Matthew Palkimas

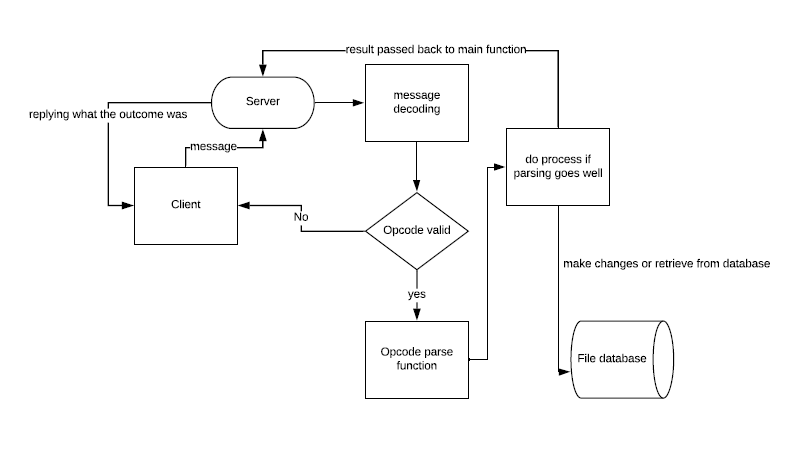
ECE 470

**1.0 Server Design**:

* 1. Server will decode messages sent in the design shown in **2.0**. If not sent correctly it will return an error message saying where it went wrong.

Note: this will be after the user has connected successfully.

* 1. The main server architecture will be spread out through this document.
  2. Overall Design after login was successful



**2.0 Message Design:**

*2.1* First three characters: Opcode

Current valid opcodes: STR, RTV, DEL, INF, LGO

In order store, retrieve, delete, information, logout

*2.2* Depending on the opcode the rest of the message will need to be encoded in a certain manner.

*2.3* To end the message it will be two /r/n and to distinguish between two strings one /r/n will be needed.

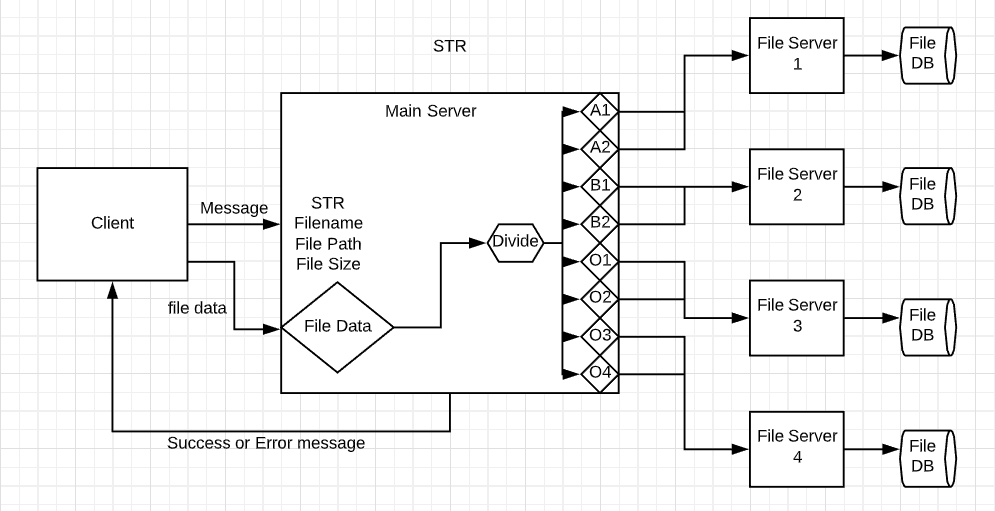
*2.4* STR:

Store

First string will be file name

Second string will be file path

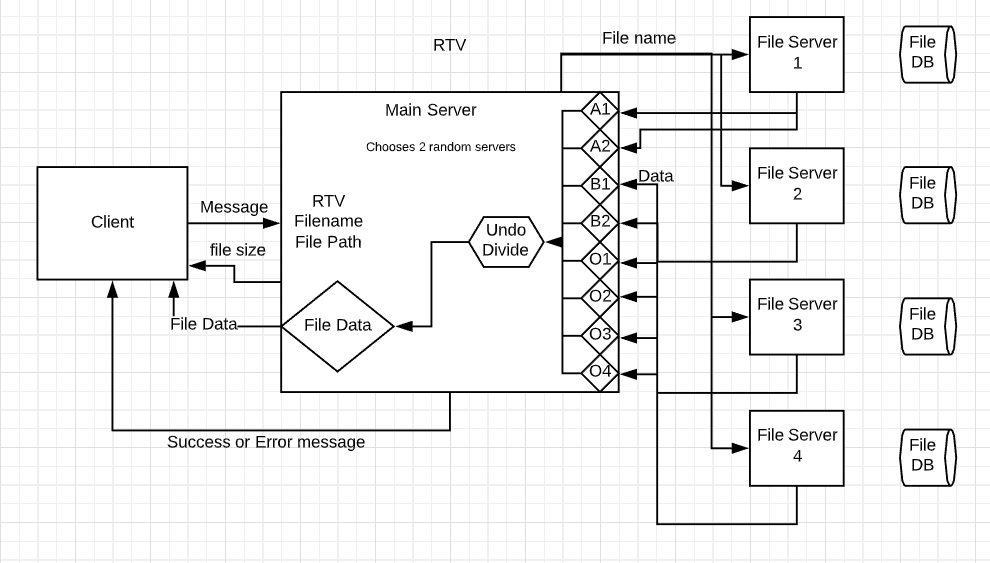
Third string will be file size



*2.5* RTV:

Retrieve

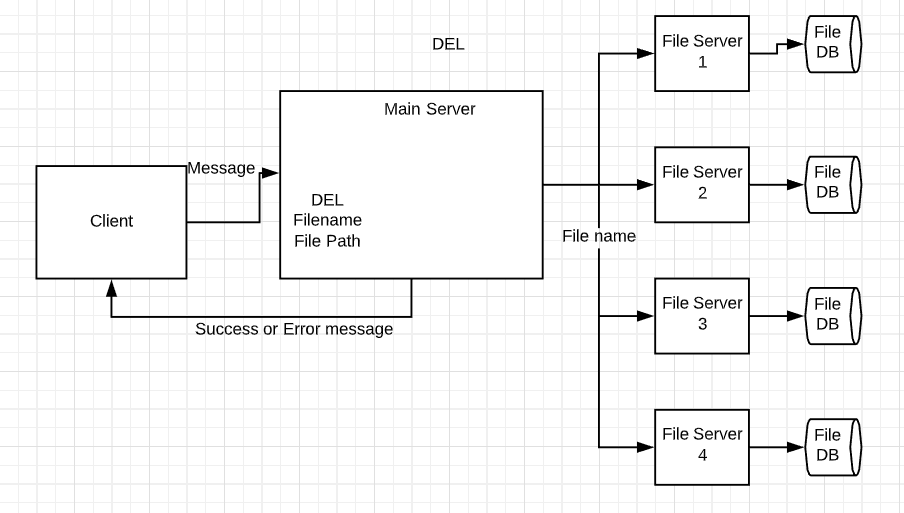
First string will be file name

Second string will be file path

*2.6* DEL:

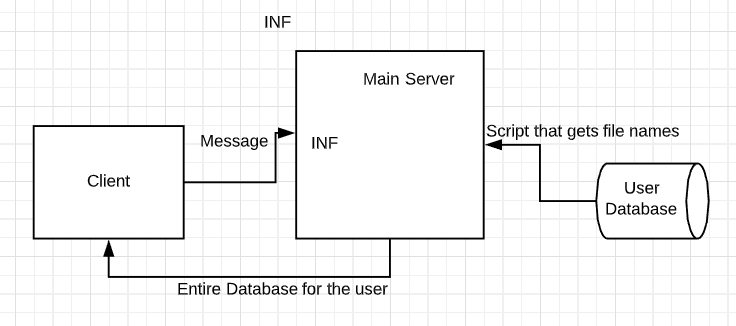
Delete

First string will be file name

Second string will be file path

*2.7* INF:

