PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Intramuros, Manila

OPERATING SYSTEMS

Using TCP: File Transfer Between Devices Over the Same Network

Submitted by:
Calungsod, Kaye
Dognidon, Donna
Gabarda, Stephen
Tagnines, Sean

Date Submitted
18-02-2021

Submitted to:

Engr. Eufemia Garcia

Introduction:

According to TechTarget, TCP (Transmission Control Protocol) is a standard that specifies how a network conversation in which application programs may share data can be formed and maintained. TCP operates in accordance with Internet Protocol (IP), which determines how computers transmit data packets to each other.

This protocol enables two endpoints in a shared computer network to create a connection that allows data to be transmitted in two directions. Any data loss is detected and immediately corrected, which is why TCP is often referred to as a reliable protocol. TCP is typically used to coordinate information in such a way that safe transmission between the server and the client is guaranteed.– With this being said, TCP is generally used to arrange data in such a way that secure transmission between the server and the client is assured. It guarantees, irrespective of the number, the integrity of data sent over the network. For this purpose, data from other higher-level protocols that enable all transmitted data to arrive are used to transmit data.

The goal of this research is to simulate file sharing between two devices connected in the same network using TCP. Our group would like to learn and explore how the operating system helps in terms of file management and networking.

Methodology:

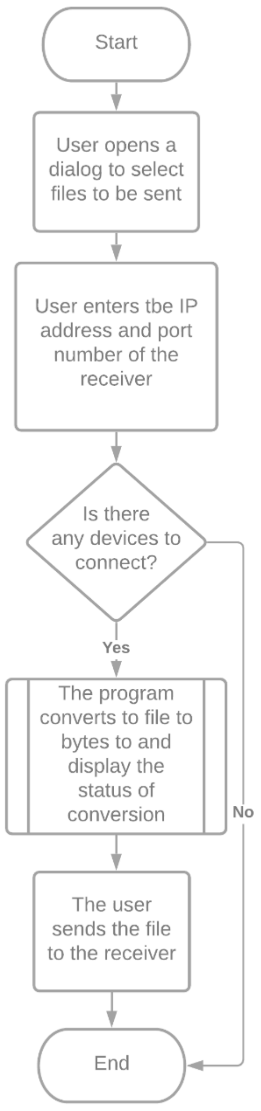

Start
User opens a dialog to select files to be sent
User enters tbe IP address and port number of the receiver
Is there any devices to connect?
Yes
The program converts to file to bytes to and display the status of conversion
No
The user sends the file to the receiver
End

Figure 1

Figure 1 represents the flow of the program when the user wants to send files using TCP. The user first opens the program and specifies which files to be sent. The user enters the IP address and port of the server. If the user connects to the server, the user can now send files. If there are no servers available, the user cannot send any files. When sending the files, the files are converted to bytes to speed up the process.

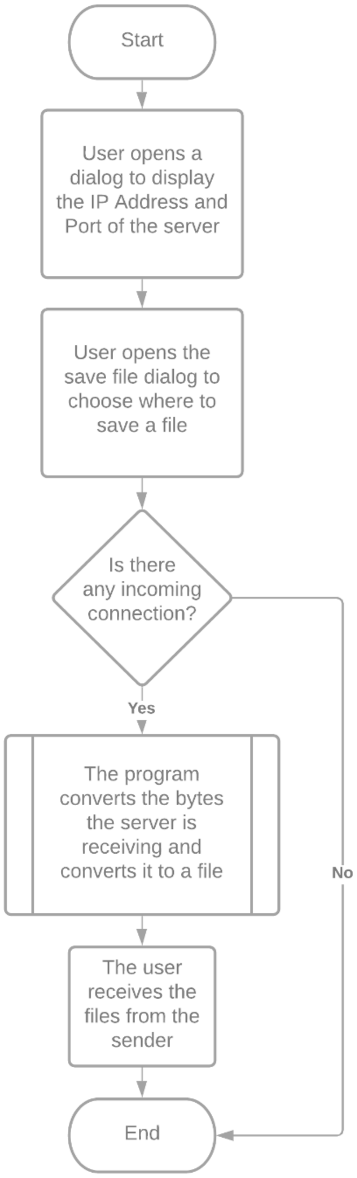

Start
User opens a dialog to display the IP Address and Port of the server
User opens the save file dialog to choose where to save a file
Is there any incoming connection?
Yes
No
The program converts the bytes the server is receiving and converts it to a file
The user receives the files from the sender
End

Figure 2

Figure 2 represents the flow for the program if the user wants to receive files. The user first opens the server to display the IP Address and Port. The user also will need to select the where to save the file to be received. If there is any incoming connection, the user can now receive files. If there is no connection, the user cannot receive files. When receiving the files, the bytes that are being received are converted into the file type that the receiver sent them. After receiving, the user can check the save folder to see the file.

