TechSight

C# TCP Chat Application

ROSHANSANTHOSH
This is a simple TCP chat application developed in C# using MonoDevelop.

SERVER
This is a multithreaded implementation for a server as I wanted the server to be able to connect to multiple clients. Texts from multiple clients are received by the server and displayed in the server application. Texts sent by the server are received by all connected clients.

The layout for the application :

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

namespace ChatServer
{
class Server : Form
{


Button StartButton = new Button();
TextBox ReceiveBox = new TextBox();
TextBox SendBox = new TextBox();
Button SendButton = new Button();
private TcpListener listener = new TcpListener (IPAddress.Any,8960);
private Thread listenThread;
private int send = 0;
TcpClient client;
List clientList = new List < TcpClient> () ;


public Server ()
{
this.SuspendLayout ();

this.StartButton.Text = "Start";
this.StartButton.Location = new System.Drawing.Point (15, 4);
this.StartButton.Size = new System.Drawing.Size (60, 30);
this.StartButton.Click += new System.EventHandler(this.startServer);
this.ReceiveBox.Location = new System.Drawing.Point (15, 100);
this.ReceiveBox.Size = new System.Drawing.Size (400, 200);
this.ReceiveBox.Multiline = true;
this.ReceiveBox.ScrollBars = ScrollBars.Vertical;
this.ReceiveBox.ReadOnly = true;
this.ReceiveBox.BackColor = Color.White;

this.SendBox.Location = new System.Drawing.Point (15, 50);
this.SendBox.Size = new System.Drawing.Size (330, 30);
this.SendBox.KeyDown += new KeyEventHandler (this.enterMsg);


this.SendButton.Location = new System.Drawing.Point (350, 50);
this.SendButton.Size = new System.Drawing.Size (60, 20);
this.SendButton.Text = "Send";
this.SendButton.Click += new System.EventHandler (this.sendMsg);


this.Controls.Add (SendButton);
this.Controls.Add (SendBox);
this.Controls.Add (StartButton);
this.Controls.Add (ReceiveBox);
this.AutoScaleDimensions = new System.Drawing.SizeF (6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.Size = new System.Drawing.Size (450, 450);
this.ResumeLayout ();
this.Text = "ChatServer";

}
```

On clicking the Start button, the **startServer** function is called which starts a new thread which listens for new clients. The **ListenForClients** function starts a new thread for every client that connects to the server. In short, each client is handled by an individual thread that takes care of the communication between the server and that client.

NOTE: Thread initialization in *startServer* and *ListenForClients* are slightly different because in *ListenForClients*, along with the function , we also pass the parameter counter to the **handleClient** function. The initialization in *startServer* doesnt have any provision for setting parameter values for the functions called.

```
private void startServer(object sender, EventArgs e)
{
  listenThread = new Thread (new ThreadStart (ListenForClients));
  listenThread.Start ();
}


private void ListenForClients()
{
  listener.Start ();
  int counter = 0;

while (true)
{
  client = listener.AcceptTcpClient ();
  counter++;
  clientList.Add (client);
  Thread clientThread = new Thread(delegate(){ handleClient(client,counter); })
  clientThread.Start ();
 }
}

private void sendMsg(object sender, EventArgs e)
{
  send = 1;
}

private void enterMsg(object sender, KeyEventArgs e)
{
  if (e.KeyCode == Keys.Enter) {
    send = 1;
 }
}

private void handleClient(object client,int i)
{
 TcpClient chat = (TcpClient)client;
 NetworkStream chatStream = chat.GetStream ();

 byte[] message = new byte[4096];
 int byteRead;
 string data;

 ASCIIEncoding encode = new ASCIIEncoding ();
 byte[] sendData = new byte[4096];

while (true)
{
 if (chatStream.DataAvailable == true)
{
 byteRead = 1;
 byteRead = chatStream.Read (message, 0, 4096);
 data = Encoding.ASCII.GetString (message, 0, byteRead);
 ReceiveBox.Text = data;
}

 if (send == 1) {
  sendData = encode.GetBytes (SendBox.Text);
  chatStream.Write (sendData, 0, sendData.Length);
  chatStream.Flush ();
  SendBox.Text = "";
  send = 0;
 }
}
chat.Close ();
}
}



public class Program
{
 public static void Main(string[] args)
{
  Application.EnableVisualStyles ();
  Application.SetCompatibleTextRenderingDefault (false);
  Application.Run (new Server ());
}
}
}
```

CLIENT
The layout for the ChatClient application is pretty much similar to the ChatServer application.

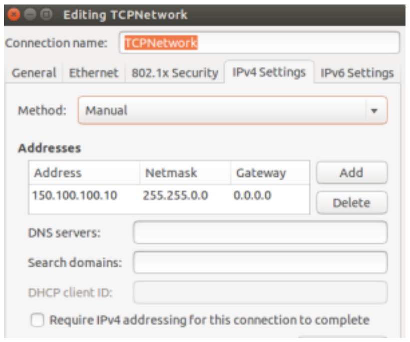
```
namespace ChatClient
{

