“Assignment#1”

Name: Ayesha Aamir **Student ID: 63687**

Q NO:1

Source Code:-

```
using System;
using System.Threading;
using System.Net.Sockets;
using System.Text;
using System.Collections;

namespace ConsoleApp1
{
    1 reference
    class Program
    {
        public static Hashtable clientsList = new Hashtable();
        0 references
        static void Main(string[] args)
        {
            TcpListener serverSocket = new TcpListener(port:8888);
            TcpClient clientSocket = default(TcpClient);
            int counter = 0;

            serverSocket.Start();
            Console.WriteLine("Chat Server Started ....");
            counter = 0;
            while ((true))
            {
                counter += 1;
                clientSocket = serverSocket.AcceptTcpClient();

                byte[] bytesFrom = new byte[10025];
                string dataFromClient = null;

                NetworkStream networkStream = clientSocket.GetStream();
                networkStream.Read(bytesFrom, offset:0, (int)clientSocket.ReceiveBufferSize);
                dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                dataFromClient = dataFromClient.Substring(startIndex:0, length:dataFromClient.IndexOf("$"));

                clientsList.Add(dataFromClient, clientSocket);

                broadcast(msg:dataFromClient + " Joined ", uName:dataFromClient, flag:false);

                Console.WriteLine(dataFromClient + " Joined chat room ");
                handleClinet client = new handleClinet();
```

```
            handleClinet client = new handleClinet();
            client.startClient(clientSocket, dataFromClient, clientsList);
        }

        clientSocket.Close();
        serverSocket.Stop();
        Console.WriteLine("exit");
        Console.ReadLine();
    }

    2 references
    public static void broadcast(string msg, string uName, bool flag)
    {
        foreach (DictionaryEntry Item in clientsList)
        {
            TcpClient broadcastSocket;
            broadcastSocket = (TcpClient)Item.Value;
            NetworkStream broadcastStream = broadcastSocket.GetStream();
            Byte[] broadcastBytes = null;

            if (flag == true)
            {
                broadcastBytes = Encoding.ASCII.GetBytes(s:uName + " says : " + msg);
            }
            else
            {
                broadcastBytes = Encoding.ASCII.GetBytes(msg);
            }

            broadcastStream.Write(broadcastBytes, offset:0, size:broadcastBytes.Length);
            broadcastStream.Flush();
        }
    }
}
```

```
2 references
public class handleClinet
{
    TcpClient clientSocket;
    string clNo;
    Hashtable clientsList;

    1 reference
    public void startClient(TcpClient inClientSocket, string clineNo, Hashtable cList)
    {
        this.clientSocket = inClientSocket;
        this.clNo = clineNo;
        this.clientsList = cList;
        Thread ctThread = new Thread(doChat);
        ctThread.Start();
    }

    1 reference
    private void doChat()
    {
        int requestCount = 0;
        byte[] bytesFrom = new byte[10025];
        string dataFromClient = null;
        Byte[] sendBytes = null;
        string serverResponse = null;
        string rCount = null;
        requestCount = 0;

        while ((true))
        {
            try
            {
                requestCount = requestCount + 1;
                NetworkStream networkStream = clientSocket.GetStream();
```

```
        while ((true))
        {
            try
            {
                requestCount = requestCount + 1;
                NetworkStream networkStream = clientSocket.GetStream();
                networkStream.Read(bytesFrom, offset:0, (int)clientSocket.ReceiveBufferSize);
                dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                dataFromClient = dataFromClient.Substring(startIndex:0, length:dataFromClient.IndexOf("$"));
                Console.WriteLine("From client - " + clNo + " : " + dataFromClient);
                rCount = Convert.ToString(requestCount);

                Program.broadcast(msg:dataFromClient, uName:clNo, flag:true);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
        }
    }
}
```

Output:-

```
C:\Users\Ayesha\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe
Chat Server Started ....
Ayesha Joined chat room
Aamir Joined chat room
From client - Ayesha : Hello have you done NP assignment?
From client - Aamir : I am doing NP assignment?
```

Q NO:2

Introduction:-

This application is in C# where users can have group chats or private conversations. Simultaneously by using the concept Multithreading. This application involves two core components, they are Server and Client. The server is multithreaded, as it needs to handle multiple clients (up to 30).

