Project 3

CIS 457 01

Lucas Myers

Completion Percentage = 100%

(Reflection at the bottom after logic)

**USER REGISTRATION**

**Text

Description automatically generated**

**Multiple users may register at the same time. The server handles this case by creating a temp file for each user separately. Upon quitting or exiting, the metadata is deleted.**

**USER QUERY**

**Text

Description automatically generated**

Bob is searching for the keyword ‘fire’

**The program searches by opening every user’s metadata file and recording which lines have one or more keywords that match. These matching lines are returned to the client. If a user quits, their files are no longer searchable/downloadable.**

**USER DOWNLOAD**

**Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated**

**BOB BEFORE ALICE BEFORE**

**Text

Description automatically generated**

Bob gets the fire hydrant image

**Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated**

**BOB AFTER ALICE AFTER**

**Downloading is handled between the clients with no interaction from the central server.**

**The program will check for errors in input, but it may not be 100% bulletproof. The clients must be in different directories for the file transfer to work properly, else it will overwrite its own file with an empty one (not sure why anyone would want to do this anyway).**

**CLIENT LOGIC**

import random  
import socket  
import os  
import csv  
import sys  
import threading  
  
  
def connect\_to\_central():  
 # create connection socket for central server  
 central\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 # attempt to connect  
 print("All files in the clientContent folder will be available for download by peers!\n")  
 command = input("Please connect to the central server by typing: CONNECT <server name/IP address> <server port>\n")  
 command = command.split(' ')  
  
 if len(command) < 3 or command[0] != 'CONNECT':  
 connect\_to\_central()  
  
 # if right format, try to connect  
 try:  
 central\_server\_socket.connect((command[1], int(command[2])))  
 # if unable to connect, try again  
 except:  
 print('Could not connect.')  
 connect\_to\_central()  
  
 # begin sending commands  
 connected(central\_server\_socket)  
  
  
def connected(central\_server\_socket):  
 # create command socket to handle metadata transfer  
 central\_server\_socket.send(str(command\_port).encode('ascii'))  
 command\_socket, addr = welcome\_socket2.accept()  
  
 # get username, hostname, connection speed  
 username = input("Please enter your username: ")  
 connection\_speed = input("Please enter your connection speed: ")  
  
 # iterate through files in clientContent and make user add description for each  
 # create an array of file names  
 files = os.listdir('./clientContent')  
 files = [f for f in files if os.path.isfile('./clientContent/' + f)]  
 files = [f for f in files if not f.startswith('.')]  
 # create an array of keywords  
 keywords = []  
 for f in files:  
 keyword = input("Please enter keywords (seperated by space) for " + f + ":")  
 keywords.append(keyword)  
 # write initial info, hostname, filename, keywords to csv file  
 with open('myFiles.csv', 'w', newline='') as file:  
 writer = csv.writer(file)  
 for f in files:  
 writer.writerow([username, socket.gethostname(), new\_port, connection\_speed, f, keywords[files.index(f)]])  
 # send csv of our initial info, files, and keywords to central server  
 with open('myFiles.csv', 'rb') as f:  
 chunk = f.read(buffer\_size)  
 while chunk: # chunk == '' indicates EOF from file  
 command\_socket.send(chunk)  
 chunk = f.read(buffer\_size)  
 # CSV FORMAT (name = myFiles):  
 # username, hostname, port, speed, filename, keywords  
  
 # at this point, our client has sent all its metadata to server  
 command\_socket.close()  
 send\_command(central\_server\_socket)  
  
  
def send\_command(central\_server\_socket):  
 # send the central server a port for our command(metadata) socket  
 central\_server\_socket.send(str(command\_port).encode('ascii'))  
 command\_socket, addr = welcome\_socket2.accept()  
  
 # COMMANDS  
 command = input(  
 "Connected to server. Please enter a command (SEARCH <keyword,keyword,...> GET <filename> or QUIT\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  
  
 # QUIT  
 if len(commandParsed) == 1:  
 if command == 'QUIT':  
 central\_server\_socket.send(command.encode('ascii'))  
 central\_server\_socket.close()  
 command\_socket.close()  
 sys.exit()  
  
 # SEARCH and GET  
 else:  
 if commandParsed[0] == 'SEARCH':  
 central\_server\_socket.send(command.encode('ascii'))  
 files = command\_socket.recv(buffer\_size).decode('ascii')  
 files = files.split('/')  
 for f in files:  
 print(f)  
 elif commandParsed[0] == 'GET':  
 # Let the central server know we are getting  
 central\_server\_socket.send(command.encode('ascii'))  
 # create a control socket for sending filename to host  
 control\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 # enter location to connect to  
 dest = input("Please specify <host port file>:")  
 while len(dest.split(' ')) != 3:  
 print("Invalid input.")  
 dest = input("Please specify <host port file>:")  
  
 dest = dest.split(' ')  
  
 # connect to host  
 try:  
 control\_socket.connect((dest[0], int(dest[1])))  
 except:  
 print("Could not connect.")  
 command\_socket.close()  
 control\_socket.close()  
 send\_command(central\_server\_socket)  
  
 # send the host our data port (for sending/receiving file)  
 control\_socket.send(str(data\_port).encode('ascii'))  
 data\_socket, addr = welcome\_socket3.accept()  
 # send file name over control socket  
 control\_socket.send(dest[2].encode('ascii'))  
  