Results:

Language Used: C#
Template Used: Windows Form Application

Note: The folder in MS Teams for Operating Systems named Group 3 - Project includes the zip file of our program and a video on how the program works.

Program Listing:

Main Form

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace File_Transfer_TCP
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        void Btn_clientClick(object sender, EventArgs e)
        {
            Client frm = new Client();
            frm.Show();
            this.Hide();
        }

        void Btn_serverClick(object sender, EventArgs e)
        {
            Server frm = new Server();
            frm.Show();
            this.Hide();
        }
        void Btn_exitClick(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Client

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

namespace File_Transfer_TCP
{
    //user interface where the client inputs the IP address and set port of the receiver to send files.
    public partial class Client : Form
    {

        private static string shortfileName = "";

        public Client()
        {
            InitializeComponent();
        }
        void Btn_backClick(object sender, EventArgs e)
        {
            MainForm frm = new MainForm();
            frm.Show();
            this.Close();
        }

        void Btn_browseClick(object sender, EventArgs e)
        { // opens an open file dialog so user can choose what file to send to the server
            OpenFileDialog openfile = new OpenFileDialog();
            openfile.Title = "File Sharing - Client";
            openfile.ShowDialog();
            txt_file.Text = openfile.FileName;
            shortfileName = openfile.SafeFileName; //to fetch the filename of the file without the
path
        }

            void Btn_sendClick(object sender, EventArgs e)
        {
                string IP = txt_ipaddress.Text;
                string fileName = txt_file.Text;
                int port = int.Parse(txt_port.Text);

                //checks if all the textboxes are empty or not
                if (txt_file.Text == "" || txt_ipaddress.Text == "" || txt_port.Text == "")
                {
                    MessageBox.Show("Please follow the instructions and fill in all the textboxes.
","Warning");
                    }
                else {  //calls the class SendTCP
                Task.Factory.StartNew(() => SendTCP(IP, port, fileName, shortfileName));
                lbl_status.Text = "Connected to the server.";
                MessageBox.Show("Succcessfully sent: \n" + shortfileName, "Notice");
                }
            }