 public class Client: Form
 {
  TcpClient newclient;
  NetworkStream clientstream;
  int start = 0;
  ASCIIEncoding encoder = new ASCIIEncoding ();
  byte[] buffer = new byte[4096];
  Button StartButton = new Button();
  TextBox sendBox = new TextBox();
  Button sendButton = new Button ();
  TextBox receiveBox = new TextBox();


public Client()
{
  ..layout..
}

private void sendMsg(object sender, KeyEventArgs e)
{
 if (e.KeyCode == Keys.Enter) {
   clientstream = newclient.GetStream ();
   buffer = encoder.GetBytes (sendBox.Text);
   clientstream.Write (buffer, 0, buffer.Length);
   clientstream.Flush ();
   sendBox.Text = "";
}
}

private void sendData(object sender, EventArgs e)
{
  clientstream = newclient.GetStream ();
  buffer = encoder.GetBytes (sendBox.Text);
  clientstream.Write (buffer, 0, buffer.Length);
  clientstream.Flush ();
  sendBox.Text = "";
}

private void startClient(object sender, EventArgs e)
{
  if (start == 0) {
  start = 1;
  newclient = new TcpClient ("150.100.100.10", 8960);
  MessageBox.Show ("Connected");
  ClientReceive ();
}
}

public void ClientReceive()
{
  clientstream = newclient.GetStream ();

new Thread (() =>
{
 byte[] data = new byte[4096];
 int i;
 string strdata;

while (true) {
  i = 0;
  i = clientstream.Read (data, 0, 4096);


if (i == 0) {   break; }

strdata = Encoding.ASCII.GetString (data, 0, i);
if(receiveBox.Text == "")
 {
   receiveBox.Text = strdata;
 }
 else
 {
   receiveBox.Text = receiveBox.Text + Environment.NewLine + strdata;
 }
 }
}).Start ();
}
}

public class Program
{
  public static void Main( string[] args)
  {
   Application.EnableVisualStyles ();
   Application.SetCompatibleTextRenderingDefault (false);
   Application.Run (new Client ());
  }
}
```

RUNNING THE APPLICATION
To test the application, run both the server and the client in the same laptop. In the client code, use 127.0.0.1 as the IP address when initializing the TCPClient. 127.0.0.1 is a loopback address and is commonly used for debugging purposes.

NOTE: It is important to start the server first before trying to connect. If the client application tries to connect to a port that's not listening the application will most probably crash. So start the server application and click on the Start button so that the port starts listening for clients. Once the server is set, start the client application and connect to the server by clicking the Connect button.

To run the application on two laptops,we first need to configure a network for the laptops to communicate on.
Create a new ethernet connection on the server laptop with the following IPv4 settings:
Address : 150.100.100.10
Netmask : 255.255.0.0

Similarly, create a new connection on the client laptop with the settings :
Address : 150.100.100.20
Netmask : 255.255.0.0


Editing TCPNetwork

Connection name: TCPNetwork

Method: Manual

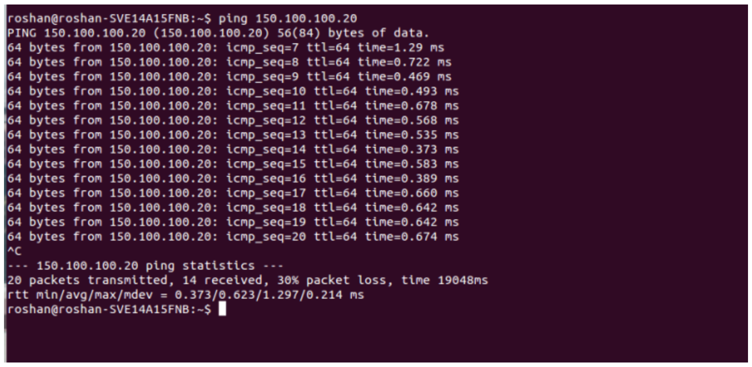
Addresses

Address	Netmask	Gateway
150.100.100.10	255.255.0.0	0.0.0.0

DNS servers:
Search domains:
DHCP client ID:
☐ Require IPv4 addressing for this connection to complete

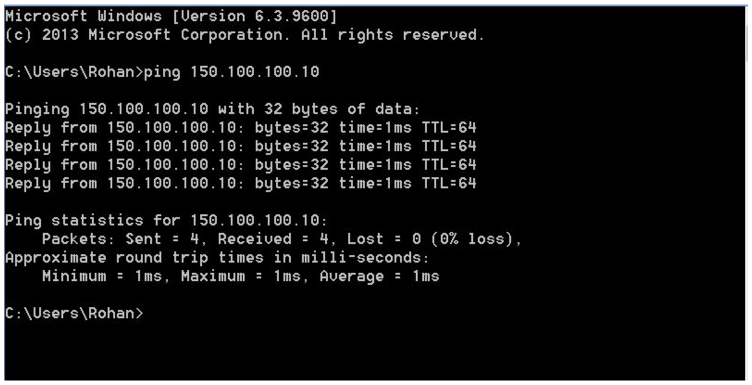
(https://roshansanthosh.files.wordpress.com/2015/05/screenshot-from-2015-05-07-170926.png)

Check the connection between the laptops using the ping command in the Terminal/ Command Prompt. Make sure to check this on both the laptops as the laptop might be configured to allow only one way communication. (Press Ctrl+C in Ubuntu to terminate the ping and see the ping statistics)



