C# Server and Client Socket Programming

Concepts and Examples

Ömer Levent Durdalı – 21702600

2021 – Spring Semester

Contents

- What is Socket Programming
- Basic TCP/IP Concepts
- Connection Concepts and Diagrams
- Client – Server Basic Project Explanation
- Single Threaded Client / Server Example
- Multi Threaded Client / Server Example

What will see

- How to create a C# socket and setup a listener server node that starts listening to any messages coming its way via the predefined IP and protocol.
- How to create a client application that will send messages to a listener server and read it using Sockets.
- Socket Programming
- TCP/IP and UDP/IP Concepts

Socket Programming

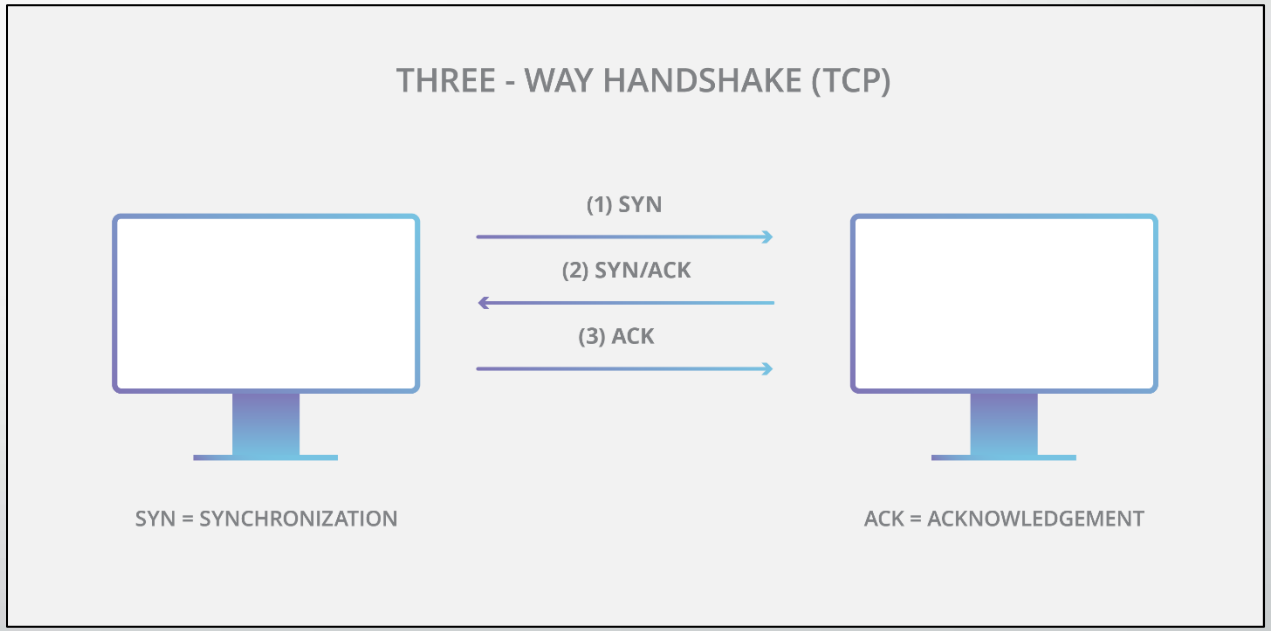
- Network programming in windows is possible with Sockets.
- A Socket is basically End-Point of To and From (Bidirectional or Directional) communication link between two different programs. (Server Program and Client Program) which are running and generally in the same network.
- My Example Will be a Directional Application meaning, is a one-way Client and Server setup where a Client connects, sends messages to the server and the server shows them using socket connection.
- For our case, .Net Framework, Windows can utilizes two namespaces, System.Net and System.Net.Sockets for managed implementation of Internet protocols.(TCP/IP and UDP/IP).
- Lastly a **Socket** = IP Address + Port Number

Socket Programming

- So, Sockets in computer networks are used to establish a connection between two or more computers
- Used to send data from one computer to another.
  - Each computer in the network is called a node.
- Sockets use nodes' IP addresses and a Ports to create a channel of communication
  - Use this channel to transfer data.
- In socket communication, one node acts as a listener and other node acts as a client.