            public void SendTCP(string remotehostIP, int Port, string filepath, string filename)
            {
                try
                {
                    if (!string.IsNullOrEmpty(remotehostIP))
                    {
                       //filepath and filename are converted to bytes.
                       //both are then copied into one byte variable - ClientData.
                    byte[] fileNameByte = Encoding.ASCII.GetBytes(filename);
                    byte[] fileData = File.ReadAllBytes(filepath);
                    byte[] clientData = new byte[4 + fileNameByte.Length + fileData.Length];
                    byte[] fileNameLen = BitConverter.GetBytes(fileNameByte.Length);
                    fileNameLen.CopyTo(clientData, 0);
                    fileNameByte.CopyTo(clientData, 4);
                    fileData.CopyTo(clientData, 4 + fileNameByte.Length);
                       //opens client socket; finds the server
                       //once the server is found, write or send the data
                    TcpClient clientSocket = new TcpClient(remotehostIP, Port);
                    NetworkStream networkStream = clientSocket.GetStream();
                    networkStream.Write(clientData, 0, clientData.GetLength(0));
                       //close
                    networkStream.Close();
                    }
                }
                catch (SocketException e)
                {
                    MessageBox.Show(e.Message);
                }
                catch (ArgumentNullException e)
                 {
                    MessageBox.Show(e.Message);
                 }
            }
        }
     }
```

Server

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


namespace File_Transfer_TCP
{
    //user interface where the server outputs the IP address and set port for the client to copy
    //server waits for connection from the client
    public partial class Server : Form
    {
        //gets the IP address of the device
        private static string ipAddress =
Dns.GetHostByName(Dns.GetHostName()).AddressList[0].ToString();
        private const int portNumber = 8080;

        /*once a file is received, class NewFileRecieved would be called*/
        public delegate void FileRecievedEventHandler(object source, string fileName);

        public event FileRecievedEventHandler NewFileRecieved;

        public Server()
        {
            InitializeComponent();
        }

        void ServerLoad(object sender, EventArgs e)
        {
            lbl_ipaddress.Text = ipAddress;
            lbl_port.Text = portNumber.ToString();
            this.NewFileRecieved += new FileRecievedEventHandler(Server_NewFileRecieved);
        }
```

```
 private void Server_NewFileRecieved(object sender, string fileName)
{
 /*used so that after the notice for the file being successfully received, program would
open folder where that file went to check*/
   this.BeginInvoke(
   new Action(
   delegate()
   {
      MessageBox.Show("Successfully received: \n" + fileName, "Notice");
      System.Diagnostics.Process.Start("explorer", txt_folderpath.Text);
   }));
}

void Btn_savetoClick(object sender, EventArgs e)
{ //opens folderbrowserdialog so that user can specify where to save the file received
   using(var saveto = new FolderBrowserDialog())
   {
      DialogResult result = saveto.ShowDialog();

      if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(saveto.SelectedPath))
      {
         string folder = saveto.SelectedPath;
         txt_folderpath.Text = folder;
      }
   }
}

void Btn_stClick(object sender, EventArgs e)
{
   //if the user didn't choose where the file received would be sent to, output an error
message
   if (txt_folderpath.Text == ""){
         MessageBox.Show("Please follow the instructions and choose a folder to save the
file to be received.","Warning");
      }
   else{
   //otherwise, start thread ReceiveTCP and transfer there the portNumber.
      Task.Factory.StartNew(() => ReceiveTCP(portNumber));
      lbl_status.Text = "Server is Running...";
```

```
    //a "\" is added in the string for the folderpath so that saving can be done properly
        txt_folderpath.Text = txt_folderpath.Text + "\\";
        MessageBox.Show("Listening on port: " + portNumber, "Notice");
        }

}

public void ReceiveTCP(int port)
{
    try
    {
        /*opens TCPListener or advertises that the server is running and can be
        connected to by the client*/
        TcpListener tcpListener = new TcpListener(IPAddress.Parse(ipAddress), port);
        tcpListener.Start();
        while (true)
        {   //automatically accepts connection from client, socket is formed for connection
            Socket handlerSocket = tcpListener.AcceptSocket();
            if (handlerSocket.Connected)
            { //if socket is connected, receive the bytes sent by the client
                string fileName = string.Empty;
                NetworkStream networkStream = new NetworkStream(handlerSocket);
                int thisRead = 0;
                int blockSize = 1024; //1024 bytes = 1Mb
                Byte[] dataByte = new Byte[blockSize];
                //locked to ensure that one thread is executing a piece of code at one time.
                lock (this)
                {   //bytes are converted back to their previous format
                    string folderPath = @txt_folderpath.Text;
                    handlerSocket.Receive(dataByte);
                    int fileNameLen = BitConverter.ToInt32(dataByte, 0);
                    fileName = Encoding.ASCII.GetString(dataByte, 4, fileNameLen);
                    //saving the file to the device using stream
                    Stream fileStream = File.OpenWrite(folderPath + fileName);
                    fileStream.Write(dataByte, 4 + fileNameLen, (1024 - (4 + fileNameLen)));
                    while (true)
                    {
                        thisRead = networkStream.Read(dataByte, 0, blockSize);
                        fileStream.Write(dataByte, 0, thisRead);
                        if (thisRead == 0)
```

