Python Drop Box for Homelab

James Parrish Thrasher

IS- 512: Systems Analysis & Design
Jeremy Shelley
10/27/2023

Table of Contents:

Project Information

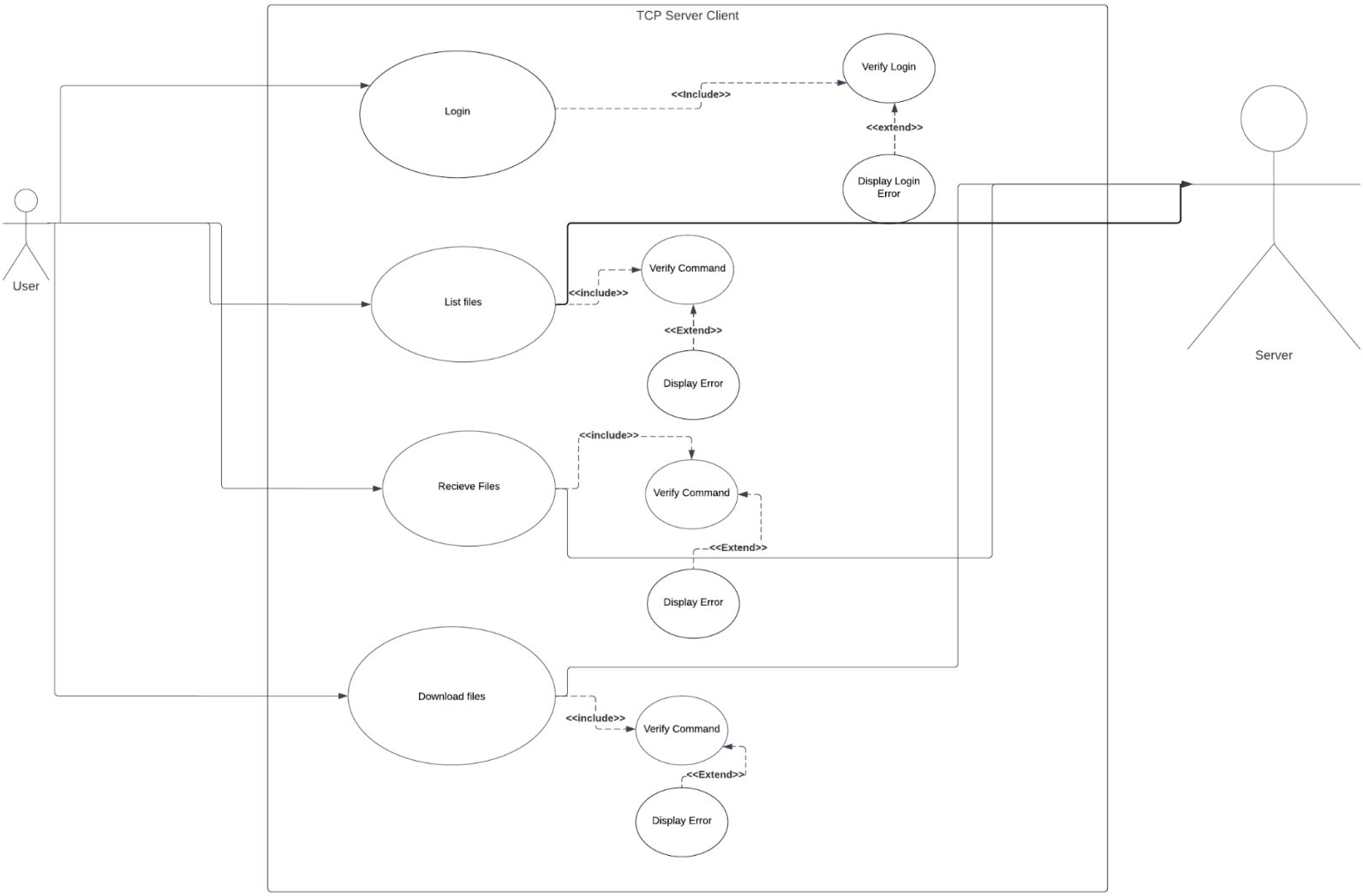
This project began with the creation of my personal homelab. I have a rack with a shelf containing my main home lab server. Which is a build 6w idle power draw. The only other item on the rack is a 1u HP server that was purchased only to run virtual machines for CTF competitions and learning to reverse engineer Malware. I wanted a program to put on that machine's main OS which is Ubuntu linux to send files to and from docker containers. This is a multiple stage project that even after this semester I will continue to change and update to suit my needs.

Currently I SSH into each of these clients to perform task but to leave this program running constantly while I am working on a project would allow me to more seamlessly upload files from another monitor or window without interrupting my work.

This project was done solely in Python3 with the logging being done in CSV format to be easily accessible in spreadsheets not only in a log format.I was okay using imported libraries for this project solely because it is on my personal network and sequestered into its own VLAN. In my opinion I do not recommend usage of this program for commercial purposes and is linked in my github.