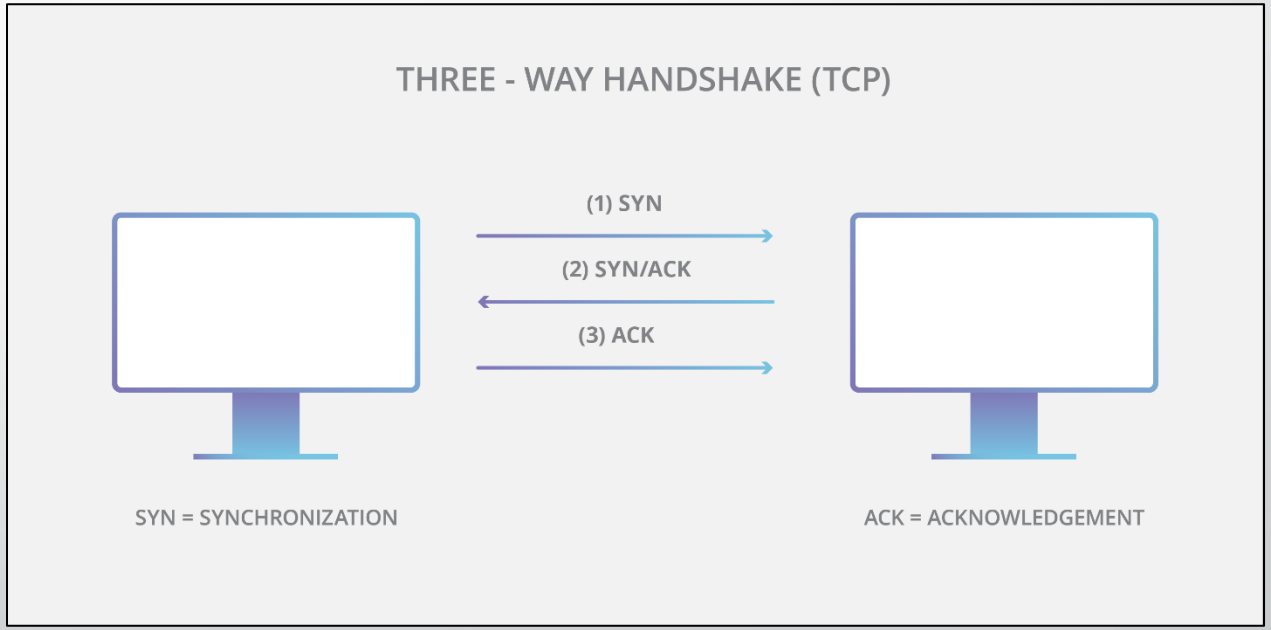
TCP/IP and UDP/IP Concepts

- **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol is a set of standardized rules that allow computers to communicate on a network such as the internet.
- **IP** is the part that obtains the address to which data is sent. **TCP** is responsible for data delivery once that IP address has been found.



Connection Concepts and Diagrams

- **Simple Explanation**: The IP address is like the phone number assigned to your smartphone. TCP is all the technology that makes the phone ring, and that enables you to talk to someone on another phone



Socket Connection - Diagram

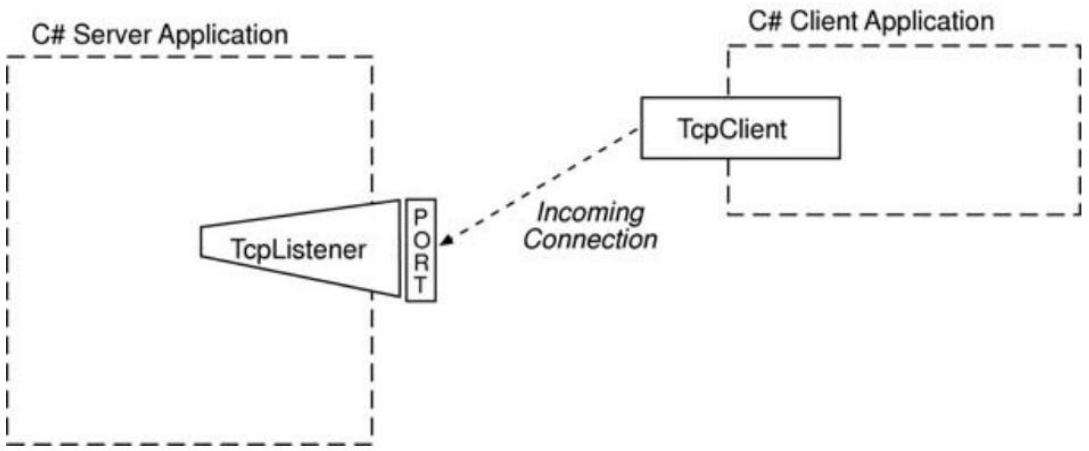


Figure 19-8: Server Application Listens on a Host and Port for Incoming TcpClient Connections

Socket Connection - Diagram

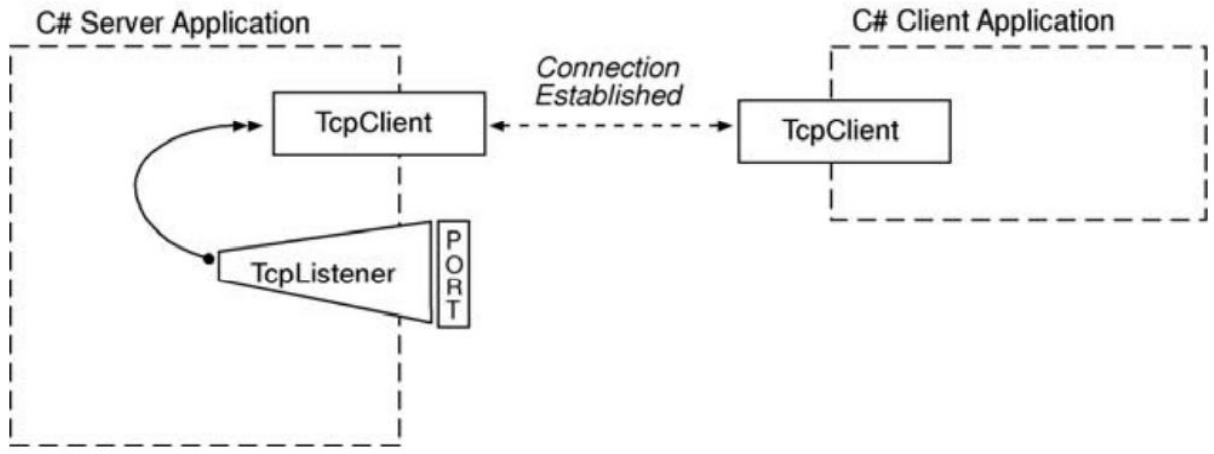


Figure 19-9: TcpListener Accepts Incoming TcpClient Connection

Socket Connection - Diagram

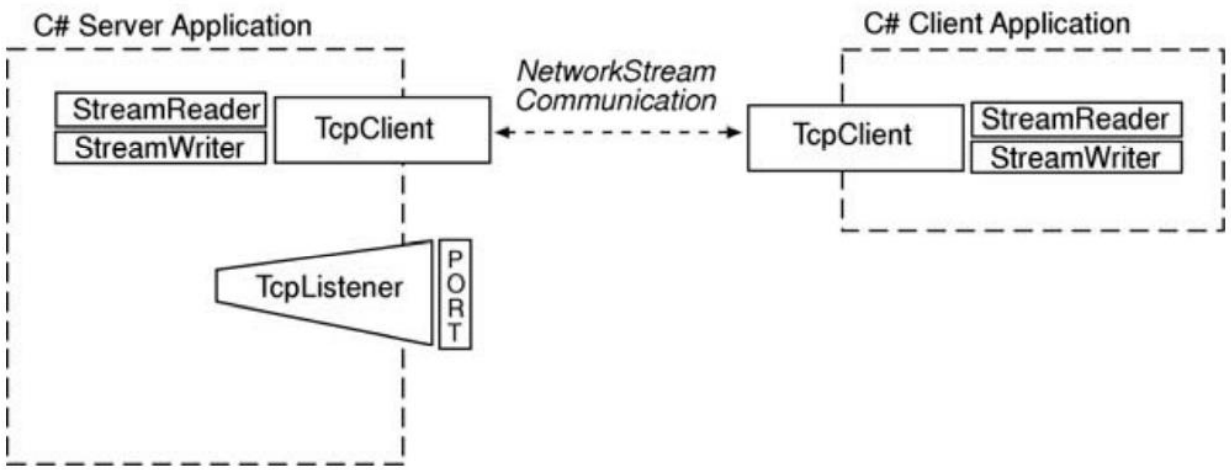


Figure 19-10: TcpClients Communicate via a NetworkStream using StreamReader and StreamWriter Objects

