PRACTICAL LAB FILE BASED ON NETWORK PROGRAMMING

PRACTICAL WORK

Submitted By

Shiv Kumar Chaudhary Kurmi

Under the guidance of

Mr. Bhavendra Sinha

In partial fulfillment for the award of the degree Of

B.TECH IN COMPUTER SCIENCE

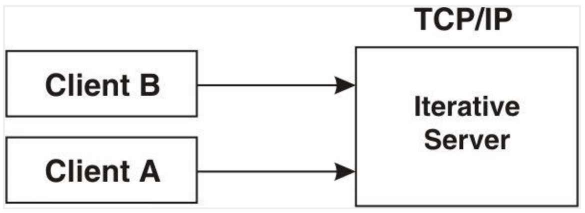
KALINGA UNIVERSITY

Village Kotni, Near Mantralaya Atal Nagar Raipur (C.G)

Session July – Dec -2020

1. Write an echo program with client and iterative server using TCP.

An iterative server handles both the connection request and the transaction involved in the call itself. Iterative servers are fairly simple and are suitable for transactions that do not last long.



Server side - server.java

```
import java.net.*; import
java.io.*;
public class TcpEchoServer { private
       static Socket socket;
       public static void main(String[] args) { try
              {
                     int port = 25002;
                     /*Creating ServerSocket*/
                     ServerSocket serverSocket = new ServerSocket(port);
                     System.out.println("Server Started and listening to the
port " + port);
                     /*Accepting connections from Clients*/ socket
                     = serverSocket.accept();
                     InputStream is = socket.getInputStream();
                     InputStreamReader isr = new InputStreamReader(is);
                     BufferedReader br = new BufferedReader(isr);
                     String name = br.readLine();
                     System.out.println("Message received from client is " +
name);
                     /* Manipulating the String*/
                     String returnMessage = "Hello " + name + " !!\n";
                     OutputStream os = socket.getOutputStream();
                     OutputStreamWriter osw = new OutputStreamWriter(os);
                     BufferedWriter bw = new BufferedWriter(osw);
                     bw.write(returnMessage);
                     System.out.println("Message sent to the client is " +
returnMessage); bw.flush();
              }
              catch (Exception e) {
                     e.printStackTrace();
              } finally { try {
              socket.close();
```

```
                        }
                        catch (Exception e){
                        }
                }
        }
}
```

Client side - client.java

```
import java.net.*; import
java.io.*;
public class TcpEchoClient { private
       static Socket socket;
       public static void main(String args[]) { try
              {
                     String host = "localhost"; int
                     port = 25002;
                     /*Determines the IP address of a host, given the host's
name.*/
                     InetAddress address = InetAddress.getByName(host);
                     /*Create client socket address*/
                     socket = new Socket(address, port); /*
                     Send the message to the Server */
                     OutputStream os = socket.getOutputStream();
                     /*Create output stream writer*/
                     OutputStreamWriter osw = new OutputStreamWriter(os);
                     /*Create buffered writer*/
                     BufferedWriter bw = new BufferedWriter(osw);
                     /*String to be passed*/
                     String Name = "Saurav";
                     String sendMessage = Name + "\n"; /*
                     writing string to writer*/
                     bw.write(sendMessage);
                     /*forces out the characters to string writer*/ bw.flush();
                     System.out.println("Message sent to the server : " +
sendMessage);
               /*Open input stream for reading purpose*/ InputStream
                            is = socket.getInputStream();
                     /*Create new input stream reader*/
                     InputStreamReader isr = new InputStreamReader(is);
                     /*Create new BufferedReader*/
                     BufferedReader br = new BufferedReader(isr);
                     /*Reading from the socket*/
                     String message = br.readLine();
                     System.out.println("Message received from the server : " +
message); }
              catch (Exception exception) { exception.printStackTrace();
              }
              finally {
              /*Closing the socket*/
```

```
                        try {
                               socket.close();
                        }
                        catch (Exception e) {
                               e.printStackTrace();
                        }
                }
        }
}
```

Output - server.java

```
$ java server.java
Server Started and listening to the port 25000
Message received from client is Saurav
Message sent to the client is Hello Saurav !!
```

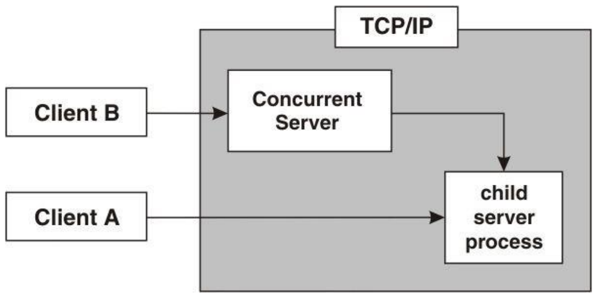
Output - client.java

```
$ java client.java
Message sent to the server : Saurav

Message received from the server : Hello Saurav !!
```

