Project 2

CIS 453

Lucas Myers

**CONNECT**

Text

Description automatically generated

**LIST**

**Graphical user interface, text, application

Description automatically generated**

**RETRIEVE**

Graphical user interface, text, application

Description automatically generated

**STORE**

**Graphical user interface, text, application

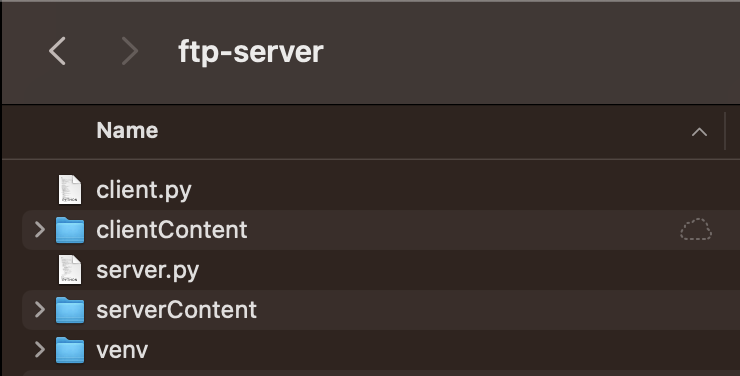
Description automatically generated**

**QUIT**

**Graphical user interface, text

Description automatically generated**

**Retrieve places the files in a folder called “clientContent” which must exist in the same directory as the client. The files are retrieved from “serverContent” which must exist in the same directory as the server. The opposite is true for the store command.**

****

**clientContent before executing the commands above**

**Graphical user interface, application

Description automatically generated**

**clientContent after executing the commands above**

**Graphical user interface, application

Description automatically generated**

**serverContent before executing the commands above**

**Graphical user interface, application

Description automatically generated**

**serverContent after executing the commands aboveGraphical user interface, application

Description automatically generated**

**The program implements error checking for input. The program will also handle cases where the requested file does not exist on the server and where the file trying to be stored does not exist on the client.**

**Text

Description automatically generated**

**Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated**

**Client Logic**

import os  
import socket  
import random  
  
  
def no\_connection():  
 # create client control socket  
 control\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 # attempt to connect  
 command = input("Please connect to a server by typing: CONNECT <server name/IP address> <server port>\n")  
 command = command.split(' ')  
  
 # if wrong format for connect command, retry  
 if len(command) < 3 or command[0] != 'CONNECT':  
 no\_connection()  
  
 # if right format, try to connect  
 try:  
 control\_socket.connect((command[1], int(command[2])))  
 # if unable to connect, try again  
 except:  
 print('Could not connect.')  
 no\_connection()  
  
 # begin sending commands  
 send\_command(control\_socket)  
  
  
def send\_command(control\_socket):  
 # send the server our port and create server data socket  
 control\_socket.send(str(new\_port).encode('ascii'))  
 data\_socket, addr = welcome\_socket.accept()  
  
 # get command as input, parse, and go  
 # checking for valid command  
 command = input("Please enter a command\n")  
 commandParsed = command.split(' ')  
 while len(commandParsed) > 2:  
 command = input("Please enter a valid command\n")  
 commandParsed = command.split(' ')  
 if len(commandParsed) == 1:  
 while commandParsed[0] not in oneCommands:  
 command = input("Please enter a valid command\n")  
 commandParsed = command.split(' ')  
 if len(commandParsed) == 2 and commandParsed[0] in twoCommands:  
 break  
 if len(commandParsed) == 2:  
 while commandParsed[0] not in twoCommands:  
 command = input("Please enter a valid command\n")  
 commandParsed = command.split(' ')  
 if len(commandParsed) == 1 and commandParsed[0] in oneCommands:  
 break  
  
 # executing command  
 if len(commandParsed) == 1:  
 # list command  
 if command == 'LIST':  
 control\_socket.send(command.encode('ascii'))  
 files = data\_socket.recv(buffer\_size).decode('ascii')  
 print(files)  
 # quit command  
 elif command == 'QUIT':  
 control\_socket.send(command.encode('ascii'))  
 control\_socket.close()  
 data\_socket.close()  
 no\_connection()  
  
