**Name: Joy Okolo**
**Course: CSC 574**
**A Readme on how to Run my code**

**Cloud/Network File System Project (TCP and Socket Programming)**

## 1. Introduction

This project implements a cloud/network-based file system using Python socket programming. It is designed to allow a user (client) to connect remotely to a server over the internet and perform basic file system operations such as:

- Uploading files from the client to the server.
- Downloading files from the server to the client.
- Executing file system commands remotely (e.g., mkdir, mv, ls, etc.).

The core concept revolves around TCP sockets, where communication happens in a reliable, ordered, and error-checked manner. No web frameworks were used. The entire communication between client and server strictly follows raw TCP socket principles, keeping it within the expectations of the project.

## 2. Program Architecture and Flow

The project consists of two main Python scripts:

- server.py: A multi-threaded TCP server that accepts multiple clients, processes their requests, handles file uploads and downloads, and executes shell commands remotely.
- client.py: A TCP client that connects to the server and provides an interface for users to interact with the server's file system.

### Server Workflow

1. The server starts and binds to a specific port (5050) using IPv4 and TCP.
2. It listens for incoming client connections.
3. Each client connection is handled in a separate thread, allowing multiple users to interact simultaneously.
4. Upon receiving messages from the client, the server interprets whether the client wants to upload a file, download a file, execute a shell command, or disconnect.
5. Proper message framing is used (64-byte header) to manage data transmission efficiently.

**Client Workflow**

1. The client connects to the server's public IP address on port 5050.
2. It provides a simple command-line interface where the user can:
    - Upload a file using the upload filename command.
    - Download a file using the download filename command.
    - Run remote shell commands like mkdir, mv, ls, and others.
    - Disconnect cleanly by typing quit.

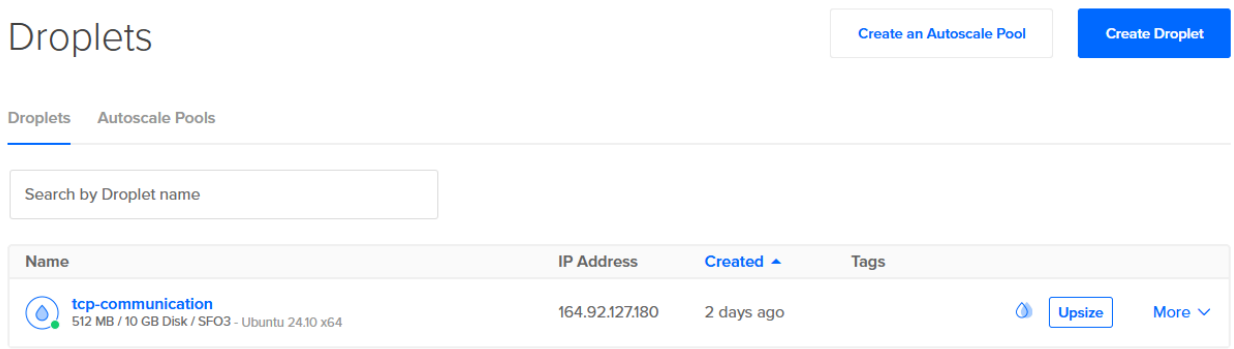
Both programs use a simple yet effective custom protocol to maintain data integrity and session consistency.

## 3. Setting Up the Server (Droplet) and Environment

### 3.1. Signing up for DigitalOcean and Creating a Server (Droplet)

1. I visited https://www.digitalocean.com/ and created a new account.
2. I clicked "Create" > "Droplet" on the dashboard
3. Selected Ubuntu 22.04 LTS as the server OS.
4. Added an SSH Key (generated using PuTTYgen) instead of using a password for more secure authentication.
5. Selected a data center region close to me for better speed.
6. Enabled "Monitoring" but left backups disabled for simplicit

After clicking Create Droplet, my cloud server was ready.




