##### **9.2 Write programs (server and client) to implement concurrent/iteraitve server to conntect multiple clients requests handled through concurrent/iterative logic using UDP/TCP socket connection.**

**OBJECTIVES:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS intraction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbersing and process relation.
5. To knows the iterative and concurrent server concept.

#### THEORY:

JAVA SOCKET PROGRAMMING

A very basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection. Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers.

CLIENT SIDE PROGRAMMING:

Establish a Socket Connection

To connect to other machine we need a socket connection. A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port.The java.net.Socket class represents a Socket. To open a socket:

Socket socket = new Socket(“127.0.0.1”, 5000)

3. First argument – IP address of Server. ( 127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

7. Second argument – TCP Port. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

To communicate over a socket connection, streams are used to both input and output the data. Closing the connection

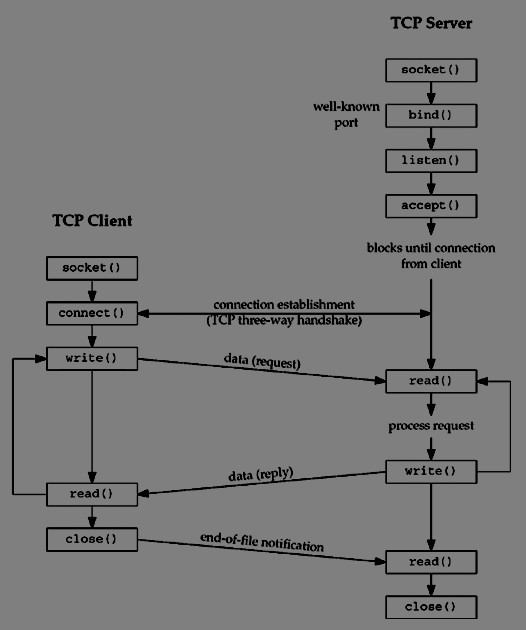
The socket connection is closed explicitly once the message to server is sent. SERVER SIDE PROGRAMMING:

Establish a Socket Connection

To write a server application two sockets are needed.

* A ServerSocket which waits for the client requests (when a client makes a new Socket())
  + A plain old Socket socket to use for communication with the client. getOutputStream() method is used to send the output through the socket. Close the Connection

After finishing, it is important to close the connection by closing the socket as well as input/output streams.



#### Data Dictionary:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
| 1 | ss | ServerSocket | Create a socket for server side communication. |
| 2 | s | Socket | Socket is created. |
| 3 | dos | DataOutputStream | Output Stream. |
| 4 | dis | DataInputStream | Input Stream. |
| 5 | cnm | String | String to display message from clients. |
| 6 | br | BufferedReader | Input data. |

**Program:**

SERVER:

import java.net.\*; import java.io.\*; class Server5

{

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

{

ServerSocket ss=new ServerSocket(5060); while(true)

{

Socket s=ss.accept();

DataOutputStream dos=new DataOutputStream(s.getOutputStream()); DataInputStream dis=new DataInputStream(s.getInputStream()); dos.writeUTF("Welcomes u");

String cnm=dis.readUTF();

ThrdComm a=new ThrdComm(dos,dis,cnm);

}

}

}

class ThrdComm extends Thread

{

DataOutputStream dos; DataInputStream dis; BufferedReader br; String str,cnm;

ThrdComm(DataOutputStream dos,DataInputStream dis,String cnm)throws Exception

{

super(cnm); this.dos=dos; this.dis=dis; this.cnm=cnm;

br=new BufferedReader(new InputStreamReader(System.in)); start();

}

public void run()

{

while(true)

{

try

{

talk();

}

catch(Exception e){}

}

}

synchronized void talk()throws Exception

{

System.out.println("Message To"+" "+cnm+":"); str=br.readLine();

dos.writeUTF(str);//sends msg to Client str=dis.readUTF();//reads msg from client System.out.println("From Client:"+" "+str);

}

}

CLIENT:

import java.net.\*; import java.io.\*;

class Client5 extends Thread

{

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

{

if(args.length!=1) return;

Client5 a=new Client5(args[0]);

}

Client5(String s1)throws Exception

{

super(s1);//naming to thread s=new Socket("localhost",5060);

dos=new DataOutputStream(s.getOutputStream()); dis=new DataInputStream(s.getInputStream());

br=new BufferedReader(new InputStreamReader(System.in)); cnm=s1;//set aurgument as client name

str="";

str=dis.readUTF();//reads msg send by server System.out.println("From Server:"+" "+str); dos.writeUTF(cnm);//client sends its name to server start();

}

public void run()