 elif len(commandParsed) == 2:  
 # retr command  
 if commandParsed[0] == 'RETR':  
 control\_socket.send(command.encode('ascii'))  
 with open('clientContent/' + commandParsed[1], 'wb') as f:  
 chunk = data\_socket.recv(buffer\_size)  
 try:  
 if chunk.decode('ascii') == 'File not found.':  
 print("File not found.")  
 os.remove('./clientContent/' + commandParsed[1])  
 else:  
 while chunk:  
 f.write(chunk)  
 chunk = data\_socket.recv(buffer\_size)  
 print("Downloaded.\n")  
 except UnicodeDecodeError:  
 while chunk:  
 f.write(chunk)  
 chunk = data\_socket.recv(buffer\_size)  
 print("Downloaded.\n")  
 # stor command  
 elif commandParsed[0] == 'STOR':  
 control\_socket.send(command.encode('ascii'))  
 try:  
 with open('clientContent/' + commandParsed[1], 'rb') as f:  
 chunk = f.read(buffer\_size)  
 while chunk: # chunk == '' indicates EOF from file  
 data\_socket.send(chunk)  
 chunk = f.read(buffer\_size)  
 print("Stored " + commandParsed[1])  
 except FileNotFoundError:  
 print('File not found.')  
 response = 'File not found.'  
 data\_socket.send(response.encode('ascii'))  
  
 data\_socket.close()  
 send\_command(control\_socket)  
  
  
# only valid commands  
oneCommands = ['LIST', 'QUIT']  
twoCommands = ['RETR', 'STOR']  
  
# create server welcome socket  
server\_ip = 'localhost'  
new\_port = random.randrange(1000, 8000)  
buffer\_size = 1024  
welcome\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
welcome\_socket.bind((server\_ip, new\_port))  
welcome\_socket.listen()  
  
# force a connection  
no\_connection()

**Server Logic**

import socket  
import threading  
import os  
  
  
def read\_command(control\_socket):  
 # get port number of client's data\_socket  
 new\_port = int(control\_socket.recv(buffer\_size).decode('ascii'))  
  
 # create client data socket  
 data\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 # connect data socket to client  
 data\_socket.connect((server\_ip, new\_port))  
  
 # receive the command from client and parse  
 command = control\_socket.recv(buffer\_size).decode('ascii')  
 if len(command.split(' ')) == 1:  
 if command == 'LIST':  
 files = os.listdir('./serverContent')  
 files = [f for f in files if os.path.isfile('./serverContent/' + f)]  
 files = [f for f in files if not f.startswith('.')]  
 files = ' '.join(files)  
 data\_socket.send(files.encode('ascii'))  
 elif command == 'QUIT':  
 data\_socket.close()  
 control\_socket.close()  
 return  
 else:  
 commandParsed = command.split(' ')  
 if commandParsed[0] == 'RETR':  
 try:  
 with open('serverContent/' + commandParsed[1], 'rb') as f:  
 chunk = f.read(buffer\_size)  
 while chunk: # chunk == '' indicates EOF from file  
 data\_socket.send(chunk)  
 chunk = f.read(buffer\_size)  
 except FileNotFoundError:  
 response = 'File not found.'  
 data\_socket.send(response.encode('ascii'))  
 if commandParsed[0] == 'STOR':  
 with open('serverContent/' + commandParsed[1], 'wb') as f:  
 chunk = data\_socket.recv(buffer\_size)  
 try:  
 if chunk.decode('ascii') == 'File not found.':  
 print("File not found.")  
 os.remove('./serverContent/' + commandParsed[1])  
 else:  
 while chunk:  
 f.write(chunk)  
 chunk = data\_socket.recv(buffer\_size)  
 print("Downloaded " + commandParsed[1] + "\n")  
 except UnicodeDecodeError:  
 while chunk:  
 f.write(chunk)  
 chunk = data\_socket.recv(buffer\_size)  
 print("Downloaded.\n")  
  
 data\_socket.close()  
 read\_command(control\_socket)  
  
  
# server parameters  
server\_ip = 'localhost'  
server\_port = 8907  
buffer\_size = 1024  
  
# create server welcome socket, bind, and listen  
welcome\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
welcome\_socket.bind((server\_ip, server\_port))  
welcome\_socket.listen()  
  
print('The server is ready to communicate')  
  
# accept incoming connections and multithreading  
while True:  
 control\_socket, addr = welcome\_socket.accept()  
 threading.Thread(target=read\_command, args=(control\_socket,)).start()

**Reflection**

There were only two problems I encountered during this project. The first problem was getting the control and data sockets working properly at the same time. I was initially creating the data socket in the wrong place, which did not allow the socket to close properly. I tried to create the data socket as a global variable then open and close the socket when needed. It turns out that closing a socket prevents any future operations from being performed on it. So, I had to overwrite the data socket with a new version of itself every time I sent a message.

The other difficulty was parsing the input properly. I could not simply re-execute my send method because I did not want to close my data socket unless the command was performed correctly. I had to use a combination of loops and conditional statements to detect errors in input correctly. This was the longest task of the entire project.

I think that the program could be refined in a few ways. The output to the terminal could be formatted nicely using a python library. The server could also report and log important information (to the terminal or a text file) like files that have been downloaded/stored, who (ip/port) is downloading/storing them, and when they are downloading/storing. I only output to the server terminal when a file is stored on the server.