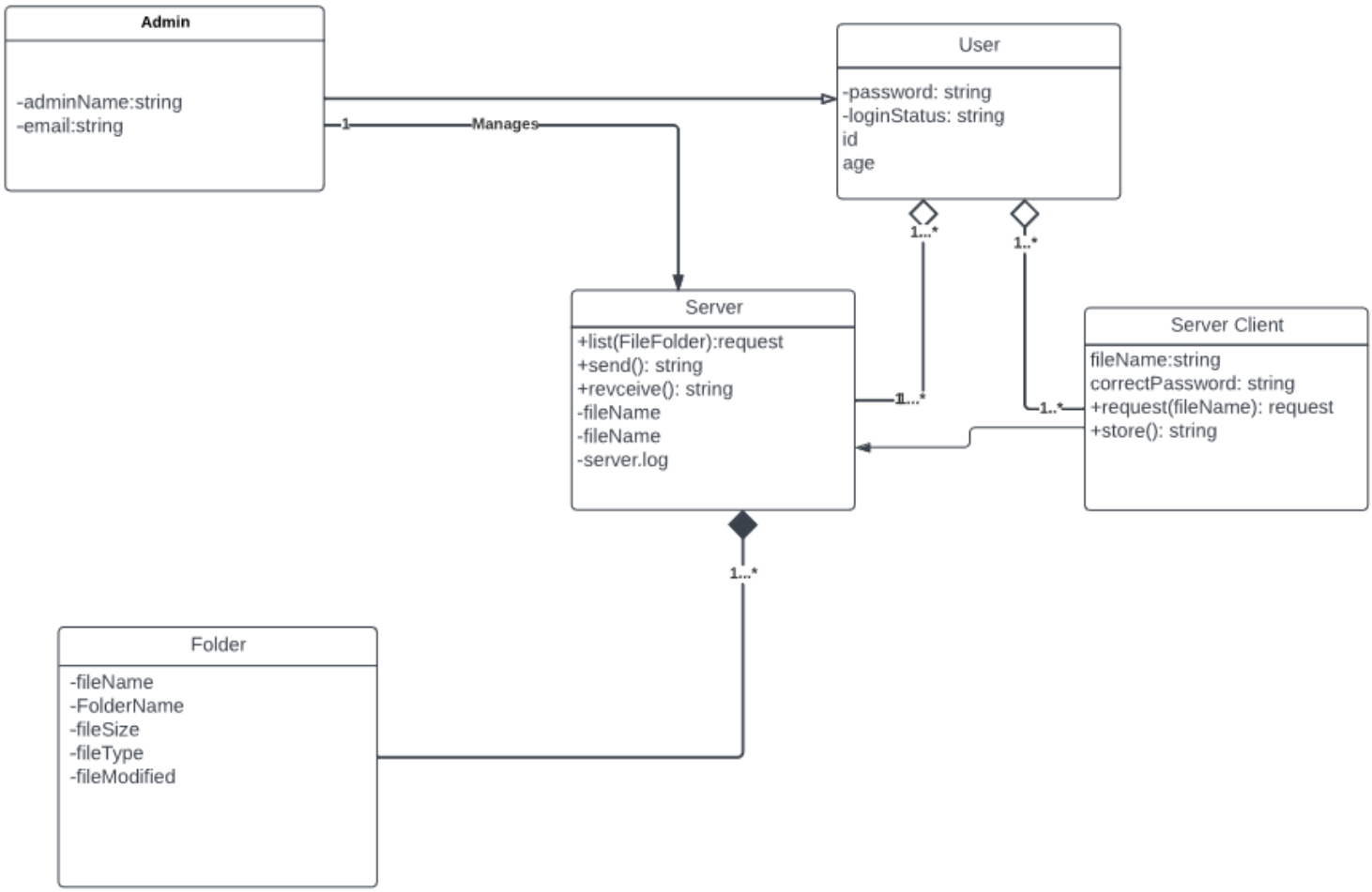
Use Case Diagram

The main users of this program will be people with small virtualization labs that are not web exposed. This means that the program only requires one password until given otherwise but usergroups with passwords are planned. All users will be able to use all functions of the system as long as they provide the correct login information. I intentionally made it where this program cannot delete or run programs inside of the folder to prevent malware from being automatically ran when uploaded.



