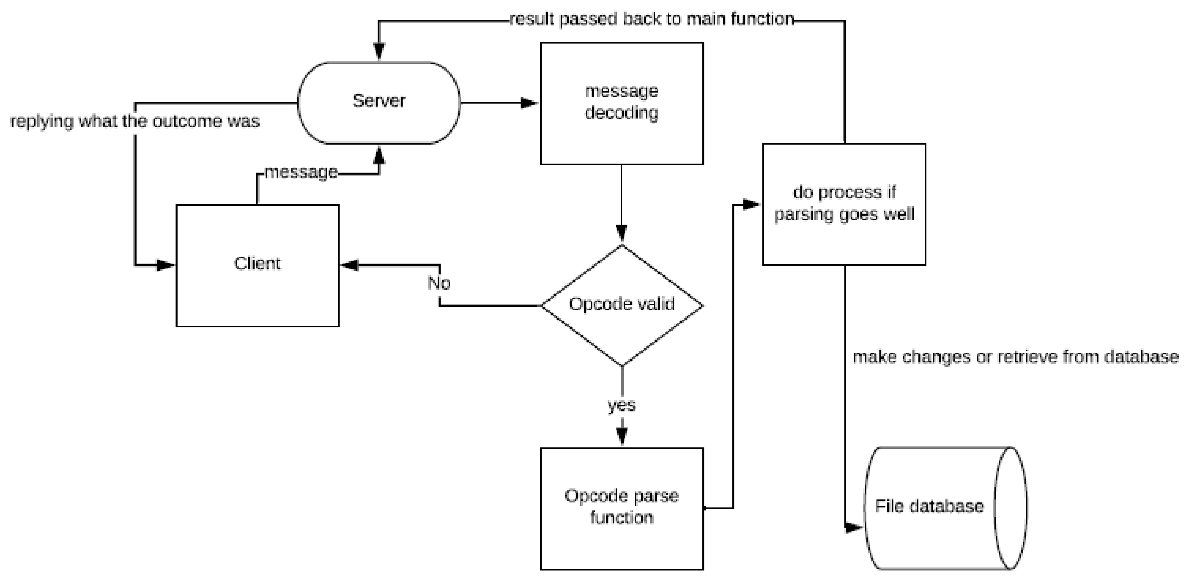
Matthew Palkimas
ECE 470

1.0 Server Design:

1.1 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.

1.2 The main server architecture will be spread out through this document.
1.3 Overall Design after login was successful


result passed back to main function
Server
message decoding
replying what the outcome was
message
do process if parsing goes well
Client
No
Opcode valid
make changes or retrieve from database
yes
Opcode parse function
File database

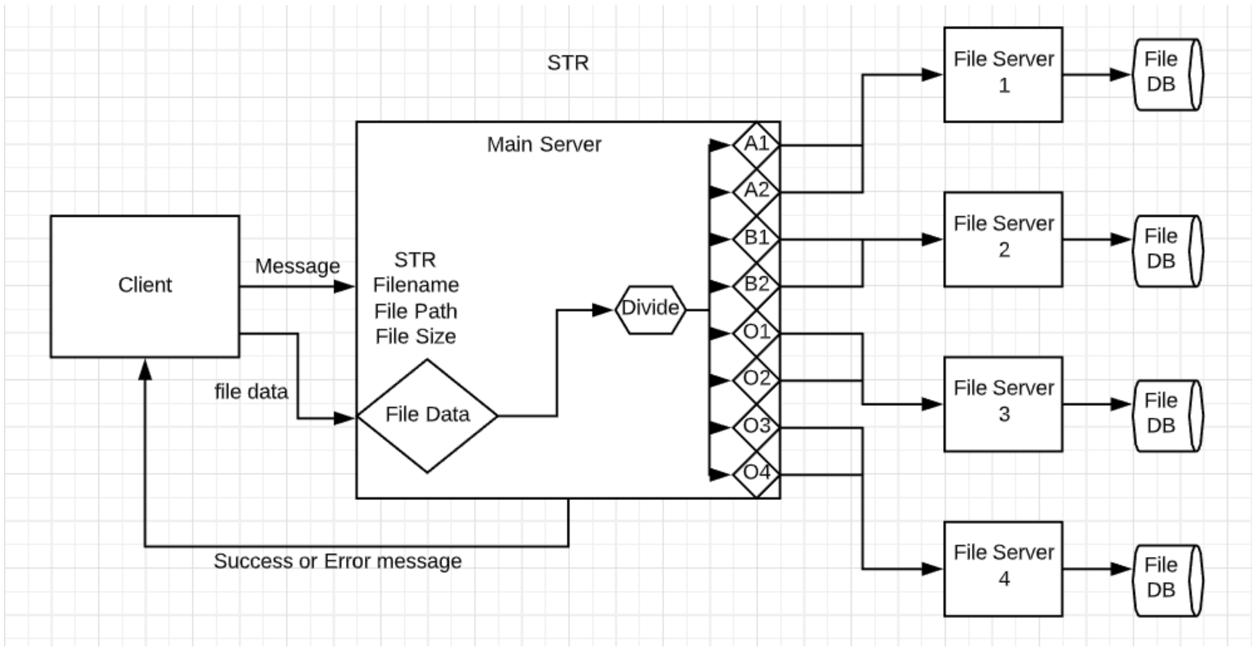
2.0 Message Design:

2.1 First three characters: Opcode
Current valid opcodes: STR, RTV, DEL, INF, LGO
In order store, retrieve, delete, information, logout

