This is a description of how the project was realized.

The project includes two files. A “client.py” and a “server.py”

In Client.py, we first connect to the server socket with:

```
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)
```

While on the server side we have:

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)
```

The start method of the server file is used to start the server and listen for new connections. It uses multithreading to manage multiple connections at the same time.

```
def start():
    server.listen()
    print(f"[LISTENING] Server is listening on {SERVER}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")
```

The handle_client method takes care of all the incoming request messages from the clients. Its first part handles the receiving messages:

```
def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msg_length = conn.recv(HEADER).decode(FORMAT)
        if msg_length:
            msg_length = int(msg_length)
            msg = conn.recv(msg_length).decode(FORMAT)
            msg_split = msg.split(" ")
            print(f"[{addr}] {msg}")
```

While the second part parses the messages for specific commands (put, get, change, bye, help).

On the client side, there is a while loop that asks the user for inputs and parses them to find the specific commands to send to the server:

```
while (1):
    command = input("Input Command:")
    commandSplit=command.split(" ")
    filename=""

    #HANDLE Bye Command
    if (command.lower() == "bye"):
```

Now lets see how the different commands were implemented:

For (bye), the client will send a disconnect message to the server:

```
#HANDLE Bye Command
if (command.lower() == "bye"):
    send(DISCONNECT_MESSAGE)
    client.close()
    exit(0)
```

As for the server,

```
if msg == DISCONNECT_MESSAGE:
    connected = False
```

For the (help) command, the client will send a help message to the server:

```
#HANDLE Help Command
elif (command.lower() == "help"):
    send("help")
```

And the server will send back a list of possible commands:

```
# HANDLE Help Command
elif msg == "help":
    conn.send(
        "Possible Commands:\n put filename\n get filename\n change oldfilename newfilename \n help\n bye".encode(
            FORMAT))
    continue
```

For (change), client will send a change message to the server:

```
#HANDLE Change Command
elif (commandSplit[0].lower()=="change"):
    changeMessage="c "+commandSplit[1]+" "+commandSplit[2]
    send(changeMessage)
```

While the server will check if the file exists before performing the name change:

```
# HANDLE Change Command
elif msg_split[0] == "c":
    if os.path.exists(msg_split[1]):    #Check if oldFile exists
        os.rename(msg_split[1],msg_split[2])    #Rename oldFile to NewFile
        conn.send("[Name was successfully changed]".encode(FORMAT))
    else:
        conn.send("[Could not change file name because file was not found]".encode(FORMAT))
    continue
```

For (put), the client will check if the file exists before sending a put message the the server. Then it will send all the data in chunks of 1024 bytes by using a buffer. Once its done sending the data, it will send a blank message to the server signalling it that it has finished sending data:

```
#HANDLE Put Command
elif (commandSplit[0].lower()=="put"):
    filename=commandSplit[1]
    if(os.path.exists(filename)):   #Check if file exists before sending it to server
        putMessage="p "+filename
        send(putMessage)    #Sends Put Message
        with open(filename, "rb") as f:
            while True:
                bytes_read = f.read(BUFFER_SIZE)    #Reads and sends file data in chunks of BUFFER_SIZE(1024)
                client.sendall(bytes_read)
                if not bytes_read:
                    endByte=b" "*1024   #Sends empty data to signify that all the data has been sent
                    client.send(endByte)
                    break
    else:
        print("File does not exist try again")
```

Once the server receives the put command, it will create a file with the provided name and await the data in chunks of 1024 bytes from the client. It breaks out the while loop when it recognizes the blank message:

```
# HANDLE Put Command
elif msg_split[0] == "p":
    conn.send("[File was put]".encode(FORMAT))
    filename = msg_split[1]
    with open(filename, "wb") as f:  #Creates new file with the name received from the client
        while True:
            data = conn.recv(BUFFER_SIZE)
            f.write(data)               #Writes the data into the file
            if data==b" "*len(data):    #Check if the all the file data has been receive
                break                   #So that we can break out of the while loop
    continue
```

Finally, the (get) command is very similar to the put command. Except its in reverse. So the server will send a file to the client instead. I used the exact same logic as with the (put) command. The client sends a get message and awaits a response from the server:

```
#HANDLE Get Command
elif (commandSplit[0].lower()=="get"):
    getMessage="g "+commandSplit[1]
    send(getMessage)    #Sends Get Message and waits for the data from the file
    with open(commandSplit[1], "wb") as g:
        while True:
            data = client.recv(BUFFER_SIZE)
            g.write(data)
            if data == b" " * len(data):    #Checks if all data has been received
                break
        send(" ")
```

Once the server receives the get command, it will open the file and read from it. It will send the data in chunks of 1024bytes to the client (similar to the (put) command implementation).

```
# HANDLE Get Command
elif msg_split[0] == "g":
    nameOfFile=msg_split[1]
    with open(nameOfFile, "rb") as fi:  #Read from file
        bytesRead = fi.read(BUFFER_SIZE)
        conn.send("[Get File]".encode(FORMAT))
        conn.sendall(bytesRead)
        while True:
            #Check if we have read all data from file
            if not bytesRead:
                endByte = b" " * 1024
                conn.send(endByte)
                break
                #Send empty data which tells the client that the data has ended
                #Then brake out of the while loop
            bytesRead = fi.read(BUFFER_SIZE)
            conn.sendall(bytesRead)
    continue
```