Source Code:-

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(mainForm: new Server());
}
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Serialization;

namespace Server
{
    3 references
    public partial class Server : Form
    {
        TcpListener listener = new TcpListener(localaddr:IPAddress.Parse("127.0.0.1"), port:5000);
        TcpClient client;
        String clNo;
        Dictionary<string, TcpClient> clientList = new Dictionary<string, TcpClient>();
        CancellationTokenSource cancellation = new CancellationTokenSource();
        List<string> chat = new List<string>();

        1 reference
        public Server()
        {
            InitializeComponent();
        }

        1 reference
        private void btnStart_Click(object sender, EventArgs e)
        {
            cancellation = new CancellationTokenSource(); //resets the token when the server restarts
            startServer();
        }

        5 references
        public void updateUI(String m)
        {
```

```
{
    this.Invoke((MethodInvoker)delegate
    {
        textBox1.AppendText(">>" + m + Environment.NewLine);
    });
}

1 reference
public async void startServer()
{
    listener.Start();
    updateUI(m:"Server Started at " + listener.LocalEndpoint);
    updateUI(m:"Waiting for Clients");
    try
    {
        int counter = 0;
        while (true)
        {
            counter++;

            client = await Task.Run(function:() => listener.AcceptTcpClientAsync(), cancellation.Token);


            byte[] name = new byte[50];
            NetworkStream stre = client.GetStream();
            stre.Read(name, offset:0, size:name.Length);
            String username = Encoding.ASCII.GetString(name);
            username = username.Substring(startIndex:0, length:username.IndexOf("$"));


            clientList.Add(username, client);
            listBox1.Items.Add(username);
            updateUI(m:"Connected to user " + username + " - " + client.Client.RemoteEndPoint);
            announce(msg:username + " Joined ", uName:username, flag:false);

            await Task.Delay(1000).ContinueWith(t:Task => sendUsersList());


            var c = new Thread(start:() => ServerReceive(client, username));
            c.Start();
```

```
                c.Start();

            }
        }
        catch (Exception)
        {
            listener.Stop();
        }

    }

    3 references
    public void announce(string msg, string uName, bool flag)
    {
        try
        {
            foreach (var Item:KeyValuePair<string,TcpClient> in clientList)
            {
                TcpClient broadcastSocket;
                broadcastSocket = (TcpClient)Item.Value;
                NetworkStream broadcastStream = broadcastSocket.GetStream();
                Byte[] broadcastBytes = null;

                if (flag)
                {


                    chat.Add(item:"gChat");
                    chat.Add(item:uName + " says : " + msg);
                    broadcastBytes = ObjectToByteArray(chat);
                }
                else
                {


                    chat.Add(item:"gChat");
                    chat.Add(msg);
                    broadcastBytes = ObjectToByteArray(chat);

                }
```

```
                }

                broadcastStream.Write(broadcastBytes, offset:0, size:broadcastBytes.Length);
                broadcastStream.Flush();
                chat.Clear();
            }
        }
        catch (Exception er)
        {

        }
    }


    1 reference
    public Object ByteArrayToObject(byte[] arrBytes)
    {
        using (var memStream = new MemoryStream())
        {
            var binForm = new BinaryFormatter();
            memStream.Write(arrBytes, offset:0, count:arrBytes.Length);
            memStream.Seek(offset:0, loc:SeekOrigin.Begin);
            var obj:object = binForm.Deserialize(memStream);
            return obj;
        }
    }

    5 references
    public byte[] ObjectToByteArray(Object obj)
    {
        BinaryFormatter bf = new BinaryFormatter();
        using (var ms = new MemoryStream())
        {
            bf.Serialize(ms, obj);
            return ms.ToArray();
        }
    }
```

```
1 reference
public void ServerReceive(TcpClient clientn, String username)
{
    byte[] data = new byte[1000];
    String text = null;
    while (true)
    {
        try
        {
            NetworkStream stream = clientn.GetStream();
            stream.Read(data, offset:0, size:data.Length);
            List<string> parts = (List<string>)ByteArrayToObject(data);

            switch (parts[0])
            {
                case "gChat":
                    this.Invoke((MethodInvoker)delegate
                    {
                        textBox1.Text += username + ": " + parts[1] + Environment.NewLine;
                    });
                    announce(msg:parts[1], uName:username, flag:true);
                    break;

                case "pChat":
                    privateChat(parts);
                    break;
            }

            parts.Clear();
        }
        catch (Exception r)
        {
            updateUI(m:"Client Disconnected: " + username);
            announce(msg:"Client Disconnected: " + username + "$", uName:username, flag:false);
            clientList.Remove(username);
```

```
                clientList.Remove(username);

                this.Invoke((MethodInvoker)delegate
                {
                    listBox1.Items.Remove(username);
                });
                sendUsersList();
                break;
            }
        }
    }

    1 reference
    private void btnServerStop_Click(object sender, EventArgs e)
    {
        try
        {
            listener.Stop();
            updateUI(m: "Server Stopped");
            foreach (var Item :KeyValuePair<string,TcpClient> in clientList)
            {
                TcpClient broadcastSocket;
                broadcastSocket = (TcpClient)Item.Value;
                broadcastSocket.Close();
            }
        }
        catch (SocketException er)
        {

        }



    }

    1 reference
    private void Private_Click(object sender, EventArgs e)
    {
        if (listBox1.SelectedIndex != -1)
        {
            String clientName = listBox1.GetItemText(listBox1.SelectedItem);
```