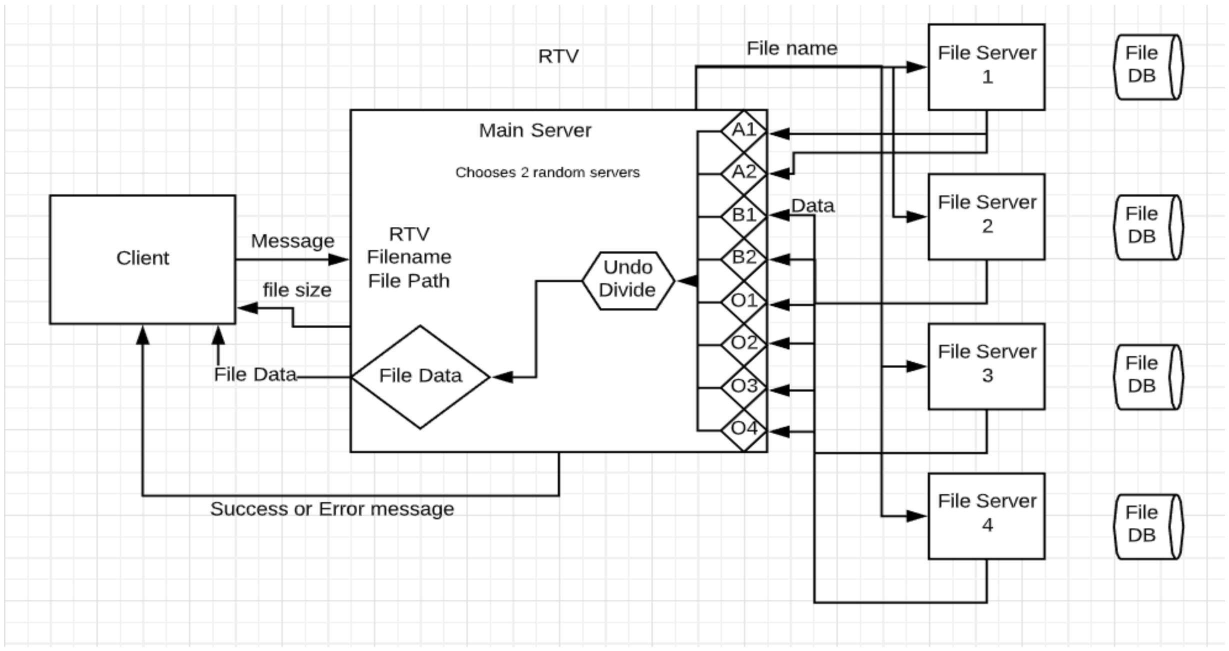
2.2 Depending on the opcode the rest of the message will need to be encoded in a certain manner.

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

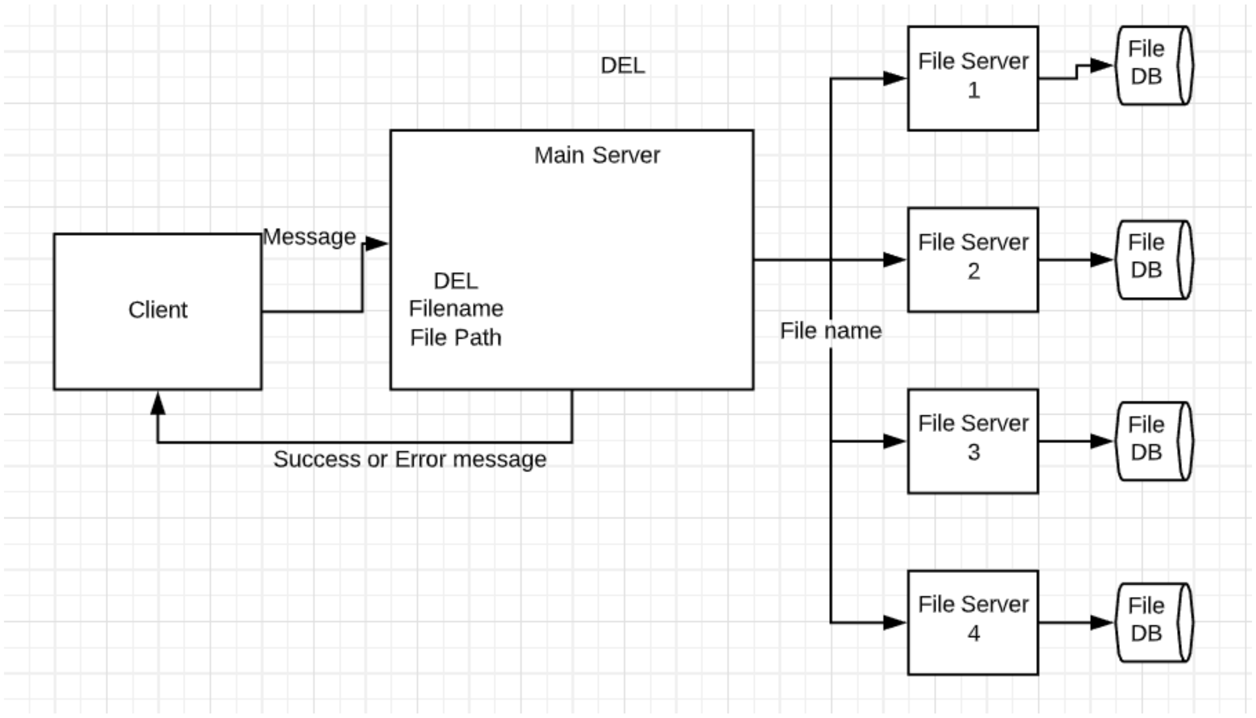
2.4 STR:
Store
First string will be file name
Second string will be file path
Third string will be file size



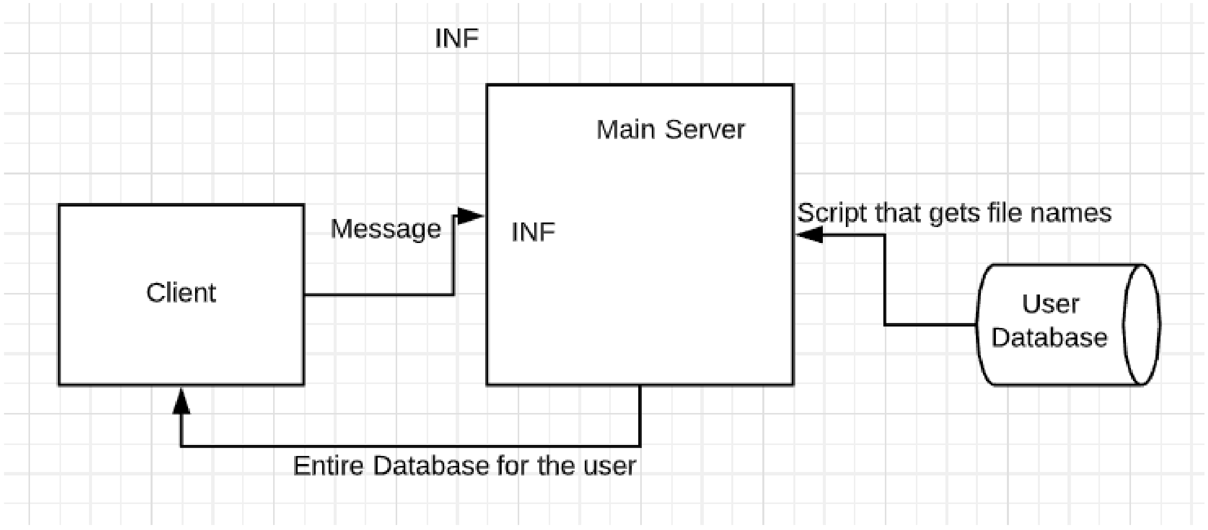
2.5 RTV:
Retrieve
First string will be file name
Second string will be file path