2. Write an echo program with client and concurrent server using TCP.

A concurrent server -



Server side - server.java

```
import java.net.*; import java.io.*; import
java.lang.*; public class Conserver { public
static void main(String args[]) { try{
                    ServerSocket serversock = new ServerSocket(6666);
                    while(true) {
                          Socket socket = serversock.accept();
```

```
                          MyClass obj = new MyClass(socket); obj.start();
                    }
              }
              catch(IOException e) {} }
}
class MyClass extends Thread {
       Socket socket;
       public MyClass(Socket s) {
socket = s;
       } public void
       run() { try {
                    DataInputStream dis = new
DataInputStream(socket.getInputStream());
                    String str; do
              {
                    Str = (String)dis.readUTF();
                    System.out.println("message = "+str);
              }
              while(!str.equals("bye")); socket.close();
              }
              catch(IOException e) {}
       }
}
```

Client side - client.java

```
import java.io.*; import java.net.*; import
java.util.Scanner; public class MyClient1 {
public static void main(String[] args) { try {
                    Scanner s1 = new Scanner(System.in);
                    String msg = new String();
                    Socket s = new Socket("localhost",6666);
                    DataOutputStream dout = new
DataOutputStream(s.getOutputStream()); do {
                    msg = s1.nextLine();
                    dout.writeUTF(msg);
                    dout.flush();
                    }
                    while(!(msg.equals("bye")));
                    dout.close();
                    s.close();
             }
             catch(Exception e) {
                    System.out.println(e); }
```

```
       }
}
```

Output - server.java

```
$ java server.java message =
hello from client1 message =
hello from client2
message = bye message
= bye
```

Output - client1

```
$ java client.java
hello from client1 bye
```

Output - client

```
$ java client.java
hello from client2 bye
```

3. Write an echo program with client and concurrent server using UDP.

Server side - server.java

```
/* udp server */ import java.io.*; import
java.net.*; public class Server { public static
final int DEFAULT_PORT = 13;
	public static void main (String[] args) throws IOException {
		DatagramSocket socket = new DatagramSocket(DEFAULT_PORT);
		DatagramPacket packet = new DatagramPacket(new byte[1], 1);
		while (true) { socket.receive
			(packet);
			System.out.println
			("Received from: " + packet.getAddress() + ":" +
packet.getPort()); byte[] outBuffer = new
java.util.Date().toString().getBytes(); packet.setData(outBuffer);
			packet.setLength(outBuffer.length);
			socket.send(packet);
		}
	}
}
```

Client side - client.java

```
/* udpclient */ import java.util.Scanner; import java.io.*;
import java.net.*; public class Client { public static void
main(String[] args) throws IOException {
		Scanner sc = new Scanner(System.in);
		int port = 13;
		InetAddress host = InetAddress.getByName("localhost");
		DatagramSocket socket = new DatagramSocket();
		socket.setSoTimeout (5000);
		//DatagramPacket(byte[], int, InetAddress, int)
		DatagramPacket packet = new DatagramPacket(new byte[256], 1,
host, port); byte[] outBuffer = sc.nextLine().getBytes();
		packet.setData(outBuffer);
		packet.setLength(outBuffer.length);
		socket.send(packet); socket.receive(packet);
		socket.close();
	}
}
```

Output - server.java

```
$ sudo java server.java
Received from: /127.0.0.1:52391
Received from: /127.0.0.1:46293
```

