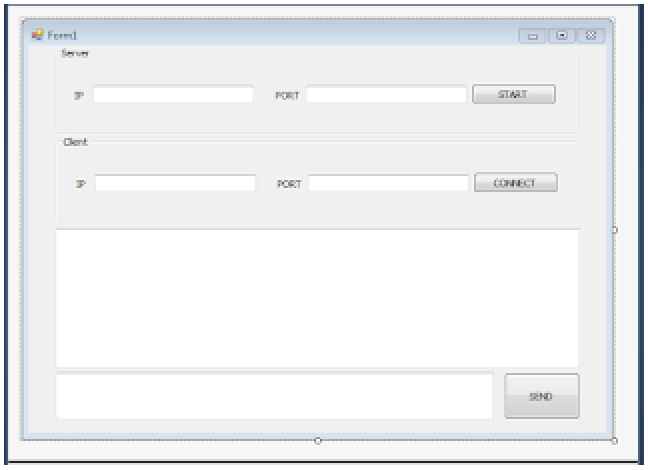
C# simplifies the network programming through its namespaces like System.Net and System.Net.Sockets. A Socket is an End-Point of To and From (Bidirectional) communication link between two programs (Server Program and Client Program ) running on the same network . We need two programs for communicating a socket application in C#. A Server Socket Program ( Server ) and a Client Socket Program ( Client ).
So let's develop a Client Server Windows Forms C# application (chat application). First let's create this Windows Form:



TextBoxes names : ServerIPtextBox, ServerPorttextBox, ClientIPtextBox, ClientPorttextBox, ChatScreentextBox, MessagetextBox.
Then add two BackgroundWorker to the project.
And add these namespaces to the project:

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

After that, declare these variables:

```
private TcpClient client;
public StreamReader STR;
public StreamWriter STW;
public string recieve;
public String TextToSend;
```

And add this code in Form1():

```
public Form1()
{
    InitializeComponent();

    IPAddress[] localIP= Dns.GetHostAddresses(Dns.GetHostName());

    foreach(IPAddress address in localIP)
    {
        if(address.AddressFamily == AddressFamily.InterNetwork)
        {
            ServerIPtextBox.Text = address.ToString();
        }
    }

}
```

In StartButton click event add this code:

```
private void StartButton_Click(object sender, EventArgs e)
{
    TcpListener listener = new TcpListener(IPAddress.Any,
int.Parse(ServerPorttextBox.Text));
    listener.Start();
    client = listener.AcceptTcpClient();
    STR = new StreamReader(client.GetStream());
    STW = new StreamWriter(client.GetStream());
    STW.AutoFlush = true;

    backgroundWorker1.RunWorkerAsync();
    backgroundWorker2.WorkerSupportsCancellation = true;

}
```

In ConnectButton click event add this code:

```
private void ConnectButton_Click(object sender, EventArgs e)
{
    client = new TcpClient();
    IPEndPoint IpEnd = new IPEndPoint(IPAddress.Parse(ClientIPtextBox.Text),
int.Parse(ClientPorttextBox.Text));

    try
    {
        client.Connect(IpEnd);

        if(client.Connected)
        {
            ChatScreentextBox.AppendText("Connected to server" + "\n");
            STW = new StreamWriter(client.GetStream());
            STR = new StreamReader(client.GetStream());
            STW.AutoFlush = true;
            backgroundWorker1.RunWorkerAsync();
            backgroundWorker2.WorkerSupportsCancellation = true;

        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
    }
}
```

In backgroundWorker1 DoWork event add this code:

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    while(client.Connected)
    {
        try
        {
            recieve = STR.ReadLine();
            this.ChatScreentextBox.Invoke(new MethodInvoker(delegate()
                {
                    ChatScreentextBox.AppendText("You:" + recieve + "\n");
                }));
            recieve = "";
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message.ToString());
        }
    }
}
```

And in backgroundWorker2 DoWork event add this code:

```
private void backgroundWorker2_DoWork(object sender, DoWorkEventArgs e)
{
    if(client.Connected)
    {
        STW.WriteLine(TextToSend);
        this.ChatScreentextBox.Invoke(new MethodInvoker(delegate()
        {
            ChatScreentextBox.AppendText("Me:" + TextToSend + "\n");
        }));
    }
    else
    {
        MessageBox.Show("Sending failed");
    }
    backgroundWorker2.CancelAsync();
}
```

Finally in SendButton click event add this code:

```
private void SendButton_Click(object sender, EventArgs e)
{
    if(MessagetextBox.Text!="")
    {
        TextToSend = MessagetextBox.Text;
        backgroundWorker2.RunWorkerAsync();
    }
    MessagetextBox.Text = "";

}
```

Then let's run the application and open a second instance to it from (%project directory%\bin\Debug). Open two instances for testing purposes in your computer, then give the IP address and Port number to the two instances. Then connect both Applications and start sending messages.

Check this video for more explanation:

```
// *** Server
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

namespace MyTcpServer
{
    class Program
    {
        static int Main(string[] args)
        {
            try
            {
                StartListener(7001).Wait();
                return 0;
            }
            catch (Exception ex)
            {
                Console.Error.WriteLine(ex);
                return -1;
            }
        }

        private static async Task StartListener(int port)
        {
            var tcpListener = TcpListener.Create(port);
            tcpListener.Start();
            for (; ; )
            {
                Console.WriteLine("[Server] waiting for clients...");
                using (var tcpClient = await tcpListener.AcceptTcpClientAsync())
                {
                    try
                    {
                        Console.WriteLine("[Server] Client has connected");
                        using (var networkStream = tcpClient.GetStream())
                        using (var reader = new StreamReader(networkStream))
                        using (var writer = new StreamWriter(networkStream) {
AutoFlush = true })
                        {
                            var buffer = new byte[4096];
                            Console.WriteLine("[Server] Reading from client");
                            var request = await reader.ReadLineAsync();
                            string.Format(string.Format("[Server] Client wrote
'{0}'", request));

                            await writer.WriteLineAsync($"[Server] to Client
{request}");
```

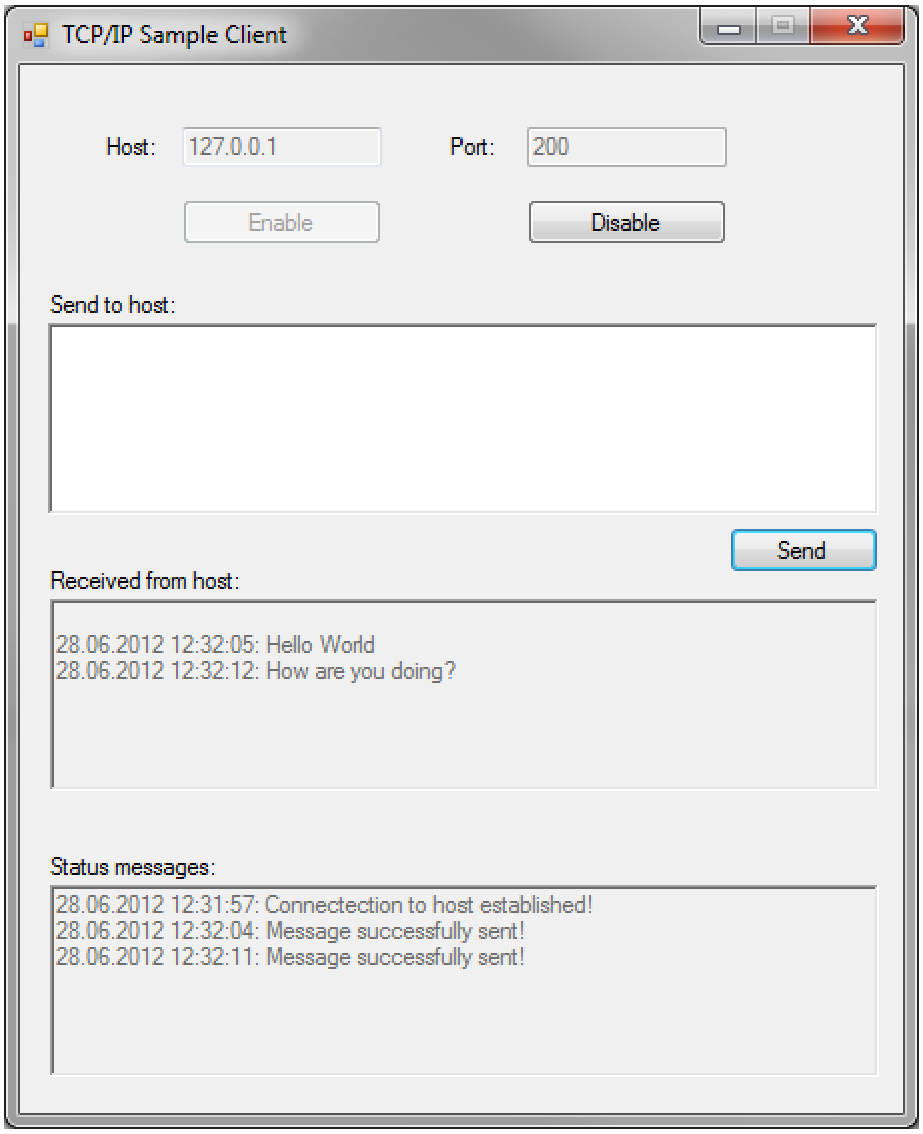
```
                            //for (int i = 0; i < 5; i++)
                            //{
                            //    await writer.WriteLineAsync("I am the server! 
HAHAHA!");
                            //    Console.WriteLine("[Server] Response has been 
written");
                            //    await Task.Delay(TimeSpan.FromSeconds(1));
                            //}
                        }
                    }
                    catch (Exception)
                    {
                        Console.WriteLine("[Server] client connection lost");
                    }
                }
            }
        }
    }
}
```

```
// *** CLIENT


using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MyClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            Task _ = ConnectAsTcpClient("127.0.0.1", 7001);
        }

        private void OutputWriteLine(string text)
        {
            output.Text = output.Text + "\n" + text;
            output.ScrollToEnd();
        }


        private void btnSend_Click(object sender, RoutedEventArgs e)
        {
            OutputWriteLine("Button_Click Send");
            ClientRequestString = "Hello Mr. Server from Button";
        }

        private async Task ConnectAsTcpClient(string ip, int port)
        {
            for (; ; )
            {
                try
                {
                    await Task.Delay(millisecondsDelay: 5000);
                    using (var tcpClient = new TcpClient())
                    {
                        OutputWriteLine("[Client] Attempting connection to server
" + ip + ":" + port);
                        Task connectTask = tcpClient.ConnectAsync(ip, port);
                        Task timeoutTask = Task.Delay(millisecondsDelay: 6100);
                        if (await Task.WhenAny(connectTask, timeoutTask) ==
timeoutTask)
                        {
                            throw new TimeoutException();
                        }


                        OutputWriteLine("[Client] Connected to server");
                        using (var networkStream = tcpClient.GetStream())
                        using (var reader = new StreamReader(networkStream))
                        using (var writer = new StreamWriter(networkStream) {
AutoFlush = true })
                        {
                            OutputWriteLine(string.Format("[Client] Writing
request '{0}'", ClientRequestString));
                            await writer.WriteLineAsync(ClientRequestString);
```

```
 // ### not good, server can send a event, at moment is closing the connection
and reconnect.
 // ### I need a event, if new data available
 // ### How can I do it?
                            try
                            {
                                for (; ; )
                                {
                                    var response = await reader.ReadLineAsync();
                                    if (response == null)
                                    {
                                        break;
                                    }
                                    OutputWriteLine(string.Format("[Client]
Server response was '{0}'", response));
                                }
                                OutputWriteLine("[Client] Server disconnected");
                            }
                            catch (IOException)
                            {
                                OutputWriteLine("[Client] Server disconnected");
                            }
                        }
                    }
                }
                catch (TimeoutException)
                {
                    // reconnect
                    OutputWriteLine("[Client] Timeout - No connection");
                }
            }
        }

        //private static readonly string ClientRequestString = "Hello Mr.
Server";
        private string ClientRequestString = "Hello Mr. Server";


    }
}
```

## .NET client

This project example shows how a client for the PLC TCP/IP server can be realized by writing a .NET4.0 application using C#.