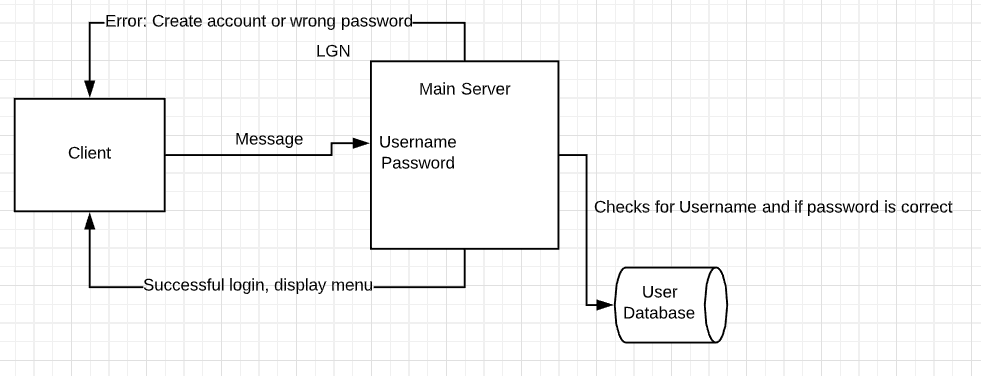
Info on Files

Will return the entire database in a structured file, see *3.6* for more info.

*2.8* LGN:

Login

Technically this opcode will not be needed to ever be sent to the server from the client as the only time the user can login will be when it first connects to the server, or logouts.

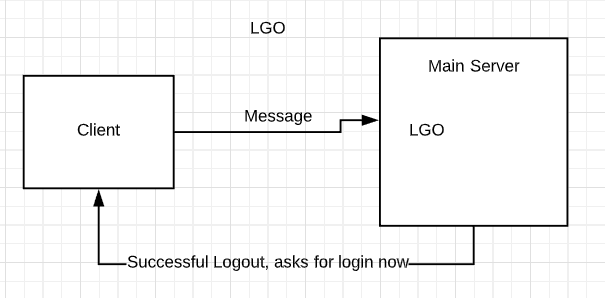


*2.9* LGO:

Logout

Just LGO will need to be sent.

Disconnects the user out of the server.



**3.0 Data Storage Protocol:**

*3.1* File system setup:

The file system will be stored locally where the file server is running. It will start off by saying the exact location of the file database which will be the current working directory joined with 'files\_$FILE\_SERVER\_NUMBER'

*3.2* When a ‘new’ user connects to the server their files/{username} directory will be created in all file servers by sending each file server the username. In the user database the directory will also be created.

*3.3* The client will not be able to create files directly in their 'files' directory but will instead have to give a path inside their ‘files’ directory.

*3.4* The server will have the capability of creating directories.

*3.5* The server will not be storing the files exactly as they are sent, instead the files will be distributed to four different servers in pieces, although it will store their names in the User database.

*3.6* As the server side database is built up, **5.0**, there will be a secondary storage for all the names and paths for the files that are being stored so that when the user requests INF the server doesn’t need to do anything complicated to return a nice built structure. (user database)

*3.7* The max file size that the server can handle will be 100MB although this could go up if I feel the need.

*3.8* A backup feature should be available to the server so that inside the files directory there will be a backup directory that of course the client doesn't have access to. But in case anything goes wrong it will be able to load from that.

**4.0 User Access Protocol:**

*4.1* The client will act as a “dumb client” between the server and the user by just passing whatever the server gives it to the user, and whatever the user gives it to the server.

*4.2* This client will send a ‘hi’ message that will be encoded as follows:

HI\r\n\r\n

In the future this message could include more than just HI to provide the server some information. But for now, this will do so I can keep the request response architecture.

Note: If the server is connected to and does not receive just this message it will disconnect immediately.

*4.3* The server will send back a message that will inform the client a success happened, and it will provide the info to ask the client to get the username and password from the user.

Formatted by the server:

1. Success, Error or Disconnect Character
2. Number of items the client needs to ask the user to fill in
3. Item number 1’s text to display to inform the user on how to fill it in correctly
4. Item number 2’s text to display to inform the user on how to fill it in correctly
5. For all numbers of course

Example server response message to “HI\r\n\r\n”

S\r\n2\r\nUsername: \r\nPassword: \r\n\r\n

*4.4* The client will be built upon this structure so that it will loop for the number of items it needs to get from the user and just display in this case Username: to the user where it will read what the user types and accordingly format a message around what the user types.

Given the user entered their username as Matthew, and their password as yeet123 the message will be formatted as below:

*4.5* Client Message format:

Matthew\r\nyeet123\r\n\r\n

As you can see it will just need to send the data the server requested. The server will be able to decode this easily.

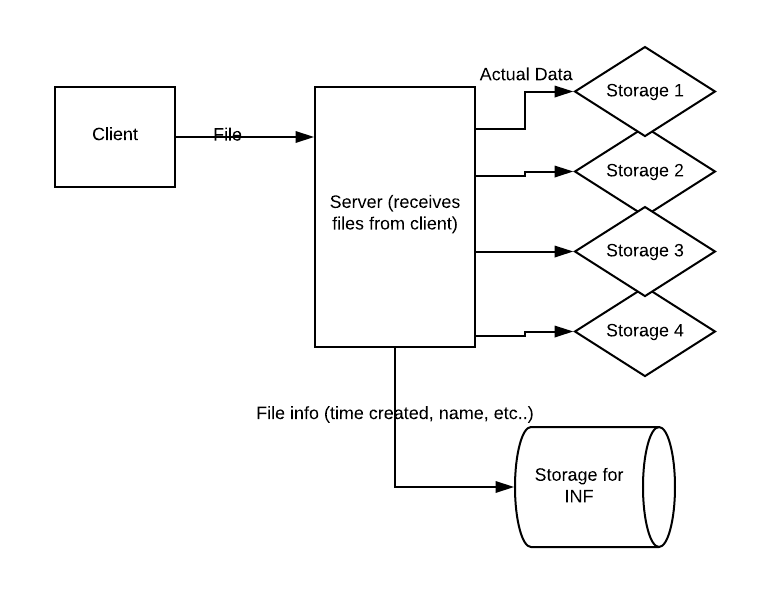
*4.6* Once the user successfully logs in then the client will follow the protocol specified in **2.0**

*4.7* The server will continue to accept the messages as in protocol **2.0** with the only difference being the responses that it gives will depend upon what the client sends now, meaning the server will follow the same response as in **4*.3***.