Output - client1

```
$ java client.java hello
from client1
```

Output - client2

```
$ java client.java hello
from client2
```

4. Write a client and server program for chatting.

Server side - server.java

```
/* chat server */ import
java.io.*;
import java.net.ServerSocket;
import java.net.Socket; import
java.net.SocketException;
```

```
public class ChatSocketServer {
       private ServerSocket severSocket = null; private
       Socket socket = null;
                           private InputStream inStream = null;
       private OutputStream outStream = null;
       public void createSocket(){
              try {
                     ServerSocket serverSocket = new ServerSocket(3339);
                     while(true){
                            socket = serverSocket.accept();
                            inStream = socket.getInputStream();
                            outStream = socket.getOutputStream();
                            System.out.println("Connected");
                            createReadThread();
                            createWriteThread();
                     }
                     }
                            catch (IOException io){
                                   io.printStackTrace();
                     }
              }
                            public void createReadThread(){
                                   Thread readThread = new Thread(){
                     public void run(){
                            while(socket.isConnected()){
                                   try {
                                          byte[] readBuffer = new byte[200];
                                          int num = inStream.read(readBuffer);
                                          if (num > 0) {
                                                 byte[] arrayBytes = new
byte[num];
                                                 System.arraycopy(readBuffer, 0,
arrayBytes, 0, num);
                                                 String recvedMessage = new
String(arrayBytes, "UTF-8");
                                                 System.out.println("Client :" +
recvedMessage);
                                                 }
                                          else{
                                                 notify();
                                          };
                                   //System.arraycopy();
                                   }
                                   catch (SocketException se){
                                          System.exit(0);
                                   }
                                   catch (IOException i){
                                          i.printStackTrace();
                                   }
                            }
                     }
                     };
              readThread.setPriority(Thread.MAX_PRIORITY);
```

```
		readThread.start();
	}
	public void createWriteThread(){
		Thread writeThread = new Thread(){
			public void run() {
				while(socket.isConnected()){
					try{
						BufferedReader inputReader = new
BufferedReader(new
							InputStreamReader(System.in));
						sleep(100);
						String typedMessage =
inputReader.readLine();
						if (typedMessage != null &&
typedMessage.length() > 0){
							synchronized (socket){

outStream.write(typedMessage.getBytes("UTF-8"));
								sleep(100);
							}
						};
					}
					catch (IOException i){
						i.printStackTrace();
					}
					catch(InterruptedException ie){
						ie.printStackTrace();
					}
				}
			}
		};
		writeThread.setPriority(Thread.MAX_PRIORITY);
		writeThread.start();
	}
	public static void main(String[] args){
		ChatSocketServer chatServer = new ChatSocketServer();
		chatServer.createSocket();
	}
}
```

Client side - client.java

```
/* chat client */ import java.io.*; import
java.net.Socket; import
java.net.SocketException; import
java.net.UnknownHostException; public class
ChatSocketClient{ private Socket socket =
null; private InputStream inStream = null;
private OutputStream outStream = null;
public void createSocket(){
             try {
                    socket = new Socket("localHost", 3339); //Binding
                    System.out.println("Connected"); inStream =
                    socket.getInputStream(); outStream =
                    socket.getOutputStream();
                    createReadThread(); //To read createWriteThread();
                    //To write
             }
             catch (UnknownHostException u){
                    u.printStackTrace(); }
             catch (IOException io){ io.printStackTrace();
             }
      }
      public void createReadThread(){
             Thread readThread = new Thread(){ public
                    void run(){
                    while(socket.isConnected()){ try {
                                        byte[] readBuffer = new byte[200]; int
                                        num = inStream.read(readBuffer);
                                        if (num > 0) { byte[]
                                               arrayBytes = new
byte[num];
                                               System.arraycopy(readBuffer, 0,
arrayBytes, 0, num);
                                               String recvedMessage = new
String(arrayBytes, "UTF-8");
                                               System.out.println("Server :" +
recvedMessage);
                                        };
                                 }
                                 catch (SocketException se){ System.exit(0);
                                 }
                                 catch (IOException i){
                                        i.printStackTrace();
                                 }
                          }
                    }
             };
             readThread.setPriority(Thread.MAX_PRIORITY);
             readThread.start();
      }
      public void createWriteThread(){
             Thread writeThread = new Thread(){ public
                    void run(){
                          while(socket.isConnected()){ try{
                                        BufferedReader inputReader = new
BufferedReader(new
                                        InputStreamReader(System.in));
                                        sleep(100);
                                        String typedMessage =
inputReader.readLine(); if (typedMessage != null &&
typedMessage.length() > 0){ synchronized (socket){

outStream.write(typedMessage.getBytes("UTF-8")); sleep(100);
                                               }
                                        };
                                 }
                                 catch(IOException i){
                                        i.printStackTrace();
                                 }
                                 catch(InterruptedException ie){
                                        ie.printStackTrace();
                                 }
                          }
                   }
             };
             writeThread.setPriority(Thread.MAX_PRIORITY);
             writeThread.start();
      }
      public static void main(String[] args)throws Exception{
             ChatSocketClient myChatClient = new ChatSocketClient();
      myChatClient.createSocket(); }
}
```

Output - server.java

```
$ java server.java
Connected Client
:hi hello
Client :how are you? i'm
fine. what about you?
Client :i’m also fine
```

Output - client.java

```
$ java client.java
Connected hi
Server :hello how
are you?
Server :I'm fine. what about you?
i’m also fine
```

5. Write a program to retrieve date and time using TCP.

Server side - server.java

```
/** day server TCP **/
import java.util.*;
import java.net.*; import
java.io.*; public class
IterServer {
	public static void main(String[] args) { try{
			ServerSocket ss=new ServerSocket(6432);
			Socket s=ss.accept();
			Date dt= new java.util.Date();
			DataOutputStream dout=new
DataOutputStream(s.getOutputStream()); String
			str; str=dt.toString();
			dout.writeUTF(str);
			ss.close();
		}
		catch(Exception e){
			System.out.println(e);
		}
	}
}
```