```
28.06.2012 12:32:05: Hello World
28.06.2012 12:32:12: How are you doing?
```

```
28.06.2012 12:31:57: Connectection to host established!
28.06.2012 12:32:04: Message successfully sent!
28.06.2012 12:32:11: Message successfully sent!
```

This sample client makes use of the .NET libraries System.Net and System.Net.Sockets which enable a programmer easy access to socket functionalities. By pressing the button **Enable**, the application attempts to cyclically (depending on the value of TIMERTICK in [ms]) establish a connection with the server. If successful, a string with a maximum length of 255 characters can be sent to the server via the "Send" button. The server will then take this string and send it back to the client. On the server side, the connection is closed automatically if the server was unable to receive new data from the client within a defined period, as specified by PLCPRJ_RECEIVE_TIMEOUT in the server sample - by default 50 seconds.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

/* ##########################################################################
################
 * This sample TCP/IP client connects to a TCP/IP-
Server, sends a message and waits for the
 * response. It is being delivered together with our TCP-
Sample, which implements an echo server
 * in PLC.
 * ##########################################################################
################ */namespace TcpIpServer_SampleClient
{
 publicpartialclassForm1 : Form
 {
    /* ######################################################################
####################
    * Constants
    * #######################################################################
################### */privateconstint RCVBUFFERSIZE = 256; // buffer size for
 receive bufferprivateconststring DEFAULTIP = "127.0.0.1";
    privateconststring DEFAULTPORT = "200";
    privateconstint TIMERTICK = 100;

    /* ######################################################################
####################
    * Global variables
    * #######################################################################
################### */privatestaticbool _isConnected; // signals whether sock
et connection is active or notprivatestaticSocket _socket; // object used for
 socket connection to TCP/IP-
ServerprivatestaticIPEndPoint _ipAddress; // contains IP address as entered i
n text fieldprivatestaticbyte[] _rcvBuffer; // receive buffer used for receiv
ing response from TCP/IP-Serverpublic Form1()
    {
     InitializeComponent();
    }

    privatevoid Form1_Load(object sender, EventArgs e)
    {
     _rcvBuffer = newbyte[RCVBUFFERSIZE];

     /* #####################################################################
#####################
     * Prepare GUI
     * ######################################################################
#################### */
     cmd_send.Enabled = false;
     cmd_enable.Enabled = true;
     cmd_disable.Enabled = false;
     rtb_rcvMsg.Enabled = false;
     rtb_sendMsg.Enabled = false;
     rtb_statMsg.Enabled = false;
     txt_host.Text = DEFAULTIP;
     txt_port.Text = DEFAULTPORT;

     timer1.Enabled = false;
     timer1.Interval = TIMERTICK;
     _isConnected = false;
    }

    privatevoid cmd_enable_Click(object sender, EventArgs e)
    {
     /* ############################################################################
#####################
     * Parse IP address in text field, start background timer and prepare GUI

     * ############################################################################
#################### */try
     {
        _ipAddress = newIPEndPoint(IPAddress.Parse(txt_host.Text), Convert.To
Int32(txt_port.Text));
        timer1.Enabled = true;
        cmd_enable.Enabled = false;
        cmd_disable.Enabled = true;
        rtb_sendMsg.Enabled = true;
        cmd_send.Enabled = true;
        txt_host.Enabled = false;
        txt_port.Enabled = false;
        rtb_sendMsg.Focus();
     }
     catch (Exception ex)
       {
        MessageBox.Show("Could not parse entered IP address. Please check spe
lling and retry. " + ex);
     }
    }

    /* #############################################################################
####################
    * Timer periodically checks for connection to TCP/IP-
Server and reestablishes if not connected
    * #############################################################################
################### */privatevoid timer1_Tick(object sender, EventArgs e)
    {
     if (!_isConnected)
        connect();
    }

    privatevoid connect()
    {
     /* ############################################################################
#####################
     * Connect to TCP/IP-
Server using the IP address specified in the text field
     * ############################################################################
#################### */try
     {
        _socket = newSocket(AddressFamily.InterNetwork, SocketType.Stream, Pr
otocolType.IP);
        _socket.Connect(_ipAddress);
        _isConnected = true;
        if (_socket.Connected)
         rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to
 host established!\n");
        else
         rtb_statMsg.AppendText(DateTime.Now.ToString() + ": A connection to
the host could not be established!\n");
     }
     catch (Exception ex)
     {
        MessageBox.Show("An error occured while establishing a connection to
the server: " + ex);
     }
    }

    privatevoid cmd_send_Click(object sender, EventArgs e)
    {
     /* ####################################################################
#####################
     * Read message from text field and prepare send buffer, which is a byte[
] array. The last
     * character in the buffer needs to be a termination character, so that t
he TCP/IP-Server knows
     * when the TCP stream ends. In this case, the termination character is '
0'.
     * #####################################################################
#################### */ASCIIEncoding enc = newASCIIEncoding();
     byte[] tempBuffer = enc.GetBytes(rtb_sendMsg.Text);
     byte[] sendBuffer = newbyte[tempBuffer.Length + 1];
     for (int i = 0; i < tempBuffer.Length; i++)
        sendBuffer[i] = tempBuffer[i];
     sendBuffer[tempBuffer.Length] = 0;

     /* ####################################################################
#####################
       * Send buffer content via TCP/IP connection
     * #####################################################################
#################### */try
     {
        int send = _socket.Send(sendBuffer);
        if (send == 0)
         thrownewException();
        else
        {
         /* ################################################################
#########################
         * As the TCP/IP-
Server returns a message, receive this message and store content in receive b
uffer.
         * When message receive is complete, show the received message in tex
t field.
         * #################################################################
######################## */
         rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Message successf
ully sent!\n");
         IAsyncResult asynRes = _socket.BeginReceive(_rcvBuffer, 0, 256, Sock
etFlags.None, null, null);
         if (asynRes.AsyncWaitHandle.WaitOne())
         {
            int res = _socket.EndReceive(asynRes);
            char[] resChars = newchar[res + 1];
            Decoder d = Encoding.UTF8.GetDecoder();
            int charLength = d.GetChars(_rcvBuffer, 0, res, resChars, 0, true
);
            String result = newString(resChars);
            rtb_rcvMsg.AppendText("\n" + DateTime.Now.ToString() + ": " + res
ult);
            rtb_sendMsg.Clear();
         }
        }
     }
     catch (Exception ex)
       {
        MessageBox.Show("An error occured while sending the message: " + ex);
     }
    }

    privatevoid cmd_disable_Click(object sender, EventArgs e)
    {
     /* #################################################################
#####################
     * Disconnect from TCP/IP-Server, stop the timer and prepare GUI
     * #################################################################
#################### */
     timer1.Enabled = false;
     _socket.Disconnect(true);
     if (!_socket.Connected)
     {
        _isConnected = false;
        cmd_disable.Enabled = false;
        cmd_enable.Enabled = true;
        txt_host.Enabled = true;
        txt_port.Enabled = true;
        rtb_sendMsg.Enabled = false;
        cmd_send.Enabled = false;
        rtb_statMsg.AppendText(DateTime.Now.ToString() + ": Connectection to
host closed!\n");
        rtb_rcvMsg.Clear();
        rtb_statMsg.Clear();
     }
    }
 }
}
```

**Description:**

This is an implementation of a Chat program using sockets. Along with basic chat capability, it has the ability to send private messages and log the ongoing chat.  There are two applications - ChatServer and ChatClient.

ChatServer manages the chat session. It maintains a list of the active chatters and forwards incoming chat messages. ChatServer is hard coded to listen on port 5555 in case you have to change it. The Chat Server is multithreaded - communication with each client is through a separate thread.  When ChatServer is started, it enters the StartListening() thread to wait for client connections:

```
private void StartListening()
{
listener = new TcpListener(listenport);
listener.Start();
while (true)
{
try
{
Socket s = listener.AcceptSocket();
clientsocket = s;
clientservice = new Thread(new ThreadStart(ServiceClient));
clientservice.Start();
}
catch(Exception e)
{
Console.WriteLine(e.ToString() );
}
}
}
```

This thread continues forever to listen for clients. When a connection request is received, the server starts a thread called ServiceClient() to service the client.  Note that when a socket is established, a copy of the socket is made. This is necessary for each client to have its own socket. When the thread is started, a Client object is created containing all the relevant info about each client and saved in an ArrayList. The Client class is shown below. Note the Client class contains a reference to it's own ServiceClient() thread and socket. This simplifies the job of the ChatServer and makes the program logic easier because the ChatServer communicates directly with Clients and can kill their threads directly instead of having to use separate lists to keep track of which socket and thread belongs to which Client.

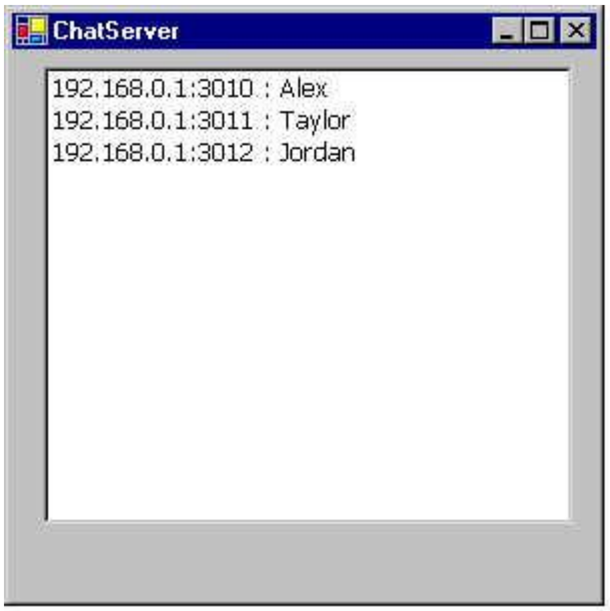
```
namespace ChatServer
{
using System.Net.Sockets;
using System.Net;
public class Client
{
private Thread clthread;
private EndPoint endpoint;
private string name;
private Socket sock;
public Client(string _name, EndPoint _endpoint, Thread _thread, Socket _sock)
{
clthread = _thread;
endpoint = _endpoint;
name = _name;
sock = _sock;
}
public override string ToString()
{
return endpoint.ToString()+ " : " + name;
}
public Thread CLThread
{
get{return clthread;}
set{clthread = value;}
}
public EndPoint Host
{
get{return endpoint;}
set{endpoint = value;}
}
public string Name
{
get{return name;}
set{name = value;}
}
public Socket Sock
{
get{return sock;}
set{sock = value;}
}
}
}
```

The ServiceClient() thread is where all the work is done. ServiceClient() defines the implementation of the server side of the application protocol. The application protocol is the semantics of message exchange between the server and client.The protocol I have used has two traits that are highly desirable when designing a protocol. First, it is stateless, meaning the messages are context-free. The processing of a message does not depend on a previous message or any other context - messages can be received at any time and in any order without confusing the application. Second, the messages are fully self describing, meaning all the data needed to process a message is contained within the message.  The server recognizes four commands, CONN, CHAT, PRIV, and GONE.  CONN establishes a new client, sends a list of current chatters, and notifies other chatters a new person has joined the group. CHAT simply forwards the incoming chat to all recipients. PRIV sends a private message to the person designated.  GONE removes an active chatter account and notifies all other members the chatter has left. GONE will also cause the death of the clients service thread by setting the bool keepalive to false. This is the cleanest way to kill the thread, however the ChatServer can kill the thread itself if it detects the connection is terminated.

The server receives the incoming messages as ASCII strings. The '|' char is used as the separator between parts of the message. A message consists of a command, and one or more other parameters required to process the message.

```
private void ServiceClient()
{
Socket client = clientsocket;
bool keepalive = true;
while (keepalive)
{
Byte[] buffer = new Byte[1024];
client.Receive(buffer);
string clientcommand = System.Text.Encoding.ASCII.GetString(buffer);
string[] tokens = clientcommand.Split(new Char[]{'|'});
Console.WriteLine(clientcommand);
if (tokens[0] == "CONN")
{
for(int n=0; n<clients.Count; n++) {
Client cl = (Client)clients[n];
SendToClient(cl, "JOIN|" + tokens[1]);
}
EndPoint ep = client.RemoteEndPoint;
Client c = new Client(tokens[1], ep, clientservice, client);
clients.Add(c);
string message = "LIST|" + GetChatterList() +"\r\n";
SendToClient(c, message);
lbClients.Items.Add(c);
}
if (tokens[0] == "CHAT")
{
for(int n=0; n<clients.Count; n++)
{
Client cl = (Client)clients[n];
SendToClient(cl, clientcommand);
}
}
if (tokens[0] == "PRIV") {
string destclient = tokens[3];
for(int n=0; n<clients.Count; n++) {
Client cl = (Client)clients[n];
if(cl.Name.CompareTo(tokens[3]) == 0)
SendToClient(cl, clientcommand);
if(cl.Name.CompareTo(tokens[1]) == 0)
SendToClient(cl, clientcommand);
}
}
if (tokens[0] == "GONE")
{
int remove = 0;
bool found = false;
int c = clients.Count;
for(int n=0; n<c; n++)
{
Client cl = (Client)clients[n];
SendToClient(cl, clientcommand);
if(cl.Name.CompareTo(tokens[1]) == 0)
{
remove = n;
found = true;
lbClients.Items.Remove(cl);
}
}
if(found)
clients.RemoveAt(remove);
client.Close();
keepalive = false;
}
}
}
```

The ChatServer is shown below. Visually, there isn't much to it, it just displays the active chatter's host ip and name



```
192.168.0.1:3010 : Alex
192.168.0.1:3011 : Taylor
192.168.0.1:3012 : Jordan
```

The ChatClient allows users to log on to the chat and send and receive messages. If the ChatClient is started without any command line parameters, it assumes the ChatServer is on the localhost. If it is not, you must supply an ip address for the ChatServer on the command line. After starting up, put your chat name in the chat entry box and click **Connect**. The ChatServer will respond with a list of current chatters which ChatClient puts in a ListBox. After that you can send messages by typing them on the send line and clicking **Send**. If you wish to send a **private** message, click Private and select a name from the ListBox. Only one name can be selected at a time. Then send the message. The receiver will get the message with a "Private from" string prepended to it.

When the client attempts to connect, a connection is established and registers with the ChatServer. EstablishConnection() uses a TcpClient to connect to the ChatServer. A NetworkStream is established that will be used to send messages. Again the ChatServer port is hard coded to 5555.

```
private void EstablishConnection()
{
statusBar1.Text = "Connecting to Server";
try
{
clientsocket = new TcpClient(serveraddress,serverport);
ns = clientsocket.GetStream();
sr = new StreamReader(ns);
connected = true;
}
catch (Exception e)
{
MessageBox.Show("Could not connect to Server","Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
statusBar1.Text = "Disconnected";
}
}
```

The RegisterWithServer() method constructs and sends a CONN command to the ChatServer. The CONN command contains the chatters name as well. The command is constructed using the '|' char as a separator. The ChatClient then receives the list of chatters and adds them to the ListBox.

```
private void RegisterWithServer()
{
try
{
string command = "CONN|" + ChatOut.Text;
Byte[] outbytes = System.Text.Encoding.ASCII.GetBytes(command.ToCharArray());
ns.Write(outbytes,0,outbytes.Length);
string serverresponse = sr.ReadLine();
serverresponse.Trim();
string[] tokens = serverresponse.Split(new Char[]{'|'});
if(tokens[0] == "LIST")
{
statusBar1.Text = "Connected";
btnDisconnect.Enabled = true;
}
for(int n=1; n<tokens.Length-1; n++)
lbChatters.Items.Add(tokens[n].Trim(new char[]{'\r','\n'}));
this.Text = clientname + ": Connected to Chat Server";
}
catch (Exception e)
{
MessageBox.Show("Error Registering","Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
}
```