**5.0 Storage Server Design**

*5.1* To run all the file servers run:

bash fs\_start.sh



*5.2* The file servers currently are organized with

Storage 1 having A1, A2

Storage 2 having B1, B2

Storage 3 having O1, O2

Storage 4 having O3, O4

*5.3* To obtain O1, O2, O3, and O4 the following protocol happens:

O1 = A1 ⊕ B1

O2 = A2 ⊕ B2

O3 = A2 ⊕ B1

O4 = A1 ⊕ A2 ⊕ B2

*5.4* During the STR protocol the main server will pad the file with the following protocol:

Result = fileSize % 4

0: appends 4 b‘\x04’

1: appends 3 b‘\x03’

2: appends 2 b‘\x02’

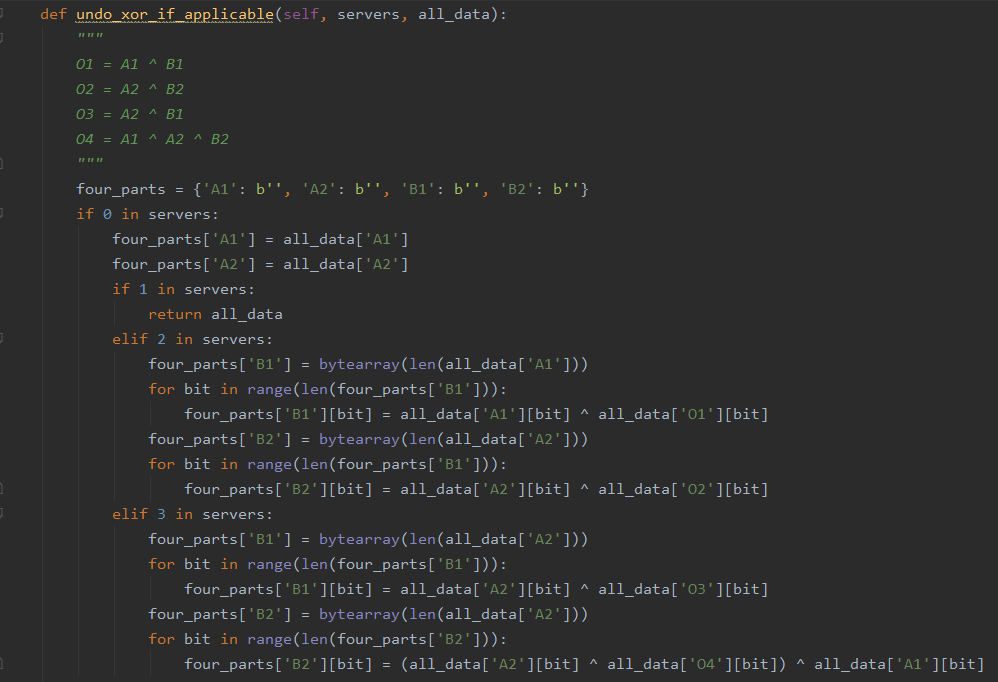
3: appends 1 b‘\x01’

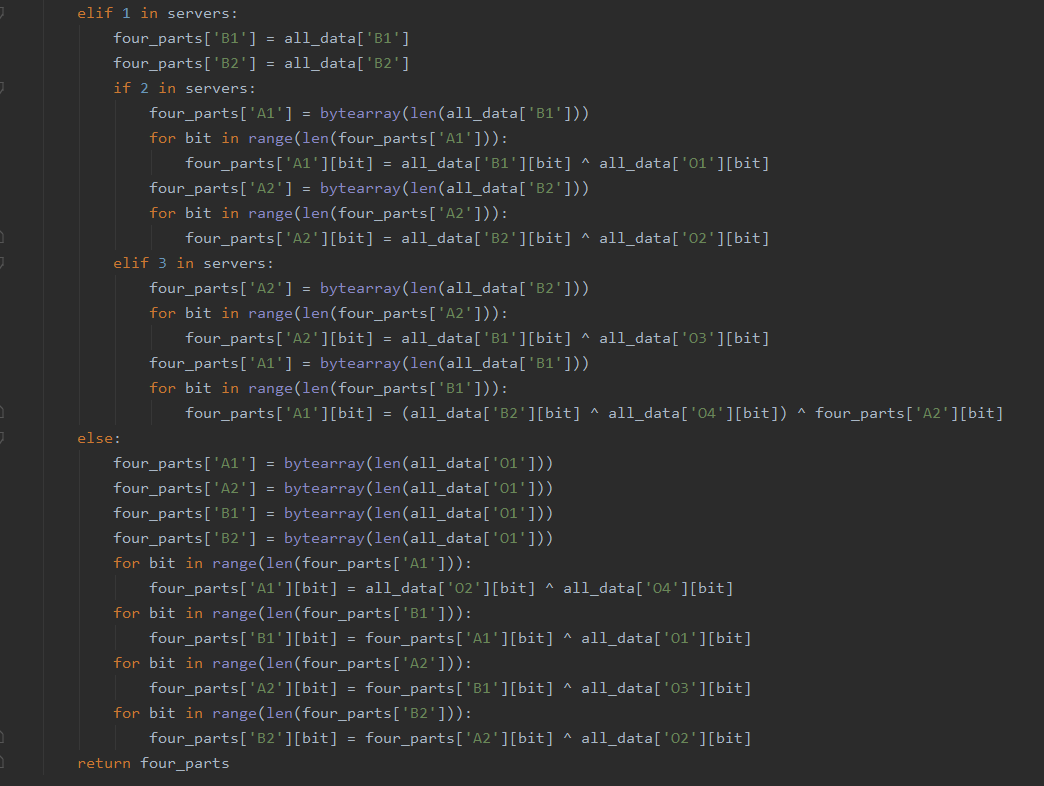
During the RTV protocol the main server will un-pad the file by reading the last byte and removing that many bytes. This happens after *5.7* finished.

*5.5* Once padded the main server will divide the file into four parts: A1, A2, B1, B2. O1, O2, O3, and O4 will be calculated as in *5.3*. It will send accordingly as stated in *5.2*.

*5.6* During the RTV protocol the main server randomly selects two of the servers and Retrieves back the two blocks of data that the servers stored. It will be able to rebuild the original file from these four blocks no matter the two servers.

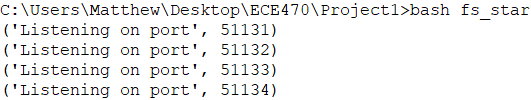
*5.7* I feel the code here would explain what happens best.



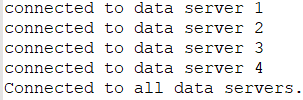


**6.0 Demonstration**

*6.1* File Servers start

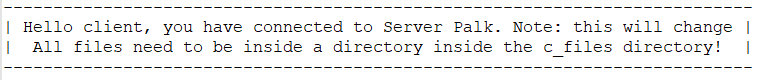


*6.2* Main server connecting to file servers

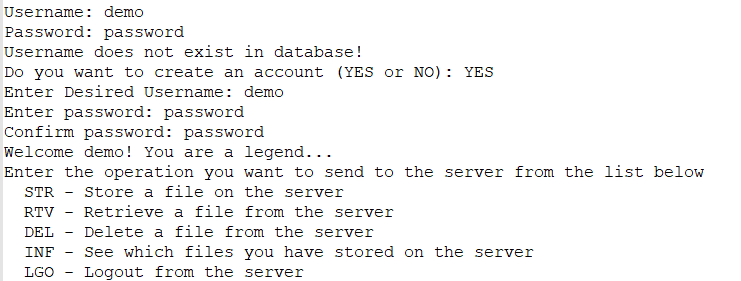


*6.3* Client connects to main server

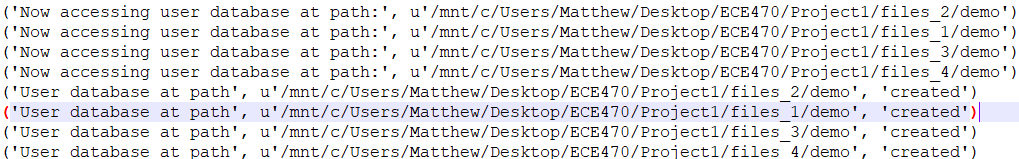




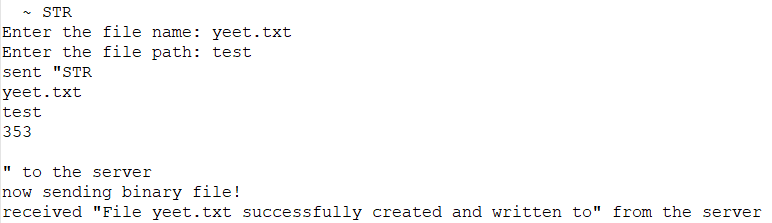
*6.4* Client Successful create an account / login, client – server – file servers