```
        {
            String clientName = listBox1.GetItemText(listBox1.SelectedItem);

            chat.Clear();
            chat.Add(item: "gChat");
            chat.Add(item: "Admin : " + inputPrivate.Text);

            byte[] byData = ObjectToByteArray(chat);
            TcpClient workerSocket = null;
            workerSocket = (TcpClient)clientList.FirstOrDefault(x :KeyValuePair<string,TcpClient> => x.Key == clientName).Value;

            NetworkStream stm = workerSocket.GetStream();
            stm.Write(byData, offset: 0, size: byData.Length);
            stm.Flush();
            chat.Clear();

        }
    }

    1 reference
    private void disconnectToolStripMenuItem_Click(object sender, EventArgs e)
    {
        try
        {
            TcpClient workerSocket = null;

            String clientName = listBox1.GetItemText(listBox1.SelectedItem);
            workerSocket = (TcpClient)clientList.FirstOrDefault(x :KeyValuePair<string,TcpClient> => x.Key == clientName).Value;
            workerSocket.Close();

        }
        catch (SocketException se)
        {
        }
    }

    2 references
    public void sendUsersList()
    {
        try
```

```
public void sendUsersList()
{
    try
    {
        byte[] userList = new byte[1024];
        string[] clist = listBox1.Items.OfType<string>().ToArray();
        List<string> users = new List<string>();

        users.Add(item:"userList");
        foreach (String name in clist)
        {
            users.Add(name);
        }
        userList = ObjectToByteArray(users);

        foreach (var Item:KeyValuePair<string,TcpClient> in clientList)
        {
            TcpClient broadcastSocket;
            broadcastSocket = (TcpClient)Item.Value;
            NetworkStream broadcastStream = broadcastSocket.GetStream();
            broadcastStream.Write(userList, offset:0, size:userList.Length);
            broadcastStream.Flush();
            users.Clear();
        }
    }
    catch (SocketException se)
    {
    }
}

1 reference
private void privateChat(List<string> text)
{
    try
    {

        byte[] byData = ObjectToByteArray(text);
```

```
            byte[] byData = ObjectToByteArray(text);

            TcpClient workerSocket = null;
            workerSocket = (TcpClient)clientList.FirstOrDefault(x:KeyValuePair<string,TcpClient> => x.Key == text[1]).Value;

            NetworkStream stm = workerSocket.GetStream();
            stm.Write(byData, offset:0, size:byData.Length);
            stm.Flush();

        }
        catch (SocketException se)
        {
        }
    }

    1 reference
    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        textBox1.SelectionStart = textBox1.TextLength;
        textBox1.ScrollToCaret();
    }
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Client
{
    7 references
    public partial class formPrivate : Form
    {
        TcpClient clientSocket = new TcpClient();
        String friend, myName;
        Thread ctThread;
        NetworkStream serverStream = default(NetworkStream);
        List<string> chat = new List<string>();


        0 references
        public String getFriend()
        {
            return this.friend;
        }

        0 references
        public String getHistory()
        {
            return history.Text;
        }

        3 references
        public void setHistory(String message)
        {
```

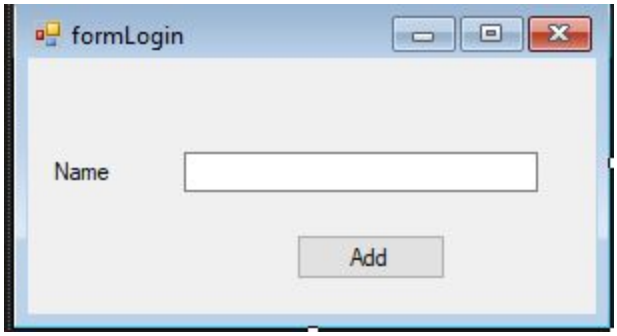
```
public formPrivate(String friend, TcpClient c, String name)
{
    InitializeComponent();
    clientSocket = c;
    this.friend = friend;
    this.myName = name;

    serverStream = clientSocket.GetStream();
    ctThread = new Thread(getMessage);
    ctThread.Start();

}


1 reference
private void history_TextChanged(object sender, EventArgs e)
{
    history.SelectionStart = history.TextLength;
    history.ScrollToCaret();
}

1 reference
public Object ByteArrayToObject(byte[] arrBytes)
{
    using (var memStream = new MemoryStream())
    {
        var binForm = new BinaryFormatter();
        memStream.Write(arrBytes, offset:0, count:arrBytes.Length);
        memStream.Seek(offset:0, loc:SeekOrigin.Begin);
        var obj:object = binForm.Deserialize(memStream);
        return obj;
    }
}