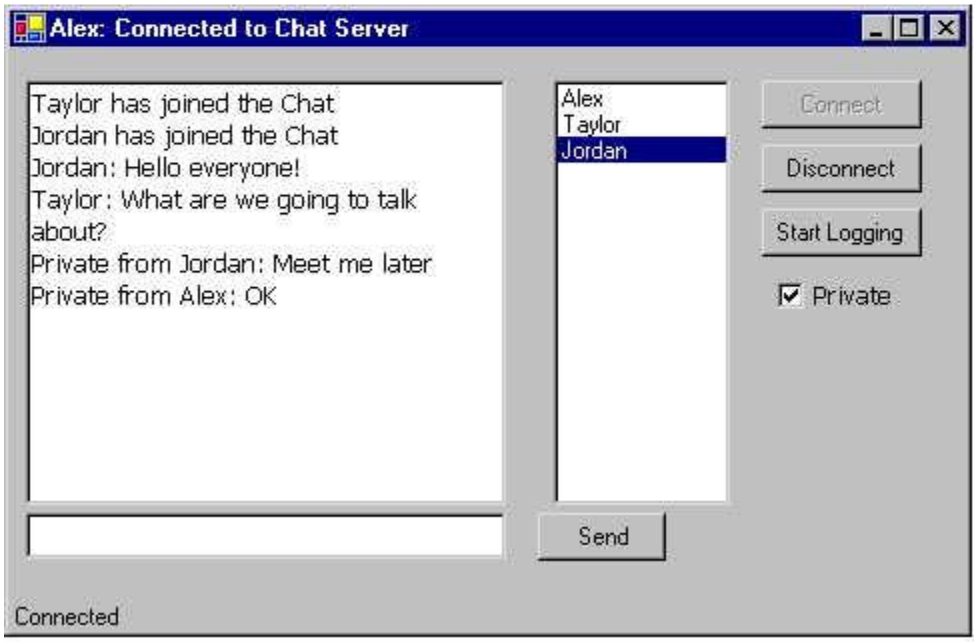
After the ChatClient has connected, it spawns a ReceiveChat() thread to handle all the ins and outs. ReceiveChat() implements the client side of the application protocol.

```
private void ReceiveChat()
{
bool keepalive = true;
while (keepalive)
{
try
{
Byte[] buffer = new Byte[2048];
ns.Read(buffer,0,buffer.Length);
string chatter = System.Text.Encoding.ASCII.GetString(buffer);
string[] tokens = chatter.Split(new Char[]{'|'});
if (tokens[0] == "CHAT")
{
rtbChatIn.AppendText(tokens[1]);
if(logging)
logwriter.WriteLine(tokens[1]);
}
if (tokens[0] == "PRIV") {
rtbChatIn.AppendText("Private from ");
rtbChatIn.AppendText(tokens[1].Trim() );
rtbChatIn.AppendText(tokens[2] + "\r\n");
if(logging){
logwriter.Write("Private from ");
logwriter.Write(tokens[1].Trim() );
logwriter.WriteLine(tokens[2] + "\r\n");
}
}
if (tokens[0] == "JOIN")
{
rtbChatIn.AppendText(tokens[1].Trim() );
rtbChatIn.AppendText(" has joined the Chat\r\n");
if(logging)
logwriter.WriteLine(tokens[1]+" has joined the Chat");
string newguy = tokens[1].Trim(new char[]{'\r','\n'});
lbChatters.Items.Add(newguy);
}
if (tokens[0] == "GONE")
{
rtbChatIn.AppendText(tokens[1].Trim() );
rtbChatIn.AppendText(" has left the Chat\r\n");
if(logging)
logwriter.WriteLine(tokens[1]+" has left the Chat");
lbChatters.Items.Remove(tokens[1].Trim(new char[]{'\r','\n'}));
}
if (tokens[0] == "QUIT")
{
ns.Close();
clientsocket.Close();
keepalive = false;
statusBar1.Text = "Server has stopped";
connected= false;
btnSend.Enabled = false;
btnDisconnect.Enabled = false;
}
}
catch(Exception e){}
}
```

```
}
```

Clicking the Start Logging button will begin writing all the chat and messages to a text file. The file name is built from the current date and time. A sub directory called "logs" will be created to store the log files.

The ChatClient is shown below.



Taylor has joined the Chat
Jordan has joined the Chat
Jordan: Hello everyone!
Taylor: What are we going to talk about?
Private from Jordan: Meet me later
Private from Alex: OK

ISSUES:  No guarantees about this code being bullet proof, its never really been tested with lots of users. Access to the Client list should be synchronized. Don't try to use the text log feature with multiple clients on the same machine or it could break if they are assigned the same file name. Sometimes the clients list of active chatters does not update properly when someone quits. I haven't figured out why yet. If you disconnect and log back in it will fix it.  Some characters are sometimes missing from the log files.

ASP.NET

# Applied C#.NET Socket Programming

-
- Ajay Yadav
- 
- Apr 13, 2021
- 
- 
- 311.8k
- 18
- 11
-

Download .NET & JAVA Office File API For Free

## Abstract

This article explains the key networking concepts, for instance ISO stack, of TCP/IP under the C# framework by employing its essential socket classes and how applications can logically and physically be distributed in a network environment. This paper rovides a close encounter with the .NET base classes for using various network protocols, particularly HTTP, TCP and UDP, to access networks and the internet as a client. Apart from that, we

shall learn how to manipulate an IP address and do a DNS lookup to establish a socket connection between the client and server. Finally, we'll examine the anatomy of sockets in depth and examine how to develop a client and server application to set up a socket connection.

## Internet Networking Protocol

A networking protocol is a standard set of rules that determine how systems will communicates across networks. Two systems that use the same protocols can communicate and understand each other despite their differences. The OSI reference model provides important guidelines to vendors, engineers and others for network communication protocol development. The internet protocol facilitates the transmission and reception of data between participating client and server applications in a packet-switched network environment.

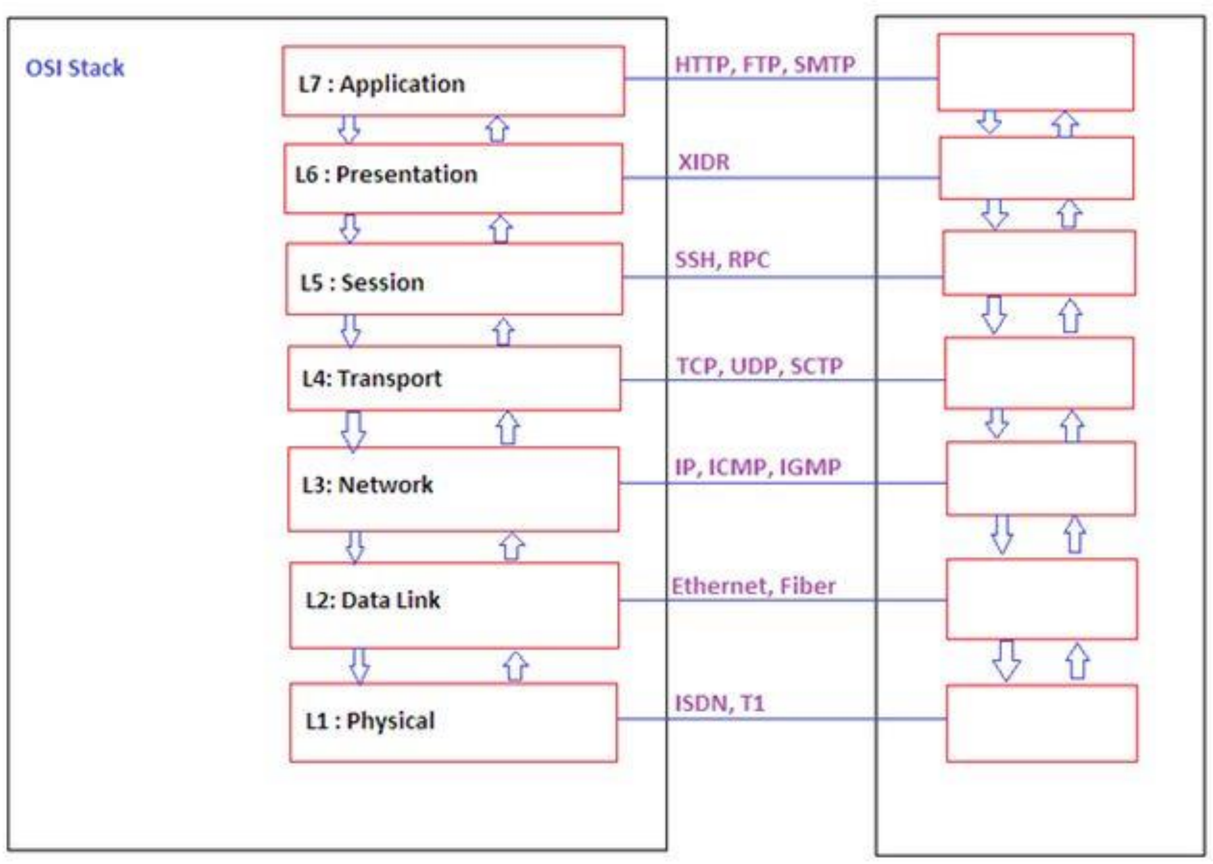


Figure 1.1 OSI Layers

Packet switched networks means data travelling along network pathways is divided into small, routable packages referred to as packet. If a communication link between two points on a network goes down, the packet can be routed through a remaining network connection to their intended destination. The internet protocols work together as a layered protocol stack that consists of the application, presentation, network, transport, Data Link Layer and physical layers. Each layer in the protocol stack provides a set of services to the layer above it.

The transport layer is responsible for process-to-process delivery. The delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship paradigm. A process on the local host called a client, needs services from a process usually on the remote host, called a server.

## Addressing

Whenever we need to deliver something to one specific destination among many, we need an address. At the Data Link Layer, we need a MAC address whereas at the Network Layer, we need a 32 bit IP address (choose from A, B, C, D and E class). A datagram in the Network Layer requires a destination IP address for delivery and a source IP address for the destination reply.

At the Transport layer, we need a transport layer address, which is called a port number, to choose among multiple processes running on the destination host. In the internet paradigm, the port numbers are between 0 and 65535 and are chosen randomly by the transport layer software running on the client host. These are called ephemeral ports (range from 1024 to 49151). The server process must also define itself with a port number. These ports, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and uses its services will not know the port number. So, the internet has decided to use universal port numbers for servers, called well-known port numbers (range 0 to 1023).

## Anatomy of a Socket

TCP and UDP are transport protocols that applications utilize to transport their data across a network. They both use ports to communicate with upper OSI layers and to keep track of various conversations that occur simultaneously. The ports have the mechanism to identify how other computers access the services. When a TCP or UDP message is formed, a source and destination port is contained within the header information along with the source and destination IP address. This makes a socket.

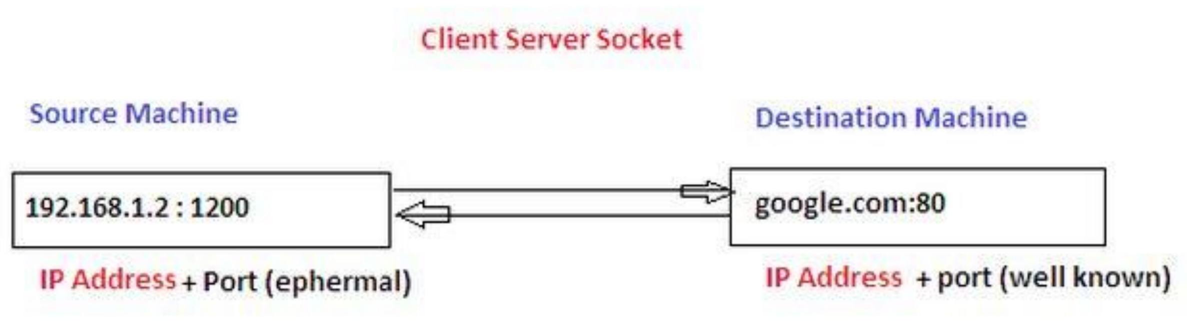


Figure 1.2 Socket Demonstration

A socket represents one end of a connection between two end points on the network, much like streams. Windows ships with an implementation of sockets called WinSock32, on the top of which are the **System.Net.Socket** APIs. The library can be used for both responding to an existing network communication and initiating a new network communication, that makes them perfect for client-server applications. They are often used for work-intensive server applications, such as web servers. The socket implementations in the S**ystem.Net.Socket** namespace can be described by the following components:

- **Socket:** This operation is used to create a new socket end point, allocating operating system resources that can then be used to execute an incoming and outgoing communication.

- **Bind:** Servers must bind a socket to an address to establish a local name. A bind enables external communication to connect to and send messages through new end points, enable the socket to read messages off of it and send its own.

- **Listen:** The socket shows its interest in message queues participation by notifying the OS by executing an optional listen operation, where message queues consume incoming requests for further processing.

- **Accept:** A socket accepts an incoming request by leaving a new socket, that can be used to read and write to the client.

- **Connect:** Used to connect to a server-side network end point. Once the target end point accepts the request, the socket can be used to read from and write data to the socket.

- **Send:** once a socket is connected to an end point, sending data initiates a message that the other point is then able to receive.

- **Receive:** after the end point to which a socket is connected sends data, a socket may consume that data by receiving it.

- **Close:** when a connection has finished its session, a socket must be closed that releases all the resources held by the socket.

There are two types of a sockets generally created, server-side sockets and client-side sockets. The server-side application whose purpose is to process network request commonly sits inside a loop and rapidly consumes and dispatches the requests. The basic pattern is to create, bind, listen to and accept until the program is shut down. For instance, a web server (such as IIS or apache) that receives and processes incoming requests. The client-side socket communicates with existing end points on the network. The only operation required is connecting to server, aside from sending and receiving data.

### Utility Networking Classes

The two namespaces of most interest for networking are **System.Net** and **System.Net.Sockets.** System.Net is generally concerned with higher-level operations, for example, downloading and uploading a file and making web requests using HTTP and other protocols, whereas System.Net.Sockets contains classes to do lower-level operations.

## IPAddress Class

An Internet Protocol (IP) address is a 32 bit or 128 bit number that uniquely identifies a specific computer on the network. TCP/IP uses two pieces of information to identify a specific program: an Internet address, used by IP and a port number, the additional address is interpreted by the transport protocol (TCP or UDP). The **IPAddress** class represents an IP address. The .NET encapsulates the IP addresses abstraction in the **IPAddress** class that can take a long integer IP argument in its constructor, or process a string with the dotted-quad representation of an IP address by using its Parse() method.

```
IPAddress ipa = IPAddress.Parse("192.168.1.10");
Console.WriteLine("IP address is="+ipa.ToString());
```

The **IPAddress** class also provides a number of constant static fields to return a special address, such as the computer loopback address.

```
Console.WriteLine("IP address is=" + IPAddress.Loopback.ToString());
Console.WriteLine("Broadcast address is=" + IPAddress.Broadcast.ToString());
```

## Dns Class

The **Dns** class is able to communicate with your default DNS server to obtain an IP address. Here, the two static methods **Resolve()** and **GetHostByAddress()** are used to retrieve the details of the host using the **IPHostEntry** class.

## IPHostEntry Class

The **IPHostEntry** class encapsulates the information related to a specific host computer. Here, the SocketPro class shows the use of the **IPAddress**, **IPHostEntry** and **Dns** classes. The program takes a list of names or IP addresses as command-line parameters and prints the name and an IP address of the local host, followed by the names and IP addresses of the hosts specified on the command line.

