Design Document 10/2/2023

RDT 1.0

Daniel Maccaline, Nathan Grady

Project Phase 2

**Phase purpose:**

The purpose of this phase is to transfer a file between a UDP client process and a UDP server process using reliable data transfer 1.0

**Code explanation**

**Client.py**

This bloc imports needed libraries.

*"""  
TCP Server  
Authors: Daniel Maccaline and Nathan Grady  
 Based on code from phase 1 (Daniel Maccaline)  
"""*import socket  
import os

The TCP server function creates a UDP socket and then binds that socket to a port number.

Then the server loops forever while it waits for packets to come in from the client

If it receives a packet then it will concatenate the packet onto the end of a byte array.

When it receives ‘stop’ it will then save the whole byte array into a bmp file and open the file in an image viewer.

Then it will clear the byte array and wait for more packets.

def TCPServer():  
  
 #string of bytes to hold passed file  
 frame=b""  
  
 #Define server port number  
 serverPort = 11000  
  
 #Create UDP Socket  
 serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 #Bind socket to port mumber  
 serverSocket.bind(('', serverPort))  
 #output message indicating ready to recieve  
 print('The server is ready to recieve')  
  
 #Loop forever, continually read messages sent to socket  
 while True:  
 #Recieve message, store message in sentence, store address of client that sent message to clientAddress  
 sentence, clientAddress = serverSocket.recvfrom(2048)  
  
 #if passed sentence = stop code  
 if(sentence==b'stop'):  
 #store created output to bmp file and open the file  
 f = open("temp.bmp", "wb")  
 f.write(frame)  
 f.close()  
 os.startfile("temp.bmp")  
  
 frame = b""  
 #Output completion statement  
 print("Finished recieving file\nFile opened in seperate window")  
  
 else:  
 #if not at end of file, concatenate sentence to frame  
 frame+=sentence  
 # Send message to client, used to prevent client from sending messages faster than server can handle them  
 serverSocket.sendto("Recieved".encode(), clientAddress)

the main function calls the TCPserver function

#Main method used to start server  
if \_\_name\_\_ == "\_\_main\_\_":  
 TCPServer()

**client.py**

this block imports libraries.

*"""  
TCP Client  
Authors: Daniel Maccaline and Nathan Grady  
 Based on code from phase 1 (Daniel Maccaline)  
"""*import time  
#import socket library  
import socket  
  
#import file dialog stuff  
import tkinter as tk  
from tkinter import filedialog

The Make\_Packets functions takes in a file and a packetSize as parameters. It will read the file and break the file into packets of size packetSize. It will return an array of all packets made from the file.

def Make\_Packets(file, packetSize):  
  
 #Read file data  
 data = file.read()  
  
 #Setup variable to track progress  
 currentIndex = packetSize  
  
 #Get first packet  
 packet = [data[0:packetSize]]  
  
 #loop and create all other packets  
 while(currentIndex < len(data)):  
 packet.append(data[currentIndex:currentIndex + packetSize])  
 currentIndex += packetSize  
  
 #Return results  
 return packet

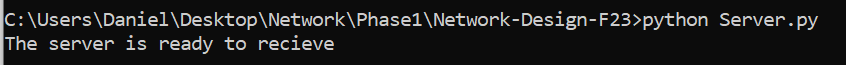
The TCPClient function has a filename as a parameter. It will call the make packet function to make a list of packets from the file. Then it will create a port. It will transmit each packet one at a time to the server waiting 0.1 seconds after each packet is sent. After the packet is sent it will send a message indicating all the packets have been sent.

#Client Functionality (called from main)  
def TCPClient(fileName):  
  
 #packet size in bytes  
 packetSize = 1024  
  
 #output message to indicate client startup/message contents  
 print('Starting Client to send image: ', fileName)  
  
 #read file in, output error if fails  
 try:  
 file = open(fileName,"rb")  
 except:  
 print("File could not be opened...")  
 return  
  
 #Create packets  
 packet = Make\_Packets(file, packetSize)  
  
 #Close files  
 file.close()  
  
#Transmit  
  
 #set server name and port to expect server at  
 serverName = 'localhost'  
 serverPort = 11000  
  
 #create UDP Socket  
 clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
  
 #Loop for each packet that needs to be sent  
 for i in range (0,len(packet)):  
 #Send packet  
 clientSocket.sendto(packet[i], (serverName, serverPort))  
 #Wait for server acknoledgement (will cause severe packet loss if client sends faster than server can recieve)  
 modifiedSentence, server = clientSocket.recvfrom(2048)  
  
 #Send code indicating end of file  
 time.sleep(0.1)  
 clientSocket.sendto(b'stop', (serverName, serverPort))  
  
 #Output statement to command line and close socket  
 print("Finished sending file")  
 clientSocket.close()

The main function is used to select a file and then call the TCPClient function to send the selected file.