Client – Server Information/Tools/ Libraries

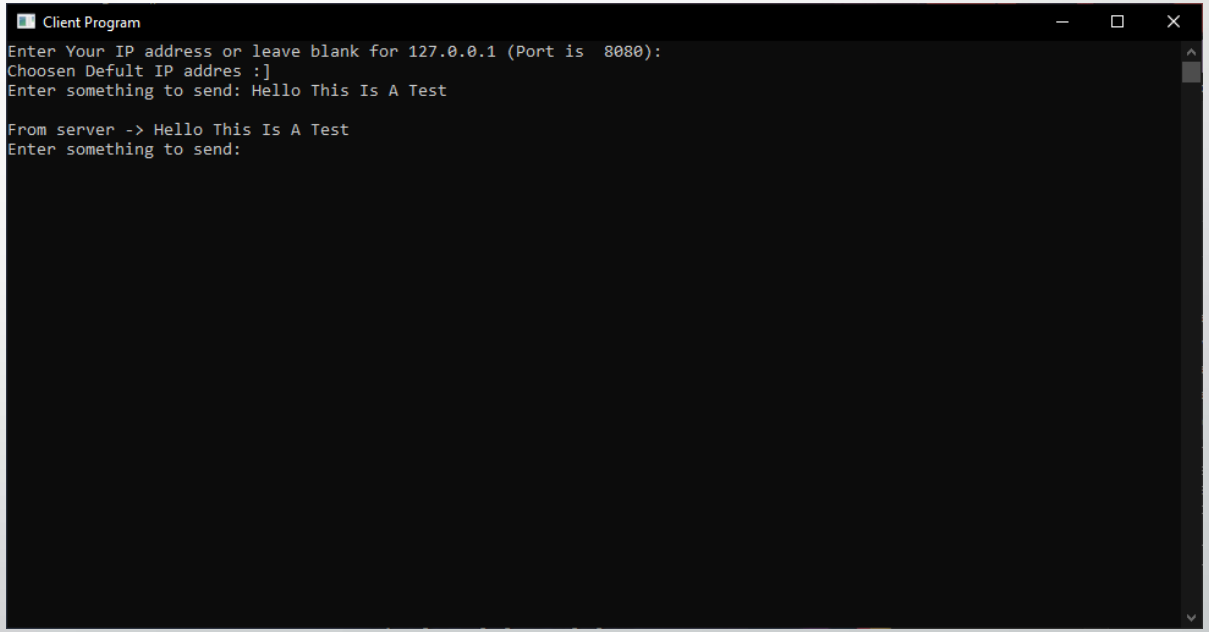
- This example will be a Directional Application.
- **Visual Studio 2019** was used during the creation of the project.
- The Client Program app will get the IP from the User to send message to the Server Program.
  - Make sure that your computers firewall **allow** the usage of the "**Port 8080**".
- Project was made as an **new Console Application (.Net Core)**

- No external libraries are needed.
- No external tools are needed.
- Project .EXE will be given to run the Program without seeing code.

Single Threaded Client – Server Example

Client - Example

- As you can see the **Client** Program can **Send** a message and **Receive** a reply from the server on the chosen **Local** IP address.



./NormalClientServer-Project-EXE/NormalClient-Project.exe

Client Imports

- To use the TCP Programming our application firstly must include the following imports

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

- After this we can start writing the program.

System.Net.Sockets and **System.Net** are two namespaces that **Microsoft .NET framework** provides and are used for managing the implementation of Internet protocols that applications use.

Code Class Explanation - Client

- Some basics, with the help of "Console.Title" is used for changing the name of the Console.
- `TcpListener` is simply a class that listens for connections from TCP network clients.
  - Listens to a specified IP (one or more) and Port.
- `IPAddress` is simply a class that stores IP address and network configuration.
- `TcpClient` is a class that provides simple methods for connecting, sending, and receiving data.
- `StreamReader` and `StreamWriter`
  - The server/client uses these StreamReader and StreamWriter objects to communicate with the client/server

```
Console.Title = "Client Program";

TcpListener listener = null;

listener = new TcpListener(ip_address_def, port);

listener.Start();

listener.Stop()

IPAddress ip_address_def = IPAddress.Parse("127.0.0.1");

TcpClient client = listener.AcceptTcpClient();

StreamReader reader = new StreamReader(client.GetStream());
StreamWriter writer = new StreamWriter(client.GetStream());
```

Client Example Explanation

- In line **7** we are setting a default fallback IP address**.**
- In line **8** we are setting a Port,
  - Line 7 and 8 are used, if the user doesn't specify an IP or Port.
- In the `if`/`else` block we get user inputs or set the default values.
- In line **14,** a new `TcpClient` is created and started with the default IP address.
- In line **18,** a new user specified `TcpClient` is created.

```
1. namespace NormalClient_Project {
2.     class Program {
3.         static void Main(string[] args) {  // Main Method
4.             Console.Title = "Client Program";
5.             string ipAdd = string.Empty;
6.             TcpClient client = new TcpClient();
7.             IPAddress ip_address_def = IPAddress.Parse("127.0.0.1");
8.             int port = 8080;
9.             try {
10.                 Console.Write("Enter Your IP address or leave blank for
                                                  127.0.0.1 (Port is  8080): ");
11.                 ipAdd = Console.ReadLine();
12.                 if (ipAdd == "") {
13.                     Console.WriteLine("Choosen Defult IP addres :]");
14.                     client = new TcpClient(ip_address_def.ToString(), port);
15.                 }
16.                 else {
17.                     Console.WriteLine("Choosen IP addres" + ipAdd);
18.                     client = new TcpClient(ipAdd, port);
19.                 }
20.               //Continue…
```