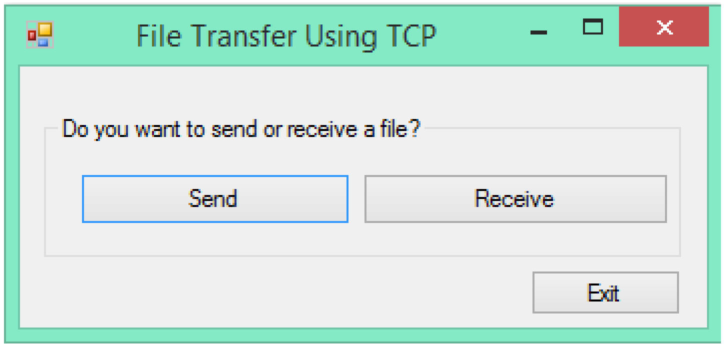
```
                                break;
                            }
                            fileStream.Close();
                        }
                        //once the file is received the NewFileReceived class above is called.
                        if (NewFileRecieved != null)
                        {
                            NewFileRecieved(this, fileName);
                        }
                        //closing socket
                           handlerSocket = null;
                    }
                }
            }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

        void Btn_backClick(object sender, EventArgs e)
        {
            MainForm frm = new MainForm();
            frm.Show();
            this.Close();
        }

    }
}
```

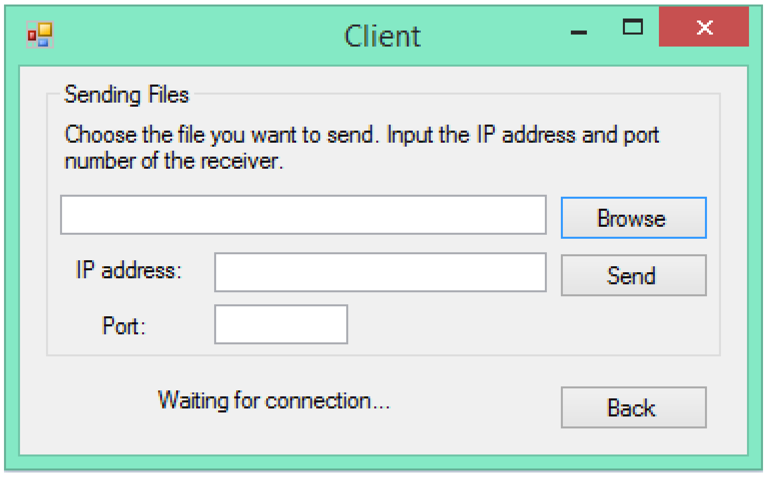
User Interface:

Main Form



The main form asks the user if they want to send or receive a file. If they click the Send button, they would open the client form. If they click the Receive button, they would open the server form. An exit button is also included in the main form to close and end the program.

Client



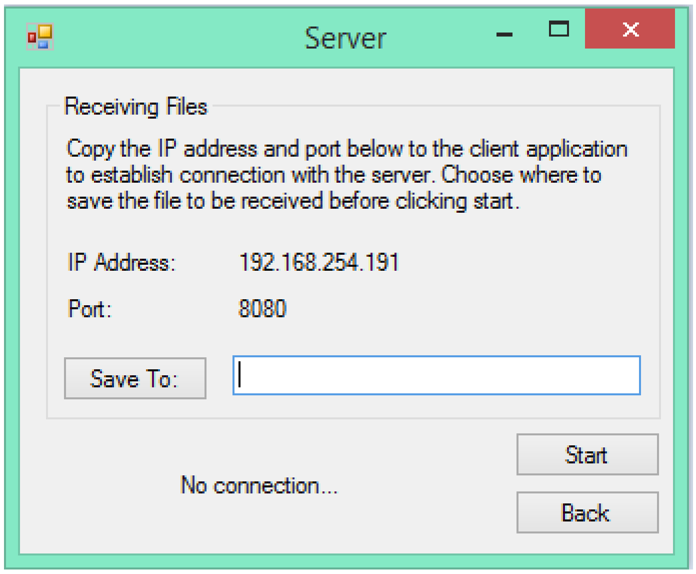
Choose the file you want to send. Input the IP address and port number of the receiver.

Waiting for connection...

The client form enables the user to send files to another device that would act as a server. It has three buttons: Browse, Send and Back. The Browse button is used to open an OpenFileDialog to fetch the file the user wants to send. In order to activate the Send button, the user needs to input the IP address and set port of the device that would act as a server. The Send button wouldn't work unless all textboxes have a proper output. The Back button redirects the user back to the main form.

Beside the Back button is a label that says "Waiting for connection…", this label is named in the program as lbl_status. It would change accordingly once a connection with the server is established.

Server



Copy the IP address and port below to the client application to establish connection with the server. Choose where to save the file to be received before clicking start.

IP Address: 192.168.254.191

Port: 8080

No connection...

