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 programing

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. Algorithm/Pseudocodes

For Basic Client Server Communication using UDP

UDP Server :

Create UDP socket.

Bind the socket to server address.

Wait until datagram packet arrives from client.

Process the datagram packet and send a reply to client.

Go back to Step 3.

UDP Client :

Create UDP socket.

Send message to server.

Wait until response from server is recieved.

Process reply and go back to step 2, if necessary.

Close socket descriptor and exit.

In this Program I have used bye as a keyword to end conversation between Client and Server.

For Basic Client Server Communication using TCP

TCP Client :

1. The client initiates connection to a server specified by hostname/IP address and port number.

2. Send data to the server using an OutputStream.

3. Read data from the server using an InputStream.

4. Close the connection.

The steps 2 and 3 can be repeated many times depending on the nature of the communication.

TCP Server:

1.Create a server socket and bind it to a specific port number

2. Listen for a connection from the client and accept it. This results in a client socket is created for the connection.

3. Read data from the client via an InputStream obtained from the client socket.

4. Send data to the client via the client socket's OutputStream.

5. Close the connection with the client.-

5.Programs (Source Code)

For Basic Client Server Communication using UDP

Client Side

```java
// Java program to illustrate Client side
// Program Done By Deepak R
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class udpBaseClient_2
{
	public static void main(String args[]) throws IOException
	{
		Scanner sc = new Scanner(System.in);

		// Step 1:Create the socket object for
		// carrying the data.
		DatagramSocket ds = new DatagramSocket();

		InetAddress ip = InetAddress.getLocalHost();
		byte buf[] = null;

		// loop while user not enters "bye"
		while (true)
		{
			String inp = sc.nextLine();

			// convert the String input into the byte array.
			buf = inp.getBytes();

			// Step 2 : Create the datagramPacket for sending
			// the data.
			DatagramPacket DpSend =
				new DatagramPacket(buf, buf.length, ip, 1234);

			// Step 3 : invoke the send call to actually send
			// the data.
			ds.send(DpSend);

			// break the loop if user enters "bye"
			if (inp.equals("bye"))
				break;
		}
	}
}
```

Server Side

```java
package lab2;
// Java program to illustrate Server side
// Program Done By Deepak R
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
public class udpBaseServer_2
{
	public static void main(String[] args) throws IOException
	{
		// Step 1 : Create a socket to listen at port 1234
		DatagramSocket ds = new DatagramSocket(1234);
		byte[] receive = new byte[65535];
		DatagramPacket DpReceive = null;
		while (true)
		{
			// Step 2 : create a DatgramPacket to receive the data.
			DpReceive = new DatagramPacket(receive, receive.length);

			// Step 3 : revieve the data in byte buffer.
			ds.receive(DpReceive);

			System.out.println("Client:-" + data(receive));

			// Exit the server if the client sends "bye"
			if (data(receive).toString().equals("bye"))
			{
				System.out.println("Client sent bye.....EXITING");
				break;
			}
			// Clear the buffer after every message.
			receive = new byte[65535];
		}
	}
	// A utility method to convert the byte array
	// data into a string representation.
	public static StringBuilder data(byte[] a)
	{
		if (a == null)
			return null;
		StringBuilder ret = new StringBuilder();
		int i = 0;
		while (a[i] != 0)
		{
			ret.append((char) a[i]);
			i++;
		}
		return ret;
	}
}
```

For Basic Client Server Communication using UDP

Client Side

```java
package lab2a;
//Program Done By Deepak R
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
public class SocketClientExample {
    public static void main(String[] args) throws UnknownHostException, IOException, ClassNotFoundException, InterruptedException{
        //get the localhost IP address, if server is running on some other IP, you need to use that
        InetAddress host = InetAddress.getLocalHost();
        Socket socket = null;
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;
        for(int i=0; i<5;i++){
            //establish socket connection to server
            socket = new Socket(host.getHostName(), 9876);
            //write to socket using ObjectOutputStream
            oos = new ObjectOutputStream(socket.getOutputStream());
            System.out.println("Sending request to Socket Server");
            if(i==4)oos.writeObject("exit");
            else oos.writeObject(""+i);
            //read the server response message
            ois = new ObjectInputStream(socket.getInputStream());
            String message = (String) ois.readObject();
            System.out.println("Message: " + message);
            //close resources
            ois.close();
            oos.close();
            Thread.sleep(100);
        }
    }
}
```