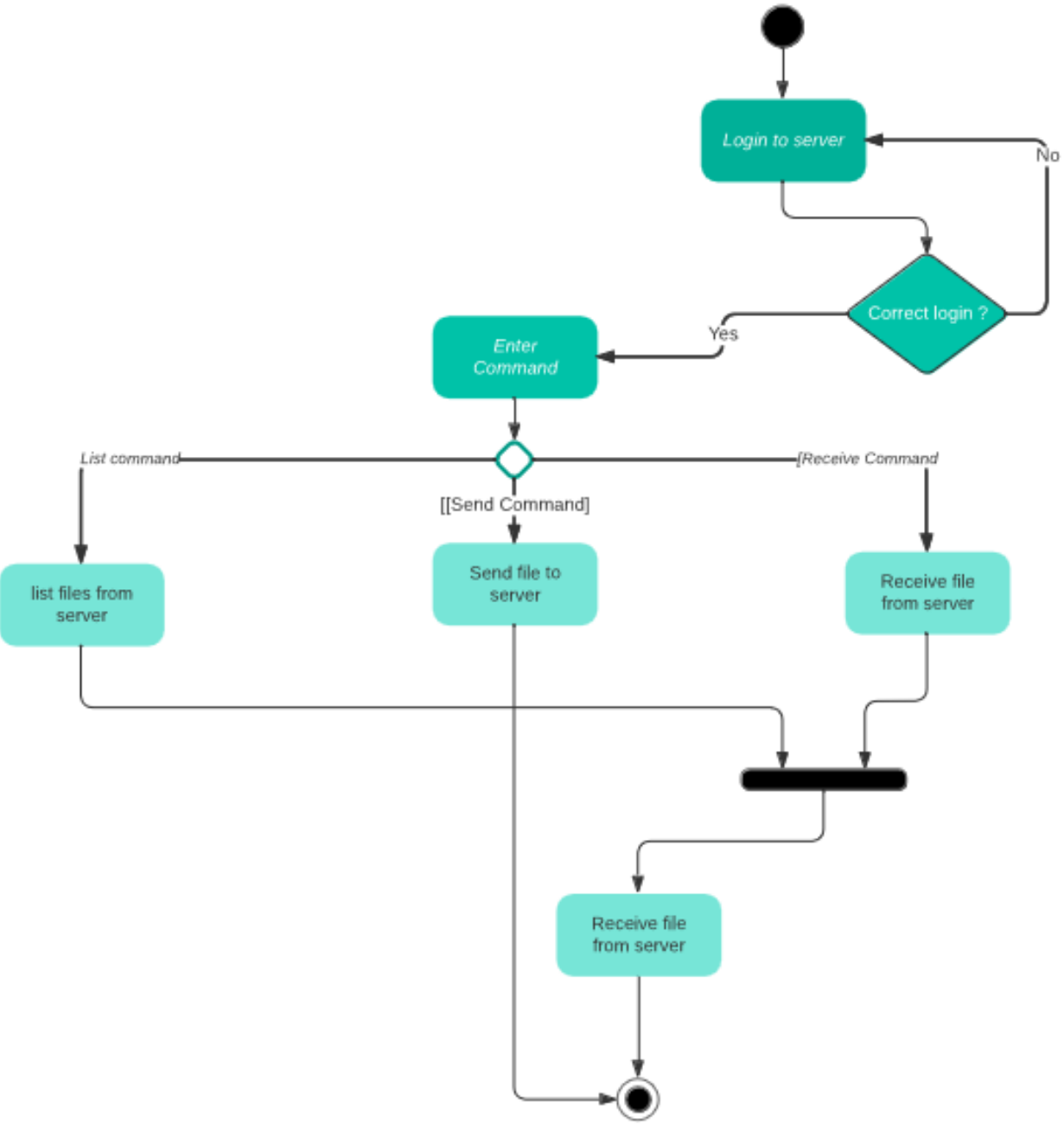
Domain Model Class Diagram

The domain model class diagram shows the connections and classes within different functions of the program.Such as the User has a password and a login status. The program's usage is to login and then send the possibility of three commands to the server which will begin those functions such as listing files which shows all of the file names in the folder. It can also send the file in packets of 1024 bytes to the server which is then written to a file created with the same name. The receive function of the server does the same thing in which the client sends the file name and contents in packets of 1024 bytes to the server to store.



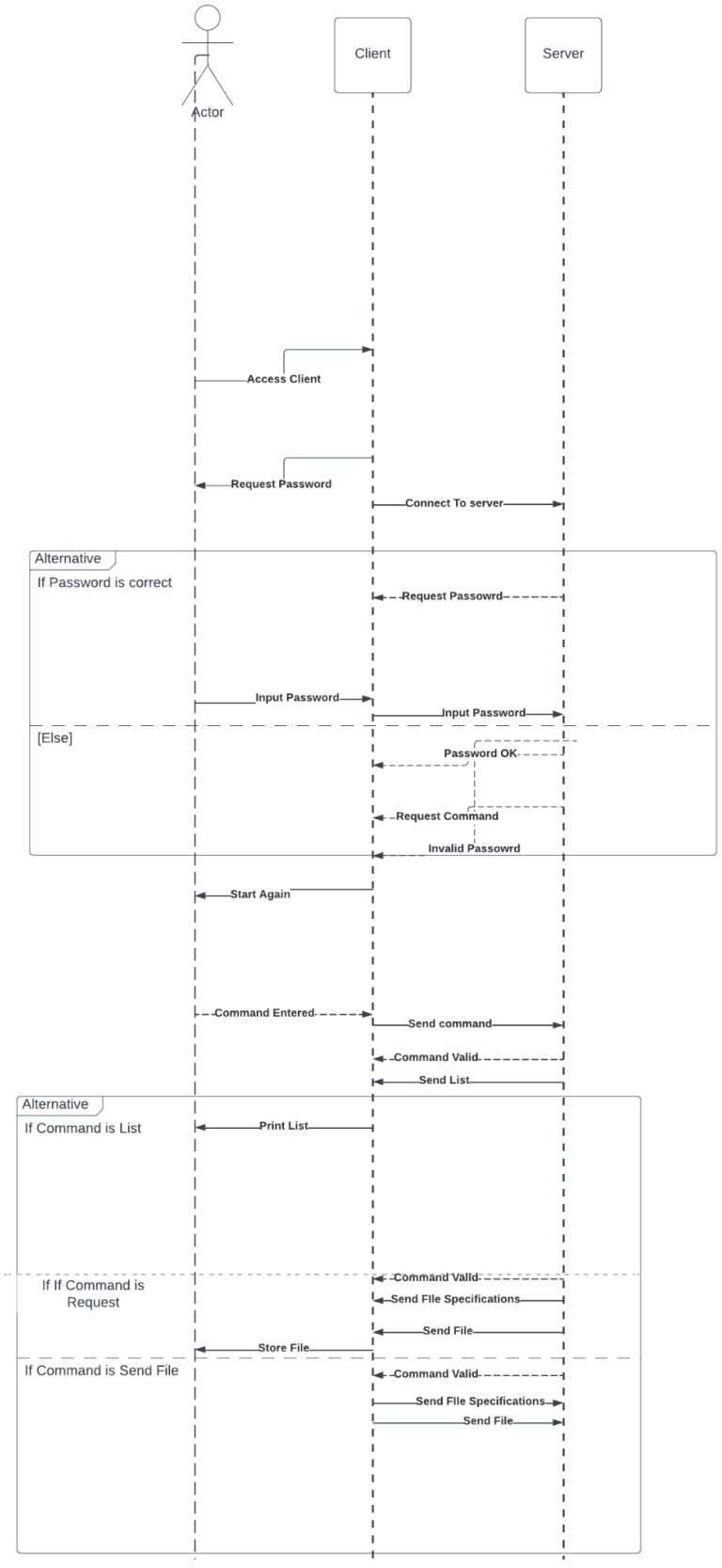
Activity Diagram

The activity diagram shows the series of events in which the activities of the program take place. Showing how an actor will begin as they are checked for the correct password all the way to receiving the files requested whether it is a list of the files or downloadings. This diagram helps give clear communication between developers and designers which is helpful for the development process but also is very useful in explaining to shareholders and marketing



Sequence Diagram

The system's sequence diagram shows the inputs and requests from the user to the server as they communicate through the client. This gives the clearest pictures since this program is so reliant on proper communication between the client and server.



Program

Initial Connection:

```
[jamest@f-is-for-friends Final Project]$ python FinalServer.py
Directory 'server-files' already exists.
Server is listening for incoming connections...
Connection from ('127.0.0.1', 39260)
Password accepted
taco
```

```
[jamest@f-is-for-friends Final Project]$ python FinalClient.py
Connected to localhost:8080
Directory 'client_files' already exists.
You have successfully connected.
Please enter the password:
Enter your password: taco
```