```
roshan@roshan-SVE14A15FNB:~$ ping 150.100.100.20
PING 150.100.100.20 (150.100.100.20) 56(84) bytes of data.
64 bytes from 150.100.100.20: icmp_seq=7 ttl=64 time=1.29 ms
64 bytes from 150.100.100.20: icmp_seq=8 ttl=64 time=0.722 ms
64 bytes from 150.100.100.20: icmp_seq=9 ttl=64 time=0.469 ms
64 bytes from 150.100.100.20: icmp_seq=10 ttl=64 time=0.493 ms
64 bytes from 150.100.100.20: icmp_seq=11 ttl=64 time=0.678 ms
64 bytes from 150.100.100.20: icmp_seq=12 ttl=64 time=0.568 ms
64 bytes from 150.100.100.20: icmp_seq=13 ttl=64 time=0.535 ms
64 bytes from 150.100.100.20: icmp_seq=14 ttl=64 time=0.373 ms
64 bytes from 150.100.100.20: icmp_seq=15 ttl=64 time=0.583 ms
64 bytes from 150.100.100.20: icmp_seq=16 ttl=64 time=0.389 ms
64 bytes from 150.100.100.20: icmp_seq=17 ttl=64 time=0.660 ms
64 bytes from 150.100.100.20: icmp_seq=18 ttl=64 time=0.642 ms
64 bytes from 150.100.100.20: icmp_seq=19 ttl=64 time=0.642 ms
64 bytes from 150.100.100.20: icmp_seq=20 ttl=64 time=0.674 ms
^C
--- 150.100.100.20 ping statistics ---
20 packets transmitted, 14 received, 30% packet loss, time 19048ms
rtt min/avg/max/mdev = 0.373/0.623/1.297/0.214 ms
roshan@roshan-SVE14A15FNB:~$
```

(https://roshansanthosh.files.wordpress.com/2015/05/new.png)
Ubuntu



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Rohan>ping 150.100.100.10

Pinging 150.100.100.10 with 32 bytes of data:
Reply from 150.100.100.10: bytes=32 time=1ms TTL=64
Reply from 150.100.100.10: bytes=32 time=1ms TTL=64
Reply from 150.100.100.10: bytes=32 time=1ms TTL=64
Reply from 150.100.100.10: bytes=32 time=1ms TTL=64

Ping statistics for 150.100.100.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms

C:\Users\Rohan>
```

(https://roshansanthosh.files.wordpress.com/2015/05/capture.jpg)
Windows

Once the connection is established, run the ChatServer application and start the server. Then run the ChatClient application and click on the connect button. A "connected" pop-up indicates that the connection has been made.

NOTE: The TCPClient object in the ChatClient program must be properly initialized depending on the situation. If the server and client are to be run on the same laptop, use 127.0.0.1 as the IP address. In the other case, when server and client are on different laptops, use the IP address of the server to initialize the TCPClient object. In this case that would be 150.100.100.10.

You can find the code here (https://github.com/rsk2327/Blog/tree/origin/master/ChatApp).

Sponsored Content

Simple Prostate Shrinking Solution Helps Men Over 50 medicalhelp.me | Sponsored
(https://prstfree.com/view/RX2x4qhUhSalSEhSalS2tnqyT0hSalS7etsULplGZHHD8T7tlzYteDuiBY?c=42316&pid=4443&tid=ob_click_id&ob_marketer=Propro_all4&ob_publisher_id=$publisher_id$&ob_section_id=$section_id$&ob_ad_id=00236a4d51f884d

Only $49 to get all the TV channels? It's now possible TV Superboost | Sponsored
(https://prough-veridated.icu/0ba598af-467b-4012-b57a-f15f21ec0a54?campaign_id=00dd6ceffd2f648f6fd422a016e8d8da70&publisher_id=$publisher_id$&pub

Forget the Blue Pill, This Can Heal Your ED! trkmeded.com | Sponsored
(https://asbunnath.com/view/S8cWaT1Y3eGH7WdZiXT1bBcrAASeOm8MGLl5eIhSalSBzqM?c=42491&pid=4443&tid=ob_click_id&ob_marketer=MED8&ob_publisher_id=$publisher_id$&ob_section_id=$section_id$&ob_ad_id=0094de9b710ee63096a

Software development
C++ CHAT CLIENT LAPTOP CONNECTION MONODEVELOP SERVER SOFTWARE DEVELOPMENT TCP VISUAL STUDIO