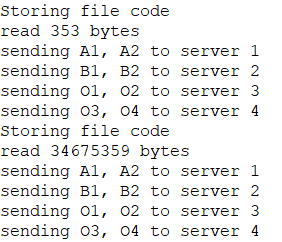


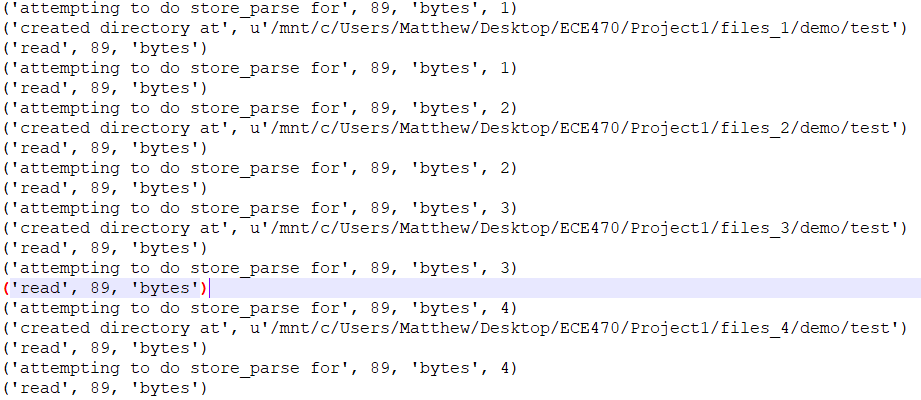


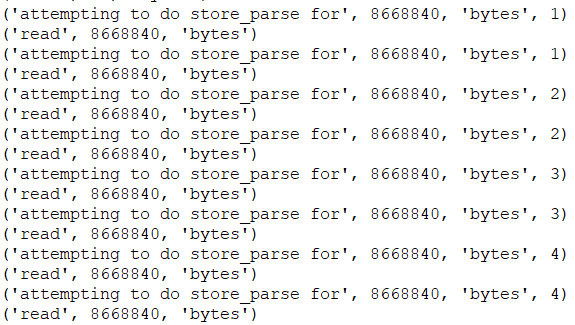


*6.5* Storing a file successfully, client – server – file servers

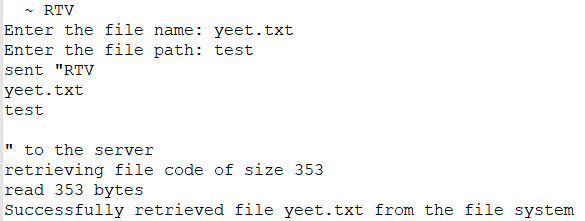


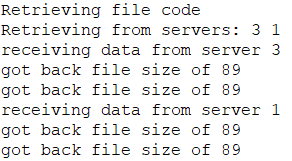


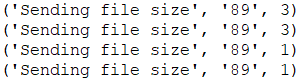




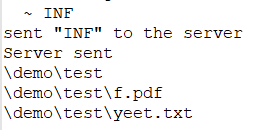
*6.6* Retrieving a file, client – server – file servers



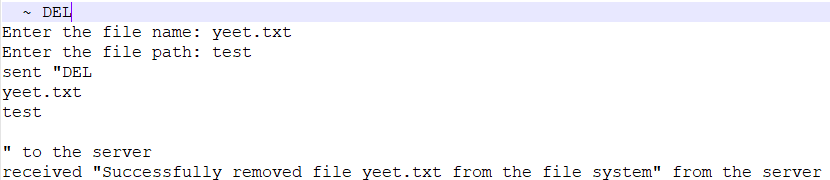


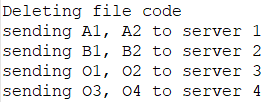


*6.7* Information request, client – server

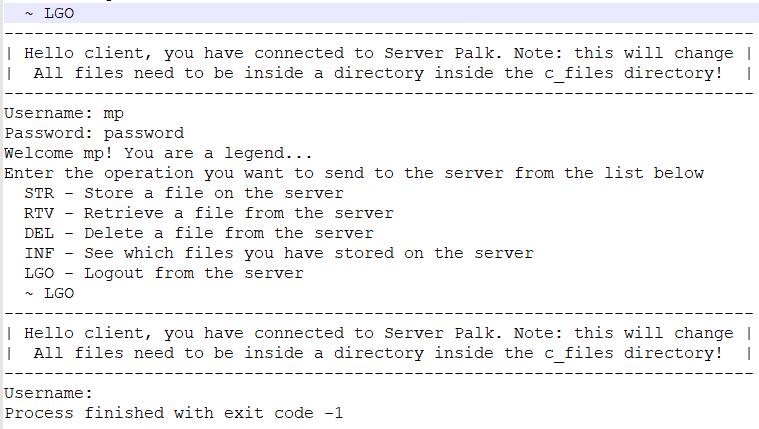


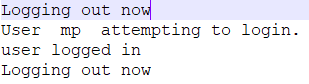
*6.8* Delete, client – server, file servers

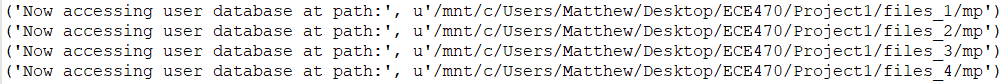




*6.9* Logout to new login, client – server, file servers (file servers is showing if a new login happens it works)







*6.10* Entire logs from the demo run can be found in Documentation directory with names:

client\_test\_demo\_output.txt

server\_main\_demo\_output.txt

file\_servers\_demo\_output.txt

**7.0** Code

*7.1* main\_server.py

import socket  
import datetime  
import time  
import os  
import glob  
import random  
  
VALID\_OPCODES = ['STR', 'RTV', 'DEL', 'INF', 'LGO']  
'''  
STR  
 Store a file  
 file name\r\nfile path\r\nfile size\r\n\r\n  
RTV  
 Retrieve a file  
 file name\r\nfile path\r\n\r\n  
 server sends file size ---> client receives that many bytes  
DEL  
 Delete a file  
 file name\r\nfile path\r\n\r\n  
INF  
 Get info of all files in database  
LGO  
 Logout of the server and go back to login screen  
'''  
  
rn = "\r\n"  
  
  
class InvalidOpCode(Exception):  
 pass  
  
  
class RecievedPartial(Exception):  
 pass  
  
  
class MyServer:  
 def \_\_init\_\_(self, ip='127.0.0.1', port=17777):  
 self.data\_server\_sockets = [socket.socket(), socket.socket(), socket.socket(), socket.socket()]  
 self.connect\_to\_data\_servs()  
 self.orig\_sock = socket.socket()  
 self.csoc = None  
 self.ip = ip  
 self.port = port  
 self.orig\_sock.bind((self.ip, self.port))  
 self.start()  
 self.connection\_time = datetime.datetime.utcnow()  
 self.username = None  
 self.password = None  
 # was commenting this out since if something goes wrong I dont want it to mess anything up...  
 self.user\_database = os.path.join(os.getcwd(), "user\_database")  
 if not os.path.exists(self.user\_database):  
 os.mkdir(self.user\_database)  
 # self.user\_database = os.path.join("C:\\Users\\Matthew\\Desktop\\ECE470\\Project1", "files")  
 self.current\_user\_database = None  
 print("Client connected to Server Palk @", self.connection\_time)  
 self.restart()  
  
 def connect\_to\_data\_servs(self):  
 list\_of\_dservs = [1, 2, 3, 4]  
 while True:  
 for dserv in list\_of\_dservs:  
 try:  
 self.data\_server\_sockets[dserv - 1].connect(('127.0.0.1', (51130 + dserv)))  
 except:  
 print("could not connect to data server", dserv)  
 continue  
 print("connected to data server", dserv)  
 list\_of\_dservs.remove(dserv)  
 break  
 if not list\_of\_dservs:  
 break  
 print("Connected to all data servers.")  
  