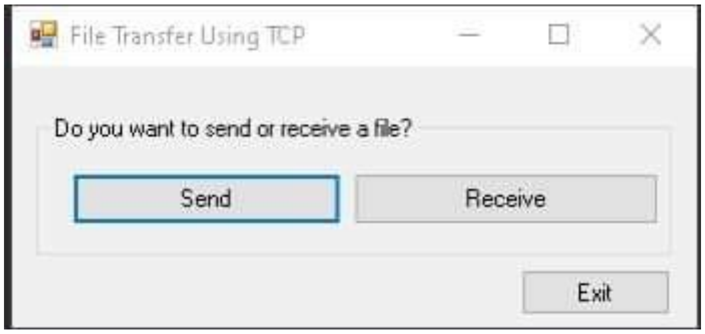
The server form enables the user to receive files from a client. It has three buttons: Save To, Start and Back. The Save To button enables the user to choose where to save th files to be received. The folder should be chosen before clicking start. If not, an error message would pop up as warning. The Start button enables the user to start listening or start looking for clients that want to connect to the server. The Back button redirects the user back to the main form.

Beside the Start and Back button is a label that says "No connection…", this label is named in the program as lbl_status. It would change accordingly once a connection with a client is established.

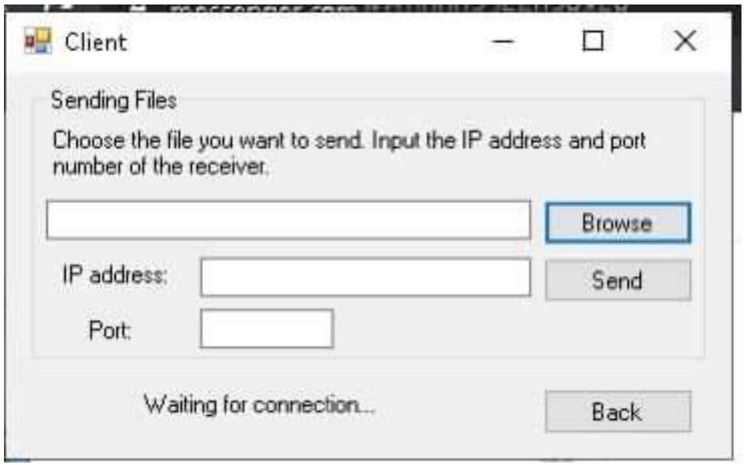
Demonstration:

A simulation of two devices enacting a file transfer using the Windows Form Application our group created.

- Open Main Form to ask if they want to send or receive a file.



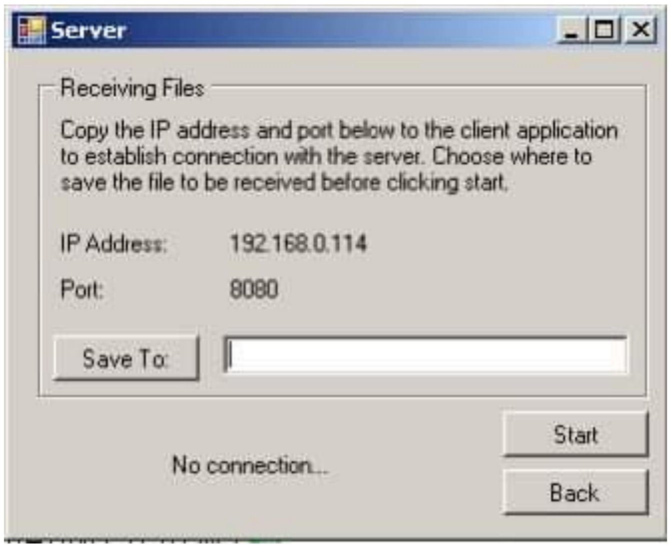
- The first device would act as the client,



Choose the file you want to send. Input the IP address and port number of the receiver.

Waiting for connection...

- The second device would act as the server.