Client side - client.java

```
/** Tcp day client **/ import java.net.*; import java.io.*;
public class Tcpdayclient { public static void main(String
arg[])throws IOException{ try{
			Socket s=new Socket("localhost",6432);
			DataInputStream dis=new
DataInputStream(s.getInputStream()); String
			msg;
			msg= dis.readUTF();
			System.out.println(msg); dis.close();
			s.close(); }
		catch(Exception e){
			System.out.println(e);
		}
	}
}
```

Output - server.java

```
$ java server.java
```

Output - client.java

```
$ java client.java
Thu Oct 08 07:08:02 UTC 2020
```

6. Write a program to retrieve date and time using UDP.

Server side - server.java

```
/* udp day server */ import java.io.*; import
java.net.*; public class DaytimeServer { public
static final int DEFAULT_PORT = 13;
        public static void main (String[] args) throws IOException {
                DatagramSocket socket = new DatagramSocket (DEFAULT_PORT );
                DatagramPacket packet = new DatagramPacket (new byte[1], 1);
                while (true) { socket.receive
                        (packet);
                        System.out.println
                        ("Received from: " + packet.getAddress () + ":" +
packet.getPort ()); byte[] outBuffer = new java.util.Date
                        ().toString
().getBytes(); packet.setData (outBuffer);
                        packet.setLength (outBuffer.length);
                        socket.send (packet);
                }
        }
}
```

Client side - client.java

```
/* udp day client */ import java.io.*; import java.net.*; public
class DaytimeClient { public static void main (String[] args)
throws IOException { int port = 13;
                InetAddress host = InetAddress.getByName ("localhost");
                DatagramSocket socket = new DatagramSocket ();
                socket.setSoTimeout (5000);
                //DatagramPacket(byte[], int, InetAddress, int)
                DatagramPacket packet = new DatagramPacket (new byte[256], 1,
host, port);
                socket.send (packet);
                packet.setLength (packet.getData ().length);
                socket.receive (packet);
```

```
              socket.close ();
              byte[] data = packet.getData (); int
       length = packet.getLength ();
       System.out.println (new String (data)); }
}
```

Output - server.java

```
$ sudo java server.java
Received from: /127.0.0.1:36208
```

Output - client.java

```
$ java client.java
Thu Oct 08 07:20:13 UTC 2020
```

7. Write a client and server routines showing Blocking I/O.

Server side - server.java

```
package crunchify.com.tutorials; import
java.io.IOException; import
java.net.InetSocketAddress; import
java.nio.ByteBuffer;
import java.nio.channels.SelectionKey; import
java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel; import
java.nio.channels.SocketChannel; import
java.util.Iterator; import java.util.Set;
/**
*      @author
Crunchify.com
*      */
public class CrunchifyNIOServer {
       @SuppressWarnings("unused")
       public static void main(String[] args) throws IOException {
              // Selector: multiplexor of SelectableChannel objects
              Selector selector = Selector.open(); // selector is open here //
              ServerSocketChannel: selectable channel for stream-oriented
listening sockets
              ServerSocketChannel crunchifySocket =
ServerSocketChannel.open();
              InetSocketAddress crunchifyAddr = new InetSocketAddress("localhost",
1111);
                        // Binds the channel's socket to a local address and configures
the socket to listen for connections
              crunchifySocket.bind(crunchifyAddr); // Adjusts
              this channel's blocking mode.
```

```
             crunchifySocket.configureBlocking(false); int
             ops = crunchifySocket.validOps();
             SelectionKey selectKy = crunchifySocket.register(selector, ops,
null);
             // Infinite loop.. //
             Keep server running
             while (true) { log("i'm a server and i'm waiting for new
                    connection and
buffer select...");
                    // Selects a set of keys whose corresponding channels are
ready for I/O operations selector.select();
                    // token representing the registration of a
SelectableChannel with a Selector
                    Set<SelectionKey> crunchifyKeys = selector.selectedKeys();
                    Iterator<SelectionKey> crunchifyIterator =
crunchifyKeys.iterator(); while
                    (crunchifyIterator.hasNext()) {
                           SelectionKey myKey = crunchifyIterator.next();
                           // Tests whether this key's channel is ready to
accept a new socket connection if
                           (myKey.isAcceptable()) {
                                  SocketChannel crunchifyClient =
crunchifySocket.accept();
                                  // Adjusts this channel's blocking mode to
false crunchifyClient.configureBlocking(false); // Operation-set bit for
                                  read operations
                                  crunchifyClient.register(selector,
SelectionKey.OP_READ);
                                  log("Connection Accepted: " +
crunchifyClient.getLocalAddress() + "\n");
                                  // Tests whether this key's channel is ready
for reading
                           }
                           else if (myKey.isReadable()) {
                           SocketChannel crunchifyClient = (SocketChannel)
                           myKey.channel();
                           ByteBuffer crunchifyBuffer =
ByteBuffer.allocate(256); crunchifyClient.read(crunchifyBuffer);
                           String result = new
                           String(crunchifyBuffer.array()).trim();
                           log("Message received: " + result); if
                           (result.equals("Crunchify")) {
                                         crunchifyClient.close();
                                         log("\nIt's time to close connection
as we got last company name 'Crunchify'"); log("\nServer will keep running.
                                         Try
running client again to establish new connection");
                                  }
                           }
                           crunchifyIterator.remove();
                    }
            }
      }
      private static void log(String str) {
      System.out.println(str); }
}
```