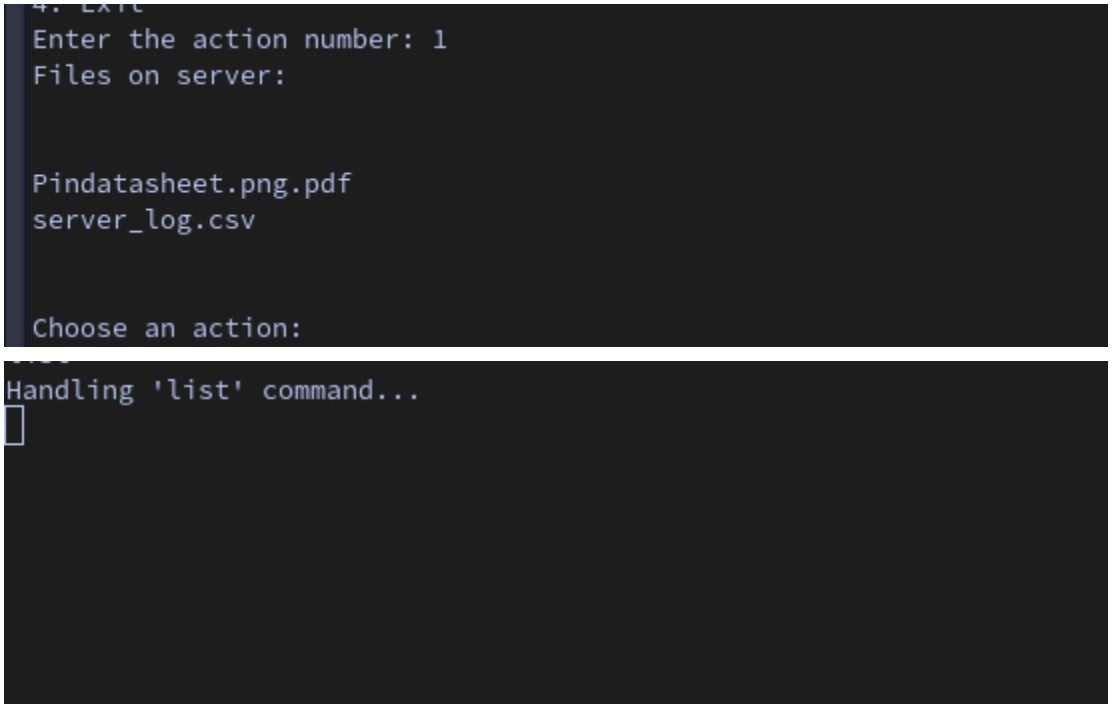
After you launch the Server.py part of the program you can run the client to connect to the Designated port and Address. The Client will then prompt for the Server Password.

Query:

```
Password accepted
Waiting for Command:
Choose an action:
1. List server files
2. Request a file
3. Upload a file
4. Exit
Enter the action number:
```

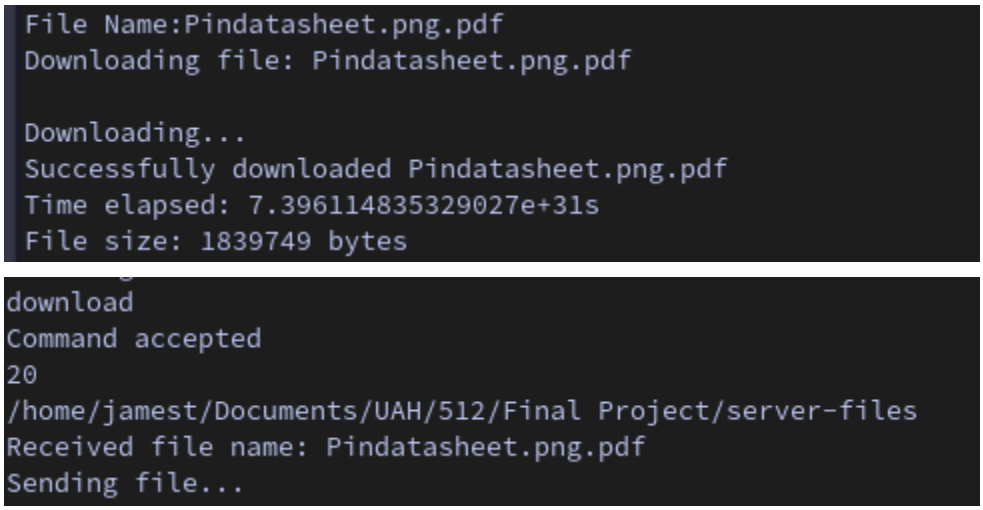
After The password has been accepted by the server the client will ask you to enter the number for the action you desire to execute.

List:



When selecting action number 1 you submit the ‘list’ command to the server in which the server will display that it was received by Handling ‘list’ command.

Request a file:



When you select Request a file option 2 on the client. It Executes the send file command on the server which displays the file directory the file name and then the client displays how long it takes to download the file once initiated along with the file size. The client sends the file in bytes in packet sizes of 1024 bytes.

Upload a file:

```
Choose an action:
1. List server files
2. Request a file
3. Upload a file
4. Exit
Enter the action number: 3
Enter the local file path to upload:Pindatasheet.png.pdf
Sending 'upload' command to the server
Server is ready to receive file content
Sending file contents
sent in packets of 1024
File 'Pindatasheet.png.pdf' uploaded successfully.
```

```
Client disconnected.
Connection from ('127.0.0.1', 37174)
Password accepted
taco
upload
Received file name length: 20
Receiving file: 'Pindatasheet.png.pdf'
Sent acknowledgment to the client
Received file size: 1839749 bytes
Sent acknowledgment to the client
Receiving...
Received file 'Pindatasheet.png.pdf' and saved it to the current working directory.
```

The Client Seds the upload command and file name following then checks to see if the server is ready to receive the content in packets of 1024. The Server then receives the file name length and name and sends it is ready to receive the file. Receiving the file size in bytes and writing it to the file it created with the name and size.

Disconnect:

```
Choose an action:
1. List server files
2. Request a file
3. Upload a file
4. Exit
Enter the action number: 4
Shutting down.
```

The exit command disconnects from the server socket.