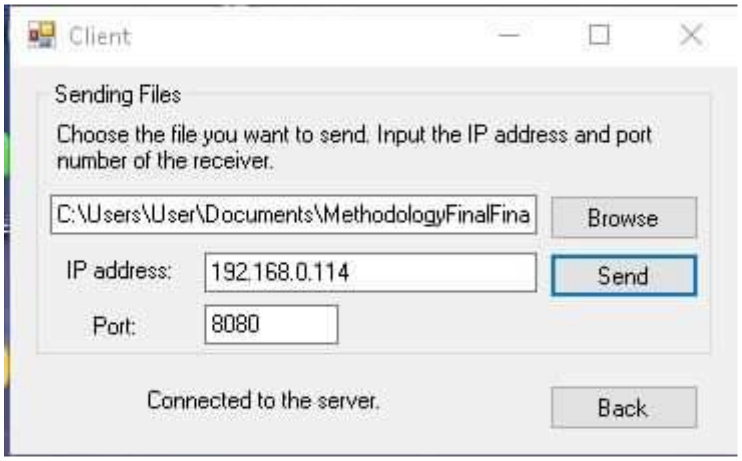
Copy the IP address and port below to the client application to establish connection with the server. Choose where to save the file to be received before clicking start.

IP Address: 192.168.0.114

Port: 8080

No connection...

- On the client side, the user would click browse and choose a file, then copy the IP Address and set port of the server.



Choose the file you want to send. Input the IP address and port number of the receiver.

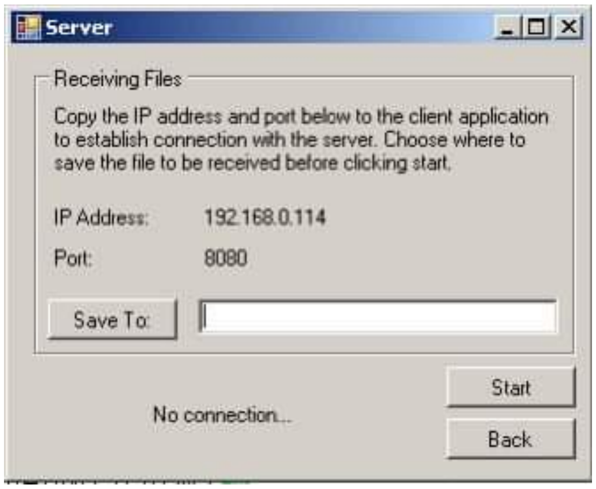
C:\Users\User\Documents\MethodologyFinalFina

IP address: 192.168.0.114

Port: 8080

Connected to the server.

- On the server side, the user would click save to and choose a folder.



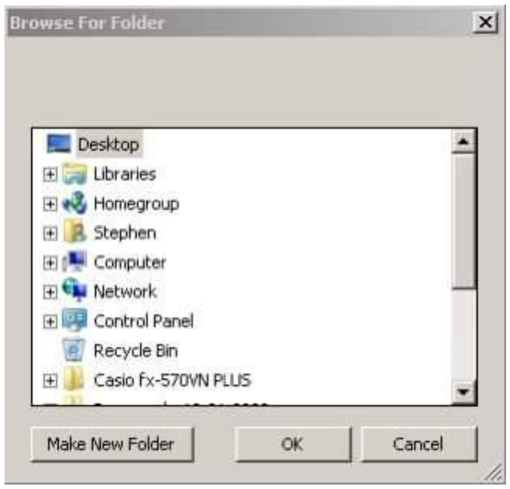
Server

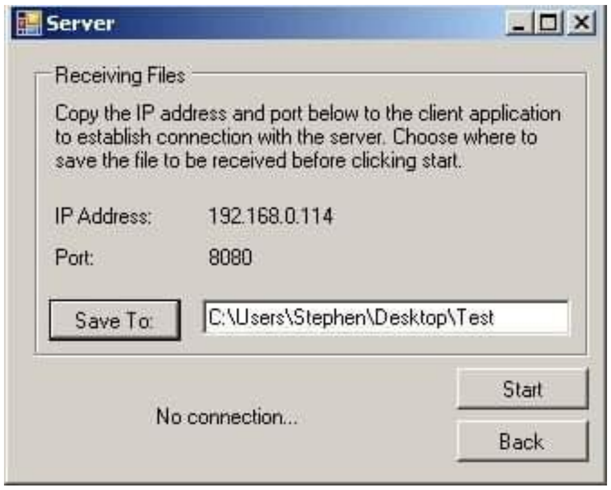
Receiving Files

Copy the IP address and port below to the client application to establish connection with the server. Choose where to save the file to be received before clicking start.