{

while(true)

{

try

{

talk();

}

catch(Exception e){}

}

}

synchronized void talk()throws Exception

{

str=dis.readUTF();//msg from server System.out.println("From Server:"+" "+str); System.out.println("Message to Server:"); str=br.readLine();

dos.writeUTF(str);

}

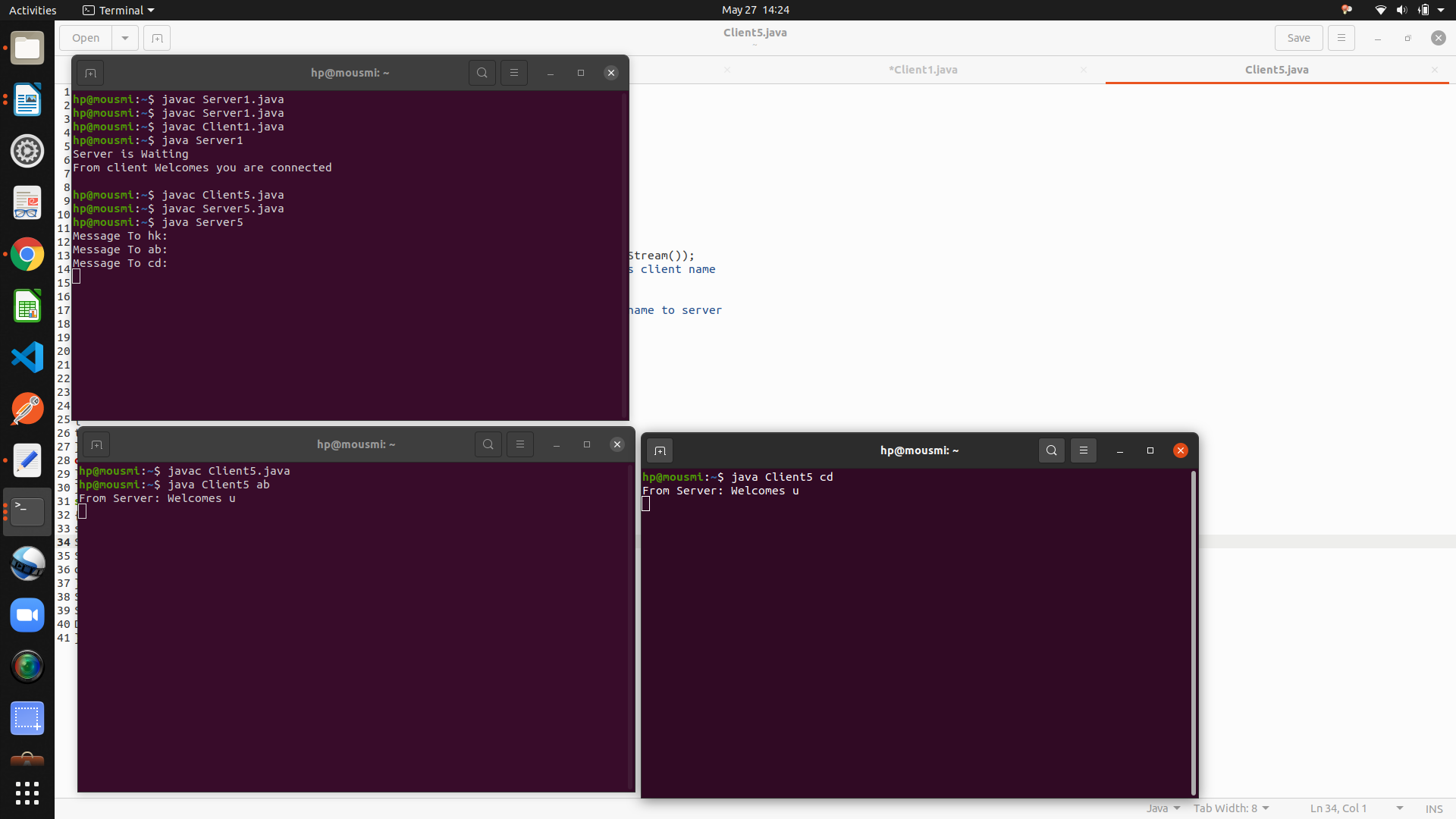
Socket s;

String str,cnm;

DataOutputStream dos; DataInputStream dis; BufferedReader br;

}

OUTPUT:

****

**Conclusion:**

Various communication protocols like TCP/UDP can be implemented using socket programming in Java to serve requests from multiple clients.

#### References:

<http://www.prasannatech.net/2008/07/socket-programming-tutorial.html>