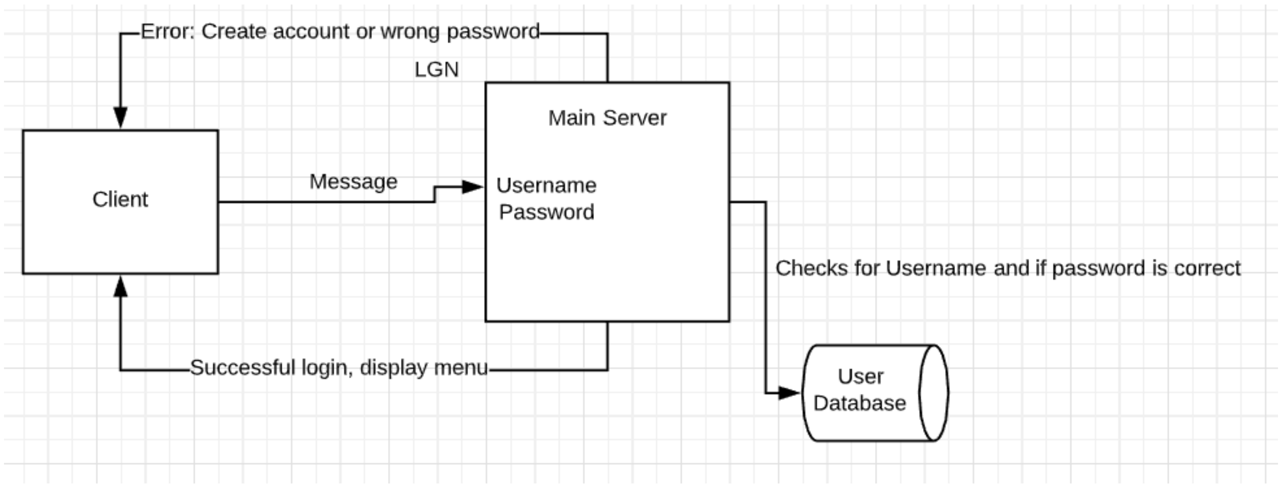
2.6 DEL:
Delete
First string will be file name
Second string will be file path



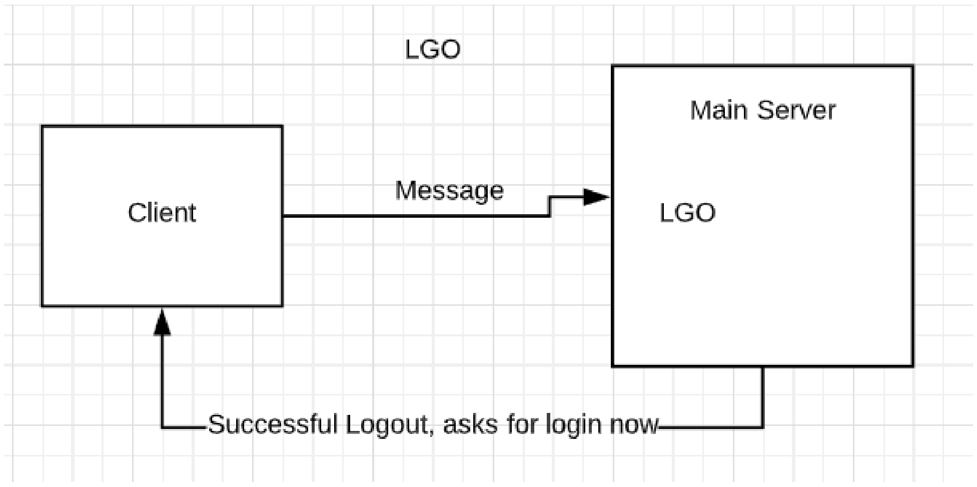
2.7 INF:
Info on Files
Will return the entire database in a structured file, see *3.6* for more info.



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.


Error: Create account or wrong password
LGN
Main Server
Client
Message
Username
Password
Checks for Username and if password is correct
Successful login, display menu
User Database

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.

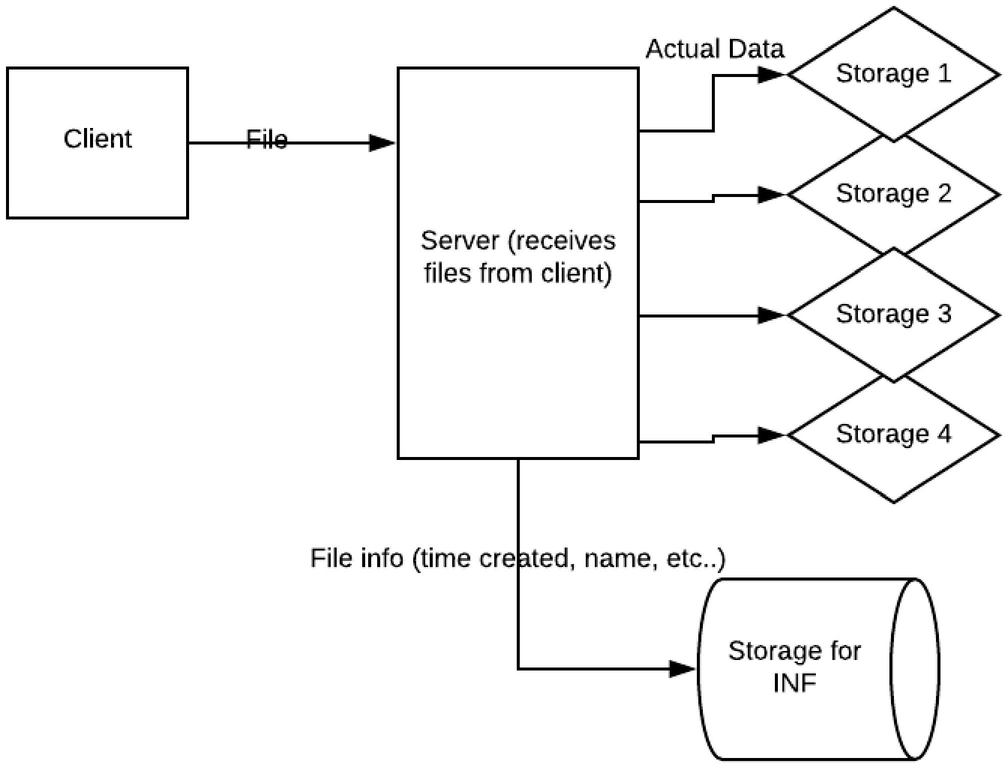
4.6 Once the user successfully logs in then the client will follow the protocol specified in **2.0**