Client Example Explanation

- Inside the Try/Catch, in line **22** we start a StreamReaders , so that we can get the received data(**32**), than we can print the received data.
- Inside the Try/Catch, in line **23** we start a StreamWriter, sends the data(**30**) to the server.
- On line **25**, the while loop checks the received message and prints the received until it receives an **"Exit"** message, than the server closes all readers and TCP received connections.
- In line **31**, writer.Flush() is used for clearing all buffered data and the data is sent to the Stream
  - So the next message dose not mix or get altered by the previous message.
  - Is used to send the data.

```
21.             //Continue…
22.             StreamReader reader = new StreamReader(client.GetStream());
23.             StreamWriter writer = new StreamWriter(client.GetStream());
24.             string s = string.Empty;
25.             while (!s.Equals("Exit") || !s.Equals("exit"))
26.             {
27.                 Console.Write("Enter something to send: ");
28.                 s = Console.ReadLine();
29.                 Console.WriteLine();
30.                 writer.WriteLine(s);
31.                 writer.Flush();
32.                 string server_string = reader.ReadLine();
33.                 Console.WriteLine(server_string);
34.             }
35.             reader.Close();
36.             writer.Close();
37.             client.Close();
38.         }//try
39.         //Continue…
```

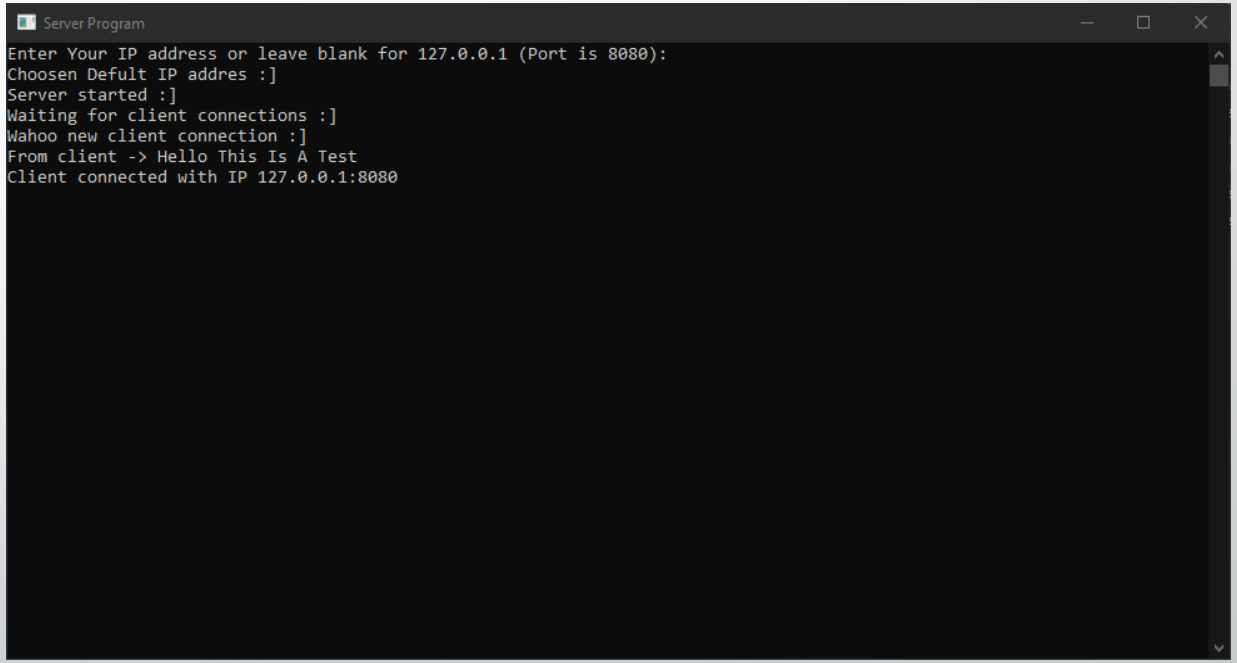
Client Example Explanation

- This part is for error catching as the program has multiple point of failures, so we have 3 different error catching.
  - `FormatException` is for IP address parsing errors from line **18**.
  - `SocketException` is for all TCP connections and TCP listeners Errors.
  - `IOException` is for all `Client` connections.

```
40.             //Continue…
41.             catch (FormatException e) {
42.                Console.WriteLine("Not an IP address closing the server.. :[");
43.                //Console.WriteLine(e);
44.             }
45.             catch (SocketException e) {
46.                 Console.WriteLine("Not an IP address closing the server :[");
47.                 //Console.WriteLine(e);
48.             }
49.             catch (IOException e) {
50.                 Console.WriteLine("Client Connection has been interrupted,
                                                    closing the server :[");
51.                 //Console.WriteLine(e);
52.             }
53.         }//Main
54.     }//class
55.}//namespace
```

Server - Example

- As you can see the **Server** Program can **Receive** a message from the Client on the chosen **Local** IP address.
- The server also logs it startup in the window.
- .EXE location ./NormalClientServer-Project-EXE



./NormalClientServer-Project-EXE/NormalServer-Project.exe

Server Imports – Same as Client

- To use the TCP Programming our application firstly must include the following imports

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

- After this we can start writing the program.

System.Net.Sockets and **System.Net** are two namespaces that **Microsoft .NET framework** provides and are used for managing the implementation of Internet protocols that applications use.