IP Address: 192.168.0.114

Port: 8080





Server

Receiving Files

Copy the IP address and port below to the client application to establish connection with the server. Choose where to save the file to be received before clicking start.

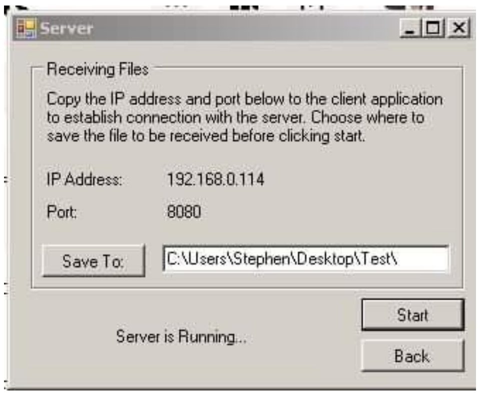
IP Address: 192.168.0.114

Port: 8080

Save To: C:\Users\Stephen\Desktop\Test

No connection...

- The server side would click start and get a message box that would inform the user that it’s listening on the set port. The status would also change from “No connection…” to “Server is Running…”



Server

Receiving Files

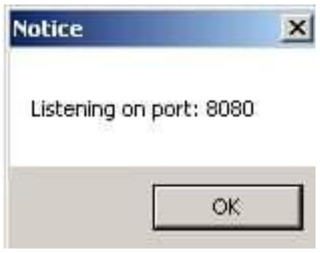
Copy the IP address and port below to the client application to establish connection with the server. Choose where to save the file to be received before clicking start.

IP Address: 192.168.0.114

Port: 8080

Save To: C:\Users\Stephen\Desktop\Test\

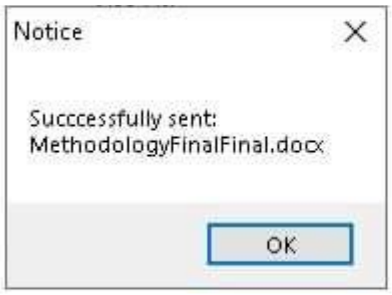
Server is Running...



Notice

Listening on port: 8080

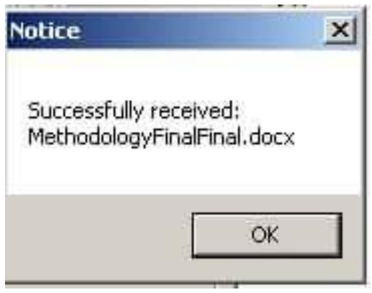
- After clicking the send button in the client side, a pop-up window saying “Successfully Sent” with the file name would appear.



Notice

Succcessfully sent:
MethodologyFinalFinal.docx

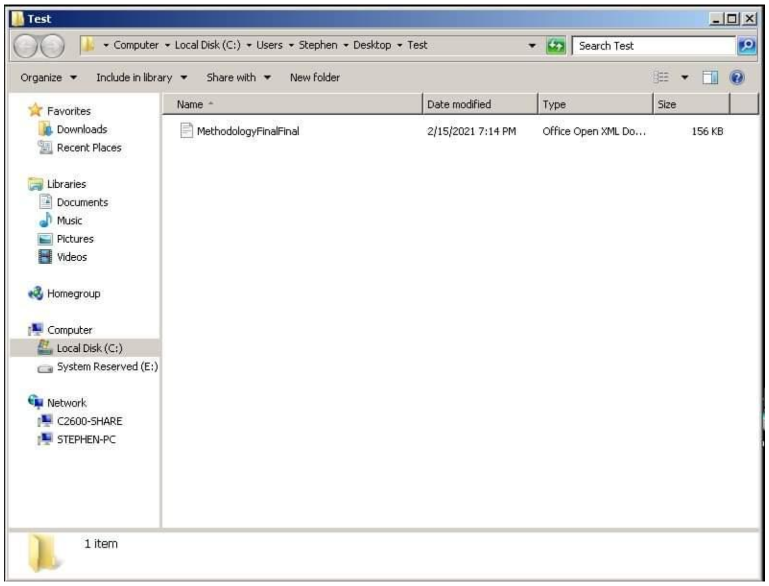
- On the server side, a window “Successfully Received” with the file name would also pop-up.



Notice

Successfully received:
MethodologyFinalFinal.docx

- After clicking ok in the pop-up window for the server side, the folder where that file is sent would automatically run to show the user that the file is successfully stored in that place.