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
```


Client
File
Server (receives files from client)
Actual Data
Storage 1
Storage 2
Storage 3
Storage 4
File info (time created, name, etc..)
Storage for INF

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:

$O1 = A1 \oplus B1$
$O2 = A2 \oplus B2$
$O3 = A2 \oplus B1$
$O4 = A1 \oplus A2 \oplus B2$

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

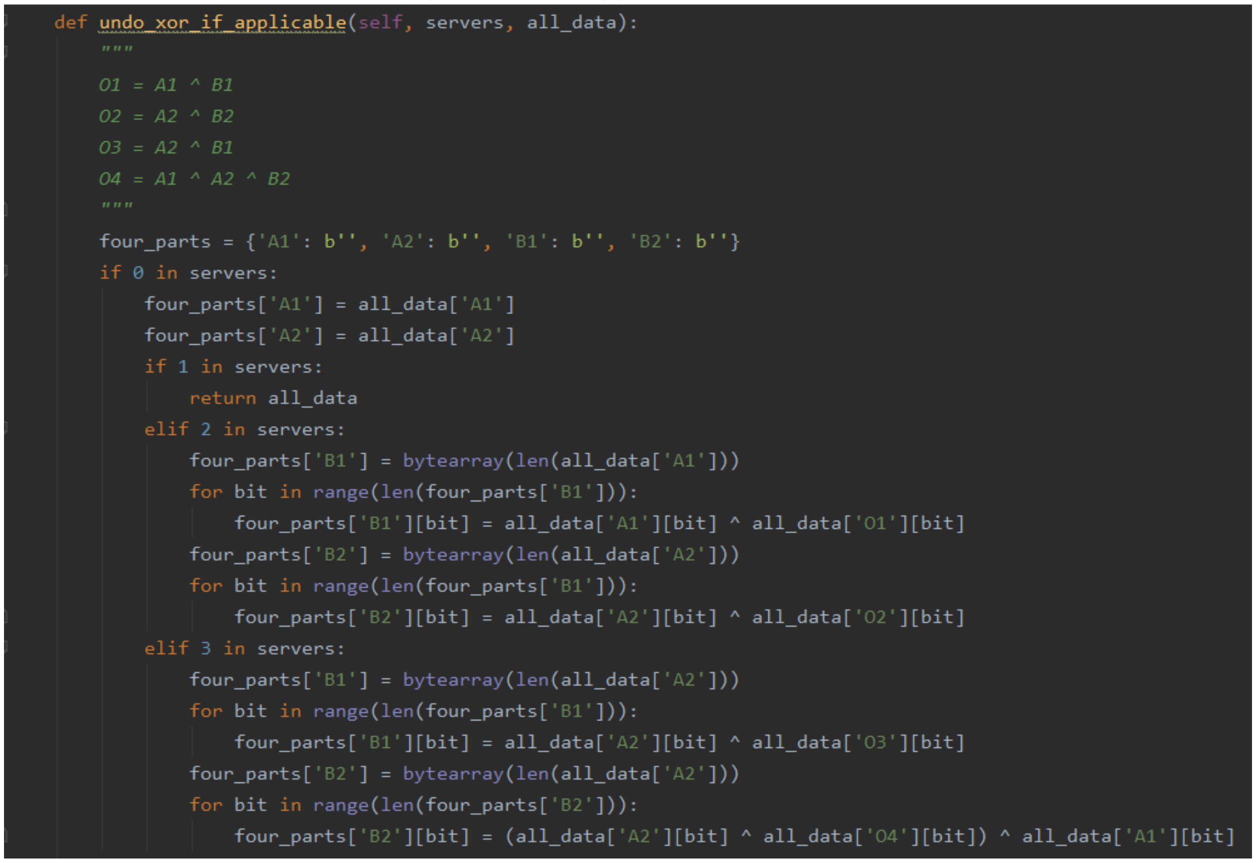
Result = fileSize % 4
0: appends 4 b'\x04'
1: appends 3 b'\x03'
2: appends 2 b'\x02'
3: appends 1 b'\x01'

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.

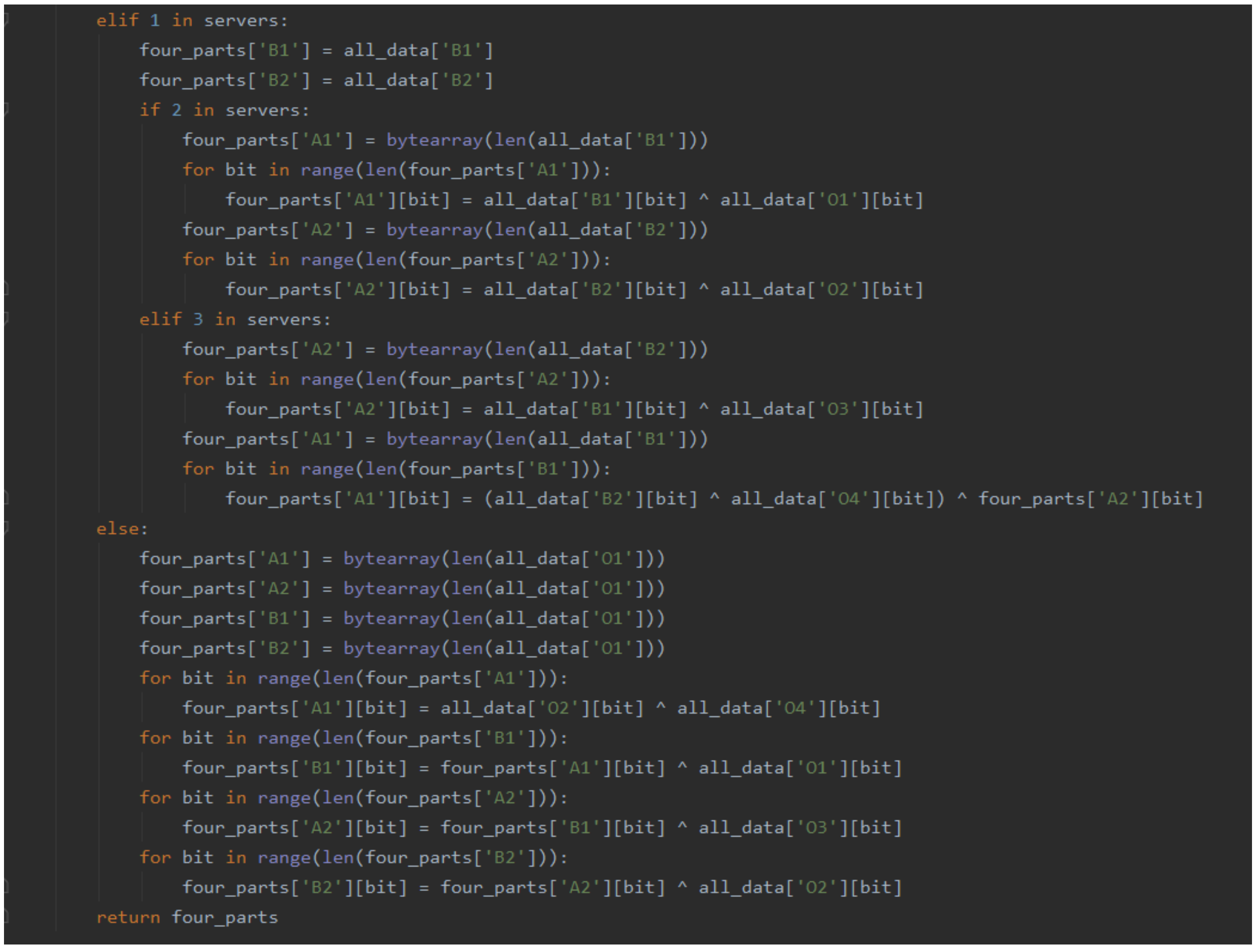
5.5 Once padded the main server will divide the file into four parts: A1, A2, B1, B2. O1, O2, O3, and O4 will be calculated as in *5.3*. It will send accordingly as stated in *5.2*.

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.