Code Class Explanation - Server

- Some basics, with the help of "Console.Title" is used for changing the name of the Console.
- `TcpListener` is simply a class that listens for connections from TCP network clients.
- `IPAddress` is simply a class that stores IP address and network configuration.
- `TcpClient` is a class that provides simple methods for connecting, sending, and receiving data.
- `StreamReader` and `StreamWriter`
  - The server/client uses these StreamReader and StreamWriter objects to communicate with the client/server

```
Console.Title = "Server Program";

TcpListener listener = null;

listener = new TcpListener(ip_address_def, port);

listener.Start();

listener.Stop()

IPAddress ip_address_def = IPAddress.Parse("127.0.0.1");

TcpClient client = listener.AcceptTcpClient();

StreamReader reader = new StreamReader(client.GetStream());
StreamWriter writer = new StreamWriter(client.GetStream());
```

Server Example Explanation

- In line **7** we are setting a default fallback IP address**.**
- In line **8** we are setting a Port,
  - Line 7 and 8 are used, if the user doesn't specify an IP or Port.
- In the `if`/`else` block we get user inputs or set the default values. Once these are set we start the `TcpListener`
- In line **14,** a new `TcpListener` is created and started with the default IP address. If the user specified an IP than in line **19**, the `TcpListener` is created accordingly.
- Lasty the user promoted with an server stared message.

```
1. namespace NormalServer_Project{
2.     class Program{
3.         private static void Main(string[] args){
4.             Console.Title = "Server Program";
5.             TcpListener listener = null;
6.             string ipAdd = string.Empty;
7.             IPAddress ip_address_def = IPAddress.Parse("127.0.0.1"); //Default
8.             int port = 8080; //Default
9.             try{
10.            Console.Write("Enter Your IP address or leave blank for 127.0.0.1");
11.                ipAdd = Console.ReadLine();
12.                 if (ipAdd == ""){
13.                     Console.WriteLine("Choosen Defult IP addres :]");
14.                     listener = new TcpListener(ip_address_def, port);
15.                     listener.Start();
16.                 }
17.                 else{
18.                     Console.WriteLine("Choosen IP addres" + ipAdd);
19.                     listener = new TcpListener(IPAddress.Parse(ipAdd), port);
20.                     listener.Start();
21.                 }
22.                 Console.WriteLine("Server started :]");
23.             //Continue…
```

Server Example Explanation

- Inside the Try/Catch, In line **29,** we start the TcpClient Listeners, so that we can assign the Clients a TCP connections. Waits for an client connection. To the set IP and Port.
- In line **31,** we start a StreamReaders , so that we can get the received data(**34**). And write the received message to the console on line **36.**
- In line **32,** we start a StreamWriter, so that we can get the received data(**37**), use writer.WriteLine() send an echo message to the Client.
- On line **34**, the while loop checks the received message and prints the received until it receives an **"Exit"** message, than the server closes all readers and TCP received connections.
- In line **31**, writer.Flush() is used for clearing all buffered data and the data is sent to the Stream
  - So the next message dose not mix or get altered by the previous message.
  - Is used to send the data.

```
24.         try{
25.         //Continue…
26.          Console.WriteLine("Server started :]");
27.          while (true){
28.             Console.WriteLine("Waiting for client connections :]");
29.             TcpClient client = listener.AcceptTcpClient();
30.             Console.WriteLine("Wahoo new client connection :]");
31.             StreamReader reader = new StreamReader(client.GetStream());
32.             StreamWriter writer = new StreamWriter(client.GetStream());
33.             string s = string.Empty;
34.             while (!(s = reader.ReadLine()).Equals("Exit") ||
                                                s.Equals("exit") || (s == null))
35.             {//while
36.                Console.WriteLine("From client -> " + s);
37.                writer.WriteLine("From server -> " + s);
38.                Console.WriteLine("Client connected with IP {0}",
                                                  client.Client.LocalEndPoint);
39.                writer.Flush();
40.              }
41.                 reader.Close();
42.                 writer.Close();
43.                 client.Close();
44.             }
45.         }//try
46.         //Continue…
```

Server Example Explanation

- This part is for error catching as the program has multiple point of failures, so we have 3 different error catching.
  - `FormatException` is for IP address parsing errors from line **19**.
  - `SocketException` is for all TCP connections and TCP listeners.
  - `IOException` is for all TCP connections and TCP listeners.
- And finally, in line **61**, after an error the server stops listing for TCP connections.

```
47.             //Continue…
48.             }//try
49.             catch (FormatException e) {
50.                 Console.WriteLine("Not an IP address closing the server :[");
51.                 //Console.WriteLine(e);
52.             }
53.             catch (SocketException e) {
54.                 Console.WriteLine("Not an IP address closing the server :[");
55.                 //Console.WriteLine(e);
56.             }
57.             catch (IOException e) {
58.                 Console.WriteLine("Client Connection has been interrupted, clo
                                                                        sing the server :[");
59.                   //Console.WriteLine(e);
60.             }
61.             finally {
62.                 if (listener != null)  {
63.                     listener.Stop();
64.                 }
65.             }
66.         }//Main()
67.     }//Class
68. }//namespace
```

Multi Threaded Client – Server Example

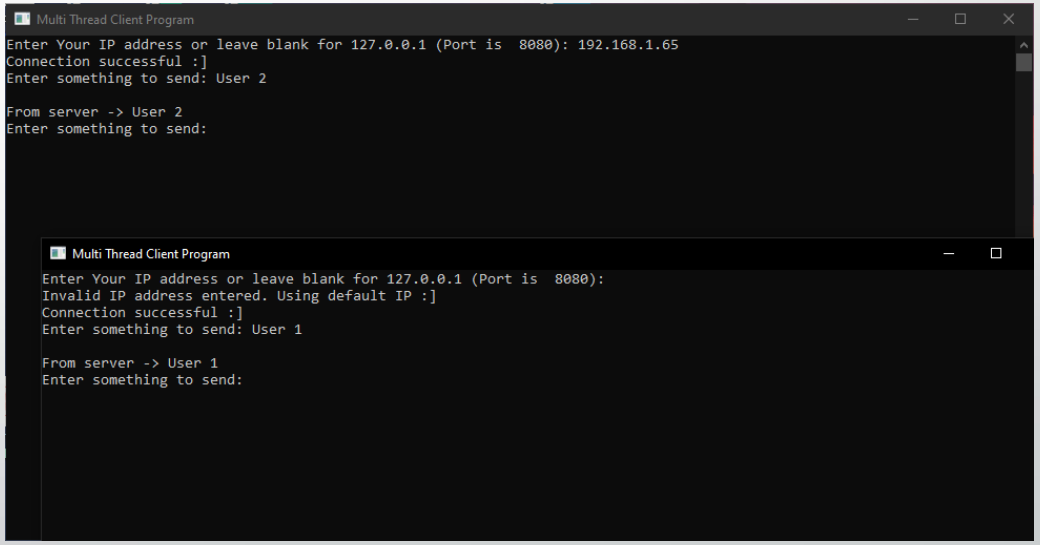
Multi Threaded and Single Threaded Difference

