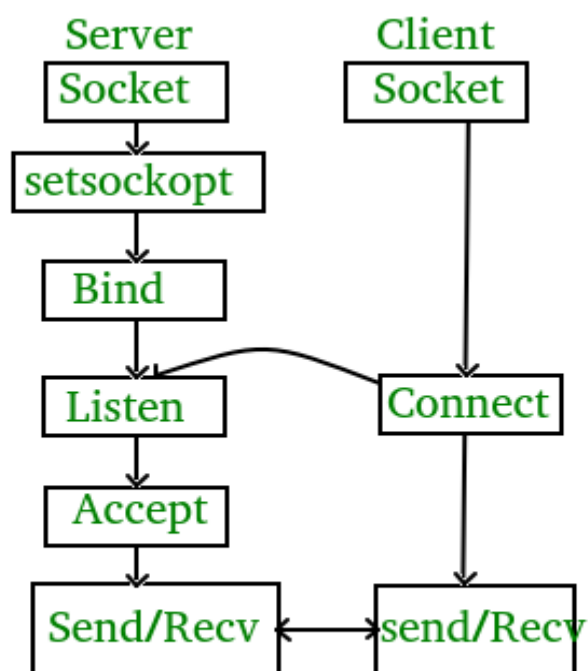


EXPERIMENT NO: 09

Aim: Socket programming using TCP or UDP

Theory: Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

State diagram for server and client model



Stages for server

- Socket creation:

```
int sockfd = socket(domain, type, protocol)
```

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

- Setsockopt:

```
int setsockopt(int sockfd, int level, int optname,  
               const void *optval, socklen_t optlen);
```

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

- Bind:

```
int bind(int sockfd, const struct sockaddr *addr,  
         socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

- Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

- Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t  
*addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

- Socket connection: Exactly same as that of server's socket creation
- Connect:

```
int connect(int sockfd, const struct sockaddr *addr,  
            socklen_t addrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Client side Program:

```
import java.net.*;  
import java.io.*;  
  
public class Client  
{  
    // 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(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();
        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);
}
}

public static void main(String args[])
{
    Client client = new Client("127.0.0.1", 5000);
}
}

```

Server side Program:

```

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

public class Server
{
    //initialize socket and input stream
    private Socket      socket      = null;
    private ServerSocket server      = null;
    private DataInputStream in        = null;

    // constructor with port

```

```

public Server(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);

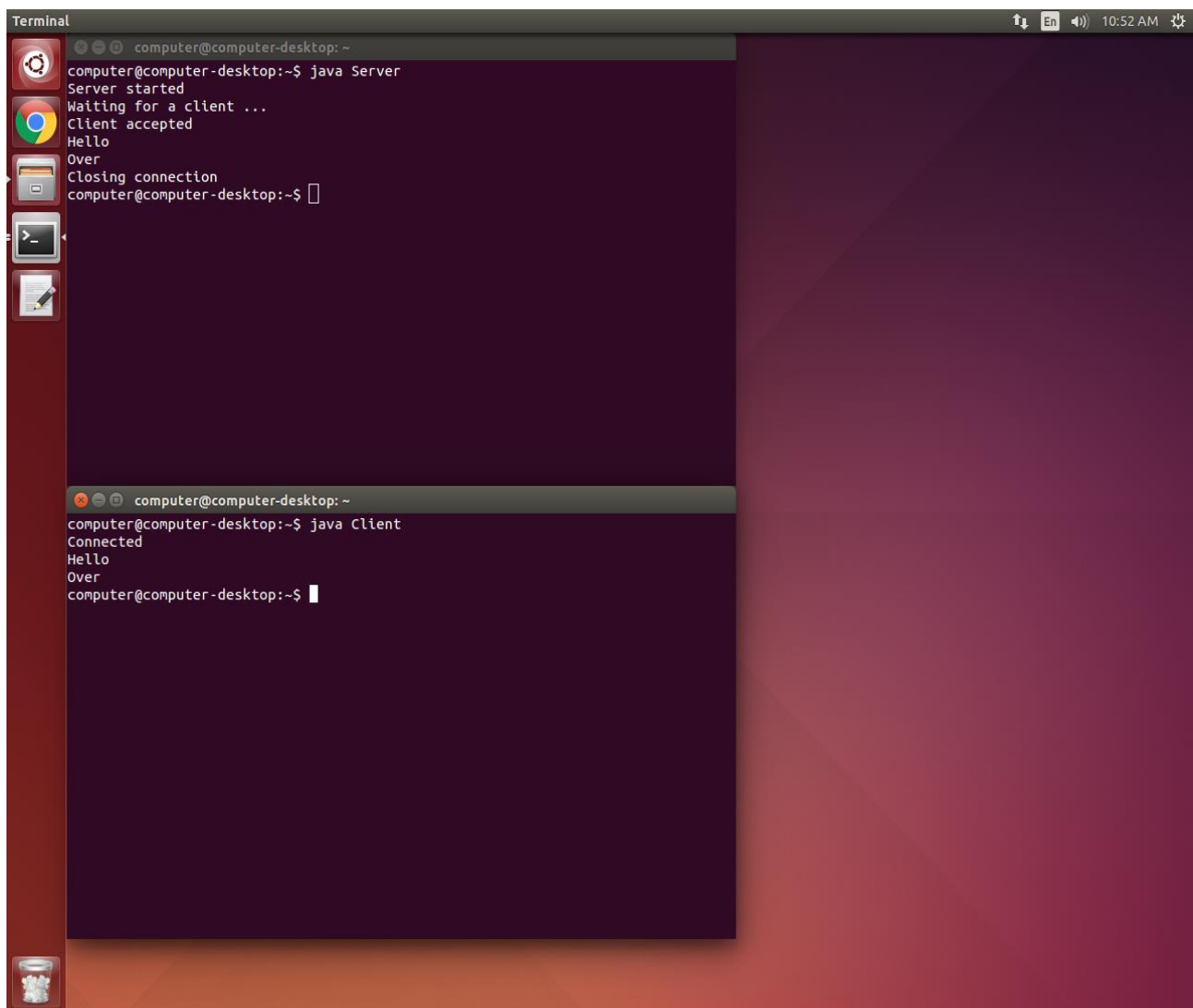
            }
            catch (IOException i)
            {
                System.out.println(i);
            }
        }
        System.out.println("Closing connection");

        // close connection
        socket.close();
        in.close();
    }
    catch (IOException i)
    {
        System.out.println(i);
    }
}

public static void main(String args[])
{
    Server server = new Server(5000);
} }

```

Output:



```
computer@computer-desktop: ~  
computer@computer-desktop:~$ java Server  
Server started  
Waiting for a client ...  
Client accepted  
Hello  
Over  
Closing connection  
computer@computer-desktop:~$  
  
computer@computer-desktop: ~  
computer@computer-desktop:~$ java Client  
Connected  
Hello  
Over  
computer@computer-desktop:~$
```