 def start(self):  
 self.orig\_sock.listen(5)  
 print("Listening on ", self.port)  
 commsoc, raddr = self.orig\_sock.accept()  
 self.csoc = commsoc  
  
 def restart(self):  
 login = self.login()  
 if not login:  
 return  
 self.update\_data\_servs()  
 self.reading\_commands()  
  
 def login(self):  
 not\_logged\_in = True  
 server\_user\_pass\_mess = "2" + rn + "Username: " + rn + "Password: " + rn  
 hello\_message = self.csoc.recv(1000).decode("utf-8")  
 if hello\_message != "HI\r\n\r\n":  
 return False  
 else:  
 self.csoc.sendall(("HI\r\n" + server\_user\_pass\_mess + rn).encode("utf-8"))  
 while not\_logged\_in:  
 # ask user for username and password  
 username\_password = self.csoc.recv(1000).decode("utf-8")  
 username\_password = username\_password.split("\r\n")  
 if len(username\_password) != 4 and username\_password[-2:] != ['', '']:  
 self.csoc.sendall(("E\r\nBadly Formatted Message\r\n"  
 + server\_user\_pass\_mess + rn).encode("utf-8"))  
 continue  
 self.username = username\_password[0]  
 password = username\_password[1]  
 if self.username not in os.listdir(self.user\_database):  
 self.csoc.sendall(("E\r\nUsername does not exist in database!\r\n"  
 + "1\r\nDo you want to create an account (YES or NO): \r\n" + rn).encode("utf-8"))  
 response = self.csoc.recv(1000).decode("utf-8").split('\r\n')  
 if response[0] == "YES":  
 self.csoc.sendall("L\r\n3\r\nEnter Desired Username: \r\nEnter password:"   
 " \r\nConfirm password: \r\n\r\n".encode("utf-8"))  
 response = self.csoc.recv(1000).decode("utf-8").split("\r\n")  
 self.current\_user\_database = os.path.join(self.user\_database, response[0])  
 while os.path.exists(self.current\_user\_database):  
 self.csoc.sendall("E\r\n"  
 "Username already exists in database!"  
 "\r\n"  
 "3"  
 "\r\n"  
 "Enter Desired Username: "  
 "\r\n"  
 "Enter password:"  
 " \r\n"  
 "Confirm password: "  
 "\r\n\r\n".encode("utf-8"))  
 response = self.csoc.recv(1000).decode("utf-8").split("\r\n")  
 self.current\_user\_database = os.path.join(self.user\_database, response[0])  
 os.mkdir(self.current\_user\_database)  
 print(self.current\_user\_database)  
 pass1 = response[1]  
 pass2 = response[2]  
 while pass1 != pass2:  
 self.csoc.sendall("E\r\n"  
 "Passwords did not match..."  
 "\r\n"  
 "2"  
 "\r\n"  
 "Enter password:"   
 " \r\n"  
 "Confirm password: "  
 "\r\n\r\n".encode("utf-8"))  
 response = self.csoc.recv(1000).decode("utf-8").split("\r\n")  
 pass1 = response[0]  
 pass2 = response[1]  
 password = response[1]  
 password\_file = open(os.path.join(self.current\_user\_database, "password.txt"), "w+")  
 password\_file.write(password)  
 password\_file.close()  
 else:  
 self.csoc.sendall(("HI\r\n" + server\_user\_pass\_mess + rn).encode("utf-8"))  
 continue  
 print("User ", self.username, " attempting to login.")  
 self.current\_user\_database = os.path.join(self.user\_database, self.username)  
 if self.check\_password(password):  
 print("user logged in")  
 not\_logged\_in = False  
 self.csoc.sendall(("S\r\nWelcome " + self.username + "! You are a legend...\r\n").encode("utf-8"))  
 else:  
 incorrect\_pass = "E\r\nPassword did not match username " + self.username + rn \  
 + server\_user\_pass\_mess + rn  
 self.csoc.sendall(incorrect\_pass.encode("utf-8"))  
 return True  
  
 def update\_data\_servs(self, logout=False):  
 if logout:  
 for dserv in self.data\_server\_sockets:  
 dserv.send("LOGOUT".encode("utf-8"))  
 else:  
 for dserv in self.data\_server\_sockets:  
 dserv.send(self.username.encode("utf-8"))  
  
 def reading\_commands(self):  
 while True:  
 try:  
 data = self.csoc.recv(1000)  
 except ConnectionResetError:  
 break  
 message\_back = self.manipulate\_data(data.decode("utf-8"), data)  
 time.sleep(0.5)  
 self.csoc.sendall(message\_back.encode("utf-8"))  
 print("Client Ended Connection @", datetime.datetime.utcnow())  
  
 def manipulate\_data(self, data\_recv, raw\_data):  
 self.opcode = data\_recv[0:3]  
 if self.opcode not in VALID\_OPCODES:  
 return "ERROR Opcode was invalid!"  
 entire\_data = self.apply\_opcode\_to\_parse(data\_recv)  
 return entire\_data  
  
 def store\_parse(self, data):  
 file\_name = data[0]  
 file\_path = data[1]  
 file\_size = int(data[2])  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 if not os.path.exists(actual\_file\_directory):  
 os.mkdir(actual\_file\_directory)  
 try:  
 f = open('temp.temp', "wb+")  
 p = open(actual\_file\_path, "wb+")  
 # this is just for the INF  
 p.close()  
 cur\_file\_size\_read = 0  
 temp = file\_size / 1000  
 extra = file\_size % 1000  
 while cur\_file\_size\_read < file\_size:  
 if not temp:  
 file\_data = self.csoc.recv(extra)  
 else:  
 file\_data = self.csoc.recv(1000)  
 f.write(file\_data)  
 cur\_file\_size\_read += len(file\_data)  
 temp -= 1  
 print("read", cur\_file\_size\_read, "bytes")  
 f.close()  
 f = open('temp.temp', 'rb')  
 temp\_byte\_array = f.read()  
 f.close()  
 os.remove('temp.temp')  
 div\_by\_4 = cur\_file\_size\_read % 4  
 if div\_by\_4 == 0:  
 padding = b'\x04\x04\x04\x04'  
 file\_size += 4  
 elif div\_by\_4 == 3:  
 padding = b'\x01'  
 file\_size += 1  
 elif div\_by\_4 == 2:  
 padding = b'\x02\x02'  
 file\_size += 2  
 else:  
 padding = b'\x03\x03\x03'  
 file\_size += 3  
 temp\_byte\_array += padding  
 file\_name\_endings = ['A1', 'A2', 'B1', 'B2', 'O1', 'O2', 'O3', 'O4']  
 four\_divisions = {'A1': b'', 'A2': b'', 'B1': b'', 'B2': b'',  
 'O1': b'', 'O2': b'', 'O3': b'', 'O4': b'',  
 '\_A1': '', '\_A2': '', '\_B1': '', '\_B2': '',  
 '\_O1': '', '\_O2': '', '\_O3': '', '\_O4': ''}  
 for i, \_end in enumerate(file\_name\_endings):  
 file\_name\_div = file\_name.split('.')  
 temp\_file\_name = file\_name\_div[0] + '\_' + \_end  
 for ext in file\_name\_div[1:]:  
 temp\_file\_name = temp\_file\_name + '.' + ext  
 four\_divisions[('\_' + \_end)] += temp\_file\_name  
 if i < 4:  
 begin = int(i \* (file\_size / 4))  
 end = int((i + 1) \* (file\_size / 4))  
 four\_divisions[\_end] += temp\_byte\_array[begin:end]  
 else:  
 four\_divisions[\_end] = bytearray(len(four\_divisions['A1']))  
 for bit in range(len(four\_divisions[\_end])):  
 if \_end == 'O1':  
 four\_divisions[\_end][bit] = four\_divisions['A1'][bit] ^ four\_divisions['B1'][bit]  
 elif \_end == 'O2':  
 four\_divisions[\_end][bit] = four\_divisions['A2'][bit] ^ four\_divisions['B2'][bit]  
 elif \_end == 'O3':  
 four\_divisions[\_end][bit] = four\_divisions['A2'][bit] ^ four\_divisions['B1'][bit]  
 elif \_end == 'O4':  
 four\_divisions[\_end][bit] = (four\_divisions['A1'][bit] ^ four\_divisions['A2'][bit]) \  
 ^ four\_divisions['B2'][bit]  
 for i, data\_server in enumerate(self.data\_server\_sockets):  
 if i == 0:  
 print("sending A1, A2 to server 1")  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_A1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['A1'])  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_A2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['A2'])  
 pause\_recv = data\_server.recv(1)  
 elif i == 1:  
 print("sending B1, B2 to server 2")  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_B1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['B1'])  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_B2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['B2'])  
 pause\_recv = data\_server.recv(1)  
 elif i == 2:  
 print("sending O1, O2 to server 3")  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_O1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['O1'])  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_O2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['O2'])  
 pause\_recv = data\_server.recv(1)  
 elif i == 3:  
 print("sending O3, O4 to server 4")  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_O3'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['O3'])  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + str(int(file\_size / 4)) + rn + four\_divisions['\_O4'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 data\_server.sendall(four\_divisions['O4'])  
 pause\_recv = data\_server.recv(1)  
 return "File " + file\_name + " successfully created and written to"  
 except Exception as e:  
 print(e)  
 return "E Something went wrong during file process"  
  