- Actually there isn't much difference from the Single Threaded application but I made some improvements overall. So it looks a bit different.
- Other than my own improvements the only **major change happens in the Server Program Code.**
- The Client Program Code does **not** need to change, but I made some improvements over the Single Threaded one.

- Server Program Code, the change is;
  - New import "`using System.Threading;`"
  - The `StreamReader`, `StreamWriter` and the `while` loop are all in an different function for easier use of `Thread` creation.

Client - Example

- There are multiple Clients that use different IP address and can send messages to the Multi Thread Server Program.
- The server also logs it startup in the window.
- The different IP address connection was the Improvement that I was talking about.
- .EXE location ./MultiThreadedClientServer-EXE



./MultiThreadedClientServer-EXE/MultiTClient-Project.exe

Server Imports – Same as Single Thread Client

- To use the TCP Programming our application firstly must include the following imports

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

- After this we can start writing the program.

System.Net.Sockets and **System.Net** are two namespaces that **Microsoft .NET framework** provides and are used for managing the implementation of Internet protocols that applications use.

Code Class Explanation - Client

- Some basics, with the help of "Console.Title" is used for changing the name of the Console.
- `TcpListener` is simply a class that listens for connections from TCP network clients.
- `IPAddress` is simply a class that stores IP address and network configuration.
- `TcpClient` is a class that provides simple methods for connecting, sending, and receiving data.
- `StreamReader` and `StreamWriter`
  - The server/client uses these StreamReader and StreamWriter objects to communicate with the client/server

```
Console.Title = "Multi Thread Client Program";

TcpListener listener = null;

listener = new TcpListener(ip_address_def, port);

listener.Start();

listener.Stop()

IPAddress ip_address_def = IPAddress.Parse("127.0.0.1");

TcpClient client = listener.AcceptTcpClient();

StreamReader reader = new StreamReader(client.GetStream());
StreamWriter writer = new StreamWriter(client.GetStream());
```

Client Example Explanation

- In line **5** we are setting a default fallback IP address**.**
- In line **6** we are setting a Port,
  - Line **5** and **6** are used, if the user doesn't specify an IP or Port.
- In the `if`/`else` block, in line **11** we make the check to see if the user gave any inputs and if they did not than, a new `TcpClient` with “ip_address” will be created with the default IP address.
- If the user specified an IP than the `TcpClient` is created accordingly, with the parsed “ipAdd” string.
- Lastly we have an `FormatException` because we receive an IP from the user.

```
1. namespace MultiTClient_Project{
2.     internal class Program    {
3.         private static void Main(string[] args)        {
4.             Console.Title = "Multi Thread Client Program";
5.             IPAddress ip_address = IPAddress.Parse("127.0.0.1"); //Default
6.             int port = 8080; //Default
7.             string ipAdd = string.Empty;
8.             try{
9.                 Console.Write("Enter Your IP address or leave blank for 127.0.0.1 (Port is  8080): ");
10.                 ipAdd = Console.ReadLine();
11.                 if (ipAdd == ""){
12.                     Console.WriteLine("Invalid IP address entered. Using default IP :]");
13.                 }
14.                 else{
15.                     ip_address = IPAddress.Parse(ipAdd);
16.                 }
17.
            }//try
18.             catch (FormatException){
19.                 Console.WriteLine("Invalid IP address entered. Using default IP of: " + ip_address.ToString());
20.             }
21.             //Continue…
```