Client side - client.java

```
package crunchify.com.tutorials; import
java.io.IOException; import
java.net.InetSocketAddress; import
java.nio.ByteBuffer;
import java.nio.channels.SocketChannel; import
java.util.ArrayList;
/**
*      @author
Crunchify.com
*      */
public class CrunchifyNIOClient { public static void main(String[] args)
       throws IOException,
InterruptedException {
             InetSocketAddress crunchifyAddr = new InetSocketAddress("localhost",
1111);
             SocketChannel crunchifyClient =
SocketChannel.open(crunchifyAddr); log("Connecting to Server on
             port 1111...");
             ArrayList<String> companyDetails = new ArrayList<String>();
             // create a ArrayList with companyName list
             companyDetails.add("Facebook"); companyDetails.add("Twitter");
             companyDetails.add("IBM"); companyDetails.add("Google");
             companyDetails.add("Crunchify");
             for (String companyName : companyDetails) { byte[] message = new
                    String(companyName).getBytes();
                    ByteBuffer buffer = ByteBuffer.wrap(message);
                    crunchifyClient.write(buffer); log("sending: " +
                    companyName);
                    buffer.clear();
                    // wait for 2 seconds before sending next message
                    Thread.sleep(2000);
             }
             crunchifyClient.close();
      }
      private static void log(String str) {
      System.out.println(str); }
}
```

Output - server.java

```
$ java server.java
i'm a server and i'm waiting for new connection and buffer select...
Connection Accepted: /127.0.0.1:1111

i'm a server and i'm waiting for new connection and buffer select... Message
received: Facebook
i'm a server and i'm waiting for new connection and buffer select... Message
received: Twitter
i'm a server and i'm waiting for new connection and buffer select... Message
received: IBM
i'm a server and i'm waiting for new connection and buffer select... Message
received: Google
i'm a server and i'm waiting for new connection and buffer select... Message
received: Crunchify

It's time to close connection as we got last company name 'Crunchify'
Server will keep running. Try running client again to establish new connection
i'm a server and i'm waiting for new connection and buffer select...
```

Output - client.java

```
$ java client.java
Connecting to Server on port 1111...
sending: Facebook sending: Twitter
sending: IBM sending: Google
sending: Crunchify
```

8. Write a client and server routines showing I/O multiplexing.

Server side - server.c

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/select.h>
#include<unistd.h>
#define MAXLINE 20 #define SERV_PORT 7134
main(int argc,char **argv){ int
i,j,maxi,maxfd,listenfd,connfd,sockfd;
       int nread,client[FD_SETSIZE];
       ssize_t n; fd_set
       rset,allset; char
       line[MAXLINE];
```

```
      socklen_t clilen;
      struct sockaddr_in cliaddr,servaddr;
      listenfd=socket(AF_INET,SOCK_STREAM,0);
      bzero(&servaddr,sizeof(servaddr)); servaddr.sin_family=AF_INET;
      servaddr.sin_port=htons(SERV_PORT);
      bind(listenfd,(struct sockaddr *)&servaddr,sizeof(servaddr));
      listen(listenfd,1); maxfd=listenfd; maxi=-1;
      for(i=0;i<FD_SETSIZE;i++)
            client[i]=-1; FD_ZERO(&allset);
            FD_SET(listenfd,&allset);
      for(; ;){ rset=allset;
            nread=select(maxfd+1,&rset,NULL,NULL,NULL);
            if(FD_ISSET(listenfd,&rset)){ clilen=sizeof(cliaddr);
                  connfd=accept(listenfd,(struct
sockaddr*)&cliaddr,&clilen); for(i=0;i<FD_SETSIZE;i++)
                  if(client[i]<0) { client[i]=connfd;
                  break;
                        }
                        if(i==FD_SETSIZE){ printf("Too
                              many clients"); exit(0);
                        }
                        FD_SET(connfd,&allset); if(connfd>maxfd)
                              maxfd=connfd;
                        if(i>maxi) maxi=i;
                        if(--nread<=0)
                        continue;
            }
            for(i=0;i<=maxi;i++){ if((sockfd=client[i])<0)
                  continue; if(FD_ISSET(sockfd,&rset)){
                  if((n=read(sockfd,line,MAXLINE))==0){
                              close(sockfd);
                              FD_CLR(sockfd,&allset); client[i]=-1;
                        } else{ printf("Line recieved from the
                        client
:%s\n",line);
                              for(j=0;line[j]!='\0';j++)
                                    line[j]=toupper(line[j]);
                                    write(sockfd,line,MAXLINE);
                        }
                        if(--nread<=0)
```