 def retrieve\_parse(self, data):  
 file\_name = data[0]  
 file\_path = data[1]  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 if not os.path.exists(actual\_file\_directory):  
 return "ERROR the file path you gave did not exist in the file system!"  
 if not os.path.exists(actual\_file\_path):  
 return "ERROR the file name you gave did not exist in the file system!"  
 try:  
 file\_name\_endings = ['A1', 'A2', 'B1', 'B2', 'O1', 'O2', 'O3', 'O4']  
 four\_divisions = {'A1': '', 'A2': '', 'B1': '', 'B2': '',  
 'O1': '', 'O2': '', 'O3': '', 'O4': ''}  
 for i, \_end in enumerate(file\_name\_endings):  
 file\_name\_div = file\_name.split('.')  
 temp\_file\_name = file\_name\_div[0] + '\_' + \_end  
 for ext in file\_name\_div[1:]:  
 temp\_file\_name = temp\_file\_name + '.' + ext  
 four\_divisions[\_end] += temp\_file\_name  
 random\_server = [0, 1, 2, 3]  
 chosen\_servers = random.sample(set(random\_server), 2)  
 print("Retrieving from servers:", chosen\_servers[0]+1, chosen\_servers[1]+1)  
 file\_data = {'A1': b'', 'A2': b'', 'B1': b'', 'B2': b'',  
 'O1': b'', 'O2': b'', 'O3': b'', 'O4': b''}  
 for fs in chosen\_servers:  
 if fs == 0:  
 ends = ['A1', 'A2']  
 elif fs == 1:  
 ends = ['B1', 'B2']  
 elif fs == 2:  
 ends = ['O1', 'O2']  
 else:  
 ends = ['O3', 'O4']  
 print("receiving data from server", fs + 1)  
 for end in ends:  
 cur\_file\_size\_read = 0  
 first\_mess = self.opcode + rn + four\_divisions[end] \  
 + rn + file\_path + rn + rn  
 self.data\_server\_sockets[fs].sendall(first\_mess.encode('utf-8'))  
 file\_size = int(self.data\_server\_sockets[fs].recv(1000).decode('utf-8').split('\r\n')[0])  
 print("got back file size of", file\_size)  
 temp = file\_size / 1000  
 extra = file\_size % 1000  
 while cur\_file\_size\_read < file\_size:  
 if not temp:  
 data = self.data\_server\_sockets[fs].recv(extra)  
 else:  
 data = self.data\_server\_sockets[fs].recv(1000)  
 file\_data[end] += data  
 cur\_file\_size\_read += len(data)  
 temp -= 1  
 file\_data = self.undo\_xor\_if\_applicable(chosen\_servers, file\_data)  
 \_padding = file\_data['B2'][-1]  
 \_padding = \_padding \* -1  
 file\_data['B2'] = file\_data['B2'][:\_padding]  
 entire\_file = file\_data['A1'] + file\_data['A2'] + file\_data['B1'] + file\_data['B2']  
 self.csoc.sendall((str(len(entire\_file)) + rn).encode("utf-8"))  
 self.csoc.sendall(entire\_file)  
 return "Successfully retrieved file " + file\_name + " from the file system"  
 except Exception as e:  
 print(e)  
 return "E Something went wrong during file process"  
  
 def delete\_parse(self, data):  
 file\_name = data[0]  
 file\_path = data[1]  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 if not os.path.exists(actual\_file\_directory) or not os.path.exists(actual\_file\_path):  
 return "ERROR The file path or file name you gave did not exist in the file system!"  
 try:  
 os.remove(actual\_file\_path)  
 file\_name\_endings = ['A1', 'A2', 'B1', 'B2', 'O1', 'O2', 'O3', 'O4']  
 four\_divisions = {'\_A1': '', '\_A2': '', '\_B1': '', '\_B2': '',  
 '\_O1': '', '\_O2': '', '\_O3': '', '\_O4': ''}  
 for i, \_end in enumerate(file\_name\_endings):  
 file\_name\_div = file\_name.split('.')  
 temp\_file\_name = file\_name\_div[0] + '\_' + \_end  
 for ext in file\_name\_div[1:]:  
 temp\_file\_name = temp\_file\_name + '.' + ext  
 four\_divisions[('\_' + \_end)] += temp\_file\_name  
 for i, data\_server in enumerate(self.data\_server\_sockets):  
 if i == 0:  
 print("sending A1, A2 to server 1")  
 first\_mess = self.opcode + rn + four\_divisions['\_A1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + four\_divisions['\_A2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 elif i == 1:  
 print("sending B1, B2 to server 2")  
 first\_mess = self.opcode + rn + four\_divisions['\_B1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + four\_divisions['\_B2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 elif i == 2:  
 print("sending O1, O2 to server 3")  
 first\_mess = self.opcode + rn + four\_divisions['\_O1'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + four\_divisions['\_O2'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 elif i == 3:  
 print("sending O3, O4 to server 4")  
 first\_mess = self.opcode + rn + four\_divisions['\_O3'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 first\_mess = self.opcode + rn + four\_divisions['\_O4'] \  
 + rn + file\_path + rn + rn  
 data\_server.sendall(first\_mess.encode('utf-8'))  
 pause\_recv = data\_server.recv(1)  
 return "Successfully removed file " + file\_name + " from the file system"  
 except:  
 return "E Something went wrong during file process"  
  
 def info\_parse(self):  
 all\_files = glob.glob(os.path.join(self.current\_user\_database, "\*\*", "\*"), recursive=True)  
 if len(all\_files) == 1:  
 return "No files in database"  
 file\_info = ""  
 for f in all\_files:  
 if "password.txt" in f:  
 continue  
 path = f.split(self.user\_database)[1]  
 file\_info = file\_info + path + "\n"  
 return file\_info  
  
