocket clientSocket = new DatagramSocket();

            // Get the IP address of the server
            InetAddress IPAddress = InetAddress.getByName("localhost");
            byte[] sendingDataBuffer = new byte[1024];
            byte[] receivingDataBuffer = new byte[1024];
            String sentence = "Hello from UDP client";
            sendingDataBuffer = sentence.getBytes();
            DatagramPacket sendingPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, IPAddress,
                    SERVICE_PORT);
            clientSocket.send(sendingPacket);
            DatagramPacket receivingPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);
            clientSocket.receive(receivingPacket);
            // Printing the received data
            String receivedData = new String(receivingPacket.getData());
            System.out.println("Sent from the server: " + receivedData);
            // Closing the socket connection with the server
            clientSocket.close();
        } catch (SocketException e) {
            e.printStackTrace();}}      You, seconds ago • Uncommitted changes

```

Figure 5 Java Program for UDP Client Side

UDP Output for Server and Client

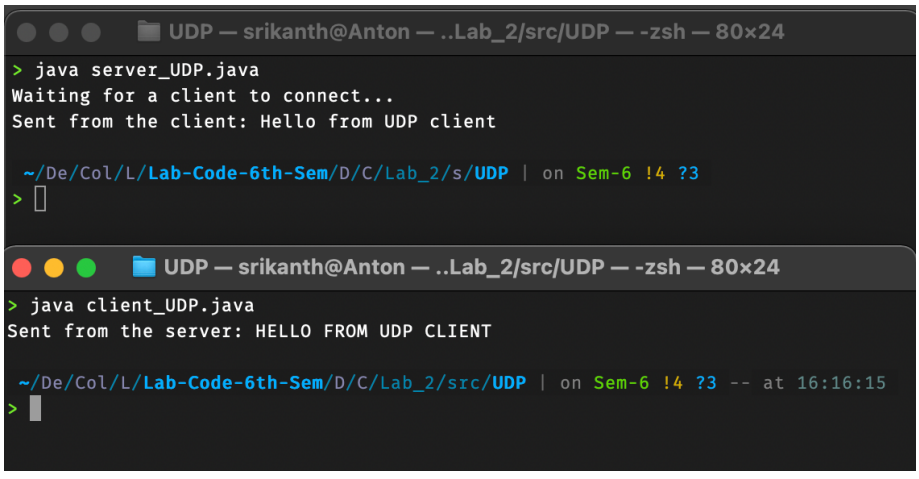

Figure 6 Java Program Output for UDP Connection

6. Analysis and Discussions

TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
1. It is a communications protocol, using which the data is transmitted between systems over the network. In this, the data is transmitted into the form of packets. It includes error-checking, guarantees the delivery and preserves the order of the data packets.	1. It is same as the TCP protocol except this doesn't guarantee the error-checking and data recovery. If you use this protocol, the data will be sent continuously, irrespective of the issues in the receiving end.
2. TCP is a connection oriented protocol	2. UDP is a connection less protocol.
3. As TCP provides error checking support and also guarantees delivery of data to the destination router this make it more reliable as compared to UDP.	3. While on other hand UDP does provided only basic error checking support using checksum so the delivery of data to the destination cannot be guaranteed in UDP as compared to that in case of TCP.
4. In TCP the data is transmitted in a particular sequence which means that packets arrive in-order at the receiver.	4. On other hand there is no sequencing of data in UDP in order to implement ordering it has to be managed by the application layer.
5. TCP is slower and less efficient in performance as compared to UDP. Also TCP is heavy-weight as compared to UDP.	5. On other hand UDP is faster and more efficient than TCP.
6. Retransmission of data packets is possible in TCP in case packet get lost or need to resend.	6. On other hand retransmission of packets is not possible in UDP.

Datagram Packets Class

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

- **DatagramPacket(byte[] buf, int length)**
  Constructs a DatagramPacket for receiving packets of length (length).
- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
  Constructs a datagram packet for sending packets of length (length) to the specified port number on the specified host.

Datagram Socket Class

A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

1. Limitations of Experiments

Data corruption is a common occurrence on the Internet, UDP is not good at error detection. Also No compensation for lost packets and the Packets can arrive out of order as this on the problem that No congestion control UDP may be light weight, but not that reliable.

2. Limitations of Results

UDP Packets are not that reliable

3. Learning happened

Learned about Socket Programming of UDP and TCP Protocols in Java.