```
                                  break;

                     }
              }
       }
}
```

Client side - client.c

```


#include<netinet/in.h>
#include<sys/types.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/select.h>
#include<unistd.h>
#define MAXLINE 20 #define
SERV_PORT 7134 main(int
argc,char **argv){
       int maxfdp1; fd_set
       rset;
       char sendline[MAXLINE],recvline[MAXLINE]; int
       sockfd;
       struct sockaddr_in servaddr; if(argc!=2){
       printf("usage tcpcli <ipaddress>");
              return;
       }
       sockfd=socket(AF_INET,SOCK_STREAM,0);
       bzero(&servaddr,sizeof(servaddr)); servaddr.sin_family=AF_INET;
       servaddr.sin_port=htons(SERV_PORT);
       inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
       connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
       printf("\n Enter data to be send : ");
       while(fgets(sendline,MAXLINE,stdin)!=NULL){
              write(sockfd,sendline,MAXLINE);
              printf("\n Line send to server is : %s",sendline);
              read(sockfd,recvline,MAXLINE);
              printf("Line recieved from the server : %s",recvline);
       } exit(0);
}
```

Output - server.c

```
$ gcc server.c -o server
$ ./server
Line recieved from the client :hi


Line recieved from the client :hello Line recieved from the client :i'm saurav
```

Output - client.c

```
$ gcc client.c -o client
$ ./client 2

 Enter data to be send : hi

 Line send to server is : hi Line
recieved from the server : HI
hello

 Line send to server is : hello Line
recieved from the server : HELLO i'm
saurav

 Line send to server is : i'm saurav
Line recieved from the server : I'M SAURAV
```

9. **Write an echo client and server program using Unix domain stream socket.**

Server side - server.c

```
//unix tcp server
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
//unix tcp server int
main()
{ int sock, connected, bytes_recieved , true = 1;
       char send_data [1024]={"Hello User!!"} , recv_data[1024];
       struct sockaddr_in server_addr,client_addr; int sin_size;
       if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              perror("Socket"); exit(1);
       }
       if (setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&true,sizeof(int)) == -1) {
              perror("Setsockopt"); exit(1);
       }
       server_addr.sin_family = AF_INET; server_addr.sin_port
       = htons(5000); server_addr.sin_addr.s_addr =
       INADDR_ANY;
       bzero(&(server_addr.sin_zero),8);
       if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr)) == -1) { perror("Unable to bind"); exit(1);
       }
       if (listen(sock, 5) == -1) {
              perror("Listen"); exit(1);
       }
       printf("\nTCPServer Waiting for client on port 5000"); fflush(stdout);
       sin_size = sizeof(struct sockaddr_in);
       connected = accept(sock, (struct sockaddr *)&client_addr,&sin_size);
       printf("\n I got a connection from (%s , %d)\n",
       inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
       send(connected, send_data,strlen(send_data), 0);
       close(connected); close(sock); return 0;
}
```

Client side - client.c

```
//unix tcp client
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h> int
main()
{
       int sock, bytes_recieved;
       char send_data[1024],recv_data[1024];
       struct hostent *host;
       struct sockaddr_in server_addr;
       host = gethostbyname("127.0.0.1");
       if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              perror("Socket");
              exit(1);
       }
       server_addr.sin_family = AF_INET;
       server_addr.sin_port = htons(5000);
       server_addr.sin_addr = *((struct in_addr *)host->h_addr);
       bzero(&(server_addr.sin_zero),8);
       if (connect(sock, (struct sockaddr *)&server_addr,sizeof(struct
sockaddr)) == -1){
              perror("Connect");
              exit(1);
       }
       bytes_recieved=recv(sock,recv_data,1024,0);
       recv_data[bytes_recieved] = '\0';
       printf("\nReceived data = %s\n " , recv_data);
       //send(sock,send_data,strlen(send_data), 0);
       close(sock);
       return 0;
}
```

Output - server.c