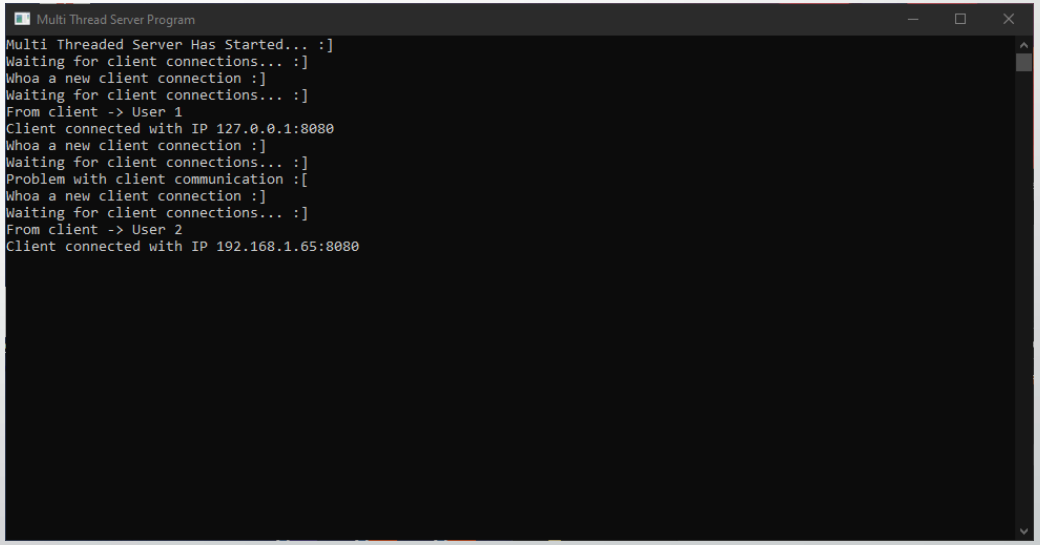
Client Example Explanation

- Inside the Try/Catch, In line **21** we start the TcpClient so that we can assign the Clients TCP connections.
- Inside the Try/Catch, in line **25** we start a StreamReaders , so that we can get the received data(**35**), than we can print the received data.
- Inside the Try/Catch, in line **26** we start a StreamWriter, sends the data(**32**) to the server.
- On line **28**, the while loop checks the received message and prints the received until it receives an **"Exit"** message, than the server closes all readers and TCP received connections.
- In line **33**, writer.Flush() is used for clearing all buffered data and the data is sent to the Stream
  - So the next message dose not mix or get altered by the previous message.
  - Is used to send the data.

```
21.          //Continue…
22.          try{
23.              TcpClient client = new TcpClient(ip_address.ToString(), port);
24.              Console.WriteLine("Connection successful :]");
25.              StreamReader reader = new StreamReader(client.GetStream());
26.              StreamWriter writer = new StreamWriter(client.GetStream());
27.              string s = String.Empty;
28.              while (!s.Equals("Exit") || !s.Equals("exit")){
29.                  Console.Write("Enter something to send: ");
30.                  s = Console.ReadLine();
31.                  Console.WriteLine();
32.                  writer.WriteLine(s);
33.                  writer.Flush();
34.                  if (!s.Equals("Exit") || !s.Equals("exit")) {
35.                      String server_string = reader.ReadLine();
36.                      Console.WriteLine(server_string);
37.                  }
38.              }//while
39.              reader.Close();
40.              writer.Close();
41.              client.Close();
42.          }//rty
43.          catch (Exception e){ Console.WriteLine(e);   }
44.      }//Main
45.  }//class
46.}//namespace
```

Server - Example

- As you can see the **Server** Program can **Receive** a message from the multiple Clients at the same time, from different IP addresses.
- The server also logs it startup in the window.
- Again, the different IP addres connection was the Improvement that I was talking about.



./MultiThreadedClientServer-EXE/MultiTServer-Project.exe

Server Imports

- To use the TCP Programming our application firstly must include the following imports

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

- After this we can start writing the program.

System.Net.Sockets and **System.Net** are two namespaces that **Microsoft .NET framework** provides and are used for managing the implementation of Internet protocols that applications use.

System.Threading is simply for thread configuration and controling

Code Class Explanation - Server

- Some basics, with the help of "Console.Title" is used for changing the name of the Console.
- `TcpListener` is simply a class that listens for connections from TCP network clients.
- `IPAddress` is simply a class that stores IP address and network configuration.
- `TcpClient` is a class that provides simple methods for connecting, sending, and receiving data.
- `StreamReader` and `StreamWriter`
  - The server/client uses these StreamReader and StreamWriter objects to communicate with the client/server

```
Console.Title = "Multi Thread Server Program";

TcpListener listener = null;

listener = new TcpListener(ip_address_def, port);

listener.Start();

listener.Stop()

IPAddress ip_address_def = IPAddress.Parse("127.0.0.1");

TcpClient client = listener.AcceptTcpClient();

StreamReader reader = new StreamReader(client.GetStream());
StreamWriter writer = new StreamWriter(client.GetStream());
```

Server Example Explanation

- `TcpListener(IPAddress.Any, port);`
  - With IPAddres.Any the Server is listening all network interfaces and IP address.
- `Thread t = new Thread(ProcessClientRequests);`
  - On every new ProcessClientRequests **function** a new thread is opened and a new connection is started.

```
namespace MultiTServer_Project{
    internal class Program{
        private static void Main(string[] args){
            Console.Title = "Multi Thread Server Program";
            TcpListener listener = null;
            int port = 8080;
            try{
                //Unlike our Single Threaded Server IPAddress.Any will listen to all Netwotk Interfaces and their Ip address
                //Ex. Wifi Module is 192.168.1.2 and Ethernet Interface is 172.0.0.1
                //IPAddress.Any will listen to both
                listener = new TcpListener(IPAddress.Any, port);
                listener.Start();
                Console.WriteLine("Multi Threaded Server Has Started... :]");
                while (true)
                {
                    Console.WriteLine("Waiting for client connections... :]");
                    TcpClient client = listener.AcceptTcpClient();
                    Console.WriteLine("Whoa a new client connection :]");
                     //Start a new Thread for every new client connection
                    Thread t = new Thread(ProcessClientRequests);
                     t.Start(client);
                }
            }//try
        //Continue…
```

Server Example Explanation

- Inside the `Try`/`Catch`, In line **19,** we start the `TcpClient` Listeners, so that we can assign the Clients a TCP connections. Waits for an client connection. To the set IP and Port.
- In line **20,** Pass `TcpClient` object to the newly created Thread/Function.
- Since we have a `while` loop for every new TcpClient we create a new Thread.
  - This wont constantly open new Thread because the `listener.AcceptTcpClient(); function locks the process and the program continues once a new Client has connected.`

```
13.             //Continue…
14.             while (true) {
15.                 Console.WriteLine("Waiting for client connections... :]");
16.                 TcpClient client = listener.AcceptTcpClient();
17.                 Console.WriteLine("Whoa a new client connection :]");
18.                  //Start a new Thread for every new client connection
19.                 Thread t = new Thread(newClientRequests);
20.                 t.Start(client);
21.             } //while
22.         }//try
23.         catch (Exception e){
24.             Console.WriteLine(e);
25.         }
26.         finally{
27.             if (listener != null) {
28.                 listener.Stop();
29.             }
30.         }
31.     }//Main()
32. //Continue…
```

Server Example Explanation newClientRequests()