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

namespace SocketPro
{
    class Program
    {
        static void GetHostInfo(string host)
         {
              try
            {
                // Attempt to resolve DNS for given host or address
                IPHostEntry hostInfo = Dns.GetHostEntry(host);

                // Display host name
                Console.WriteLine("\tHost Name: " + hostInfo.HostName);

                // Display list of IP addresses for this host
                Console.Write("\tIP Addresses: ");
                foreach (IPAddress ipaddr in hostInfo.AddressList)
                {
                    Console.Write(ipaddr.ToString() + " ");
                }
                Console.WriteLine();
            }
            catch (Exception)
            {
                Console.WriteLine("\tUnable to resolve host: " + host + "\n");
            }
        }
        static void Main(string[] args)
        {
            foreach (String arg in args)
            {
                Console.WriteLine("\nTesting for="+arg);
                GetHostInfo(arg);
            }
            Console.ReadKey();
        }
    }
}
```

After compiling this program, open the application executable over the command prompt with essential arguments such as host name or IP address, then the output would be as follows.

## Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) classes are used to connect and send data between two socket endpoints (IP address + port). TCP offers **TcpClient** and **TcpListener** classes to establish a socket connection on well-known ports such as 80, 443 and 21. The **TcpClient** class enables creation and of use a TCP connection while **TcpListener** enables listening for an incoming TCP connection request with its **Start()** method. When a connection request arrives, you can use the **AcceptSocket()** method to return a socket for communication with the remote machine.

To demonstrate the operation of these classes, you need to build two applications. In the first application, we'll set up a socket connection on a specific port to implement the file sending functionality and in the second application, we'll listen for that connection to receive the file.

**TcpSend Application**

Here, in the following demonstration, we are setting up a socket connection on port 2112 to send an uploaded file to the server from the localhost. This example creates the TcpClient using a host name and a port number. After retrieving an instance of the NetworkStream class, you open the source code file and begin to read bytes.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace TcpAction
{
      public partial class tcpSend : Form
      {
          public tcpSend()
           {
               InitializeComponent();
           }

           private void btnSend_Click(object sender, EventArgs e)
          {
                string HostName = txtHost.Text;
               int prt = Int32.Parse(txtPort.Text);

               TcpClient tc = new TcpClient(HostName,prt);

               NetworkStream ns = tc.GetStream();
```

```

                FileStream fs = File.Open(txtFile.Text, FileMode.Open
);

                int data = fs.ReadByte();

                while (data != -1)
                 {
                    ns.WriteByte((byte)data);
                     data = fs.ReadByte();
                }
                 fs.Close();
                ns.Close();
                 tc.Close();
            }

            private void btnOpen_Click(object sender, EventArgs e)
             {
                OpenFileDialog ofd = new OpenFileDialog();
                 ofd.ShowDialog();
                string str = ofd.FileName;
                 txtFile.Text = str.ToString();
            }
        }
 }
```

**TcpReceive Application**

On the other side of the connection, the **Tcp Receive** application showcases the received file after the transmission is finished. Here, you use the **TcpClient** object returned by **AcceptTcpClient()** to open a new stream for reading. We are creating a **StreamReader** class to convert the incoming network data into a string.

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Windows.Forms;

namespace tcpReceive
{
      public partial class Form1 : Form
      {
          public delegate void testDelegate(string str);
           public Form1()
          {
            InitializeComponent();
            Thread t = new Thread(new ThreadStart(ListenData));
            t.Start();
        }

        public void test(string str)
        {
            txtData.Text = str;
        }
        public void ListenData()
        {
            IPAddress ipad = IPAddress.Parse("127.0.0.1");
            Int32 prt = 2112;
            TcpListener tl = new TcpListener(ipad, prt);
            tl.Start();

            TcpClient tc = tl.AcceptTcpClient();

            NetworkStream ns = tc.GetStream();
            StreamReader sr = new StreamReader(ns);

            string result = sr.ReadToEnd();
            Invoke(new testDelegate(test), new object[] { result });

            tc.Close();
            tl.Stop();
        }
    }
}
```

After successfully compiling both files, first run the **Tcp Receive** application to put it into listen mode to receive the file, then execute the Tcp **Send Data** application and later mention there the host name as 127.0.0.1 (localhost), port (2112) and upload a file, finally hitting the **Send Data** button.

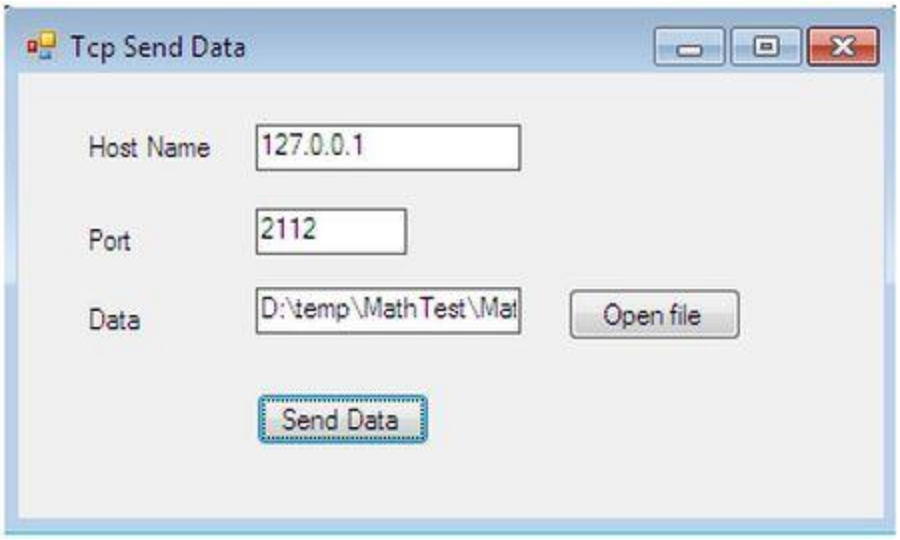
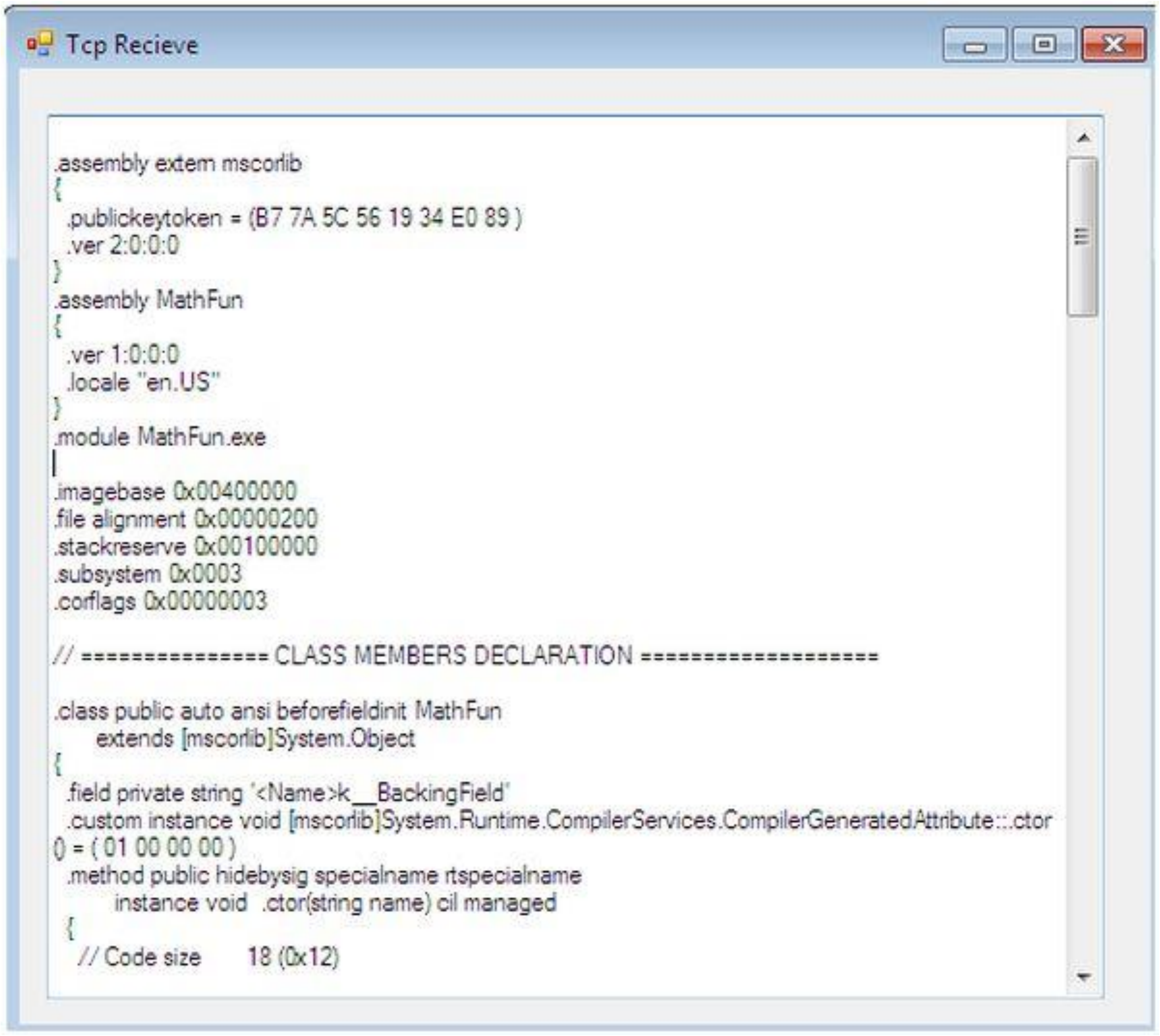


Figure 1.3 Data Sending

When we upload a file and send it across the network, the **Tcp Receive** application receives this file and displays its contents into a TextBox as follows:



```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
.assembly MathFun
{
  .ver 1:0:0:0
  .locale "en.US"
}
.module MathFun.exe

.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003
.corflags 0x00000003

// =============== CLASS MEMBERS DECLARATION ===================

.class public auto ansi beforefieldinit MathFun
       extends [mscorlib]System.Object
{
  .field private string '<Name>k__BackingField'
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::.ctor
() = ( 01 00 00 00 )
  .method public hidebysig specialname rtspecialname
          instance void  .ctor(string name) cil managed
  {
    // Code size       18 (0x12)
```

Figure 1.4 Data Receiving

## Socket in Depth

It is essential to describe a brief overview and history of sockets on Microsoft Windows before we begin describing the details of the .NET socket classes. A socket was initially created for the Berkeley Software Distribution (BSD) of UNIX. A version of sockets for Microsoft Windows called WinSock 1.1 was initially released in 1992 and is currently on version 2.0; In order to allow access to the underlying sockets interface, Microsoft implemented a .NET Socket class, that is a wrapper around the WinSock socket functions and has most of the versatility (and complexity) of sockets interface exposed. Then three higher-level socket classes, **TcpClient**, **TcpListener** and **UdpClient**, were implemented by using the .NET Socket wrapper class.

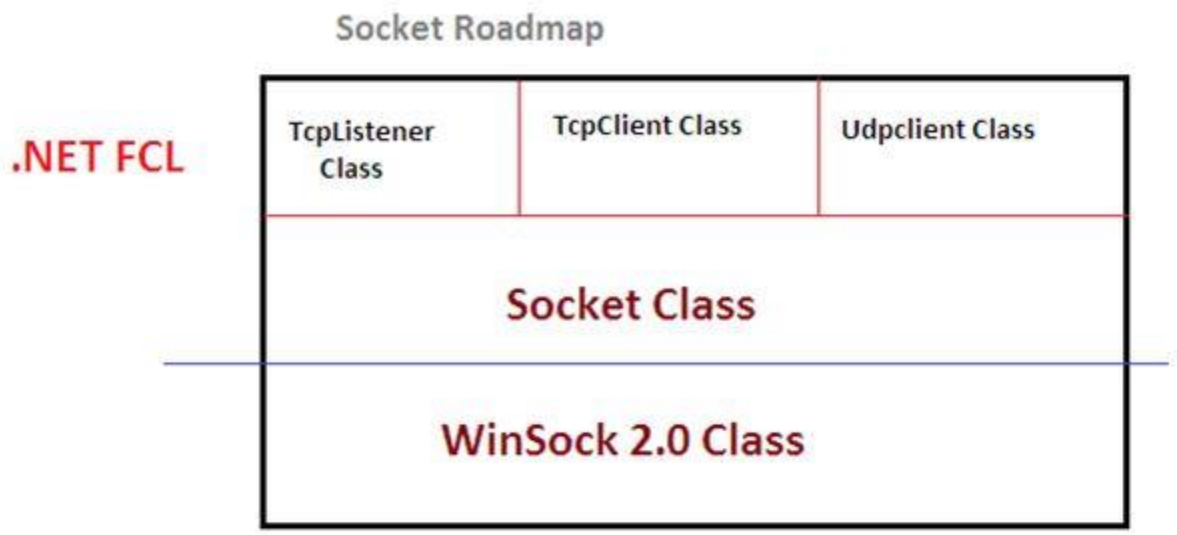


Figure 1.5 .NET Socket Classes

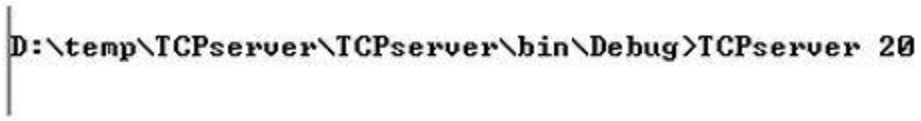
## TCP Socket Server

The server's job is to set up endpoint for clients to connect to and passively wait for connections. The typical TCP server first constructs a **TcpListener** instance, specifying the local address and port and calls the Start() method. This socket listens for incoming connections on the specified port then calls the **AcceptTcpClient()** method of the **TcpListener** class to get the next incoming client connection. Upon establishment of a new client connection, an instance of **TcpClient** for the new connection is created and returned by the **AcceptTcpClient()** call. Thereafter, communication with the client is done using the Read() and Write() methods of the **NetworkStream** class and finally the client socket connection and stream are closed using the Close() methods of **NetworkStream** and **TcpClient.**

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

namespace TCPserver
{
    class Program
     {
        private const int BUFSIZE = 32;

        static void Main(string[] args)
        {
            if (args.Length > 1) // Test for correct of args
                throw new ArgumentException("Parameters: [<Port>]");

            int servPort = (args.Length == 1) ? Int32.Parse(args[0]) : 7;

            TcpListener listener = null;

            try
            {
                // Create a TCPListener to accept client connections
                listener = new TcpListener(IPAddress.Any, servPort);
                listener.Start();
            }
            catch (SocketException se)
            {
                Console.WriteLine(se.Message);
                Environment.Exit(se.ErrorCode);
            }

            byte[] rcvBuffer = new byte[BUFSIZE]; // Receive buffer
            int bytesRcvd; // Received byte count

            for (; ; )
            { // Run forever, accepting and servicing connections

                TcpClient client = null;
                NetworkStream ns = null;
                try
                {
                    client = listener.AcceptTcpClient(); // Get client connection
                    ns = client.GetStream();
                    Console.Write("Handling client - ");

                    // Receive until client closes connection
                    int totalBytesEchoed = 0;
```

```
                      while ((bytesRcvd = ns.Read(rcvBuffer, 0, rcvBuffer.Length)) > 0)
                        {
                          ns.Write(rcvBuffer, 0, bytesRcvd);
                            totalBytesEchoed += bytesRcvd;
                      }
                        Console.WriteLine("echoed {0} bytes.", totalBytesEchoed);

                        ns.Close();
                      client.Close();

                  }
                    catch (Exception e)
                  {
                        Console.WriteLine(e.Message);
                      ns.Close();
                    }
              }
          }
      }
  }
```