```
$ gcc server.c -o server
$ ./server


TCPServer Waiting for client on port 5000
 I got a connection from (127.0.0.1 , 39112)
```

Output - client.c

```
$ gcc client.c -o client
$ ./client
```

```
Received data = Hello User!!
```

10. Write an echo client and server program using Unix domain Datagram socket.

Server side - server.c

```
// unix udp server
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h> int
main()
{ int sock;
       int addr_len, bytes_read; char
       recv_data[1024];
       struct sockaddr_in server_addr , client_addr; if
       ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
```

```
              perror("Socket"); exit(1);
       }
       server_addr.sin_family = AF_INET; server_addr.sin_port
       = htons(5000); server_addr.sin_addr.s_addr =
       INADDR_ANY;
       bzero(&(server_addr.sin_zero),8);
       if (bind(sock,(struct sockaddr *)&server_addr, sizeof(struct sockaddr))
== -1){ perror("Bind");
              exit(1);
       }
       addr_len = sizeof(struct sockaddr);
       printf("\nUDPServer Waiting for client on port 5000"); fflush(stdout);
       bytes_read = recvfrom(sock,recv_data,1024,0,(struct sockaddr
*)&client_addr, &addr_len); recv_data[bytes_read]
       = '\0'; printf("\n(%s , %d) said :
",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
       printf("%s\n", recv_data); fflush(stdout); return 0;
}
```

Client side - client.c

```
// unix udp client
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h> int
main()
{ int sock;
       struct sockaddr_in server_addr;
       struct hostent *host; char
       send_data[1024];
       host= (struct hostent *) gethostbyname((char *)"127.0.0.1");
       if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
              perror("socket"); exit(1);
       }
       server_addr.sin_family = AF_INET; server_addr.sin_port
       = htons(5000);
       server_addr.sin_addr = *((struct in_addr *)host->h_addr);
       bzero(&(server_addr.sin_zero),8);
```

```
       printf("Type Something :"); gets(send_data);
       sendto(sock, send_data, strlen(send_data), 0,(struct sockaddr
*)&server_addr, sizeof(struct sockaddr)); }
```

Output - server.c

```
$ gcc server.c -o server
$ ./server

UDPServer Waiting for client on port 5000
(127.0.0.1 , 48905) said : hi
```

Output - client.c

```
$ gcc client.c -o client
$ ./client
Type Something :hi
```

11. Write a client and server program to implement file transfer.

Server side - server.java

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException; import
java.io.OutputStream; import
java.net.ServerSocket; import
java.net.Socket;
public class TCPFileTransferServer { public static void
      main(String[] args) throws IOException {
             ServerSocket servsock = new ServerSocket(12345);
             File myFile = new File("./Study.txt"); while
             (true) {
                    System.out.println("Waiting for client"); Socket
                    sock = servsock.accept();
                  byte[] mybytearray = new byte[(int) myFile.length()];
                  BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(myFile)); bis.read(mybytearray, 0,
                    mybytearray.length);
                    OutputStream os = sock.getOutputStream();
                    System.out.println("Sending file data to client");
                    os.write(mybytearray, 0, mybytearray.length);
                    System.out.println("Sent file successfully to client");
                    os.flush(); sock.close();
             }
      }
}
```

Client side - client.java

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream; import
java.net.Socket;
public class TCPFileTransferClient { public static void
      main(String[] argv) throws Exception {
             Socket sock = new Socket("127.0.0.1", 12345);
             System.out.println("Connected to server"); byte[]
             mybytearray = new byte[1024];
             InputStream is = sock.getInputStream();
             FileOutputStream fos = new FileOutputStream("./Readtext.txt");
             BufferedOutputStream bos = new BufferedOutputStream(fos);
             int bytesRead;
             System.out.println("Receiving file ");
             while ((bytesRead = is.read(mybytearray)) != -1) {
                    bos.write(mybytearray, 0, bytesRead);
             }
             System.out.println("File Copied Successfully");
             bos.close(); sock.close();
      }
}
```

Output - server.java

```
$ java server.java
Waiting for client
Sending file data to client
Sent file successfully to client
Waiting for client
```

Output - client.java