 def apply\_opcode\_to\_parse(self, data):  
 if self.opcode == 'STR':  
 print("Storing file code")  
 entire\_data = data.split('\r\n')[1:]  
 if len(entire\_data) != 5 or entire\_data[-2:] != ['', '']:  
 return "ERROR Did not receive all the parameters required for opcode:" + self.opcode  
 return self.store\_parse(entire\_data)  
 elif self.opcode == 'RTV':  
 print("Retrieving file code")  
 entire\_data = data.split('\r\n')[1:]  
 if len(entire\_data) != 4 or entire\_data[-2:] != ['', '']:  
 return "ERROR Did not receive all the data!"  
 return self.retrieve\_parse(entire\_data)  
 elif self.opcode == 'DEL':  
 print("Deleting file code")  
 entire\_data = data.split('\r\n')[1:]  
 if len(entire\_data) != 4 or entire\_data[-2:] != ['', '']:  
 return "ERROR Did not receive all the data!"  
 return self.delete\_parse(entire\_data)  
 elif self.opcode == 'INF':  
 return self.info\_parse()  
 elif self.opcode == 'LGO':  
 print("Logging out now")  
 self.update\_data\_servs(logout=True)  
 self.restart()  
  
 def undo\_xor\_if\_applicable(self, servers, all\_data):  
 *"""  
 O1 = A1 ^ B1  
 O2 = A2 ^ B2  
 O3 = A2 ^ B1  
 O4 = A1 ^ A2 ^ B2  
 """* four\_parts = {'A1': b'', 'A2': b'', 'B1': b'', 'B2': b''}  
 if 0 in servers:  
 four\_parts['A1'] = all\_data['A1']  
 four\_parts['A2'] = all\_data['A2']  
 if 1 in servers:  
 return all\_data  
 elif 2 in servers:  
 four\_parts['B1'] = bytearray(len(all\_data['A1']))  
 for bit in range(len(four\_parts['B1'])):  
 four\_parts['B1'][bit] = all\_data['A1'][bit] ^ all\_data['O1'][bit]  
 four\_parts['B2'] = bytearray(len(all\_data['A2']))  
 for bit in range(len(four\_parts['B1'])):  
 four\_parts['B2'][bit] = all\_data['A2'][bit] ^ all\_data['O2'][bit]  
 elif 3 in servers:  
 four\_parts['B1'] = bytearray(len(all\_data['A2']))  
 for bit in range(len(four\_parts['B1'])):  
 four\_parts['B1'][bit] = all\_data['A2'][bit] ^ all\_data['O3'][bit]  
 four\_parts['B2'] = bytearray(len(all\_data['A2']))  
 for bit in range(len(four\_parts['B2'])):  
 four\_parts['B2'][bit] = (all\_data['A2'][bit] ^ all\_data['O4'][bit]) ^ all\_data['A1'][bit]  
 elif 1 in servers:  
 four\_parts['B1'] = all\_data['B1']  
 four\_parts['B2'] = all\_data['B2']  
 if 2 in servers:  
 four\_parts['A1'] = bytearray(len(all\_data['B1']))  
 for bit in range(len(four\_parts['A1'])):  
 four\_parts['A1'][bit] = all\_data['B1'][bit] ^ all\_data['O1'][bit]  
 four\_parts['A2'] = bytearray(len(all\_data['B2']))  
 for bit in range(len(four\_parts['A2'])):  
 four\_parts['A2'][bit] = all\_data['B2'][bit] ^ all\_data['O2'][bit]  
 elif 3 in servers:  
 four\_parts['A2'] = bytearray(len(all\_data['B2']))  
 for bit in range(len(four\_parts['A2'])):  
 four\_parts['A2'][bit] = all\_data['B1'][bit] ^ all\_data['O3'][bit]  
 four\_parts['A1'] = bytearray(len(all\_data['B1']))  
 for bit in range(len(four\_parts['B1'])):  
 four\_parts['A1'][bit] = (all\_data['B2'][bit] ^ all\_data['O4'][bit]) ^ four\_parts['A2'][bit]  
 else:  
 four\_parts['A1'] = bytearray(len(all\_data['O1']))  
 four\_parts['A2'] = bytearray(len(all\_data['O1']))  
 four\_parts['B1'] = bytearray(len(all\_data['O1']))  
 four\_parts['B2'] = bytearray(len(all\_data['O1']))  
 for bit in range(len(four\_parts['A1'])):  
 four\_parts['A1'][bit] = all\_data['O2'][bit] ^ all\_data['O4'][bit]  
 for bit in range(len(four\_parts['B1'])):  
 four\_parts['B1'][bit] = four\_parts['A1'][bit] ^ all\_data['O1'][bit]  
 for bit in range(len(four\_parts['A2'])):  
 four\_parts['A2'][bit] = four\_parts['B1'][bit] ^ all\_data['O3'][bit]  
 for bit in range(len(four\_parts['B2'])):  
 four\_parts['B2'][bit] = four\_parts['A2'][bit] ^ all\_data['O2'][bit]  
 return four\_parts  
  
 def check\_password(self, password):  
 try:  
 pass\_file = open(os.path.join(self.current\_user\_database, "password.txt"), "r")  
 except:  
 return False  
 correct\_password = pass\_file.read()  
 if password != correct\_password:  
 return False  
 else:  
 return True  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 da\_server = MyServer()  
 # # wait for incoming connections  
 # while True:  
 # print("Listening on ", port)  
 #  
 # commsoc, raddr = serversoc.accept()  
 #  
 # MyServer(commsoc)  
 #  
 # commsoc.close()  
 #  
 # # close the server socket  
 # serversoc.close()

*7.2* client\_test.py

import socket  
import os  
  
rn = "\r\n"  
  