Now, compile this application and open a command prompt. Here specify the application name with a port such as 20 (if you don't specify any port then the default is 7) on which you want to establish communication as in the following:

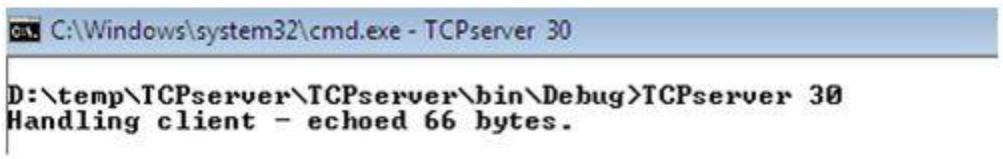


```
D:\temp\TCPserver\TCPserver\bin\Debug>TCPserver 20
```

To test whether the standard echo server is running or not, telnet to that port such as 30 on the server as in the following:

```
Telnet 127.0.0.1 30
```

Enter some data at the Telnet terminal and exit. If the server establishes a socket connection on port 30 properly then the server will respond by showing the entered character bytes as follows:



```
C:\Windows\system32\cmd.exe - TCPserver 30

D:\temp\TCPserver\TCPserver\bin\Debug>TCPserver 30
Handling client - echoed 66 bytes.
```

Figure 1.6 Server in listening mode

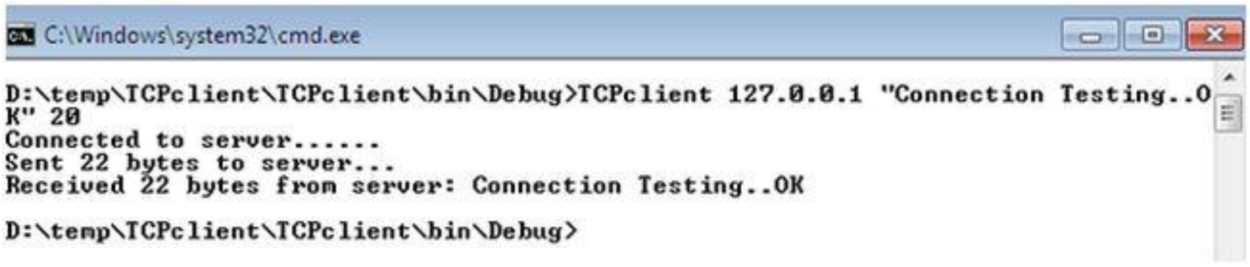
## TCP Client Socket

A TCP client initiates communication with a server that is passively waiting to be contacted. In TCP the client typically goes through the three steps. First construct an instance of **TcpClient**. A TCP connection can be created implicitly in the constructor by specifying the remote host and port, or explicitly using the Connect() method. Second, communicate using the socket's stream. A connected instance of **TcpClient** contains a **NetworkStream** that can be used like any other .NET I/O stream. Finally, close the connection by calling the Close() method of **TcpClient**.

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

namespace TCPclient
{
    class Program
     {
          static void Main(string[] args)
         {
              if ((args.Length < 2) || (args.Length > 3))
             {
                  throw new ArgumentException("Parameters: <Server> <Word> [<Port>]");
             }

             String server = args[0]; // Server name or IP address

             // Convert input String to bytes
              byte[] byteBuffer = Encoding.ASCII.GetBytes(args[1]);

              // Use port argument if supplied, otherwise default to 7
             int servPort = (args.Length == 3) ? Int32.Parse(args[2]) : 7;

             TcpClient client = null;
              NetworkStream ns = null;

          try
         {
             // Create socket that is connected to server on specified port
            client = new TcpClient(server, servPort);

            Console.WriteLine("Connected to server......");

            ns = client.GetStream();

            // Send the encoded string to the server
             ns.Write(byteBuffer, 0, byteBuffer.Length);

             Console.WriteLine("Sent {0} bytes to server...", byteBuffer.Length);

             int totalBytesRcvd = 0; // Total bytes received so far
            int bytesRcvd = 0; // Bytes received in last read

            // Receive the same string back from the server
             while (totalBytesRcvd < byteBuffer.Length)
            {
                if ((bytesRcvd = ns.Read(byteBuffer, totalBytesRcvd,
              byteBuffer.Length - totalBytesRcvd)) == 0)
                {
                  Console.WriteLine("Connection closed prematurely.");
                    break;
               }
                totalBytesRcvd += bytesRcvd;
            }
             Console.WriteLine("Received {0} bytes from server: {1}", totalBytesRcvd,
             Encoding.ASCII.GetString(byteBuffer, 0, totalBytesRcvd));

         }
          catch (Exception e)
         {
             Console.WriteLine(e.Message);
         }
```

```
                finally
               {
                   ns.Close();
                  client.Close();
                }

            }
        }
}
```

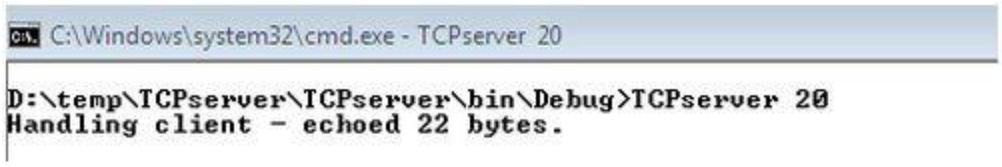
Finally, open the TCPclient.exe in the command prompt and send some string data via port 20 to server 127.0.0.1. It will echo back the output as follows:



```
D:\temp\TCPclient\TCPclient\bin\Debug>TCPclient 127.0.0.1 "Connection Testing..O
K" 20
Connected to server......
Sent 22 bytes to server...
Received 22 bytes from server: Connection Testing..OK

D:\temp\TCPclient\TCPclient\bin\Debug>
```

Figure 1.7 Connecting Sockets

At first, it is mandatory to keep the server in listening mode before executing the client. Once the data begins to come from the client side, the total number of entered bytes yields immediately as followis:



```
D:\temp\TCPserver\TCPserver\bin\Debug>TCPserver 20
Handling client - echoed 22 bytes.
```

Figure 1.8 Server in listening mode

You can test that the socket was establish on port 20 via the Windows netstat command to ensure the client server socket connectivity as follows:

```
C:\Windows\system32>netstat -nabo

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:20             0.0.0.0:0              LISTENING       900
 [TCPserver.exe]
```

Figure 1.9 socket testing

## Synopsis

This article has described socket programming using **TcpListener** and **TcpClient** classes to create client-server applications. Here, we have learned how to handle the details of establishing a network connection between client and server applications, moreover how to send data between the server and client so they can do useful operations. You have also encountered the essential .NET base classes of the **System.Net** namespace that deals with opening the client connection on the network.

## Socket programming in c using TCP/IP

Before you start learning socket programming in C, you should basic knowledge of IP addresses, TCP, and UDP. In this article, I shall describe TCP/IP and write a socket program using the TCP/IP API.

### TCP (Transmission control protocol)

A TCP (transmission control protocol) is a connection-oriented communication. It is an intermediate layer of the application layer and internet protocol layer in the OSI model. TCP is designed to send the data packets over the network. It ensures that data is delivered to the correct destination.

TCP creates a connection between the source and destination node before transmitting the data and keeps the connection alive until the communication is active.

In TCP before sending the data it breaks the large data into smaller packets and cares the integrity of the data at the time of reassembling at the destination node. Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

TCP also offers the facility of retransmission, when a TCP client sends data to the server, it requires an acknowledgment in return. If an acknowledgment is not received, after a certain amount of time transmitted data will be lost and TCP automatically retransmits the data.

The communication over the network in TCP/IP model takes place in form of a client-server architecture. ie, the client begins the communication and establishes a connection with a server.

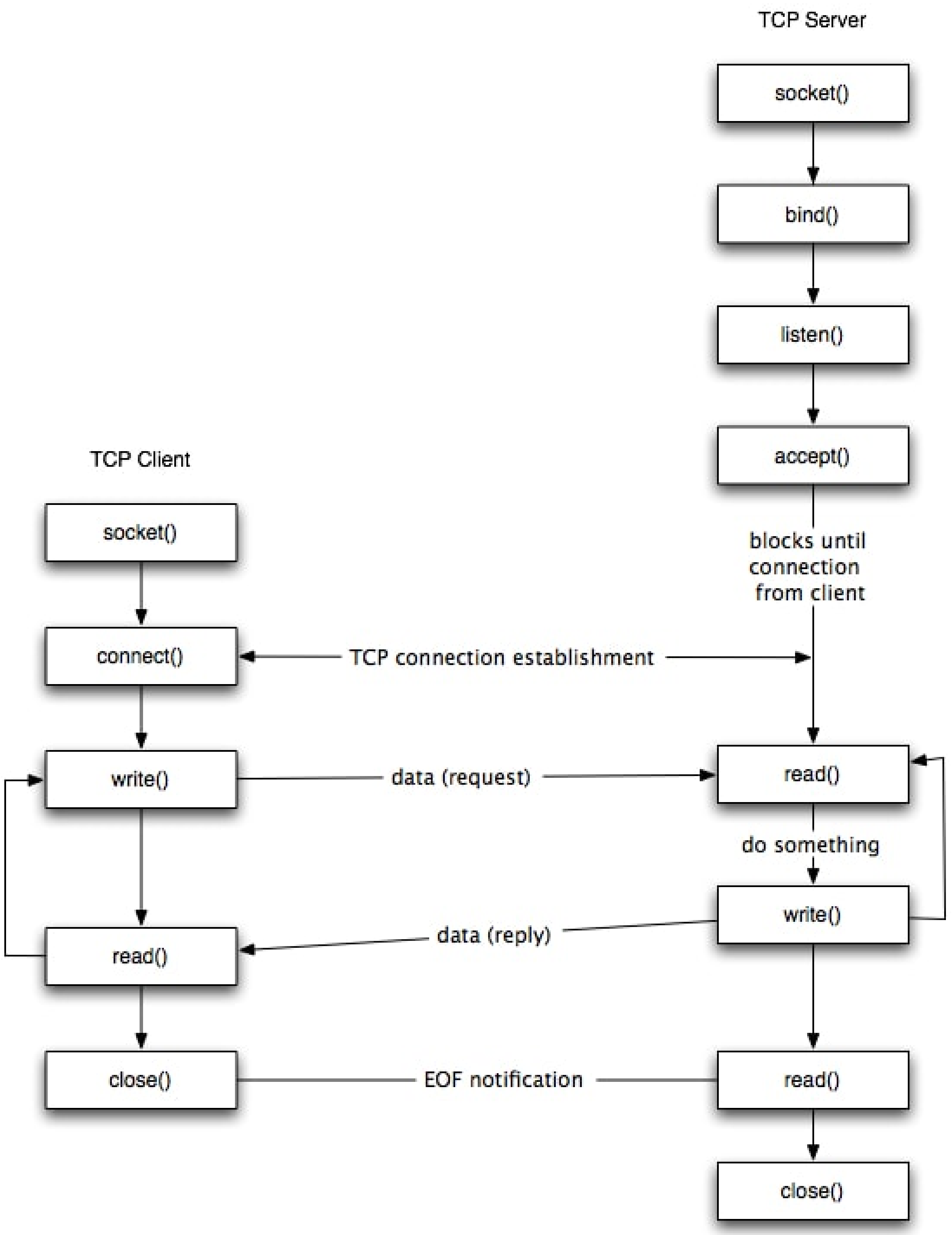
For more understanding let's create a server that continuously runs and establish the connection after getting a request from the client.

**Note:** here I am creating the server and client for Linux.

In this example, After the connection with a client, the server will wait for a message from the client. After getting the message server will check the received message and send a proper response as per the received message.

## Sequence of socket API calls and data flow:

To better understand check out the sequence of socket API calls and data flow for TCP client-server communication. The left-hand column represents the client and the right-hand side is the server.


TCP Server
socket()
bind()
listen()
accept()
blocks until connection from client
TCP Client
socket()
connect()
TCP connection establishment
write()
data (request)
read()
do something
write()
data (reply)
read()
close()
EOF notification
read()
close()

## Steps to create a client using TCP/IP  API

- Create a socket with the socket() system call.
- Initialize the socket address structure as per the server and connect the socket to the address of the server using the connect() system call.
- Receive and send the data using the recv() and send().
- Close the connection by calling the close() function.

## Steps to create a server using TCP/IP API

- Create a socket with the socket() system call.
- Initialize the socket address structure and bind the socket to an address using the bind() system call.
- Listen for connections with the listen() system call.
- Accept a connection with the accept() system call. This call typically blocks until a client connects to the server.
- Receive and send data by using the recv() and send().
- Close the connection by using the close().

If you are a beginner and want to learn TCP/IP, then you can check this course "**TCP/IP Networking for Developers**" that created by **Steve Evans**, and the **rating** of this course is around **4.7**. The good thing is that **FREE TRIAL** is available and you can also access thousands of courses that are created by industry experts.

**Today Grab your free Trial**

## Example of Socket programming in C using TCP/IP:

As we know in socket programming network nodes (sockets) are communicating with each other over the network. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. In this example code, we will create two-node, one node for the server and the other for the client. So let's see the example code for client-server socket programming in C.

## Example Source code for TCP/IP client in C Linux:

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<unistd.h>

//Create a Socket for server communication

short SocketCreate(void)

{

   short hSocket;

   printf("Create the socket\n");

   hSocket = socket(AF_INET, SOCK_STREAM, 0);

   return hSocket;

}

//try to connect with server

int SocketConnect(int hSocket)

{

   int iRetval=-1;

   int ServerPort = 90190;

   struct sockaddr_in remote= {0};

   remote.sin_addr.s_addr = inet_addr("127.0.0.1"); //Local Host
```

```
    remote.sin_family = AF_INET;

    remote.sin_port = htons(ServerPort);

    iRetval = connect(hSocket,(struct sockaddr *)&remote,sizeof(struct sockaddr_in));

    return iRetval;

}

// Send the data to the server and set the timeout of 20 seconds

int SocketSend(int hSocket,char* Rqst,short lenRqst)

{

    int shortRetval = -1;

    struct timeval tv;

    tv.tv_sec = 20;  /* 20 Secs Timeout */

    tv.tv_usec = 0;

    if(setsockopt(hSocket,SOL_SOCKET,SO_SNDTIMEO,(char *)&tv,sizeof(tv)) < 0)

    {

        printf("Time Out\n");

        return -1;

    }

    shortRetval = send(hSocket, Rqst, lenRqst, 0);

    return shortRetval;

}

//receive the data from the server

int SocketReceive(int hSocket,char* Rsp,short RvcSize)

{

    int shortRetval = -1;

    struct timeval tv;

    tv.tv_sec = 20;  /* 20 Secs Timeout */

    tv.tv_usec = 0;
    if(setsockopt(hSocket, SOL_SOCKET, SO_RCVTIMEO,(char *)&tv,sizeof(tv)) < 0)
    {
        printf("Time Out\n");
        return -1;
    }
    shortRetval = recv(hSocket, Rsp, RvcSize, 0);
    printf("Response %s\n",Rsp);
    return shortRetval;
}
//main driver program
int main(int argc, char *argv[])
{
    int hSocket, read_size;
    struct sockaddr_in server;
    char SendToServer[100] = {0};
    char server_reply[200] = {0};
    //Create socket
    hSocket = SocketCreate();
    if(hSocket == -1)
    {
        printf("Could not create socket\n");
        return 1;
    }
    printf("Socket is created\n");
    //Connect to remote server
    if (SocketConnect(hSocket) < 0)
    {
        perror("connect failed.\n");
        return 1;
    }
    printf("Sucessfully conected with server\n");
    printf("Enter the Message: ");
    gets(SendToServer);
    //Send data to the server
    SocketSend(hSocket, SendToServer, strlen(SendToServer));
    //Received the data from the server
    read_size = SocketReceive(hSocket, server_reply, 200);
    printf("Server Response : %s\n\n",server_reply);
    close(hSocket);
    shutdown(hSocket,0);
    shutdown(hSocket,1);
    shutdown(hSocket,2);
    return 0;
}
```