Logging:

The server log file shows any passwords accepted and upon disconnect logs all of the commands done by that client

Server Code

FinalServer.py > accept_password

```python
import os
import struct
from datetime import datetime
import csv
import logging     # "logging" is not accessed
from tqdm import tqdm     # "tqdm" is not accessed


log_file = "server_log.csv"

buffer_size=1024


# Define the log_session function as a global function
def log_session(addr, password_accepted, commands):
    with open(log_file, mode='a', newline='') as log_csv:
        log_writer = csv.writer(log_csv)
        if password_accepted:
            log_writer.writerow([datetime.now(), addr, "Accepted", f"Password: {commands[0]}", ', '.join(commands[1:])])
        else:
            log_writer.writerow([datetime.now(), addr, "Rejected", f"Password: {commands[0]}", ', '.join(commands[1:])])
def check_dir():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    new_directory_name = "server-files"
    new_directory_path = os.path.join(current_directory, new_directory_name)

    if not os.path.exists(new_directory_path):
        os.mkdir(new_directory_path)
        os.chdir(new_directory_path)
        print(f"Directory '{new_directory_name}' created successfully.")
    else:
        os.chdir(new_directory_path)  # Change the working directory to the existing directory
        print(f"Directory '{new_directory_name}' already exists.")

```

FinalServer.py > accept_password

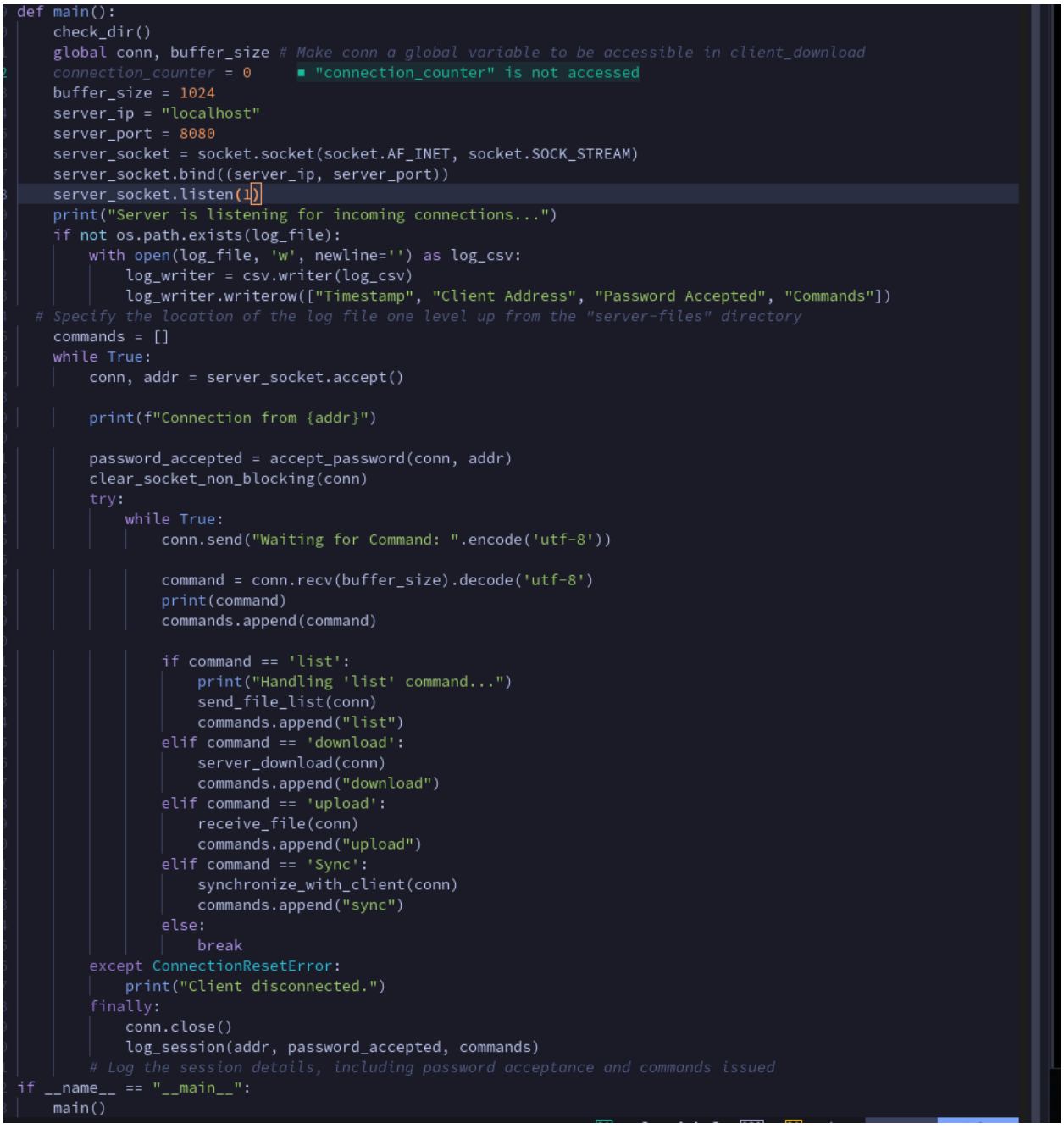
```python

def accept_password(conn, addr):
    try:
        correct_password = 'taco'
        conn.send("Please enter the password: ".encode('utf-8'))
        received_password = conn.recv(buffer_size).decode('utf-8')

        if received_password == correct_password:
            conn.send("Password accepted".encode('utf-8'))
            print("Password accepted")
            print(received_password)
            # Log the password acceptance
            log_session(addr, "Accepted", ["Password: " + received_password])
            return True
        else:
            print(f"Incorrect password: {received_password}")
            conn.send("Invalid password. Access denied".encode('utf-8'))
            print(received_password)
            # Log the password rejection
            log_session(addr, "Rejected", ["Password: " + received_password])
            return False
    except ConnectionResetError:
        print("Connection reset by the client.")
        # Log the connection reset
        log_session(addr, "Connection Reset", [])
        return False
    except Exception as e:
        print(f"Error during password acceptance: {str(e)}")
        # Log the password acceptance error
        log_session(addr, "Error", [str(e)])
        return False

def clear_socket_non_blocking(server_socket):
    server_socket.setblocking(0)  # Set the socket to non-blocking mode
    while True:
        try:
            data = server_socket.recv(1024)
            if not data:
                break
        except BlockingIOError:
            break  # No more data to receive
    server_socket.setblocking(1)

#This is the server
# Function to handle file uploads from the client