1 reference
public byte[] ObjectToByteArray(Object obj)
```

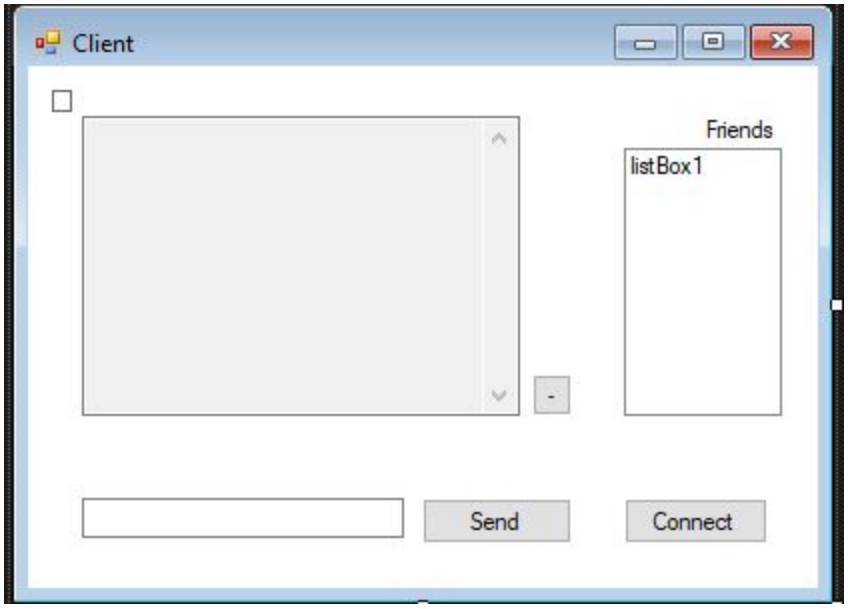
```
public byte[] ObjectToByteArray(Object obj)
{
    BinaryFormatter bf = new BinaryFormatter();
    using (var ms = new MemoryStream())
    {
        bf.Serialize(ms, obj);
        return ms.ToArray();
    }
}

3 references
private void btnSend_Click(object sender, EventArgs e)
{
    try
    {
        if (!inputPrivate.Text.Equals(""))
        {
            chat.Clear();
            chat.Add(item:"pChat");
            chat.Add(friend);
            chat.Add(myName);
            chat.Add(inputPrivate.Text);

            byte[] outStream = ObjectToByteArray(chat);

            serverStream.Write(outStream, offset:0, size:outStream.Length);
            serverStream.Flush();

            this.Invoke((MethodInvoker)delegate
            {
                history.Text = history.Text + Environment.NewLine + inputPrivate.Text;
                inputPrivate.Text = "";
            });
        }
    }
    catch (Exception er)
    {

    }
}
```

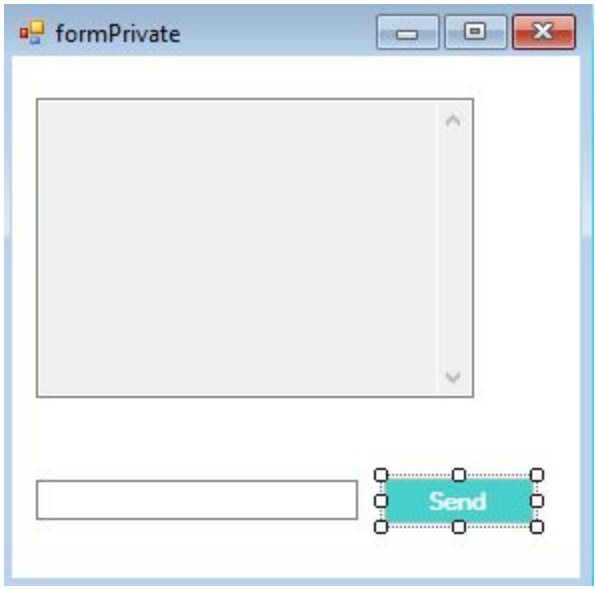
```
1 reference
public void getMessage()
{
    try
    {
        while (true)
        {
            byte[] inStream = new byte[10025];
            serverStream.Read(inStream, offset:0, size:inStream.Length);


            List<string> parts = (List<string>)ByteArrayToObject(inStream);

            if ((parts[2].Equals(friend)))
            {
                setHistory(parts[3]);
            }

            else if (parts[0].Equals(obj:'\0'))
            {
                setHistory(message:"Client Left");
                ctThread.Abort();
                clientSocket.Close();
                break;
            }
            parts.Clear();
        }
    }
    catch (Exception e)
    {
        ctThread.Abort();
        clientSocket.Close();

    }

}
```

```
0 references
bool SocketConnected(TcpClient s)
{
    bool part1 = s.Client.Poll(microSeconds:1000, SelectMode.SelectRead);
    bool part2 = (s.Available == 0);
    if (part1 && part2)
        return false;
    else
        return true;
}