Name	Date modified	Type	Size
MethodologyFinalFinal	2/15/2021 7:14 PM	Office Open XML Do...	156 KB

1 item

Discussion:

In our research we have touched over the following topics:

Threads

- A thread is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as lightweight processes. Our program is a single threaded process used to enact a half-duplex file transmission. Since the program can enable a device to both act as a client and a server (send and receive files) but not at the same time, the transmission can be two-way but it works asynchronously.

Socket Programming

A socket is a communications connection point (endpoint) that you can name and address in a network.

- Socket programming shows how to use socket APIs to establish communication links between remote and local processes. The processes that use a socket can reside on the same system or different systems on different networks. Sockets allow devices to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP.

In our program, socket programming was used to enact the file transfer.

TCPListener and **TCPClient**

TCPListener, listens for connections from TCP network clients.

- The TcpListener class provides simple methods that listen for and accept incoming connection requests in blocking synchronous mode. You can use either a TcpClient or a Socket to connect with a TcpListener. Create a TcpListener using an IPEndPoint, a Local IP address and port number, or just a port number. Specify Any for the local IP address and 0 for the local port number if you want the underlying service provider to assign those values for you.

TCPClient, provides client connections for TCP network services.

- The TcpClient class provides simple methods for connecting, sending, and receiving stream data over a network in synchronous blocking mode. In order forTcpClient to connect and exchange data, a TcpListener or Socket created with the TCP ProtocolType must be listening for incoming connection requests.

OpenDialog

- OpenDialog class displays a standard dialog box that prompts the user to open a file. This allows the user to check whether a file exists and open it. In our program, we used this to fetch the file to be sent by the client device.

FolderBrowserDialog

- The FolderBrowserDialog class prompts the user to select a folder. This class provides a way to prompt the user to browse, create, and eventually select a folder. Use this class when you only want to allow the user to select folders, not files. Browsing of the folders is done through a tree control. Only folders from the file system can be selected; virtual folders cannot. In our program, we used this to specify the file path the file received would be stored to.

In this paper, our group presented an overview of the Transfer Control Protocol (TCP) file transfer using socket programming. We demonstrated its fundamental features – client-server socket programming – in a C# Windows form application.

.

With the implementation of this project, the researchers learned that TCP is responsible for transporting data and ensuring it gets delivered to the destination application or device that IP has defined. We also learned how threading works in a client-server network and how the compilers and assemblies made it easier to call the operating system in helping transfer files from one device to another connected over the same network.

One of the most significant advantages of file transfer using TCP is that it is an open protocol suite – meaning it's highly scalable. Additionally, as a routable protocol, it can determine the most efficient path through the network. The program we made, when enacted and build upon with more complexity, can build up the internet architecture we currently enjoy now.

References:

- Hunt C. (2013). *TCP/IP Network Administration, 3rd Edition*. Oreilly. Retrieved from: https://www.oreilly.com/library/view/tcpip-network-administration/0596002971/ch01.html

- Rosencrance L., Gerwig K,, Novotny A. (2021). *TCP/IP (Transmission Control Protocol/Internet Protocol)*. SearchNetworking. Retrieved from: https://searchnetworking.techtarget.com/definition/TCP-IP

- *Introduction to Threads and Multithreading in OS | Studytonight*. (2020). Studytonight. Retrieved from: https://www.studytonight.com/operating-system/multithreading

- *IBM Knowledge Center*. (2018). IBM Knowledge Center. Retrieved from: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rzab6/rzab6soxoverview.htm

- K. (2021c). *TcpListener Class (System.Net.Sockets)*. Microsoft Docs. Retrieved from: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcplistener?view=net-5.0

- K. (2021c). *TcpClient Class (System.Net.Sockets)*. Microsoft Docs. Retrieved from: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcpclient?view=net-5.0

- D. (2021b). *OpenFileDialog Class (System.Windows.Forms)*. Microsoft Docs. Retrieved from: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.openfiledialog?view=net-5.0

- D. (2021a). *FolderBrowserDialog Class (System.Windows.Forms)*. Microsoft Docs. Retrieved from: https://docs.microsoft.com/enus/dotnet/api/system.windows.forms.folderbrowserdialog?view=net-5.0