## Example Source code for TCP/IP server in C Linux:

```
#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<unistd.h>

short SocketCreate(void)

{

    short hSocket;

    printf("Create the socket\n");

    hSocket = socket(AF_INET, SOCK_STREAM, 0);

    return hSocket;

}

int BindCreatedSocket(int hSocket)

{

    int iRetval=-1;

    int ClientPort = 90190;

    struct sockaddr_in  remote= {0};

    /* Internet address family */

    remote.sin_family = AF_INET;

    /* Any incoming interface */

    remote.sin_addr.s_addr = htonl(INADDR_ANY);

    remote.sin_port = htons(ClientPort); /* Local port */

    iRetval = bind(hSocket,(struct sockaddr *)&remote,sizeof(remote));

    return iRetval;

}

int main(int argc, char *argv[])

{
```

```
int socket_desc, sock, clientLen, read_size;

struct sockaddr_in server, client;

char client_message[200]= {0};

char message[100] = {0};

const char *pMessage = "hello aticleworld.com";

//Create socket

socket_desc = SocketCreate();

if (socket_desc == -1)

{

    printf("Could not create socket");

    return 1;

}

printf("Socket created\n");

//Bind

if( BindCreatedSocket(socket_desc) < 0)

{

    //print the error message

    perror("bind failed.");

    return 1;

}

printf("bind done\n");

//Listen

listen(socket_desc, 3);

//Accept and incoming connection

while(1)

{

    printf("Waiting for incoming connections...\n");
```

```
clientLen = sizeof(struct sockaddr_in);

//accept connection from an incoming client

sock = accept(socket_desc,(struct sockaddr *)&client,(socklen_t*)&clientLen);

if (sock < 0)

{

   perror("accept failed");

   return 1;

}

printf("Connection accepted\n");

memset(client_message, '\0', sizeof client_message);

memset(message, '\0', sizeof message);

//Receive a reply from the client

if( recv(sock, client_message, 200, 0) < 0)

{

   printf("recv failed");

   break;

}

printf("Client reply : %s\n",client_message);

if(strcmp(pMessage,client_message)==0)

{

   strcpy(message,"Hi there !");

}

else

{

   strcpy(message,"Invalid Message !");

}

// Send some data
```

```
        if( send(sock, message, strlen(message), 0) < 0)
        {
            printf("Send failed");
            return 1;
        }
        close(sock);
        sleep(1);
    }
    return 0;
}
```

# Thực hành: Lập trình socket Tcp cơ bản (+video)

Đăng bởi
**Mai Chi**
-
23/01/2019
0

Facebook

Twitter

Pinterest

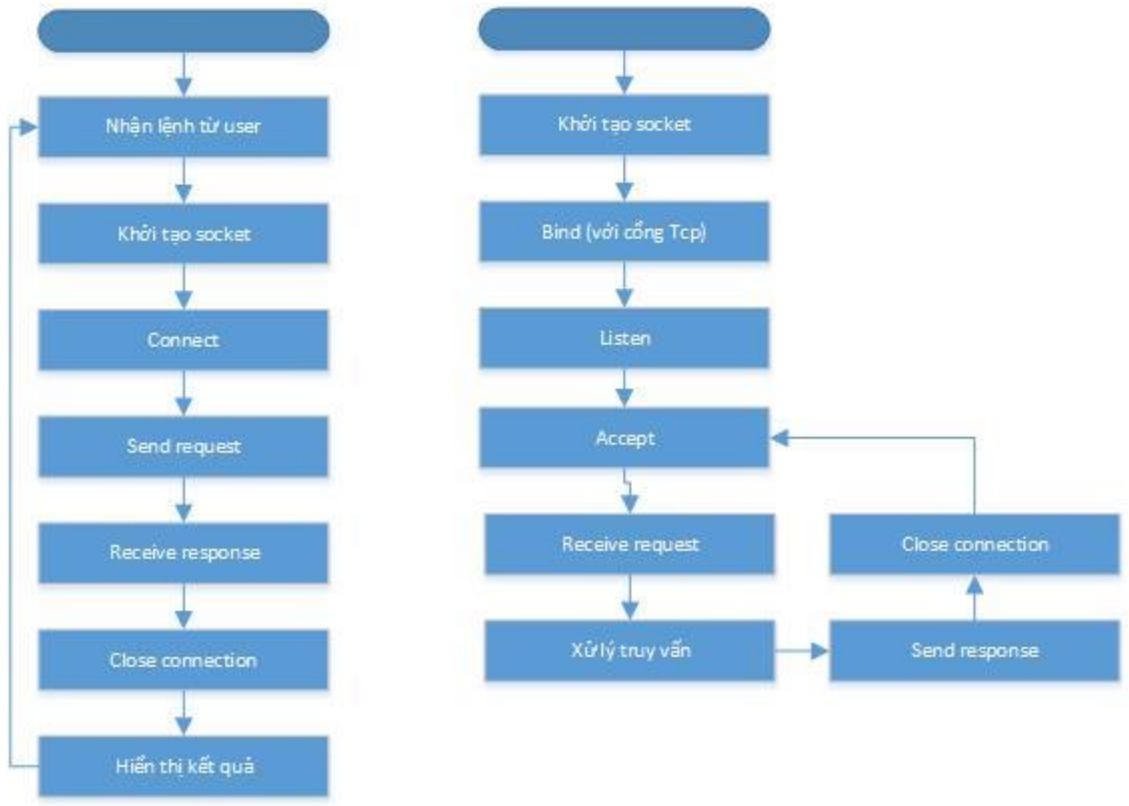
Linkedin

ReddIt

Hướng dẫn tự học lập trình socket (mạng) với C#  Thực hành: Lập trình socket Tcp cơ bản (+video)

Tương tự như đối với socket Udp, chúng ta cũng bắt đầu tiếp xúc với socket Tcp thông qua thực hiện một bài thực hành. Bài thực hành này cũng có chung yêu cầu với bài thực hành đã thực hiện đối với socket Udp: xây dựng một bộ ứng dụng theo mô hình client/server cho phép biến đổi một chuỗi ký tự người dùng nhập thành dạng chữ in hoa.

Sơ đồ flowchart của bài toán này như sau:



Sơ đồ flowchart các bước khi lập trình với Tcp socket

**Bước 1**. Tạo một Solution trống đặt tên là TcpSocket

**Bước 2**. Xây dựng lần lượt hai project thuộc kiểu Console App trong solution trên với tên gọi lần lượt là Server và Client.

**Bước 3**. Thiết lập để có thể debug đồng thời cả hai chương trình, trong đó Server sẽ khởi động trước.

**Bước 4**. Mở file Program.cs của Client và viết code như sau:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace Client
{
  internal class Program
  {
    private static void Main(string[] args)
```

```
{
    Console.Title = "Tcp Client";
    // yêu cầu người dùng nhập ip của server
    Console.Write("Server IP address: ");
    var serverIpStr = Console.ReadLine();
    // chuyển đổi chuỗi ký tự thành object thuộc kiểu IPAddress
    var serverIp = IPAddress.Parse(serverIpStr);
    // yêu cầu người dùng nhập cổng của server
    Console.Write("Server port: ");
    var serverPortStr = Console.ReadLine();
    // chuyển chuỗi ký tự thành biến kiểu int
    var serverPort = int.Parse(serverPortStr);
    // đây là "địa chỉ" của tiến trình server trên mạng
    // mỗi endpoint chứa ip của host và port của tiến trình
    var serverEndpoint = new IPEndPoint(serverIp, serverPort);
    var size = 1024; // kích thước của bộ đệm
    var receiveBuffer = new byte[size]; // mảng byte làm bộ đệm
    while (true)
    {
        // yêu cầu người dùng nhập một chuỗi
        Console.ForegroundColor = ConsoleColor.Green;
        Console.Write("# Text >>> ");
        Console.ResetColor();
        var text = Console.ReadLine();
        // khởi tạo object của lớp socket để sử dụng dịch vụ Tcp
        // lưu ý SocketType của Tcp là Stream
        var socket = new Socket(SocketType.Stream, ProtocolType.Tcp);
        // tạo kết nối tới Server
        socket.Connect(serverEndpoint);
        // biến đổi chuỗi thành mảng byte
        var sendBuffer = Encoding.ASCII.GetBytes(text);
        // gửi mảng byte trên đến tiến trình server
        socket.Send(sendBuffer);
        // không tiếp tục gửi dữ liệu nữa
        socket.Shutdown(SocketShutdown.Send);
        // nhận mảng byte từ dịch vụ Tcp và lưu vào bộ đệm
        var length = socket.Receive(receiveBuffer);
        // chuyển đổi mảng byte về chuỗi
        var result = Encoding.ASCII.GetString(receiveBuffer, 0, length);
        // xóa bộ đệm (để lần sau sử dụng cho yên tâm)
```

```
            Array.Clear(receiveBuffer, 0, size);
            // không tiếp tục nhận dữ liệu nữa
            socket.Shutdown(SocketShutdown.Receive);
            // đóng socket và giải phóng tài nguyên
            socket.Close();
            // in kết quả ra màn hình
            Console.WriteLine($">>> {result}");
         }
      }
   }
}
```

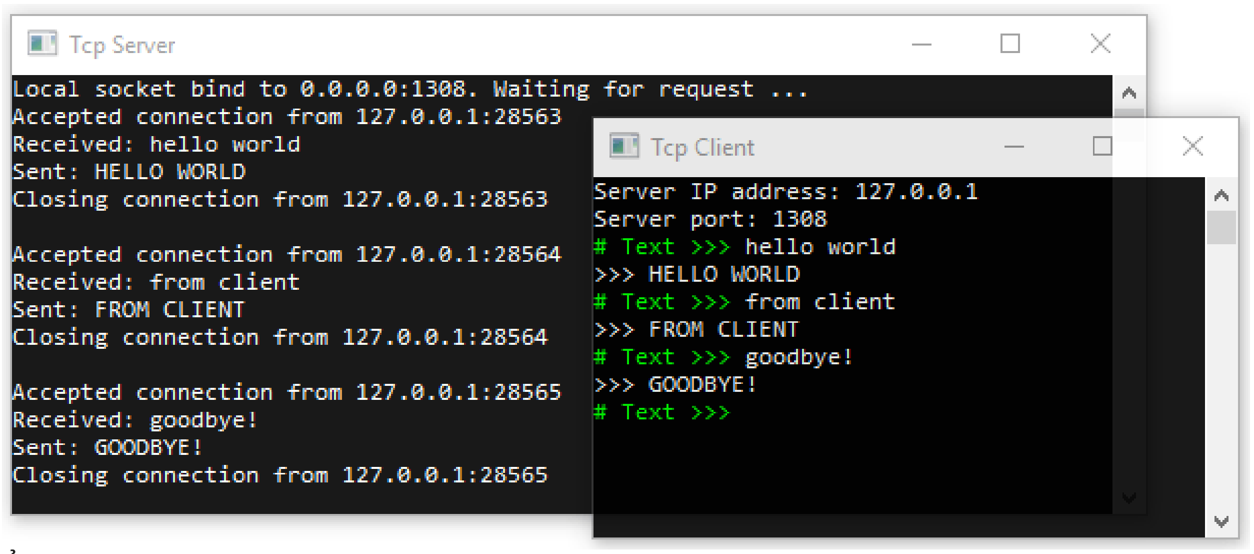
**Bước 5**. Mở file Program.cs của Server và viết code như sau:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace Server
{
   internal class Program
   {
      private static void Main(string[] args)
      {
         Console.Title = "Tcp Server";
         // giá trị Any của IPAddress tương ứng với Ip của tất cả các giao diện mạng trên máy
         var localIp = IPAddress.Any;
         // tiến trình server sẽ sử dụng cổng tcp 1308
         var localPort = 1308;
         // biến này sẽ chứa "địa chỉ" của tiến trình server trên mạng
         var localEndPoint = new IPEndPoint(localIp, localPort);
         // tcp sử dụng đồng thời hai socket:
         // một socket để chờ nghe kết nối, một socket để gửi/nhận dữ liệu
         // socket listener này chỉ làm nhiệm vụ chờ kết nối từ Client
         var listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
         // yêu cầu hệ điều hành cho phép chiếm dụng cổng tcp 1308
         // server sẽ nghe trên tất cả các mạng mà máy tính này kết nối tới
         // chỉ cần gói tin tcp đến cổng 1308, tiến trình server sẽ nhận được
         listener.Bind(localEndPoint);
         // bắt đầu lắng nghe chờ các gói tin tcp đến cổng 1308
```

```
        listener.Listen(10);
        Console.WriteLine($"Local socket bind to {localEndPoint}. Waiting for request ...");
        var size = 1024;
        var receiveBuffer = new byte[size];
        while (true)
        {
          // tcp đòi hỏi một socket thứ hai làm nhiệm vụ gửi/nhận dữ liệu
          // socket này được tạo ra bởi lệnh Accept
          var socket = listener.Accept();
          Console.WriteLine($"Accepted connection from {socket.RemoteEndPoint}");
          // nhận dữ liệu vào buffer
          var length = socket.Receive(receiveBuffer);
          // không tiếp tục nhận dữ liệu nữa
          socket.Shutdown(SocketShutdown.Receive);
          var text = Encoding.ASCII.GetString(receiveBuffer, 0, length);
          Console.WriteLine($"Received: {text}");
          // chuyển chuỗi thành dạng in hoa
          var result = text.ToUpper();
          var sendBuffer = Encoding.ASCII.GetBytes(result);
          // gửi kết quả lại cho client
          socket.Send(sendBuffer);
          Console.WriteLine($"Sent: {result}");
          // không tiếp tục gửi dữ liệu nữa
          socket.Shutdown(SocketShutdown.Send);
          // đóng kết nối và giải phóng tài nguyên
          Console.WriteLine($"Closing connection from {socket.RemoteEndPoint}rn");
          socket.Close();
          Array.Clear(receiveBuffer, 0, size);
        }
      }
    }
}
```

Chúng ta có thể để ý, code của cả Client và Server đều tái sử dụng phần khung của nội dung thực hành 1 Udp nhưng thay thế phần truyền thông Udp bằng Tcp. So sánh với code của thực hành 1 Udp để thấy được sự khác biệt.