```

```python
def receive_file(conn):

    try:
        # Receive the file name length from the client
        file_name_length = struct.unpack("I", conn.recv(4))[0]
        print(f"Received file name length: {file_name_length}")

        # Receive the file name
        file_name = conn.recv(file_name_length).decode('utf-8')
        print(f"Receiving file: '{file_name}'")

        # Send an acknowledgment to the client
        conn.send("1".encode('utf-8'))
        print("Sent acknowledgment to the client")

        # Receive the file size
        file_size = struct.unpack("Q", conn.recv(8))[0]
        print(f"Received file size: {file_size} bytes")

        # Send an acknowledgment to the client
        conn.send("1".encode('utf-8'))
        print("Sent acknowledgment to the client")

        # Create a local file for writing in the current working directory
        output_file = open(file_name, "wb")
        bytes_received = 0
        print("Receiving...")

        while bytes_received < file_size:
            data = conn.recv(1024)  # Adjust the buffer size as needed
            output_file.write(data)
            bytes_received += len(data)
        output_file.close()
        print(f"Received file '{file_name}' and saved it to the current working directory.")
    except Exception as e:
        print(f"Error receiving file: {str(e)}")
```

```python
def send_file_list(client_socket):
    try:
        directory = os.path.join(os.path.dirname(os.path.abspath(__file__)), "server-files")
        if os.path.exists(directory) and os.path.isdir(directory):
            file_list = os.listdir(directory)  # Get a list of files in the directory
            file_list_str = "\n".join(file_list)  # Convert the list to a string with newline separators
            client_socket.send(file_list_str.encode('utf-8'))  # Send the list to the client:

        else:

            client_socket.send("No files available.".encode('utf-8'))
    except Exception as e:
        print(f"Error sending file list: {str(e)}")

def server_download(conn):
    try:
        # Send "Command accepted" message
        print("Command accepted")
        conn.send("Command accepted".encode('utf-8'))

        # Receive the file name length
        file_name_length = struct.unpack("h", conn.recv(2))[0]
        print(file_name_length)

        # Receive the file name itself
        print(os.getcwd())
        file_name = conn.recv(file_name_length).decode('utf-8')
        print(f"Received file name: {file_name}")

        # Construct the full path to the file in the "server-files" directory
        file_path = os.path.join(os.getcwd(), file_name)

        if os.path.isfile(file_path):
            # Send the file size to the client
            conn.send(struct.pack("i", os.path.getsize(file_path)))
            conn.recv(buffer_size)  # Wait for the client's acknowledgment
            print("Sending file...")

            with open(file_path, "rb") as content:
                while True:
                    data = content.read(buffer_size)
                    if not data:
                        break
                    conn.send(data)
            conn.recv(buffer_size)  # Wait for the client's acknowledgment
        else:
            print(f"File '{file_name}' not found in the 'server-files' directory")
            conn.send(struct.pack("i", -1))
    except Exception as e:
        print(f"Error during file download: {str(e)}")
```

FinalServer.py > receive_file_list

```python

def receive_file_list(server_socket):
    try:
        # Accept a client connection
        client_socket, client_address = server_socket.accept()      "client_address" is not accessed

        # Receive the length of the data
        data_length_bytes = client_socket.recv(struct.calcsize('!Q'))
        data_length = struct.unpack('!Q', data_length_bytes)[0]

        # Receive the serialized data
        data = b""
        while len(data) < data_length:
            packet = client_socket.recv(data_length - len(data))
            if not packet:
                break
            data += packet

        # Unpack the received data
        file_list_str = struct.unpack(f"{data_length}s", data)[0].decode('utf-8')

        if file_list_str == "No files available.":
            print("No files available on the client.")
        else:
            file_list = file_list_str.split('\n')
            print("Received File List:")
            for filename in file_list:
                print(filename)

        client_socket.close()
    except Exception as e:
        print(f"Error receiving file list: {str(e)}")

def synchronize_with_client(conn):
    client_files = eval(conn.recv(1024).decode('utf-8'))

    server_files = get_server_files()

    files_to_send = {}
    for file, client_timestamp in client_files.items():
        server_timestamp = server_files.get(file, 0)
        if not os.path.exists(file) or client_timestamp > server_timestamp:
            files_to_send[file] = server_timestamp

    conn.send(str(files_to_send).encode('utf-8'))
    conn.close()

def get_server_files():
    server_files = {}
    for file in os.listdir('.'):
        if os.path.isfile(file):
            timestamp = os.path.getmtime(file)
            server_files[file] = timestamp
    return server_files
```



```python
def main():
    check_dir()
    global conn, buffer_size # Make conn a global variable to be accessible in client_download
    connection_counter = 0      ■ "connection_counter" is not accessed
    buffer_size = 1024
    server_ip = "localhost"
    server_port = 8080
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((server_ip, server_port))
    server_socket.listen(1)
    print("Server is listening for incoming connections...")
    if not os.path.exists(log_file):
        with open(log_file, 'w', newline='') as log_csv:
            log_writer = csv.writer(log_csv)
            log_writer.writerow(["Timestamp", "Client Address", "Password Accepted", "Commands"])
  # Specify the location of the log file one level up from the "server-files" directory
    commands = []
    while True:
        conn, addr = server_socket.accept()

        print(f"Connection from {addr}")

        password_accepted = accept_password(conn, addr)
        clear_socket_non_blocking(conn)
        try:
            while True:
                conn.send("Waiting for Command: ".encode('utf-8'))

                command = conn.recv(buffer_size).decode('utf-8')
                print(command)
                commands.append(command)

                if command == 'list':
                    print("Handling 'list' command...")
                    send_file_list(conn)
                    commands.append("list")
                elif command == 'download':
                    server_download(conn)
                    commands.append("download")
                elif command == 'upload':
                    receive_file(conn)
                    commands.append("upload")
                elif command == 'Sync':
                    synchronize_with_client(conn)
                    commands.append("sync")
                else:
                    break
        except ConnectionResetError:
            print("Client disconnected.")
        finally:
            conn.close()
            log_session(addr, password_accepted, commands)
        # Log the session details, including password acceptance and commands issued
if __name__ == "__main__":
    main()
```