```
$ java client.java
Connected to server
Receiving file
File Copied Successfully
```

12. Write a client and server program to implement the remote command execution

Server side - server.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File; import
java.io.IOException; import
java.io.InputStream; import
java.io.InputStreamReader;
```

```
import java.io.OutputStream; import
java.io.OutputStreamWriter; import
java.net.ServerSocket; import
java.net.Socket;
public class TCPRemoteCommandServer {
	private static Socket socket;
	public static void main(String[] args) {
		try { int port = 25000;
			ServerSocket serverSocket = new ServerSocket(port);
			System.out.println("Server Started and listening to the
port 25000");
			//Server is running always. This is done using this
while(true) loop while (true) {
				//Reading the message from the client socket
				= serverSocket.accept();
				InputStream is = socket.getInputStream();
				InputStreamReader isr = new InputStreamReader(is);
				BufferedReader br = new BufferedReader(isr);
				String command = br.readLine();
				System.out.println("Command received from client is
: & " + command);
				String returnMessage;
				TCPRemoteCommandServer ss = new
TCPRemoteCommandServer(); returnMessage = ss.executeCommand(command);
				System.out.println("Command output : $ " +
returnMessage);
				//Sending the response back to the client.
				OutputStream os = socket.getOutputStream();
				OutputStreamWriter osw = new
OutputStreamWriter(os);
				BufferedWriter bw = new BufferedWriter(osw);
				bw.write(returnMessage); bw.flush();
			} }
		catch (Exception e) {
			e.printStackTrace();
		}  finally {
			try { socket.close();
			}
			catch (Exception e) {
			}
		}
	}
	private String executeCommand(String command) {
		StringBuffer output = new StringBuffer(); try
		{
			Process p = Runtime.getRuntime().exec(command);
```

```
                    p.waitFor();
                    BufferedReader reader = new BufferedReader( new
                    InputStreamReader(p.getInputStream())
                    );
                    String line;
                    while ((line = reader.readLine()) != null) {
                          output.append(line).append("\n");
                    } }
              catch (IOException e1) {}  catch
       (InterruptedException e2) {} return
       output.toString(); }
}
```

Client side - client.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStream; import
java.io.InputStreamReader; import
java.io.OutputStream; import
java.io.OutputStreamWriter; import
java.net.InetAddress; import
java.net.Socket;
public class TCPRemoteCommandClient {
	private static Socket socket;
	public static void main(String args[]) { try
		{
			String host = "localhost"; int
			port = 25000;
			InetAddress address = InetAddress.getByName(host);
			socket = new Socket(address, port);
			//Send the message to the server
			OutputStream os = socket.getOutputStream();
			OutputStreamWriter osw = new OutputStreamWriter(os);
			BufferedWriter bw = new BufferedWriter(osw);
			String cmd = "ls";
			String sendMessage = cmd + "\n"; bw.write(sendMessage);
			bw.flush();
			System.out.println("Command sent to the server : " +
sendMessage);
			//Get the return message from the server
			InputStream is = socket.getInputStream();
			InputStreamReader isr = new InputStreamReader(is);
			BufferedReader br = new BufferedReader(isr);
			String line;
			//line = br.readLine();
			while ((line = br.readLine()) != null) {
				System.out.println(line);
			}
			System.out.println("Command output received from the
```

```
server :\n " + line);
		}
		catch (Exception exception) { exception.printStackTrace();
		}
		finally {
		//Closing the socket try {
			socket.close();
			}
			catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}
```

Output - server.java

```
$ java server.java
Server Started and listening to the port 25000
Command received from client is : $ ls
Command output : $ client.java server.java
```

Output - client.java

```
$ java client.java
Command sent to the server : ls

client.java server.java
Command output received from the server :
null
```

13. Write a client program that gets a number from the user and sends the number to the server for conversion into hexadecimal and gets the result from the server.

Server side - server.java

```
import java.io.*; import java.net.*; public
class Hex_server { public static void
main(String[] args){ try{
                    ServerSocket ss=new ServerSocket(6432);
                    Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());

                    String str=(String)dis.readUTF();
                    Integer n=Integer.parseInt(str);
                    String hex1=Integer.toHexString(n);
                    System.out.println("hexadecimal value of user input = "+
hex1); ss.close();
             }
             catch(Exception e){
                    System.out.println(e);
             }
      }
}
```

Client side - client.java

```
import java.io.*; import java.net.*; import
java.util.*; public class Hex_client { public
static void main(String[] args){ try{
                    Scanner w = new Scanner(System.in);
                    Socket s = new Socket("localhost",6432);
                    DataOutputStream dout=new
DataOutputStream(s.getOutputStream()); String
                    number;
                    number=w.nextLine();
                    dout.writeUTF(number);
                    dout.flush();
                    dout.close();
                    s.close(); }
             catch(Exception e){
                    System.out.println(e);
             }
       }
}
```

Output - server.java

```
$ java server.java hexadecimal value
of user input = 20 $ java
server.java hexadecimal value of
user input = 23
```

Output - client.java

```
$ java client.java
32
$ java client.java
35
```