```
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
```

6.0 Demonstration

6.1 File Servers start

```
C:\Users\Matthew\Desktop\ECE470\Project1>bash fs_star
('Listening on port', 51131)
('Listening on port', 51132)
('Listening on port', 51133)
('Listening on port', 51134)
```

6.2 Main server connecting to file servers

```
connected to data server 1
connected to data server 2
connected to data server 3
connected to data server 4
Connected to all data servers.
```

6.3 Client connects to main server

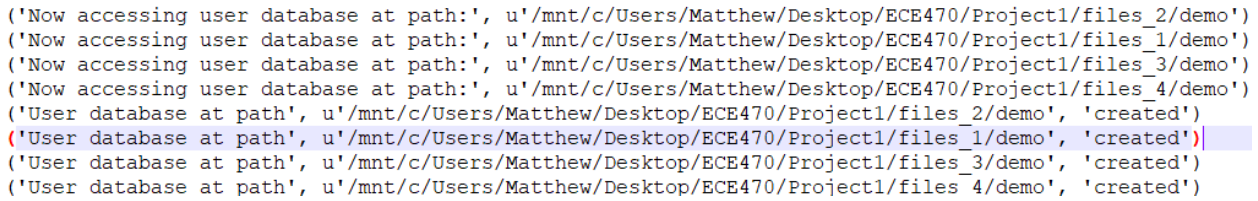
```
Listening on  17777
Client connected to Server Palk @ 2020-03-05 16:50:02.435276
```

```
-------------------------------------------------------------------------
| Hello client, you have connected to Server Palk. Note: this will change |
|  All files need to be inside a directory inside the c_files directory!  |
-------------------------------------------------------------------------
```

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

```
Username: demo
Password: password
Username does not exist in database!
Do you want to create an account (YES or NO): YES
Enter Desired Username: demo
Enter password: password
Confirm password: password
Welcome demo! You are a legend...
Enter the operation you want to send to the server from the list below
  STR - Store a file on the server
  RTV - Retrieve a file from the server
  DEL - Delete a file from the server
  INF - See which files you have stored on the server
  LGO - Logout from the server
```

```
C:\Users\Matthew\Desktop\ECE470\Project1\user_database\demo
User  demo  attempting to login.
user logged in
```

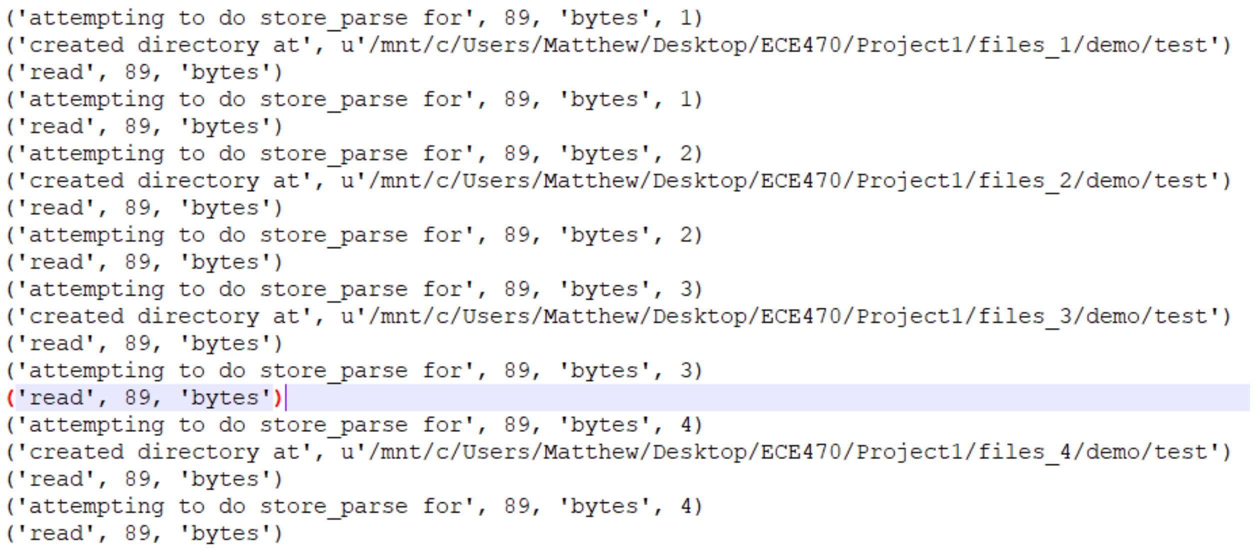