 # receive and write the file  
 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.")  
 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")  
 data\_socket.close()  
 control\_socket.close()  
  
 command\_socket.close()  
 send\_command(central\_server\_socket)  
  
  
def send\_file(control\_socket):  
 # receive the data port and connect ot hosts data socket  
 data\_port = int(control\_socket.recv(buffer\_size).decode('ascii'))  
 data\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 data\_socket.connect((server\_ip, data\_port))  
 # accept file name  
 command = control\_socket.recv(buffer\_size).decode('ascii')  
 # open file and send to host  
 try:  
 with open('clientContent/' + command, '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'))  
  
 data\_socket.close()  
 control\_socket.close()  
  
  
# only valid commands  
oneCommands = ['QUIT']  
twoCommands = ['SEARCH', 'GET']  
  
# info for server  
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()  
  
# info for command socket  
command\_port = random.randrange(1000, 8000)  
welcome\_socket2 = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
welcome\_socket2.bind((server\_ip, command\_port))  
welcome\_socket2.listen()  
  
# info for data socket  
data\_port = random.randrange(1000, 8000)  
welcome\_socket3 = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
welcome\_socket3.bind((server\_ip, data\_port))  
welcome\_socket3.listen()  
  
# start client thread  
threading.Thread(target=connect\_to\_central).start()  
# start server thread  
while True:  
 control\_socket, addr = welcome\_socket.accept()  
 threading.Thread(target=send\_file, args=(control\_socket,)).start()

**SERVER LOGIC**

import os  
import socket  
import threading  
import csv  
import random  
  
  
def central\_server(central\_server\_socket):  
 # receive command port (metadata) from client and connect command socket  
 command\_port = int(central\_server\_socket.recv(buffer\_size).decode('ascii'))  
 command\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 command\_socket.connect((server\_ip, command\_port))  
  
 # receive client's metadata and store in a temp csv file  
 fileNumber = random.randrange(1, 9999999999999999999999)  
 fileName = 'temp' + str(fileNumber) + '.csv'  
  
 # ensure the file does not already exist  
 files = os.listdir('./serverContent')  
 files = [f for f in files if os.path.isfile('./serverContent/' + fileName)]  
 files = [f for f in files if not f.startswith('.')]  
 while fileName in files:  
 fileNumber = random.randrange(1, 9999999999999999999999)  
 fileName = 'temp' + str(fileNumber) + '.csv'  
  
 # write host's metadata to a file  
 with open('serverContent/' + fileName, 'wb') as f:  
 chunk = command\_socket.recv(buffer\_size)  
 while chunk:  
 f.write(chunk)  
 chunk = command\_socket.recv(buffer\_size)  
  
 # open the file and print user connected  
 with open('serverContent/' + fileName, 'r', newline='') as f:  
 reader = csv.reader(f)  
 user = next(reader)  
 active\_user = user[0]  
 print(active\_user + " connected")  
  
 # at this point, our server has all the metadata from the  
 # connected user.  
 receive\_command(central\_server\_socket, active\_user, fileName)  
  
  
def receive\_command(central\_server\_socket, active\_user, fileName):  
 # create command port to handle transfer of metadata  
 command\_port = int(central\_server\_socket.recv(buffer\_size).decode('ascii'))  
 command\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 command\_socket.connect((server\_ip, command\_port))  
  
 # RECEIVE COMMANDS  
 command = central\_server\_socket.recv(buffer\_size).decode('ascii')  
 if len(command.split(' ')) == 1:  
 # QUIT  
 # Remove the host's metadata, close sockets  
 os.remove('./serverContent/' + fileName)  
 command\_socket.close()  
 central\_server\_socket.close()  
 print(active\_user + " has disconnected")  
 return  
  
 else:  
 commandParsed = command.split(' ')  
 # SEARCH  
 if commandParsed[0] == 'SEARCH':  
  
 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('.')]  
  
 searched = []  
 keywords = commandParsed[1].split(',')  
 for f in files:  
 with open('serverContent/' + f, 'r', newline='') as file:  
 reader = csv.reader(file)  
 for row in reader:  
 terms = row[5].split(' ')  
 added = False  
 for keyword in keywords:  
 if keyword in terms:  
 searched.append(  
 "Hostname: " + row[1] + " Port: " + row[2] + " Filename: " + row[4] + " Speed: " +  
 row[  
 3] + '/')  
 added = True  
 if added:  
 continue  
 searched = ''.join(searched)  
 command\_socket.send(searched.encode('ascii'))  
  
 else:  
 # GET  
 pass  
  
 command\_socket.close()  
 receive\_command(central\_server\_socket, active\_user, fileName)  
  
  
# 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:  
 central\_server\_socket, addr = welcome\_socket.accept()  
 threading.Thread(target=central\_server, args=(central\_server\_socket,)).start()

**Reflection**

There were two main challenges during the creation of this program. The first problem was receiving empty files from the peer. This was due to running the clients in the same directory. Running the clients in the same directory led to both clients opening the file, which resulted in deleting the contents. This was solved by running the clients in separate directories. In a realistic scenario, nobody would likely be running two of these clients in the same directory.

The second problem occurred when two users are trying to connect at the same time. The server would attempt to write the metadata to the same file. This resulted in collisions in the file, which prevented the commands from working properly (failure to parse the metadata). This was solved by creating a temp metadata file for each user upon connection. This way the server would never be writing to the same file in more than one instance.