  
class SizeOfFileError(Exception):  
 pass  
  
  
def do\_client\_stuff(csoc):  
 not\_done = True  
 local\_filesys\_path = os.path.join(os.getcwd(), "c\_files")  
 if not os.path.exists(local\_filesys\_path):  
 os.mkdir(local\_filesys\_path)  
 print("---------------------------------------------------------------------------\n"  
 "| Hello client, you have connected to Server Palk. Note: this will change |\n"  
 "| All files need to be inside a directory inside the c\_files directory! |\n"  
 "---------------------------------------------------------------------------")  
 not\_logged\_in = True  
 csoc.sendall("HI\r\n\r\n".encode("utf-8"))  
 while not\_logged\_in:  
 server\_message = csoc.recv(1000).decode("utf-8").split("\r\n")  
 code = server\_message[0]  
 code\_flag = 0  
 if code == 'E':  
 code\_flag = 1  
 error\_message = server\_message[1]  
 print(error\_message)  
 elif code == 'S':  
 print(server\_message[1])  
 break  
 number\_of\_inputs = int(server\_message[1 + code\_flag])  
 final\_mess = ""  
 for num\_i in range(0, number\_of\_inputs):  
 \_input = input(server\_message[num\_i + 2 + code\_flag])  
 final\_mess += \_input + rn  
 final\_mess += rn  
 csoc.sendall(final\_mess.encode("utf-8"))  
 while not\_done:  
 file\_size = 0  
 sending\_file = False  
 retrieving\_file = False  
 mess = input("Enter the operation you want to send to the server from the list below\n"  
 " STR - Store a file on the server\n"  
 " RTV - Retrieve a file from the server\n"  
 " DEL - Delete a file from the server\n"  
 " INF - See which files you have stored on the server\n"  
 " LGO - Logout from the server\n"  
 " ~ ")  
 if mess == "STR":  
 sending\_file = True  
 if mess == "RTV":  
 retrieving\_file = True  
 if mess == "INF":  
 inf\_mess = "INF\r\n\r\n"  
 csoc.sendall(inf\_mess.encode("utf-8"))  
 print("sent \"" + mess + "\" to the server")  
 data = csoc.recv(1000)  
 print("Server sent\n" + data.decode("utf-8"))  
 continue  
 if mess == "LGO":  
 csoc.sendall("LGO\r\n\r\n".encode("utf-8"))  
 return 'logout'  
 file\_name = input("Enter the file name: ")  
 file\_path = input("Enter the file path: ")  
 file\_dircetory\_path = os.path.join(local\_filesys\_path, file\_path)  
 full\_path = os.path.join(local\_filesys\_path, file\_path, file\_name)  
 if sending\_file:  
 if not os.path.exists(full\_path):  
 print("file path was invalid!")  
 continue  
 file\_size = os.path.getsize(full\_path)  
 whole\_mess = mess + rn + file\_name + rn + file\_path + rn + str(file\_size) + rn + rn  
 else:  
 whole\_mess = mess + rn + file\_name + rn + file\_path + rn + rn  
 csoc.sendall(whole\_mess.encode("utf-8"))  
 print("sent \"" + whole\_mess + "\" to the server")  
 if sending\_file:  
 print("now sending binary file!")  
 f = open(full\_path, "rb")  
 file\_data = f.read()  
 if len(file\_data) != file\_size:  
 raise SizeOfFileError  
 csoc.sendall(file\_data)  
 f.close()  
 data = csoc.recv(1000)  
 print("received \"" + data.decode("utf-8") + "\" from the server")  
 elif retrieving\_file:  
 data = csoc.recv(1000)  
 file\_size = data.decode("utf-8").split("\r\n")[0]  
 if file\_size[0] == 'E':  
 print("got back:\n", file\_size)  
 continue  
 else:  
 file\_size = int(file\_size)  
 print("retrieving file code of size", file\_size)  
 cur\_file\_size\_read = 0  
 if not os.path.exists(file\_dircetory\_path):  
 os.mkdir(file\_dircetory\_path)  
 f = open(full\_path, "wb+")  
 temp = file\_size / 1000  
 extra = file\_size % 1000  
 while cur\_file\_size\_read < file\_size:  
 if not temp:  
 file\_data = csoc.recv(extra)  
 else:  
 file\_data = csoc.recv(1000)  
 f.write(file\_data)  
 cur\_file\_size\_read += len(file\_data)  
 temp -= 1  
 print("read", cur\_file\_size\_read, "bytes")  
 f.close()  
 print(csoc.recv(1000).decode('utf-8'))  
 else:  
 data = csoc.recv(1000)  
 print("received \"" + data.decode("utf-8") + "\" from the server")  
 print("Ended baseTCPProtocol")  
 return 'EXIT'  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # create the socket  
 # defaults family=AF\_INET, type=SOCK\_STREAM, proto=0, filno=None  
 return\_code = ''  
 commsoc = socket.socket()  
 # port = int(input("Enter the port your server is on: "))  
 # connect to localhost:5000  
 port = 17777  
 commsoc.connect(("localhost", port))  
  
 while return\_code != 'EXIT':  
 return\_code = do\_client\_stuff(commsoc)  
  
 # close the comm socket  
 commsoc.close()

*7.3* file\_server1.py

import socket  
import os  
  
FILE\_SERVER\_NUMBER = 1  
rn = '\r\n'  
  
  
class FileServer:  
 def \_\_init\_\_(self, ip='127.0.0.1', port=51130):  
 self.orig\_sock = socket.socket()  
 self.ip = ip  
 self.port = port + FILE\_SERVER\_NUMBER  
 self.orig\_sock.bind((self.ip, self.port))  
 self.ssoc = None  
 self.not\_connected = True  
 self.file\_storage\_path = os.path.join(os.getcwd(), "files\_" + str(FILE\_SERVER\_NUMBER))  
 if not os.path.exists(self.file\_storage\_path):  
 os.mkdir(self.file\_storage\_path)  
 self.current\_user\_database = None  
 self.return\_code = None  
 self.start()  
  
 def start(self):  
 self.orig\_sock.listen(5)  
 while self.return\_code != 'SHUTDOWN':  
 print("Listening on port", self.port)  
 self.ssoc, raddr = self.orig\_sock.accept()  
 self.get\_ready()  
  
 def restart(self):  
 self.orig\_sock.close()  
 self.not\_connected = True  
 return 'RESTART'  
  
 def get\_ready(self):  
 self.current\_user\_database = os.path.join(self.file\_storage\_path,  
 self.ssoc.recv(1000).decode("utf-8"))  
 print("Now accessing user database at path:", self.current\_user\_database)  
 if not os.path.exists(self.current\_user\_database):  
 os.mkdir(self.current\_user\_database)  
 print("User database at path", self.current\_user\_database, "created")  
 self.ready\_to\_recv()  
  
 def ready\_to\_recv(self):  
 while True:  
 data = self.ssoc.recv(1000).decode("utf-8")  
 data = data.split("\r\n")  
 if data[0] == 'LOGOUT':  
 self.get\_ready()  
 elif data[0] == 'STR':  
 self.ssoc.sendall(b'\x00')  
 self.store\_parse(data[1:])  
 self.ssoc.sendall(b'\x00')  
 elif data[0] == 'RTV':  
 self.retrieve\_parse(data[1:])  
 elif data[0] == 'DEL':  
 self.delete\_parse(data[1:])  
 self.ssoc.sendall(b'\x00')  
  
 def store\_parse(self, data):  
 file\_name = data[1]  
 file\_path = data[2]  
 file\_size = int(data[0])  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 print("attempting to do store\_parse for", file\_size, "bytes", FILE\_SERVER\_NUMBER)  
 if not os.path.exists(actual\_file\_directory):  
 os.mkdir(actual\_file\_directory)  
 print("created directory at", actual\_file\_directory)  
 try:  
 f = open(actual\_file\_path, "wb+")  
 cur\_file\_size\_read = 0  
 temp = file\_size / 1000  
 extra = file\_size % 1000  
 while cur\_file\_size\_read < file\_size:  
 if not temp:  
 file\_data = self.ssoc.recv(extra)  
 else:  
 file\_data = self.ssoc.recv(1000)  
 f.write(file\_data)  
 cur\_file\_size\_read += len(file\_data)  
 temp -= 1  
 print("read", cur\_file\_size\_read, "bytes")  
 f.close()  
 return "File " + file\_name + " successfully created and written to in directory " + actual\_file\_path  
 except:  
 return "Something went wrong during file process"  
  
 def retrieve\_parse(self, data):  
 file\_name = data[0]  
 file\_path = data[1]  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 if not os.path.exists(actual\_file\_directory):  
 return "ERROR the file path you gave did not exist in the file system!"  
 if not os.path.exists(actual\_file\_path):  
 return "ERROR the file name you gave did not exist in the file system!"  
 self.ssoc.sendall((str(os.path.getsize(actual\_file\_path)) + rn).encode("utf-8"))  
 print("Sending file size", str(os.path.getsize(actual\_file\_path)), FILE\_SERVER\_NUMBER)  
 try:  
 f = open(actual\_file\_path, "rb")  
 file\_contents = f.read()  
 self.ssoc.sendall(file\_contents)  
 f.close()  
 return "Successfully sent full file"  
 except:  
 return "Something went wrong during file process"  
  
 def delete\_parse(self, data):  
 file\_name = data[0]  
 file\_path = data[1]  
 actual\_file\_directory = os.path.join(self.current\_user\_database, file\_path)  
 actual\_file\_path = os.path.join(actual\_file\_directory, file\_name)  
 if not os.path.exists(actual\_file\_directory) or not os.path.exists(actual\_file\_path):  
 return "ERROR The file path or file name you gave did not exist in the file system!"  
 try:  
 os.remove(actual\_file\_path)  
 return "Successfully removed file " + file\_name + " from the file system"  
 except:  
 return "Something went wrong during file process"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 fserv = FileServer()

*7.4* All other file servers are exactly the same as file\_server1.py with the exception that at the top of the code the variable FILE\_SERVER\_NUMBER on line 4 changes to the file server number it is.

file\_server2.py



file\_server3.py



file\_server4.py