#Main, used to start TCPClient and send name of passed file  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 #Open file select menue  
 root = tk.Tk()  
 root.withdraw()  
 file\_path = filedialog.askopenfilename()  
  
 #check if file is selected  
 if len(file\_path) <= 1:  
 #output error if no input file provided  
 print("Error: No input file specified")  
 else:  
 #pass input file name to client  
 TCPClient(file\_path)

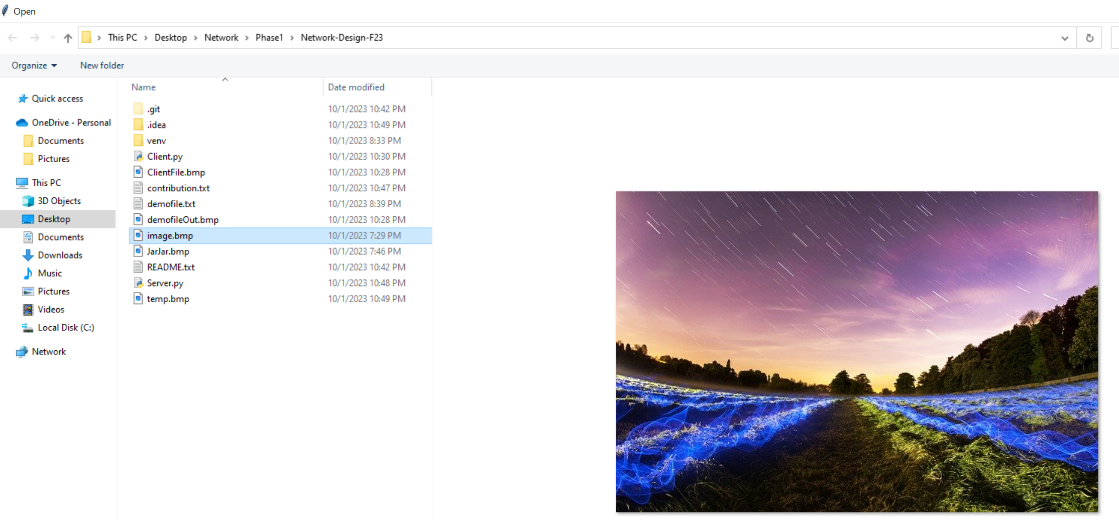
**Execution Example**



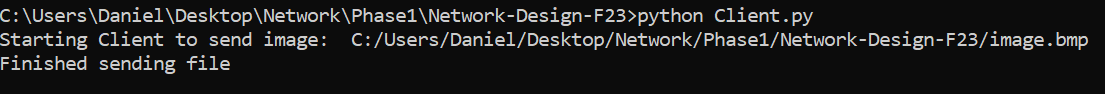
Command line command used to start the server code. In this case, the Server.py file is in the folder Network-Design-F23. In the above, you can also see the output provided by the server before the client is run.



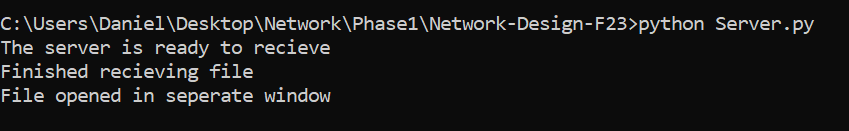
Command line command used to start the client code. In this case, the Client.py file is in the folder Network-Design-F23. Note after the above is run, a standard file select window will open



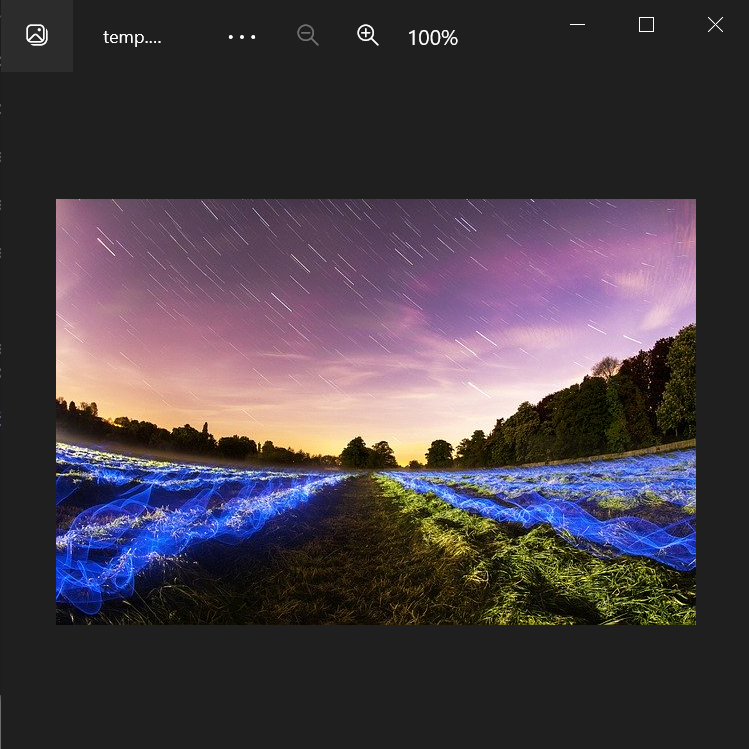
File Select window opened when client.py is run



Example output produced from the client, assuming the server is running.



Example output produced by the server. Note that the server does not exit after these messages. As such, Client can be re-run without re-starting the server.



Window opened by the server containing the received image.