```
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_2/demo')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_1/demo')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_3/demo')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_4/demo')
('User database at path', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_2/demo', 'created')
('User database at path', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_1/demo', 'created')
('User database at path', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_3/demo', 'created')
('User database at path', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_4/demo', 'created')
```

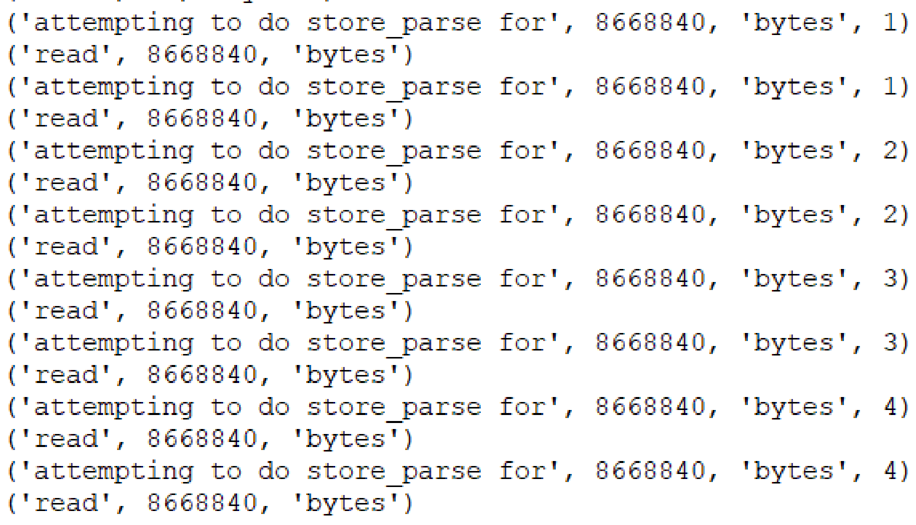
6.5 Storing a file successfully, client – server – file servers

```
  ~ STR
Enter the file name: yeet.txt
Enter the file path: test
sent "STR
yeet.txt
test
353

" to the server
now sending binary file!
received "File yeet.txt successfully created and written to" from the server
```

```
Storing file code
read 353 bytes
sending A1, A2 to server 1
sending B1, B2 to server 2
sending O1, O2 to server 3
sending O3, O4 to server 4
Storing file code
read 34675359 bytes
sending A1, A2 to server 1
sending B1, B2 to server 2
sending O1, O2 to server 3
sending O3, O4 to server 4
```



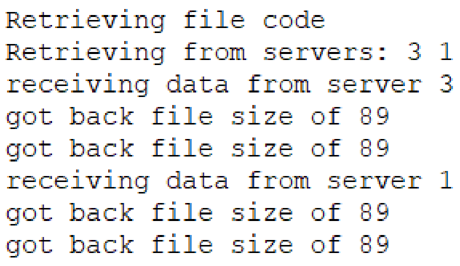
```
('attempting to do store_parse for', 89, 'bytes', 1)
('created directory at', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_1/demo/test')
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 1)
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 2)
('created directory at', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_2/demo/test')
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 2)
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 3)
('created directory at', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_3/demo/test')
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 3)
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 4)
('created directory at', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_4/demo/test')
('read', 89, 'bytes')
('attempting to do store_parse for', 89, 'bytes', 4)
('read', 89, 'bytes')
```



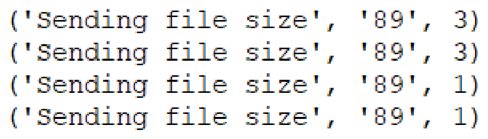
```
('attempting to do store_parse for', 8668840, 'bytes', 1)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 1)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 2)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 2)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 3)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 3)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 4)
('read', 8668840, 'bytes')
('attempting to do store_parse for', 8668840, 'bytes', 4)
('read', 8668840, 'bytes')
```

6.6 Retrieving a file, client – server – file servers

```
  ~ RTV
Enter the file name: yeet.txt
Enter the file path: test
sent "RTV
yeet.txt
test

" to the server
retrieving file code of size 353
read 353 bytes
Successfully retrieved file yeet.txt from the file system
```

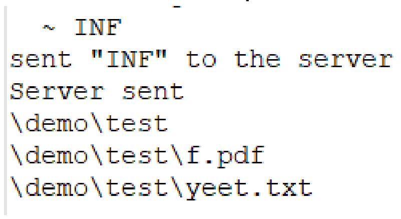


```
Retrieving file code
Retrieving from servers: 3 1
receiving data from server 3
got back file size of 89
got back file size of 89
receiving data from server 1
got back file size of 89
got back file size of 89
```



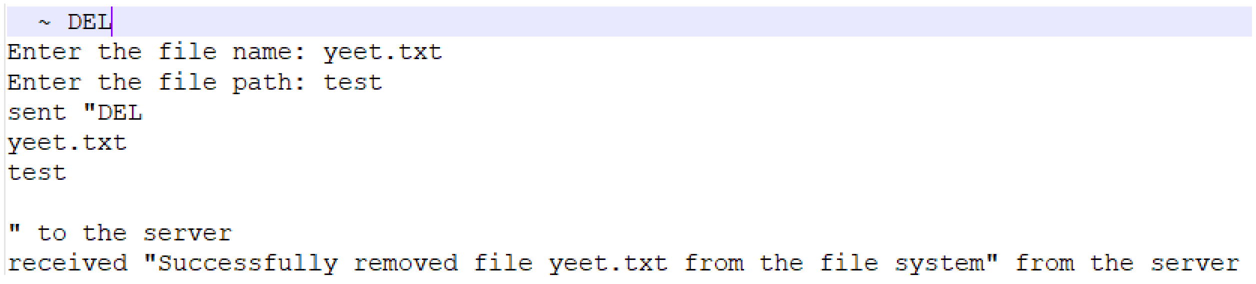
```
('Sending file size', '89', 3)
('Sending file size', '89', 3)
('Sending file size', '89', 1)
('Sending file size', '89', 1)
```

6.7 Information request, client – server

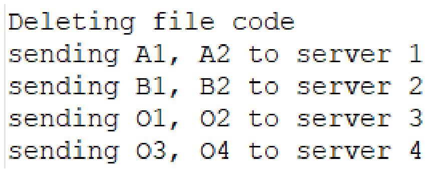