}
```

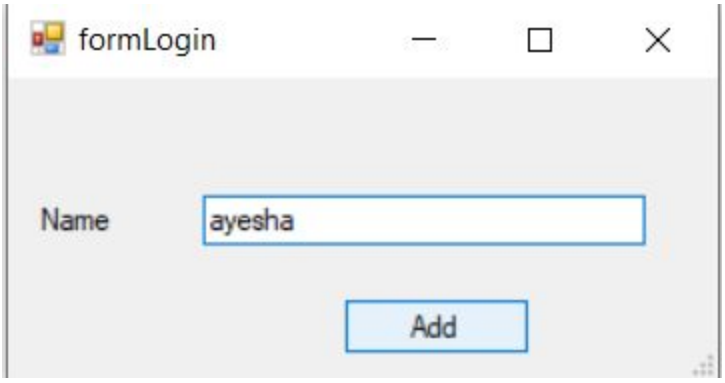
```
namespace Client
{
    0 references
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        0 references
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            formLogin fLogin = new formLogin();
            Boolean flag = true;

                if (fLogin.ShowDialog() == DialogResult.OK)
                {
                    if (fLogin.Textb() != "")
                    {
                        flag = false;
                        formMain form = new formMain();
                        form.setName(title: fLogin.Textb());
                        Application.Run(form);
                    }
                    else
                    {
                        fLogin.slblU(v: "Please enter");
                        //MessageBox.Show("Please enter");
                    }
                }
                else
                {
                    Application.Exit();