**Bước 6**. Dịch và chạy thử hệ thống



```
Local socket bind to 0.0.0.0:1308. Waiting for request ...
Accepted connection from 127.0.0.1:28563
Received: hello world
Sent: HELLO WORLD
Closing connection from 127.0.0.1:28563

Accepted connection from 127.0.0.1:28564
Received: from client
Sent: FROM CLIENT
Closing connection from 127.0.0.1:28564

Accepted connection from 127.0.0.1:28565
Received: goodbye!
Sent: GOODBYE!
Closing connection from 127.0.0.1:28565
```

```
Server IP address: 127.0.0.1
Server port: 1308
# Text >>> hello world
>>> HELLO WORLD
# Text >>> from client
>>> FROM CLIENT
# Text >>> goodbye!
>>> GOODBYE!
# Text >>>
```

Ảnh chụp màn hình chương trình tcp client/server

Chương trình Client về hình thức bề ngoài không có gì khác biệt so với chương trình viết ở phần thực hành 1 của Udp. Tuy nhiên, phần truyền thông đã thay đổi hoàn toàn.

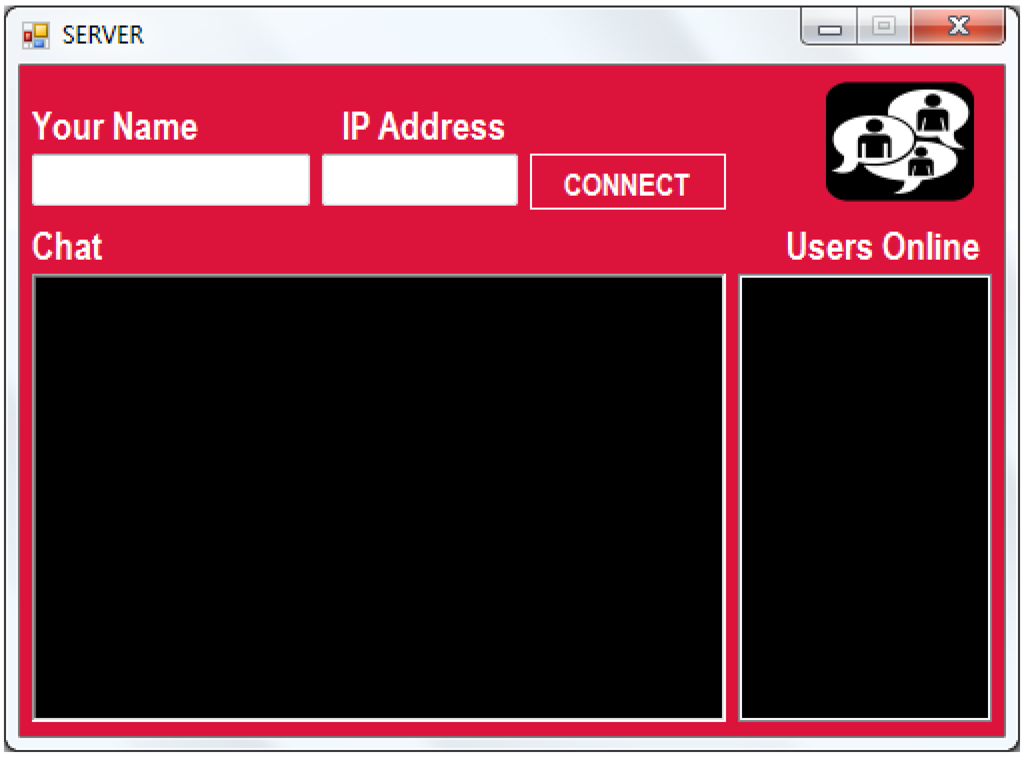
Trong phần tiếp theo chúng ta sẽ phân tích code của Client và Server để hiểu được cách lập trình với socket Tcp.

# Create Chat Group Using VB.Net

4 YEARS AGO
7 MINUTE
READ

Hello friends, in this post I will discuss about how to create a simple chat group app that is connected on the same network, using Visual Studio IDE. Such an application is certainly very useful to exchange information, for example if two employees who want to communicate but the first employee is on the 1st floor while the other employees are on the 2nd floor. That's one example of the benefits of this kind of application and of course there are still benefits other.

Now open and create a new project in Visual Studio, then design a form as server more or less as shown below

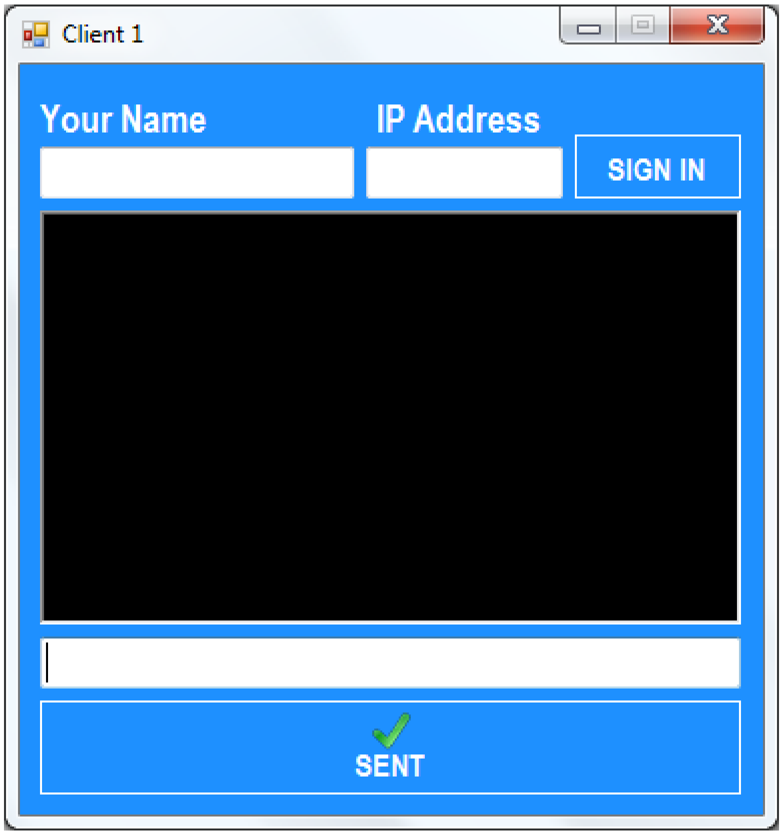


**NOTES :**

Componen Form Server :

- 4 Label
- 2 TextBox
- 2 RichTextBox
- 1 Button
- 1 PictureBox

Next, create a new project again and design the form as a client more or less like in the picture below



**NOTES :**

Componen Form Client :

- 2 Label
- 3 TextBox
- 2 Button
- 1 RichTextBox

Now, we go back to the project form server, then create a module and name it **CPcontrolClass** and enter the following code

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Threading
Imports System.IO

Public Class CPcontrolClass
    Public Event MessageReceived(ByVal sender As CPcontrolClass, ByVal Data
As String)
```

```
    Public serverIP As IPAddress = IPAddress.Parse("127.0.0.1")
    Public serverPORT As Integer = 8080
    Public server As TcpListener
    Private comThread As Thread
    Public isListening As Boolean = True
    Private client As TcpClient
    Private clientData As StreamReader

    Public Sub New()
        server = New TcpListener(serverIP, serverPORT)
        server.Start()
        comThread = New Thread(New ThreadStart(AddressOf Listening))
        comThread.Start()
    End Sub
    Public Sub ipnya()
        Dim addrs As IPAddress
        Dim x As Byte = 0
        Do
            addrs = Dns.GetHostEntry(Dns.GetHostName).AddressList(x)
            x += 1
        Loop Until (addrs.AddressFamily = AddressFamily.InterNetwork)
    End Sub
    Private Sub Listening()
        Do Until isListening = False
            If server.Pending = True Then
                client = server.AcceptTcpClient
                clientData = New StreamReader(client.GetStream)
            End If

            Try
                RaiseEvent MessageReceived(Me, clientData.ReadLine)
                My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Beep)
            Catch ex As Exception
            End Try
            Thread.Sleep(100)
        Loop
    End Sub
End Class
```

**NOTES :**

**127.0.0.1** is the local IP Address, if you want to use with a different device, please replace it with your device's ip address.

After that, import some of the required libraries on the server form

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Runtime.InteropServices
Imports System.Threading
Imports System.IO

Public Class Form1

End Class
```

Then, declare some global variables

```
Public Class Form1
    Private server As CPcontrolClass
    Dim log As String
    Dim ipserver As IPAddress
    Private TCPListener As TcpListener
    Private clientCollection As New Hashtable()
    Private usernameCollection As New Hashtable()
    Private Shared connectID As Long = 0
    Private serverThread As Thread
    Public Delegate Sub updateListboxDelegate(ByVal str As String)
    Dim clientThread As System.Threading.Thread
    Dim client As New TcpClient
    Dim reader As BinaryReader
    Dim writer As BinaryWriter
    Dim message As String
    Dim username As String
    Delegate Sub updateMessageDelegate(ByVal message As String)
    Delegate Sub updateUserDelegate(ByVal message As String)
End Class
```

Then, create some of the following functions

```
Public Structure ClientData
        Public structSocket As TcpClient
        Public structThread As Thread
        Public structReader As BinaryReader
        Public structWriter As BinaryWriter
    End Structure

    Sub updateMessageWork(ByVal message As String)
        RichTextBox1.AppendText(message + vbNewLine)
    End Sub

    Sub updateMessage(ByVal message As String)
        If RichTextBox1.InvokeRequired Then
            Invoke(New updateMessageDelegate(AddressOf updateMessageWork),
message)
        Else
            RichTextBox1.AppendText(message + vbNewLine)
        End If
    End Sub

    Sub updateUserWork(ByVal message As String)
        Dim userlist() As String = Split(message, Chr(13))
        Dim i As Integer
        lbUser.Items.Clear()
        For i = 0 To userlist.Length - 1
            lbUser.Items.Add(userlist(i))
        Next
    End Sub

    Sub updateUser(ByVal message As String)
        If lbUser.InvokeRequired Then
            Invoke(New updateUserDelegate(AddressOf updateUserWork), message)
        End If
    End Sub

    Public Sub readSocket()
        Dim realId As Long = connectID
        Dim CData As New ClientData
        CData = CType(clientCollection(realId), ClientData)
        Dim message As String

        Dim found As Boolean = False

        While True
            If CData.structSocket.Connected Then
                Try
```

```
                    message = CData.structReader.ReadString()
                    If message.Substring(0, 3) = "MES" Then
                        For Each Client As ClientData In
clientCollection.Values
                            If Client.structSocket.Connected Then
                                Client.structWriter.Write(message)
                            End If
                        Next
                    ElseIf message.Substring(0, 3) = "USR" Then
                        For Each user As String In
usernameCollection.Values()
                            If user = message.Substring(4) Then
                                found = True
                            End If
                        Next
                        If found = True Then
                            message = "err"
                            For Each Client As ClientData In
clientCollection.Values()
                                If Client.structSocket.Connected Then
                                    Client.structWriter.Write(message)
                                    Exit For
                                End If
                            Next
                            Try
                                Dim client As ClientData =
CType(clientCollection(realId), ClientData)
                                client.structThread.Abort()
                            Catch e As Exception
                                SyncLock Me
                                    clientCollection.Remove(realId)
                                    usernameCollection.Remove(realId)
                                End SyncLock
                            End Try
                        Else
                            usernameCollection.Add(realId,
message.Substring(4))
                            Dim userlist As String = ""
                            For Each user As String In
usernameCollection.Values()
                                userlist = userlist & user & Chr(13)
                            Next
                            For Each Client As ClientData In
clientCollection.Values()
                                If Client.structSocket.Connected Then
```

```
                                    Client.structWriter.Write("USR " &
userlist)
                                End If
                            Next
                        End If
                    ElseIf message.Substring(0, 3) = "DIS" Then
                        usernameCollection.Remove(realId)
                        Dim userlist As String = ""
                        For Each user As String In
usernameCollection.Values()
                            userlist = userlist & user & Chr(13)
                        Next

                        For Each Client As ClientData In
clientCollection.Values()
                            If Client.structSocket.Connected Then
                                Client.structWriter.Write("USR " & userlist)
                            End If
                        Next
                    End If
                Catch e As Exception
                    Exit While
                End Try
            End If
        End While
        CloseTheThread(realId)
    End Sub
    Private Sub CloseTheThread(ByVal realId As Long)
        Try
            Dim client As ClientData = CType(clientCollection(realId),
ClientData)
            client.structThread.Abort()
        Catch e As Exception
            SyncLock Me
                clientCollection.Remove(realId)
                usernameCollection.Remove(realId)
            End SyncLock
        End Try
    End Sub

    Private Sub OnlineReceived(ByVal sender As CPcontrolClass, ByVal Data As
String)
    End Sub
    Public Sub readSocketClient()
        While True
            Try
```

```
                message = reader.ReadString()
                If message.Substring(0, 3) = "MES" Then
                    updateMessage(message.Substring(4))
                ElseIf message.Substring(0, 3) = "USR" Then
                    updateUser(message.Substring(4))
                ElseIf message.Substring(0, 3) = "DIS" Then
                    updateUser(message.Substring(4))
                End If
                If message = "err" Then
                    Exit Sub
                End If
            Catch e As Exception
                Exit While
            End Try
        End While
    End Sub

    Public Sub writeSocket(ByVal message As String)
        If client.Connected Then
            writer.Write(message)
        End If
    End Sub
    Sub startConnect()
        Try
            client = New TcpClient()
            client.Connect(ipserver, 8888)
            reader = New BinaryReader(client.GetStream())
            writer = New BinaryWriter(client.GetStream())

            writer.Write("USR  " + TextBox1.Text)
            username = TextBox1.Text
            clientThread = New System.Threading.Thread(AddressOf
readSocketClient)
            clientThread.Start()
            server = New CPcontrolClass
            AddHandler server.MessageReceived, AddressOf OnlineReceived

            RichTextBox1.Visible = True
            lbUser.Visible = True



        Catch ex As Exception
            Exit Sub
        End Try
    End Sub
    Public Sub New()
        InitializeComponent()
    End Sub
    Public Sub waitingForClient()
        Dim CData As New ClientData
        While True
            CData.structSocket = TCPListener.AcceptTcpClient
            Interlocked.Increment(connectID)
            CData.structThread = New Thread(AddressOf readSocket)
            CData.structReader = New
BinaryReader(CData.structSocket.GetStream())
            CData.structWriter = New
BinaryWriter(CData.structSocket.GetStream())
            SyncLock Me
                clientCollection.Add(connectID, CData)
            End SyncLock
            CData.structThread.Start()
        End While
    End Sub
```