```
  ~ INF
sent "INF" to the server
Server sent
\demo\test
\demo\test\f.pdf
\demo\test\yeet.txt
```

6.8 Delete, client – server, file servers

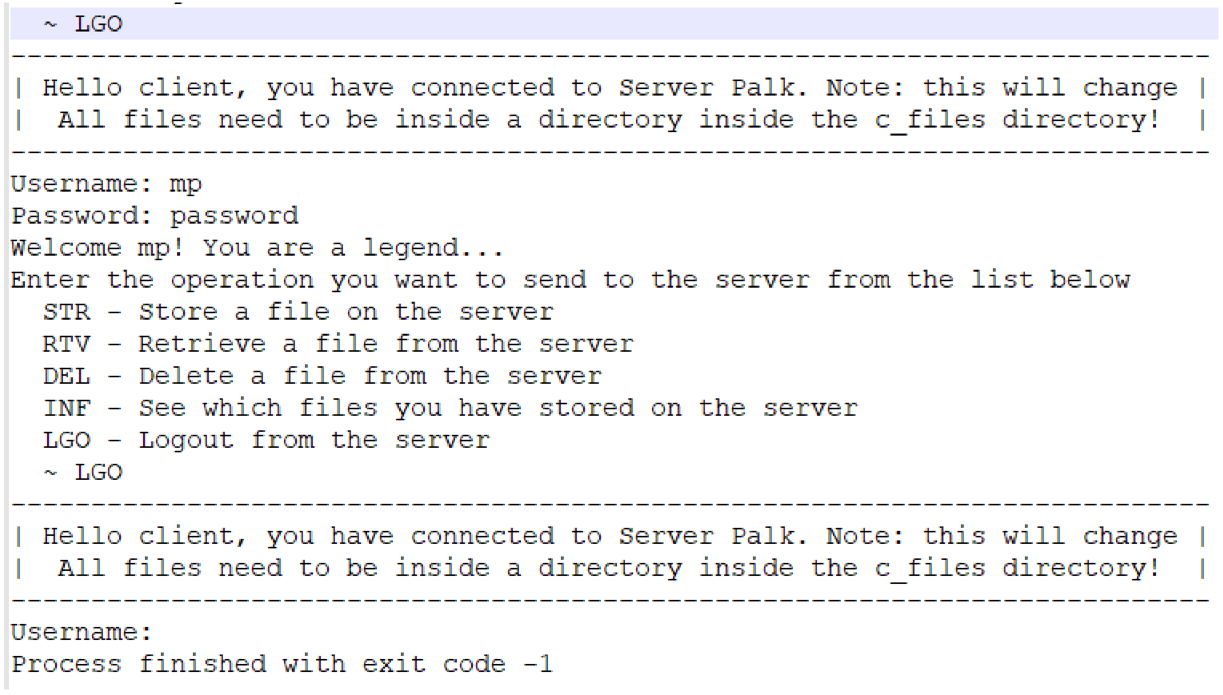


```
  ~ DEL
Enter the file name: yeet.txt
Enter the file path: test
sent "DEL
yeet.txt
test

" to the server
received "Successfully removed file yeet.txt from the file system" from the server
```

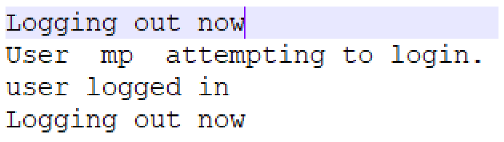


```
Deleting file code
sending A1, A2 to server 1
sending B1, B2 to server 2
sending O1, O2 to server 3
sending O3, O4 to server 4
```

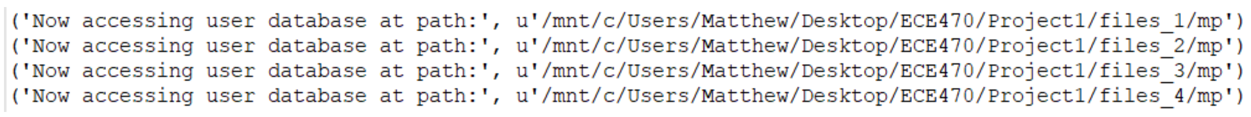
6.9 Logout to new login, client – server, file servers (file servers is showing if a new login happens it works)



```
  ~ LGO
---------------------------------------------------------------------------
| Hello client, you have connected to Server Palk. Note: this will change |
|  All files need to be inside a directory inside the c_files directory!  |
---------------------------------------------------------------------------
Username: mp
Password: password
Welcome mp! You are a legend...
Enter the operation you want to send to the server from the list below
  STR - Store a file on the server
  RTV - Retrieve a file from the server
  DEL - Delete a file from the server
  INF - See which files you have stored on the server
  LGO - Logout from the server
  ~ LGO
---------------------------------------------------------------------------
| Hello client, you have connected to Server Palk. Note: this will change |
|  All files need to be inside a directory inside the c_files directory!  |
---------------------------------------------------------------------------
Username:
Process finished with exit code -1
```



```
Logging out now
User  mp  attempting to login.
user logged in
Logging out now
```



```
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_1/mp')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_2/mp')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_3/mp')
('Now accessing user database at path:', u'/mnt/c/Users/Matthew/Desktop/ECE470/Project1/files_4/mp')
```

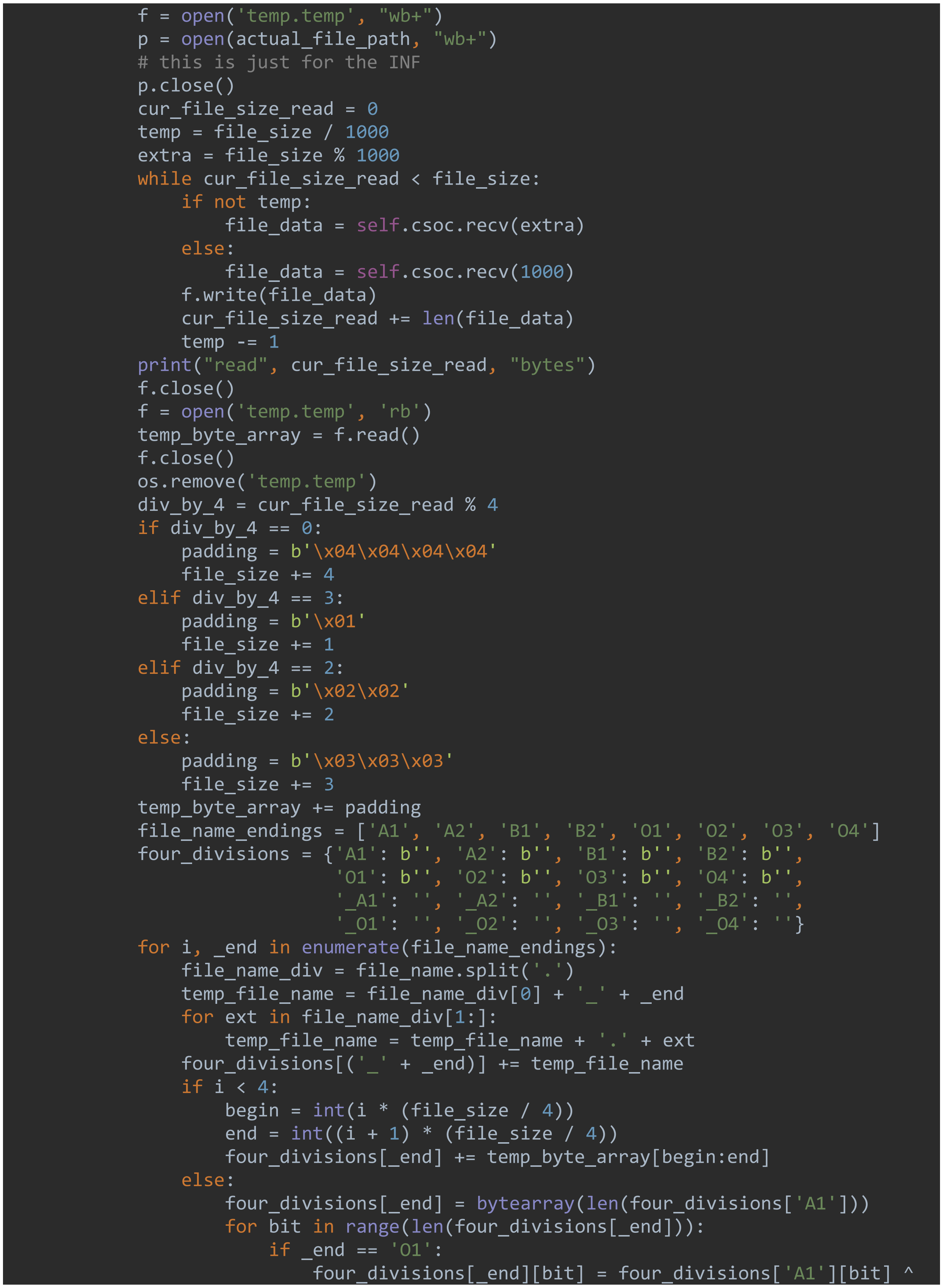
6.10 Entire logs from the demo run can be found in Documentation directory with names:

client_test_demo_output.txt
server_main_demo_output.txt
file_servers_demo_output.txt

7.0 Code

7.1 main_server.py