                }

        }
```

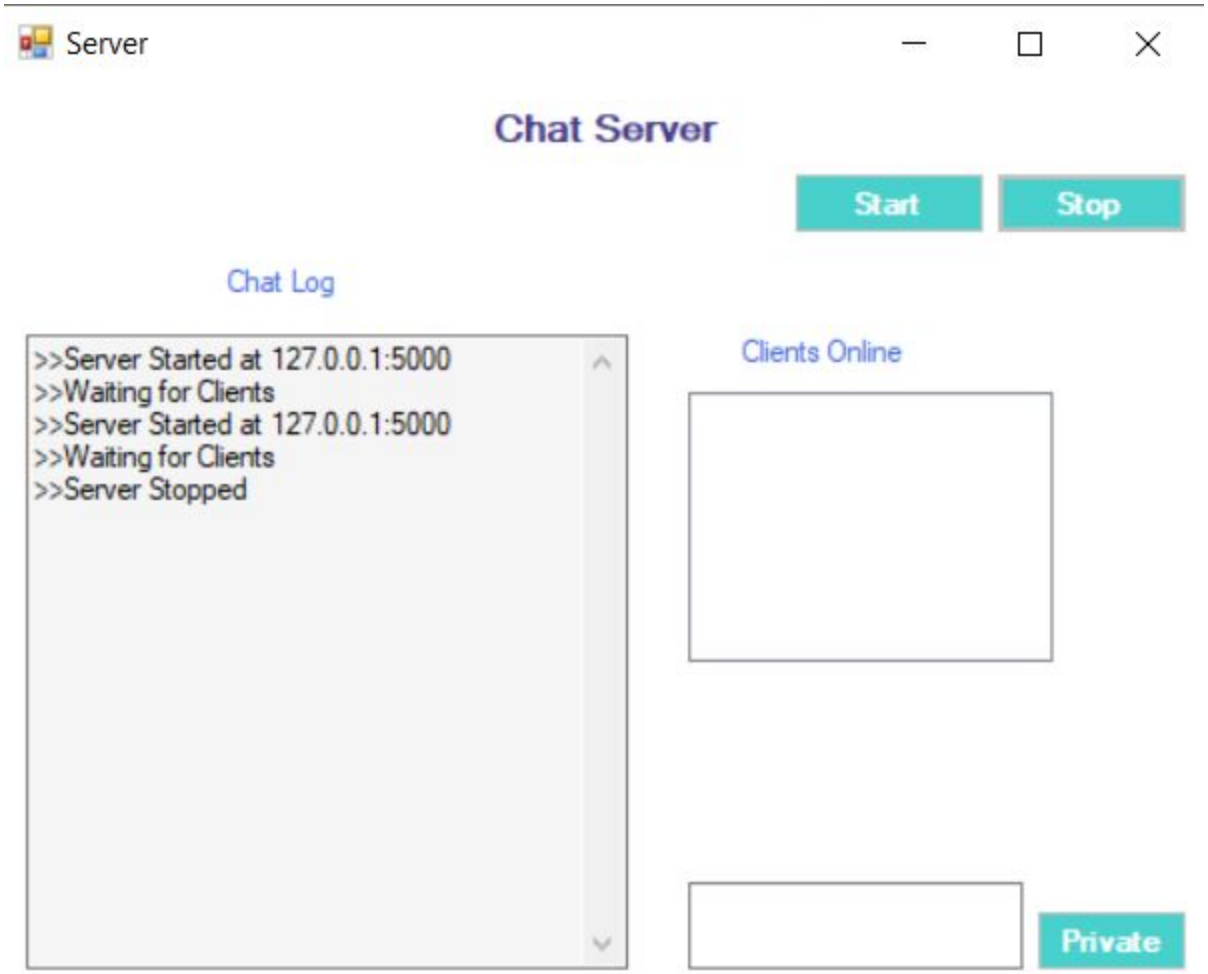

formLogin
Name
Add

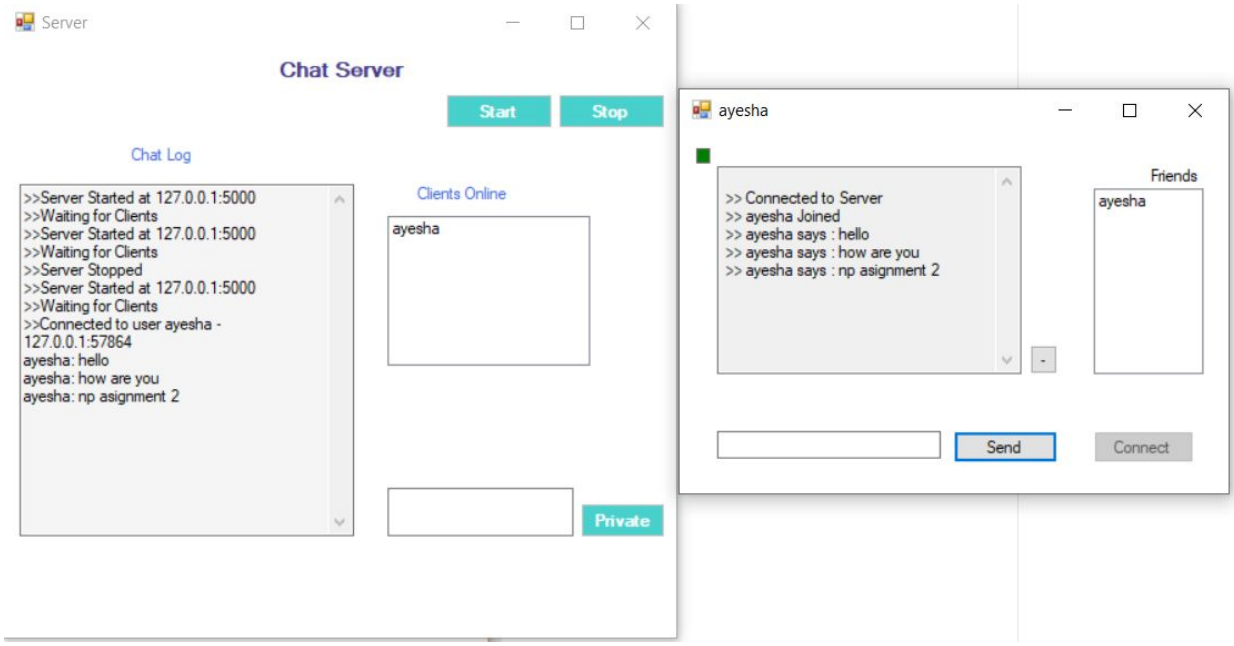

Client
Friends
listBox1
-
Send
Connect

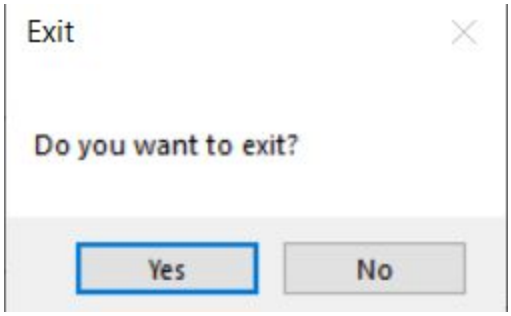

formPrivate
Send

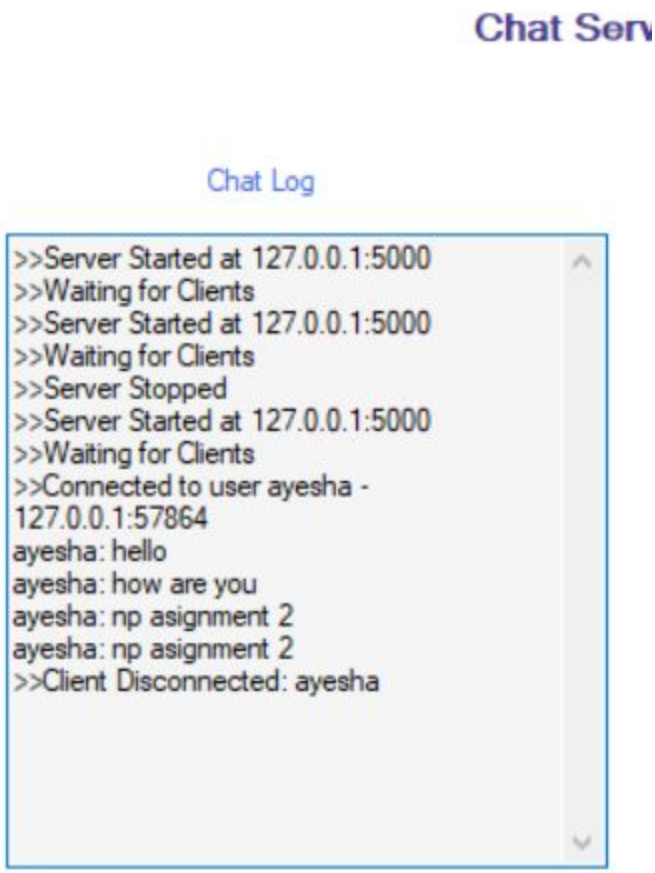
Output:-


formLogin
Name
ayesha
Add


Server
Chat Server
Start
Stop
Chat Log
>>Server Started at 127.0.0.1:5000
>>Waiting for Clients
>>Server Started at 127.0.0.1:5000
>>Waiting for Clients
>>Server Stopped
Clients Online
Private


Server
Chat Server
Start
Stop
Chat Log
>>Server Started at 127.0.0.1:5000
>>Waiting for Clients
>>Server Started at 127.0.0.1:5000
>>Waiting for Clients
>>Server Stopped
>>Server Started at 127.0.0.1:5000
>>Waiting for Clients
>>Connected to user ayesha - 127.0.0.1:57864
ayesha: hello
ayesha: how are you
ayesha: np asignment 2
Clients Online
ayesha
Private
ayesha
>> Connected to Server
>> ayesha Joined
>> ayesha says : hello
>> ayesha says : how are you
>> ayesha says : np asignment 2
Friends
ayesha
Send
Connect


Exit
Do you want to exit?
Yes
No



Q NO:3

Blocking:-

The default mode of socket calls is blocking. A blocking call does not return to your program until the event you requested has been completed. For example, if you issue a blocking recvfrom() call, the call does not return to your program until data is available from the other socket application. A blocking accept() call does not return to your program until a client connects to your socket program.

Source Code:-