Server Side

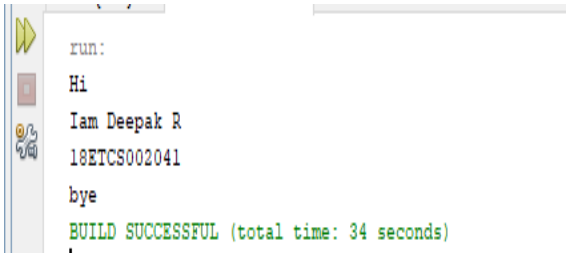
```java
package lab2a;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.net.ServerSocket;
import java.net.Socket;
//Program Done By Deepak R
public class SocketServerExample {
    //static ServerSocket variable
    private static ServerSocket server;
    //socket server port on which it will listen
    private static int port = 9876;
    public static void main(String args[]) throws IOException, ClassNotFoundException{
        //create the socket server object
        server = new ServerSocket(port);
        //keep listens indefinitely until receives 'exit' call or program terminates
        while(true){
            System.out.println("Waiting for the client request");
            //creating socket and waiting for client connection
            Socket socket = server.accept();
            //read from socket to ObjectInputStream object
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
            //convert ObjectInputStream object to String
            String message = (String) ois.readObject();
            System.out.println("Message Received: " + message);
            //create ObjectOutputStream object
            ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
            //write object to Socket
            oos.writeObject("Hi Client "+message);
            //close resources
            ois.close();
            oos.close();
            socket.close();
            //terminate the server if client sends exit request
            if(message.equalsIgnoreCase("exit")) break;
        }
        System.out.println("Shutting down Socket server!!");
        //close the ServerSocket object
        server.close();
    }

}
```

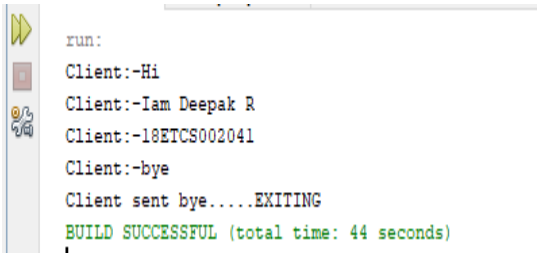
Outputs/Testcases

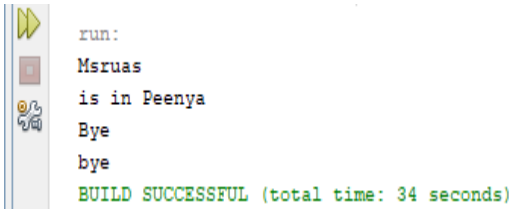
For Basic Client Server Communication using UDP

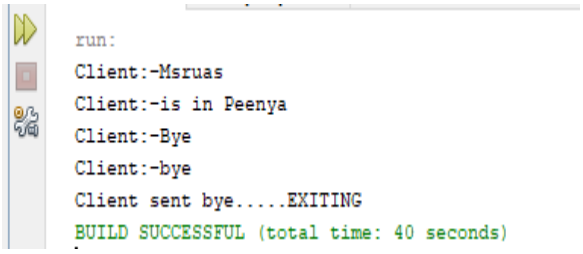
Client Side



Server Side

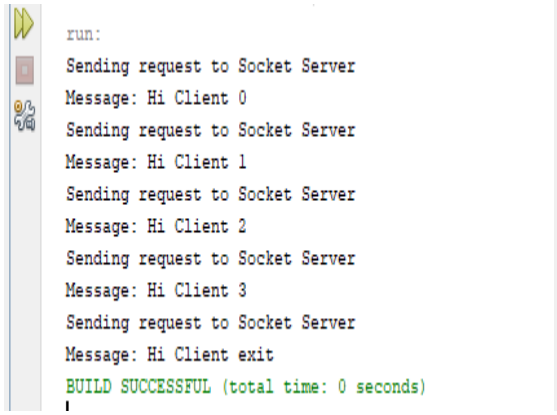




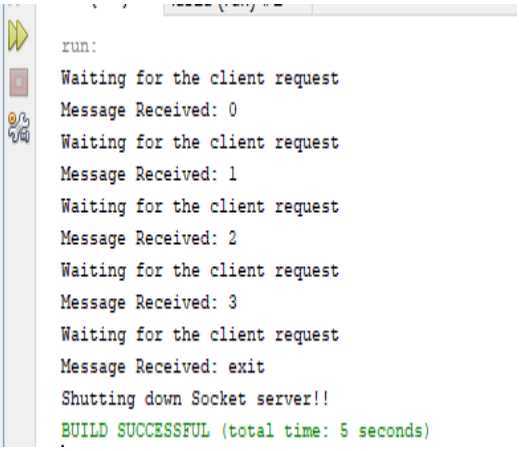


For Basic Client Server Communication using UDP

Client Side



Server Side



5. Analysis and Discussions

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.

In UDP, the client does not form a connection with the server like in TCP and instead, It just sends a datagram. Similarly, the server need not to accept a connection and just waits for datagrams to arrive. We can call a function called connect() in UDP but it does not result anything like it does in TCP. There is no 3 way handshake. It just checks for any immediate errors and store the peer's IP address and port number. connect() is storing peers address so no need to pass server address and server address length arguments in sendto().

1. Limitations of Experiments

None

2. Limitations of Results

Checked only for Small Samples it results may or may not vary if checked for large data.

3. Learning happened

We learned about Basic Client Server Communication using UDP and Basic Client Server Communication using TCP which are basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection.

4. Recommendations

None