### 3.3. Installing PuTTY and PuTTYgen

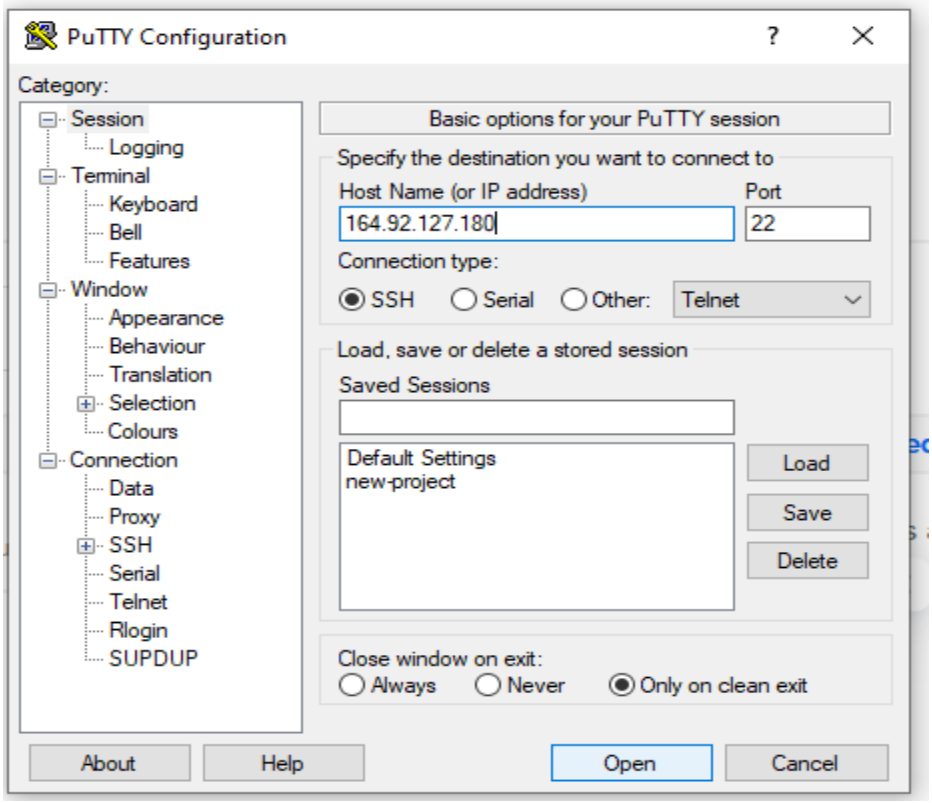
1. Downloaded PuTTY from https://www.putty.org/.
2. Installed both PuTTY (for connecting via SSH) and PuTTYgen (for generating SSH keys).
3. Used PuTTYgen to create a new SSH key pair:
    - Saved the private key as **joy-key.ppk.**
    - Copied the OpenSSH public key and added it to my DigitalOcean droplet during creation.

### 3.4. Connecting to the Droplet via PuTTY

1. Opened PuTTY.
2. Entered my Droplet's public IP address.
3. Loaded the private key file under Connection > SSH > Auth.
4. Connected successfully and logged in as the root user.

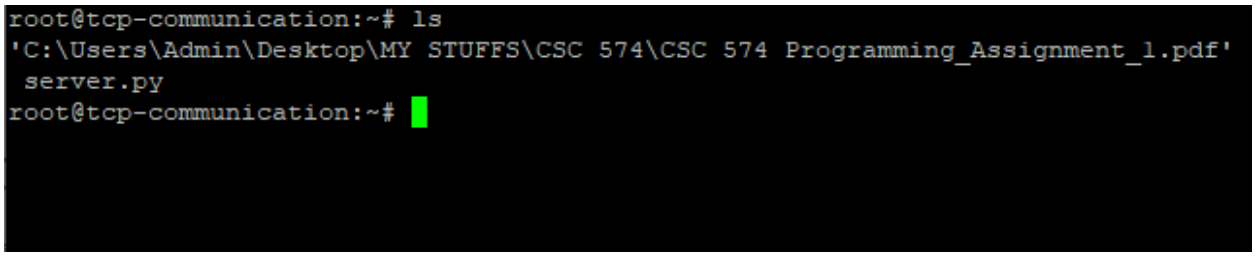
### 4.3. Running the Server

Logged into the server via PuTTY and I logged in as a root user



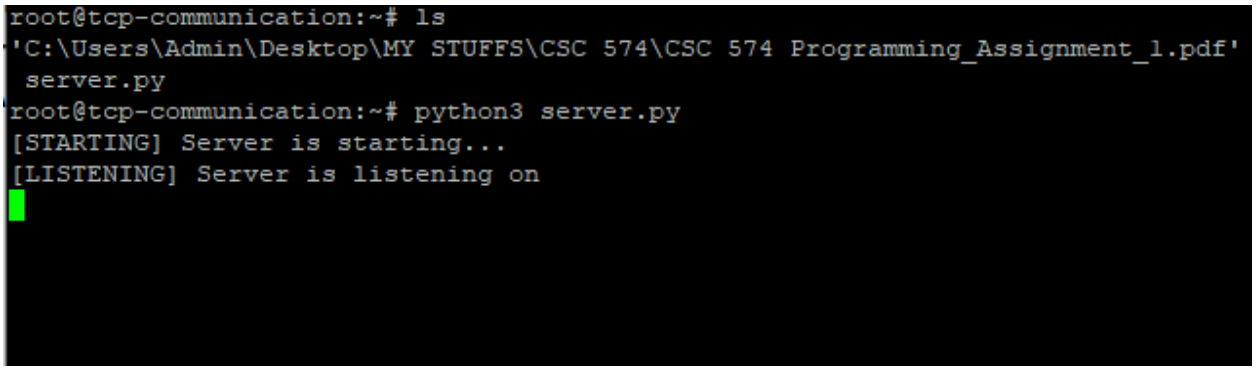

```
login as: root
Authenticating with public key "rsa-key-20250427"
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Tue Apr 29 21:12:34 UTC 2025

  System load:  0.0               Processes:             100
  Usage of /:   24.2% of 8.55GB   Users logged in:       0
  Memory usage: 36%               IPv4 address for eth0: 164.92.127.180
  Swap usage:   0%                IPv4 address for eth0: 10.48.0.5

82 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


*** System restart required ***
Last login: Tue Apr 29 20:39:31 2025 from 137.216.185.69
root@tcp-communication:~#
```

I used the command ls to see the list of files and folders in my server file system and i have only server.py which i used the nano command and pasting my server.py code, then saved. I used this step to generate server.py in my server.