Client Code

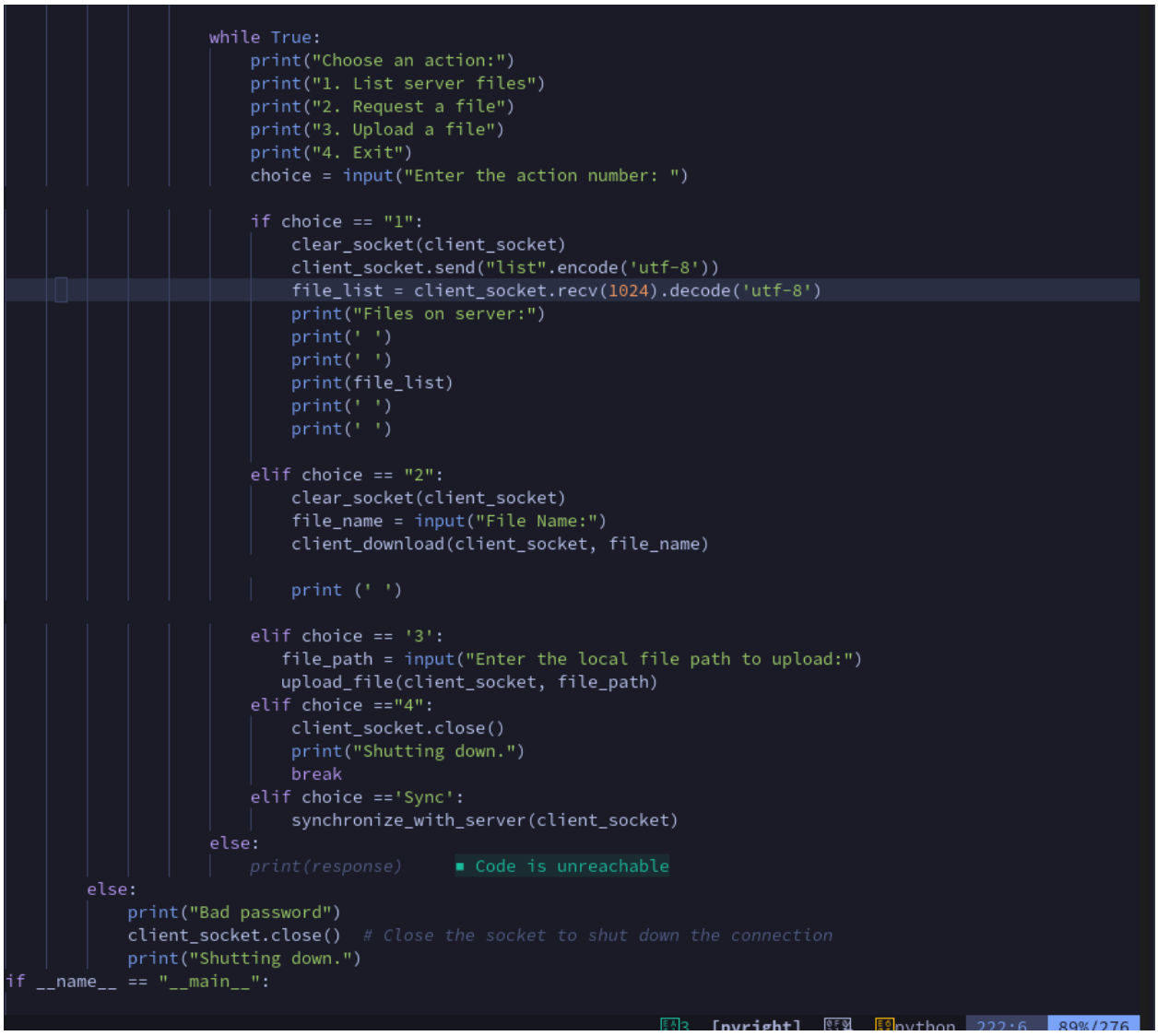
```python
import socket
import os
from tqdm import tqdm      # "tqdm" is not accessed
import struct
#standard buffer stuff will be changed in main if needed
buffer_size= 1024
def connect_to_server(server_ip, server_port):
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_ip, server_port))
        print(f"Connected to {server_ip}:{server_port}")
        return client_socket
    except Exception as e:
        print(f"Connection failed: {str(e)}")
        return None

def request_password_from_user():
    password = input("Enter your password: ")
    return password

def clear_socket(client_socket):
    client_socket.setblocking(0)  # Set the socket to non-blocking mode
    while True:
        try:
            data = client_socket.recv(1024)
            if not data:
                break
        except BlockingIOError:
            break  # No more data to receive
    client_socket.setblocking(1)
def check_dir():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    new_directory_name = "client_files"
    new_directory_path = os.path.join(current_directory, new_directory_name)

    if not os.path.exists(new_directory_path):
        os.mkdir(new_directory_path)
        os.chdir(new_directory_path)  # Change the working directory to the new directory
        print(f"Directory '{new_directory_name}' created successfully.")
    else:
        os.chdir(new_directory_path)  # Change the working directory to the existing directory
        print(f"Directory '{new_directory_name}' already exists.")
```

```python
import socket
import os
from tqdm import tqdm      ■ "tqdm" is not accessed
import struct
#standard buffer stuff will be changed in main if needed
buffer_size= 1024
def connect_to_server(server_ip, server_port):
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_ip, server_port))
        print(f"Connected to {server_ip}:{server_port}")
        return client_socket
    except Exception as e:
        print(f"Connection failed: {str(e)}")
        return None

def request_password_from_user():
    password = input("Enter your password: ")
    return password

def clear_socket(client_socket):
    client_socket.setblocking(0)  # Set the socket to non-blocking mode
    while True:
        try:
            data = client_socket.recv(1024)
            if not data:
                break
        except BlockingIOError:
            break  # No more data to receive
    client_socket.setblocking(1)
def check_dir():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    new_directory_name = "client_files"
    new_directory_path = os.path.join(current_directory, new_directory_name)

    if not os.path.exists(new_directory_path):
        os.mkdir(new_directory_path)
        os.chdir(new_directory_path)  # Change the working directory to the new directory
        print(f"Directory '{new_directory_name}' created successfully.")
    else:
        os.chdir(new_directory_path)  # Change the working directory to the existing directory
        print(f"Directory '{new_directory_name}' already exists.")
```

```python
def client_download(client_socket, file_name):
    print(f"Downloading file: {file_name}")

    try:
        # Send "download" command to the server
        client_socket.send("download".encode('utf-8'))
    except Exception as e:
        return f"Couldn't make a server request. Make sure a connection has been established. Error: {str(e)}"

    try:
        # Wait for the "Command accepted" message from the server
        response = client_socket.recv(buffer_size).decode('utf-8')
        if response != "Command accepted":
            return f"Server did not accept the command. Server response: {response}"
    except Exception as e:
        return f"Error waiting for command acceptance: {str(e)}"


    try:
        # Send the file name length and name
        file_name_encoded = file_name.encode('utf-8')
        client_socket.send(struct.pack("h", len(file_name_encoded)))
        client_socket.send(file_name_encoded)

        # Receive the file size
        file_size = struct.unpack("i", client_socket.recv(4))[0]
        if file_size == -1:
            print("File does not exist. Make sure the name was entered correctly")
            return
        # Send acknowledgment to the server
        client_socket.send("1".encode('utf-8'))

        # Create a local file for writing
        output_file = open(file_name, "wb")
        bytes_received = 0
        print("\nDownloading...")
        while bytes_received < file_size:
            data = client_socket.recv(buffer_size)
            output_file.write(data)
            bytes_received += len(data)
        output_file.close()
        print(f"Successfully downloaded {file_name}")

        # Tell the server that the client is ready to receive the download performance details
        client_socket.send("1".encode('utf-8'))
        # Get performance details
        time_elapsed = struct.unpack("f", client_socket.recv(4))[0]
        print(f"Time elapsed: {time_elapsed}s\nFile size: {file_size} bytes")
    except Exception as e:
        print(f"Error downloading file: {str(e)}")
        return
```

```python
def upload_file(client_socket, file_name):
    try:
        # Check if the file exists in the current directory
        if os.path.exists(file_name) and os.path.isfile(file_name):
            # Combine the file name with the current directory to create the file path
            file_path = os.path.join(os.getcwd(), file_name)

            # Send the "upload" command to the server
            print("Sending 'upload' command to the server")
            client_socket.send("upload".encode('utf-8'))

            # Send the file name length and name
            file_name_encoded = file_name.encode('utf-8')
            client_socket.send(struct.pack("I", len(file_name_encoded)))