- In line **35**, this method takes one argument of type **Object**, which is cast immediately to a **TcpClient object.**
- The reason this cast is necessary is because the newClientRequests() method has the signature of a ParameterizedThreadStart.
  - Meaning to execute a static method and an instance method. With sent parameters
  - https://docs.microsoft.com/en-us/dotnet/api/system.threading.parameterizedthreadstart?view=net-5.0

```
34.     private static void newClientRequests(object argument) {
35.         //Same Code Taken From Single Threaded Server Program
36.         TcpClient client = (TcpClient)argument;
37.         try{
38.             StreamReader reader = new StreamReader(client.GetStream());
39.             StreamWriter writer = new StreamWriter(client.GetStream());
40.             string s = String.Empty;
41.             while (!(s = reader.ReadLine()).Equals("Exit") ||
                                        s.Equals("exit") || (s == null))
42.             {
43.                 Console.WriteLine("From client -> " + s);
44.                 writer.WriteLine("From server -> " + s);
45.                 //Show Connected IP
46.                 Console.WriteLine("Client connected with IP {0}",
                                        client.Client.LocalEndPoint);
47.                 writer.Flush();
48.             }
49.             reader.Close();
50.             writer.Close();
51.             client.Close();
52.             Console.WriteLine("Client connection terminated :[");
53.         }//try
54.     //Continue…
```

Server Example Explanation newClientRequests()

- In line **38,** we start a `StreamReaders` , so that we can get the received data(**41**). And write the received message to the console on line **43.**
- In line **39,** we start a `StreamWriter`, so that we can get the received data(**46**), use writer.WriteLine() send an echo message to the Client.
- On line **41**, the while loop checks the received message and prints the received until it receives an **"Exit"** message, than the server closes all readers and TCP received connections.
- In line **47**, writer.Flush() is used for clearing all buffered data and the data is sent to the Stream
  - So the next message dose not mix or get altered by the previous message.
  - Is used to send the data.

```
34.     private static void newClientRequests(object argument) {
35.         //Same Code Taken From Single Threaded Server Program
36.         TcpClient client = (TcpClient)argument;
37.         try{
38.             StreamReader reader = new StreamReader(client.GetStream());
39.             StreamWriter writer = new StreamWriter(client.GetStream());
40.             string s = String.Empty;
41.             while (!(s = reader.ReadLine()).Equals("Exit") ||
                                        s.Equals("exit") || (s == null))
42.             {
43.                 Console.WriteLine("From client -> " + s);
44.                 writer.WriteLine("From server -> " + s);
45.                 //Show Connected IP
46.                 Console.WriteLine("Client connected with IP {0}",
                                        client.Client.LocalEndPoint);
47.                 writer.Flush();
48.             }
49.             reader.Close();
50.             writer.Close();
51.             client.Close();
52.             Console.WriteLine("Client connection terminated :[");
53.         }//try
54.     //Continue…
```

Server Example Explanation newClientRequests()

- This part is for error catching as the program has multiple point of failures, so we have 1 error catching.
  - `IOException` is for all TCP connections and TCP listeners.
- And finally, in line **60**, after an error the server stops listing for TCP connections.

```
55. //Continue…
56. // Have only one exception unlike the Single Threaded Server Program
                                        because we don't have any IP inputs here
57.             catch (IOException){
58.                 Console.WriteLine("Problem with client communication :[");
59.             }
60.             finally {
61.                 if (client != null){
62.                     client.Close();
63.                 }
64.             }
65.         }//newClientRequests()
66.     }//class
67. }//namespace
```

Show All IP Address Listened - Server

- `ShowServerNetworkConfig()` This function can be used to show all IP that are being listed by the fowling line
  - `listener = new TcpListener(IPAddress.Any, port);`
- To run this put call the function in the main function in side the `try/catch.`

```
private static void ShowServerNetworkConfig(){
        Console.ForegroundColor = ConsoleColor.Yellow;
        NetworkInterface[] adapters = NetworkInterface.GetAllNetworkInterfaces();
        foreach(NetworkInterface adapter in adapters){
            Console.WriteLine(adapter.Description);
            Console.WriteLine("\tAdapter Name: " + adapter.Name);
            Console.WriteLine("\tMAC Address: " + adapter.GetPhysicalAddress());
            IPInterfaceProperties ip_properties = adapter.GetIPProperties();
            UnicastIPAddressInformationCollection addresses =
                                                ip_properties.UnicastAddresses;
            foreach(UnicastIPAddressInformation address in addresses){
                Console.WriteLine("\tIP Address: " + address.Address);
            }//foreach-UnicastIPAddressInformation
        }//foreach-NetworkInterface
        Console.ForegroundColor = ConsoleColor.White;
}//ShowServerNetworkConfig()
```

```
try{
    //Unlike our Single Threaded Server IPAddress.Any will listen
              to all Netwotk Interfaces and their Ip address
    //Ex. Wifi Module is 192.168.1.2 and Ethernet Interface is 172.0.0.1
    //IPAddress.Any will listen to both
    ShowServerNetworkConfig(); // Shows all IP that are being listened.
    listener = new TcpListener(IPAddress.Any, port);
    listener.Start();
    Console.WriteLine("Multi Threaded Server Has Started... :]");
    while (true)
    {
```

Program .EXE

- **File - MultiThreadedClientServer-EXE**
  - MultiTServer-Project.exe
  - MultiTClient-Project.exe
- **File - NormalClientServer-Project-EXE**
  - NormalClient-Project.exe
  - NormalServer-Project.exe

Thank You, That’s All

References - 1

- TCP Diagrams - https://www.cloudflare.com/en-gb/learning/ddos/glossary/tcp-ip/
- Socket Diagrams - https://www.pulpfreepress.com/csharp-for-artists-1st-edition/
- https://www.geeksforgeeks.org/socket-programming-in-c-sharp/
- https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-client-sockets
- https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-create-a-socket
- https://docs.microsoft.com/en-us/dotnet/framework/network-programming/
- https://docs.microsoft.com/en-us/dotnet/api/system.threading.parameterizedthreadstart?view=net-5.0

References - 2

- http://csharp.net-informations.com/communications/csharp-chat-client.htm
- http://csharp.net-informations.com/communications/csharp-chat-server-programming.htm
- http://csharp.net-informations.com/communications/csharp-chat-server.htm
- http://csharp.net-informations.com/communications/csharp-multi-threaded-socket.htm
- http://csharp.net-informations.com/communications/csharp-server-socket.htm
- http://csharp.net-informations.com/communications/csharp-socket-programming.htm
- https://www.youtube.com/watch?v=QrdfegS3iDg
  - https://foxlearn.com/articles/chat-tcp-ip-client-server-in-csharp-98.html