```
using System;
using System.Threading;
using System.Net.Sockets;
using System.Text;
using System.Collections;
using System.Net;

namespace ConsoleApp1
{
    0 references
    class Program
    {
        1 reference
        private static Socket ConnectSocket(string server, int port)
        {
            Socket s = null;
            IPHostEntry hostEntry = null;

            hostEntry = Dns.GetHostEntry(server);

            foreach (IPAddress address in hostEntry.AddressList)
            {
                IPEndPoint ipe = new IPEndPoint(address, port);
                Socket tempSocket =
                    new Socket(ipe.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

                tempSocket.Connect(ipe);

                if (tempSocket.Connected)
                {
                    s = tempSocket;
                    break;
                }
                else
                {
                    continue;
                }
            }
            return s;
        }
```

```
1 reference
private static string SocketSendReceive(string server, int port)
{
    string request = "GET / HTTP/1.1\r\nHost: " + server +
        "\r\nConnection: Close\r\n\r\n";
    Byte[] bytesSent = Encoding.ASCII.GetBytes(request);
    Byte[] bytesReceived = new Byte[256];
    string page = "";

    using (Socket s = ConnectSocket(server, port))
    {

        if (s == null)
            return ("Connection failed");

        s.Send(bytesSent, bytesSent.Length, socketFlags:0);

        int bytes = 0;
        page = "Default HTML page on " + server + ":\r\n";

        do
        {
            bytes = s.Receive(bytesReceived, bytesReceived.Length, socketFlags:0);
            page = page + Encodi [local variable) byte[] bytesReceived eceived, index:0, count:bytes);
        }
        while (bytes > 0);
    }

    return page;
}

0 references
static void Main(string[] args)
{
    string host;
    int port = 80;
```

```
0 references
static void Main(string[] args)
{
    string host;
    int port = 80;

    if (args.Length == 0)
        host = Dns.GetHostName();
    else
        host = args[0];

    string result = SocketSendReceive(host, port);
    Console.WriteLine(result);
    Console.ReadLine();
}
```

Output:-

```
C:\Users\Ayesha\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe
Default HTML page on DESKTOP-GJ8J36C:
HTTP/1.1 404 Not Found
Content-Type: text/html; charset=us-ascii
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 24 Feb 2021 09:13:07 GMT
Connection: close
Content-Length: 315