```python
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:
```



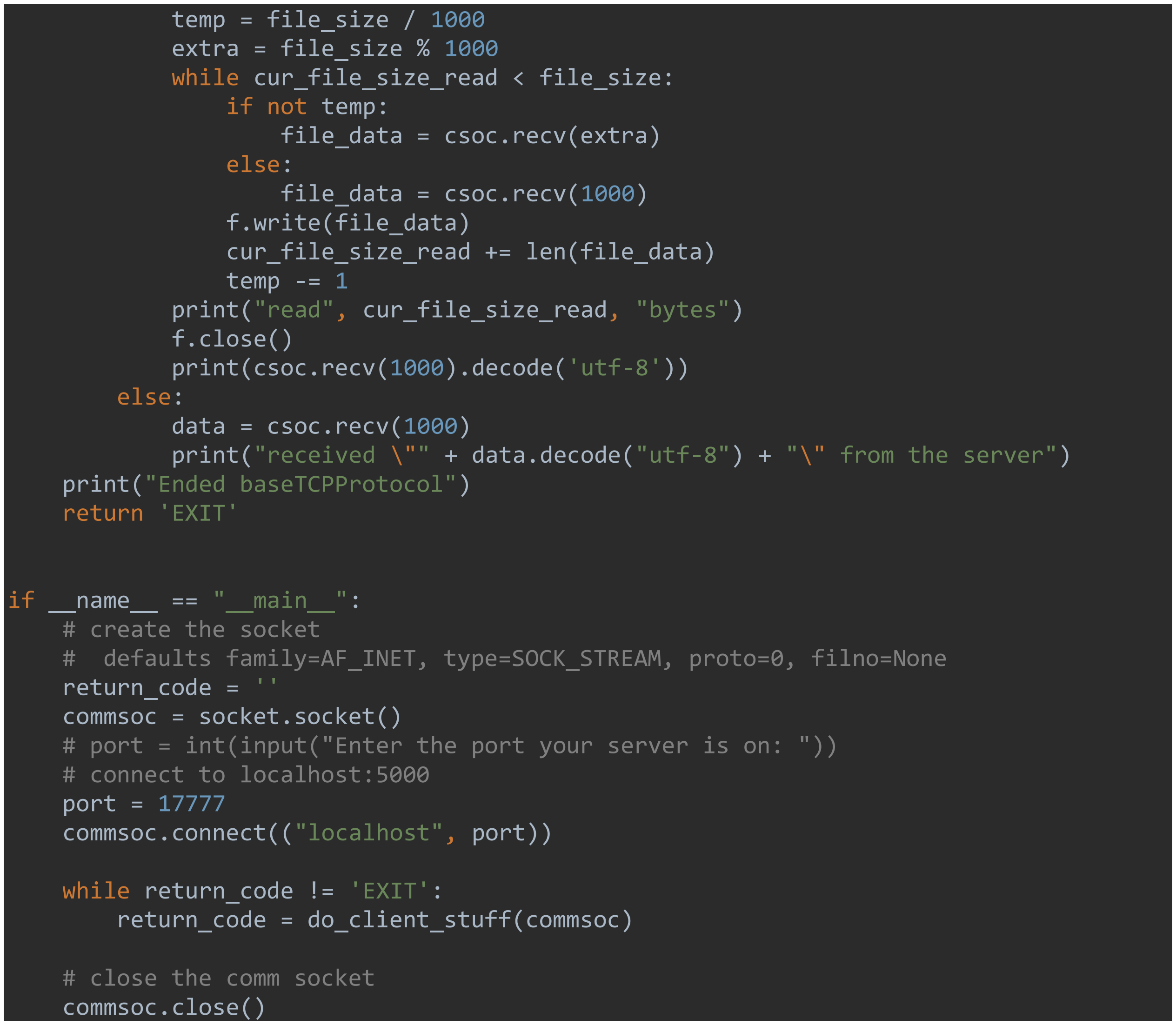
```python
            f = open('temp.temp', "wb+")
            p = open(actual_file_path, "wb+")
            # this is just for the INF
            p.close()
            cur_file_size_read = 0
            temp = file_size / 1000
            extra = file_size % 1000
            while cur_file_size_read < 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+")
```



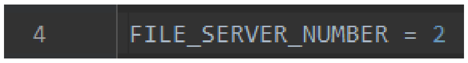
```
            temp = file_size / 1000
            extra = file_size % 1000
            while cur_file_size_read < file_size:
                if not temp:
                    file_data = csoc.recv(extra)
                else:
                    file_data = csoc.recv(1000)
                f.write(file_data)
                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()
```

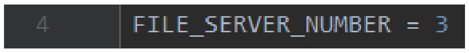
7.4 All other file servers are exactly the same as file_server1.py with the exception that at the top of the code the variable FILE_SERVER_NUMBER on line 4 changes to the file server number it is.

file_server2.py



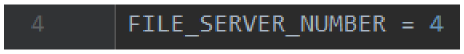
```
4    FILE_SERVER_NUMBER = 2
```

file_server3.py



```
4    FILE_SERVER_NUMBER = 3
```

file_server4.py



```
4    FILE_SERVER_NUMBER = 4
```