Then, double-click Button1 and enter the following code

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        If TextBox1.Text = "" Or TextBox3.Text = "" Then
            MsgBox("please complete the field", MsgBoxStyle.Exclamation)
        Else
            Dim addr As IPAddress = IPAddress.Parse(TextBox3.Text)
            ipserver = addr
            TCPListener = New TcpListener(addr, 8888)
            TCPListener.Start()
            serverThread = New Thread(AddressOf waitingForClient)
            serverThread.Start()
            startConnect()
            MsgBox("Connection Successful As " + TextBox1.Text,
MsgBoxStyle.Information)
            TextBox1.Clear()
            TextBox3.Clear()
        End If
End Sub
```

Next, enter the following code in the Form_Closing event

```
Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
        writeSocket("DIS:")
        If Not (clientThread Is Nothing) Then
            If clientThread.IsAlive Then clientThread.Abort()
        End If
        If Not (reader Is Nothing) Then
            reader.Close()
        End If
        If Not (writer Is Nothing) Then
            writer.Close()
        End If
        If Not (client Is Nothing) Then
            client.Close()
        End If
        Me.Dispose()
End Sub
```

Now, we have finished on the server part, now we move on client project. Same as server, create a module and name it **TCPcontroller**, then enter the following code

```
Imports System.IO
Imports System.Net
Imports System.Net.Sockets

Public Class TCPcontroller
    Public client As TcpClient
    Public clientData As StreamWriter

    Public Sub New(ByVal Host As String, ByVal Port As Integer)
        On Error Resume Next
        client = New TcpClient(Host, Port)
        clientData = New StreamWriter(client.GetStream)
    End Sub

    Public Sub send(ByVal Data As String)
        On Error Resume Next
        clientData.Write(Data & vbCrLf)
        clientData.Flush()
    End Sub
End Class
```

Next, go to the form section, then import some of the required libraries

```
Imports System.Drawing.Text
Imports System.Net.Sockets
Imports System.Text
Imports System.IO
Imports System.Net

Public Class Form1

End Class
```

Then, declare the following global variables

```
Public Class Form1
    Private Declare Function mciSendString Lib "winmm.dll" Alias
"mciSendStringA" (ByVal lpstrCommand As String, ByVal lpstrReturnString As
    String, ByVal uReturnLength As Integer, ByVal hwndCallback As Integer) As
Integer
    Private clientnya As TCPcontroller
    Dim FileName As String
    Dim pfc As New PrivateFontCollection()
    Dim status As Byte
    Dim clientThread As System.Threading.Thread
    Dim client As New TcpClient
    Dim screen As Screen
    Dim reader As BinaryReader
    Dim writer As BinaryWriter
    Dim message As String
    Dim username As String
    Delegate Sub updateMessageDelegate(ByVal message As String)
End Class
```

Then, make some of the following functions

```
Sub updateMessageWork(ByVal message As String)
        RichTextBox1.AppendText(message + vbNewLine)
    End Sub
    Sub updateMessage(ByVal message As String)
```

```
        If RichTextBox1.InvokeRequired Then
            Invoke(New updateMessageDelegate(AddressOf updateMessageWork),
message)
        Else
            RichTextBox1.AppendText(message + vbNewLine)
        End If
    End Sub
    Sub startConnect()
        Try
            client = New TcpClient()
            client.Connect(IPAddress.Parse(TextBox3.Text), 8888)
            reader = New BinaryReader(client.GetStream())
            writer = New BinaryWriter(client.GetStream())
            writer.Write("USR  " & TextBox2.Text)
            username = TextBox2.Text
            clientThread = New System.Threading.Thread(AddressOf
readSocketClient)
            clientThread.Start()
        Catch ex As Exception
            Exit Sub
        End Try
    End Sub

    Public Sub readSocketClient()
        While True
            Try
                message = reader.ReadString()
                If message.Substring(0, 3) = "MES" Then
                    updateMessage(message.Substring(4))

                End If
                If message = "err" Then
                    MsgBox("Problem Connection", MsgBoxStyle.Information,
"Information")
                    message = reader.ReadString()
                    End
                End If
            Catch e As Exception
                Exit While
            End Try
        End While
    End Sub

    Public Sub writeSocket(ByVal message As String)
        Try
            If client.Connected Then
```

```
                writer.Write(message)
            End If
        Catch ex As Exception
            Exit Sub
        End Try
    End Sub
```

Next, double-click the Sign In Button and enter the following code

```
Private Sub ButtonSignIn_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonSignIn.Click
        If ButtonSignIn.Text = "SIGN IN" Then
            If TextBox2.Text = "" Or TextBox3.Text = "" Then
                MsgBox("please complete all fields", MsgBoxStyle.Exclamation)
            Else
                clientnya = New TCPcontroller(TextBox3.Text, 8080)
                startConnect()
                ButtonSignIn.Text = "SIGN OUT"
                ButtonSignIn.FlatAppearance.MouseOverBackColor = Color.Red
                MsgBox("You Are Connected As " + TextBox2.Text,
MsgBoxStyle.Information)
                TextBox2.Clear()
                TextBox3.Clear()
            End If

        Else
            writeSocket("DIS:")
            If Not (clientThread Is Nothing) Then
                If clientThread.IsAlive Then clientThread.Abort()
            End If
            If Not (reader Is Nothing) Then
                reader.Close()
            End If
            If Not (writer Is Nothing) Then
                writer.Close()
            End If
            If Not (client Is Nothing) Then
                client.Close()
            End If
            MsgBox("You Sign Out", MsgBoxStyle.Information)
            ButtonSignIn.Text = "SIGN IN"
            ButtonSignIn.FlatAppearance.MouseOverBackColor = Color.Green
        End If
End Sub
```

Next, double-click the send button and enter the following code

```
Private Sub ButtonSent_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonSent.Click
        If ButtonSent.Text = "SIGN IN" Then
            MsgBox("You Must Sign In First", MessageBoxIcon.Exclamation)
        Else
            If TextBox1.Text = "" Then
                MsgBox("Message Can not Empty.", MessageBoxIcon.Exclamation)
            Else
                writeSocket("MES:" & username & " : " & TextBox1.Text)
                TextBox1.Text = ""
                TextBox1.Focus()
            End If
        End If
End Sub
```

and finally, enter the following code in the Form_Closing event

```
Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
        writeSocket("DIS:")
        If Not (clientThread Is Nothing) Then
            If clientThread.IsAlive Then clientThread.Abort()
        End If
        If Not (reader Is Nothing) Then
            reader.Close()
        End If
        If Not (writer Is Nothing) Then
            writer.Close()
        End If
        If Not (client Is Nothing) Then
            client.Close()
        End If
        Dispose()
End Sub
```

Up here, the program is ready to run using one device. If you want to use multiple devices, do not forget to change the IP Address of the device you want to use

# File Transfer Using C#

*Published on July 12 2010 by Inno Design*

Today as we told you last time going to tell you how we can implement file transfer in LAN

This will let you allow file transferring even through TELNET.

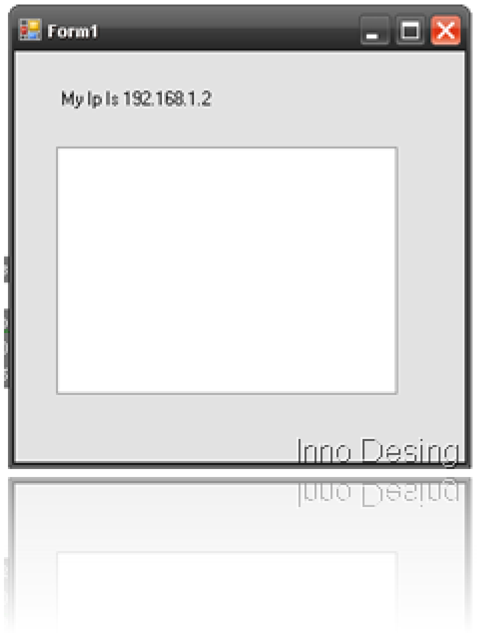
## SERVER:

So let's start with designing the server part.

for server we'll just need a label and a List Box like the previous one

Difference is Just in coding only.

So this is how it gonna look like



For File transferring we prefer to use TCP because it confirms the complete and correct order of transfer unlike UDP which just gives preference to speed.

So here we start coding from

Using statements are as follows

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;
```

Server Variables are like this

```
List<object> AiSocket= new List<object>();

delegate void additem(object obj);

delegate void additem1(string str);
```

The form load event of the form will be like this

```
private void Form1_Load(object sender, EventArgs e)
{
IPHostEntry ip = Dns.GetHostByName(Dns.GetHostName());
label1.Text = "My Ip Is " + ip.AddressList[0].ToString();
Thread thrdlistener = new Thread(new ThreadStart(listen));
thrdlistener.Start();
}
```

1st line returns the IpHostEntry, with that we extract the Own IP address. Note that we are programming for LAN so the IP address we'll get from here will not be the public IP which is provided to us by our ISP, Its our Private IP which will be unique in our Local Network.

The process of file transferring is easy to understand. Please note that it is very basic application. In this we just have described the way of file transferring over LAN.

In this process we'll have a client like last time and a server. There'll be a tcpListener  object which will listen to one port and when it'll get any message it'll create a File and Ask OS for a Lock over that file and will write the content of the message on that file.

```
void listen()
{
TcpListener tcplisten = new TcpListener(1095);
tcplisten.Start();
while (true)
{
TcpClient handlerSocket = tcplisten.AcceptTcpClient();
if (handlerSocket.Connected)
{
if (listBox1.InvokeRequired)
{
additem mydel = new additem(additem3);
listBox1.BeginInvoke(mydel,handlerSocket);
}
lock (this)
{
AiSocket.Add(handlerSocket);
}
Thread thdhandler = new Thread(new ThreadStart(handlerThread));
thdhandler.Start();
}
}
}
```

Here 1st Line shows an Object of TCPListener which listen to port 1095 in the previous Example we took a UDPlient and listen to port 8080. in 2nd line listener start listening.

then An infinite loop starts. when tcp listener will be connected with some client (can be TELNET) then it'll return a TCP client object as shown in 4th line.

Now the when the if statement is true we'll be inserting that member into listbox.

Now we can't do that using this thread as listbox were not created under scope of this thread it was created under scope of Main thread, so we'll use shown technique for doing that.

check for if invokeis required if yes then use begin invoke method and pass a delegate that matches with the method in term of signature where we'll add member into ListBox.

Now as we get the message another thread is created to write the file on destination directory.

For writing file the method is like this

```
void handlerThread()
{
TcpClient handlerSocket = (TcpClient)AiSocket[AiSocket.Count - 1];
NetworkStream Nw = new NetworkStream(handlerSocket.Client);
int thisRead = 0;
int Blocksize = 1024;
Byte[] dataByte = new Byte[Blocksize];
lock (this)
{
Stream strm = File.OpenWrite("C:\\readme.txt");
while (true)
{
thisRead = Nw.Read(dataByte, 0, Blocksize);
strm.Write(dataByte, 0, thisRead);
if (thisRead == 0)
break;
}
strm.Close();
if (listBox1.InvokeRequired)
{
additem1 mydel = new additem1(additem2);
listBox1.BeginInvoke(mydel,"Trasnfer Done");
}
handlerSocket = null;
}
}
```
Now we need a little concentration over this method.

TcpClient handlerSocket = (TcpClient)AiSocket[AiSocket.Count - 1];
this line will return the tcpclient that we received most recently. in next line we are receiving a Network stream from which we'll get the content of the message received in Bytes.
Another variable id offset point and next one is block size for file Writing.

in next line we asked for the lock and created a file.

Now it'll run an Infiniteloop.

using Network stream we got the Message in Byte form. and then after we wrote that on the fileand then got out of the loop

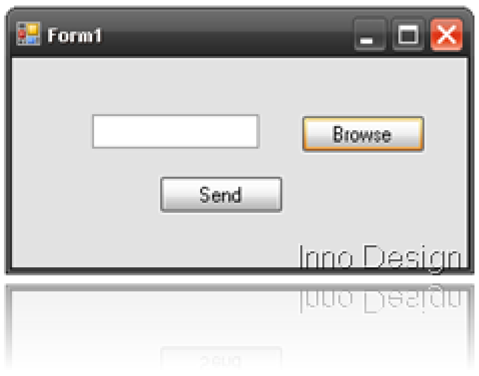
stream is closed and another work we are left is to insert another item in List box that we are done with the transfer.

here's the method for inserting Items Into ListBox

```
void additem3(object obj)
{
TcpClient handlerSocket = (TcpClient)obj;
listBox1.Items.Add(handlerSocket.Client.RemoteEndPoint.ToString() + " Connected");
}
void additem2(string str)
{
listBox1.Items.Add(str);
}
```
So here we completed Our Server

## Client:

Now let's start with Client.



So as you can say we just need two buttons and one text box placed as shown.
So Let's start with Coding that's our game for sure.

The using statements are as Follows

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.IO;
using System.Net.Sockets;
```

On the click event of browse button all we need to do is to open a open file dialog box and get path of the file we want to transfer. Here's the click event

```
private void button1_Click(object sender, EventArgs e)
{
OpenFileDialog ofpd = new OpenFileDialog();
ofpd.ShowDialog();
textBox1.Text = ofpd.FileName;
}
```

We extracted the path and put that into our text box.

Now we have to send this file to our server here's the code for sending

```
private void button2_Click(object sender, EventArgs e)
{
Stream Fs = File.OpenRead(textBox1.Text);
Byte[] buffer = new Byte[Fs.Length];
Fs.Read(buffer, 0, buffer.Length);
TcpClient socket = new TcpClient("Localhost", 1095);
NetworkStream nw = socket.GetStream();
nw.Write(buffer, 0, buffer.Length);
nw.Close();
}
```

we took the file in stream, read the stream and put the content into byte form so that we could transfer that over network.

In next line we took a TCPClient object and attempt connection to the server on port 1095. If the server is running there, it'll connect otherwise it'll return an exception.

TCPClient's Constructor takes Two parameter (the method is overloaded so other options are also available you are advised to explore other too), One is the address of the server's pc and other is the port number which server is listening to.

We created a Network Stream and using that we sent the data and then closed the networkstream. It is that easy.

You can transfer or write anything to the file using TELNET as well.

Just go to run and write "telnet"

a command window will open type there ‘o localhost 1095” for our case. and start writing whatever you want.

So hope you enjoyed this Post.

Please leave your questions and please give suggestions.