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN""http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>Not Found</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html; charset=us-ascii"></HEAD>
<BODY><h2>Not Found</h2>
<hr><p>HTTP Error 404. The requested resource is not found.</p>
</BODY></HTML>
```

Q NO:4

Introduction:-

A socket is defined as the end point of a two way communication between two processes running over a network. Inter-process communication can be achieved using sockets. After a connection between the server and client, i.e., the server process and the client process is established, they can communicate for the purpose of exchanging data using sockets.

Need of Asynchronous Sockets:-

Asynchronous programming enables you to execute tasks sans the need of holding up the execution flow or responsiveness of your application. This in turn helps to improve the performance and responsiveness of your application.
You can also build synchronous sockets, but such sockets don't scale well since they block your thread. Asynchrony can perform resource-intensive I/O operations sans the need to block the main or the executing thread of your application.

Implementation:-

To implement a TCP server-client socket communication, you would typically need to create a server process that should start at a particular port and also a client process that can start on any port and send a connection request to the server. The server process after it is started, listens for incoming connection requests at the port on which it has been started.

Difference b/w Asynchronous and Synchronous :-

Consider a server application that is listening on a specific port to get data from clients. In synchronous receiving, while the server is waiting to receive data from a client, if the stream is empty the main thread will block until the request for data is satisfied. Hence, the server cannot do anything else until it receives data from the client. If another client attempts to connect to the server at that time, the server cannot process that request because it is blocked on the first client. This behavior is not acceptable for a real-world application where we need to support multiple clients at the same time.

In asynchronous communication, while the server is listening or receiving data from a client, it can still process connection requests from other clients as well as receive data from those clients. When a server is receiving asynchronously, a separate thread (at the OS level) listens on the socket and will invoke a callback function (specified when the asynchronous listening was commenced) when a socket event occurs. This callback function in turn will respond and

process that socket event. For example, if the remote program writes some data to the socket, a "read data event" (callback function you specify) is invoked; it knows how to read the data from the socket at that point.

Q NO:5

TCP goes bad:-

1) The operating system may be buggy, and you can't escape it. it may be inefficient, and you have to put up with it. it may be optimized for conditions other than the ones you are facing, and you may not be able to return it.
2) Tcp makes it very difficult to try harder; you can set a few socket options, but beyond that you have to tolerate the built in flow control.
3) Tcp may have lots of features you don't need. it may waste bandwidth, time, or effort on ensuring things that are irrelevant to the task at hand.
4) Tcp has no block boundaries; you must create your own routers on the internet today that are out of memory. they can't pay much attention to tcp flying by, and try to help it. design assumptions of tcp break down in this environment.
5) Tcp has relatively poor throughput on a lossy, high bandwidth, high latency link, such as a satellite connection or an overfull t1.
6) Tcp cannot be used for broadcast or multicast transmission.
7) Tcp cannot conclude a transmission without all data in motion being explicitly acked.
8) Startup latency is significant. it takes at least twice rtt to start getting data back.
9) Tcp allows a window of at most 64k, and the locking mechanism means that packet loss is misdetected. tcp stalls easily under packet loss. tcp is more throttled by rtt than bandwidth.
10) Tcp transfer servers have to maintain a separate socket (and often separate thread) for each client.
11) Load balancing is crude and approximate. especially on local networks that allow collisions, two simultaneous tcp transfers have a tendency to fight with each other, even if the sender is the same.

UDP goes bad:-

1) There are no guarantees with udp. a packet may not be delivered, or delivered twice, or delivered out of order; you get no indication of this unless the listening program at the other end decides to say something. tcp is really working in the same

environment; you get roughly the same services from ip and udp. however, tcp makes up for it fairly well, and in a standardized manner.

2) Udp has no flow control. implementation is the duty of user programs.
3) Routers are quite careless with udp. they never retransmit it if it collides, and it seems to be the first thing dropped when a router is short on memory. udp suffers from worse packet loss than tcp.

Q NO:6

Source Code:-

```
using System;
using System.Net.Sockets;
using System.IO;

namespace ConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                bool status = true;
                string servermessage = "";
                string clientmessage = "";
                TcpListener tcpListener = new TcpListener(port: 8100);
                tcpListener.Start();
                Console.WriteLine("Server Started");
                Socket socketForClient = tcpListener.AcceptSocket();
                Console.WriteLine("Client Connected");
                NetworkStream networkStream = new NetworkStream(socketForClient);
                StreamWriter streamwriter = new StreamWriter(networkStream);
                StreamReader streamreader = new StreamReader(networkStream);
                while (status)
                {
                    if (socketForClient.Connected)
                    {
                        servermessage = streamreader.ReadLine();
```

```
            while (status)
            {
                if (socketForClient.Connected)
                {
                    servermessage = streamreader.ReadLine();
                    Console.WriteLine("Client:" + servermessage);
                    if ((servermessage == "bye"))
                    {
                        status = false;
                        streamreader.Close();
                        networkStream.Close();
                        streamwriter.Close();
                        return;
                    }
                    Console.Write("Server:");
                    clientmessage = Console.ReadLine();
                    streamwriter.WriteLine(clientmessage);
                    streamwriter.Flush();
                }
            }
            streamreader.Close();
            networkStream.Close();
            streamwriter.Close();
            socketForClient.Close();
            Console.WriteLine("Exiting");
        }
        catch (Exception e)
        {
            Console.WriteLine(e.ToString());
```

```
        }
        streamreader.Close();
        networkStream.Close();
        streamwriter.Close();
        socketForClient.Close();
        Console.WriteLine("Exiting");
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
```