            client_socket.send(file_name_encoded)

            # Send the file size
            file_size = os.path.getsize(file_path)
            client_socket.send(struct.pack("Q", file_size))

            acknowledgment = client_socket.recv(1).decode('utf-8')

            if acknowledgment == "1":
                # The server is ready to receive the file content
                print("Server is ready to receive file content")

                # Send the file's contents
                print("Sending file contents")
                with open(file_path, "rb") as file:
                    while True:
                        data = file.read(1024)  # Adjust the buffer size as needed
                        if not data:
                            break
                        client_socket.send(data)
                        print(f"Sent {len(data)} bytes of data")

                print(f"File '{file_name}' uploaded successfully.")
            else:
                print("Server is not ready to receive the file content. Aborting upload.")
        else:
            print(f"File '{file_name}' does not exist in the current directory.")
    except Exception as e:
        print(f"Error uploading file: {str(e)}")

def synchronize_with_server(client_socket):


    local_files = check_dir()
    client_socket.send(str(local_files).encode('utf-8'))

    files_to_download = eval(client_socket.recv(1024).decode('utf-8'))      ▪ "files_to_download" is not accesse

    # Download missing or updated files from the server

    client_socket.close()
```

```
def check_folder():
    local_files = {}
    for file in os.listdir('.'):
        if os.path.isfile(file):
            timestamp = os.path.getmtime(file)
            local_files[file] = timestamp
    return local_files
def main():
    server_ip = "localhost"
    server_port = 8080
    client_socket = connect_to_server(server_ip, server_port)
    check_dir()
    if client_socket:
        print("You have successfully connected.")


        password_prompt= client_socket.recv(1024).decode('utf-8')

        print(password_prompt)

        password = request_password_from_user()

        client_socket.send(password.encode('utf-8'))

        # Receive the server's response
        response = client_socket.recv(1024).decode('utf-8')
        print(response)  # Print the response from the server

        if response == "Password accepted":
            clear_socket(client_socket)
            while True:

                response = client_socket.recv(1024).decode('utf-8')
                print(response)
                if response == "Waiting for Command: ":
                    # Continue with other actions

                    while True:
                        print("Choose an action:")
                        print("1. List server files")
                        print("2. Request a file")
                        print("3. Upload a file")
                        print("4. Exit")
                        choice = input("Enter the action number: ")

                        if choice == "1":
                            clear_socket(client_socket)
                            client_socket.send("list".encode('utf-8'))
                            file_list = client_socket.recv(1024).decode('utf-8')
                            print("Files on server:")
                            print(' ')
                            print(' ')
                            print(file_list)
                            print(' ')
                            print(' ')
```



```
                while True:
                    print("Choose an action:")
                    print("1. List server files")
                    print("2. Request a file")
                    print("3. Upload a file")
                    print("4. Exit")
                    choice = input("Enter the action number: ")

                    if choice == "1":
                        clear_socket(client_socket)
                        client_socket.send("list".encode('utf-8'))
                        file_list = client_socket.recv(1024).decode('utf-8')
                        print("Files on server:")
                        print(' ')
                        print(' ')
                        print(file_list)
                        print(' ')
                        print(' ')

                    elif choice == "2":
                        clear_socket(client_socket)
                        file_name = input("File Name:")
                        client_download(client_socket, file_name)

                        print (' ')

                    elif choice == '3':
                       file_path = input("Enter the local file path to upload:")
                       upload_file(client_socket, file_path)
                    elif choice =="4":
                        client_socket.close()
                        print("Shutting down.")
                        break
                    elif choice =='Sync':
                        synchronize_with_server(client_socket)
                else:
                    print(response)      ■ Code is unreachable
        else:
            print("Bad password")
            client_socket.close()  # Close the socket to shut down the connection
            print("Shutting down.")
if __name__ == "__main__":
```

Conclusion:

Prior to this program I had to interrupt my ssh window to send scp commands to copy back and forth off of my containers. Now I have created more convenience while working on projects. The next iteration of the project will include automatically uploading new files from the client to the server. Eventually I would like to enable multiple connections at a time if it was ever brought into a situation where multiple people were working on the server like will be occurring for a challenge in early 2024.