To run my server.py, i used the commande “python3 server.py”

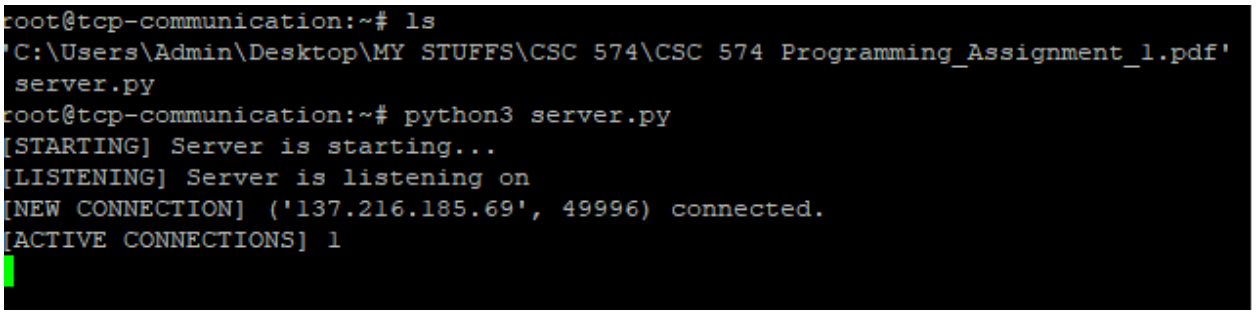
This started the server, making it listen and ready to accept client connections as seen in the screenshot below




### 4.4. Running the Client

On my local machine inside PyCharm's terminal: I ran my client.py code

Once run, I connected successfully to the server and could start sending commands.




## 5. Functional Demonstrations

### 5.1. Uploading Files

- Typed upload C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf

```
>>> upload C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf
[+] File uploaded.
>>> ls
C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf
server.py

>>>
```

- Confirmed that example.txt was transferred to the server.
- 

```
root@tcp-communication:~# ls
'C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf'
 server.py
root@tcp-communication:~# python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on
[NEW CONNECTION] ('137.216.185.69', 49996) connected.
[ACTIVE CONNECTIONS] 1
[+] Received file: C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_
Assignment_1.pdf
```

### 5.2. Downloading Files

- Typed download CSC 574\CSC 574 Programming_Assignment_1.pdf
- 

```
>>> download C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf
[+] File downloaded.
>>>
```

- Successfully pulled CSC 574\CSC 574 Programming_Assignment_1.pdf from the server to my local machine.

### 5.3. Remote Command Execution

- Created directories: mkdir newfolder

```
>>> ls
C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf
server.py

>>> mkdir newfolder
[+] Command executed.

>>> mkdir testfolder
[+] Command executed.
```

```
root@tcp-communication:~# ls
'C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf'
 newfolder
 server.py
 testfolder
root@tcp-communication:~#
```

- Moved files: mv "C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf" newfolder/

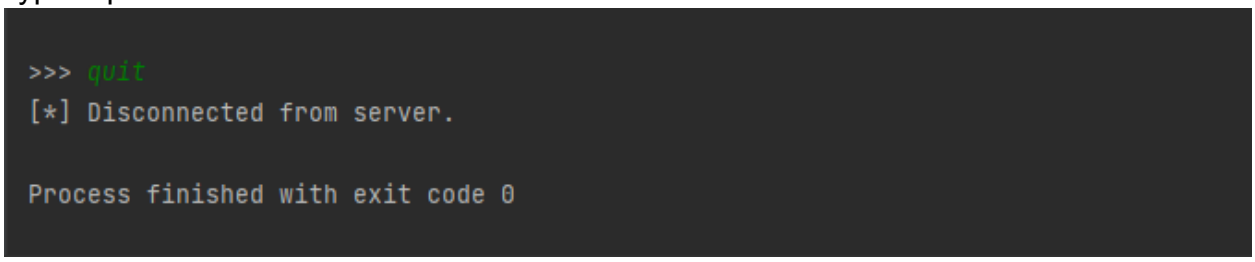
```
>>> mv "C:\Users\Admin\Desktop\MY STUFFS\CSC 574\CSC 574 Programming_Assignment_1.pdf" newfolder/
[+] Command executed.

>>> ls
newfolder
server.py
testfolder
```

- Listed contents: ls
- Removed folders using rm -r testfolder.

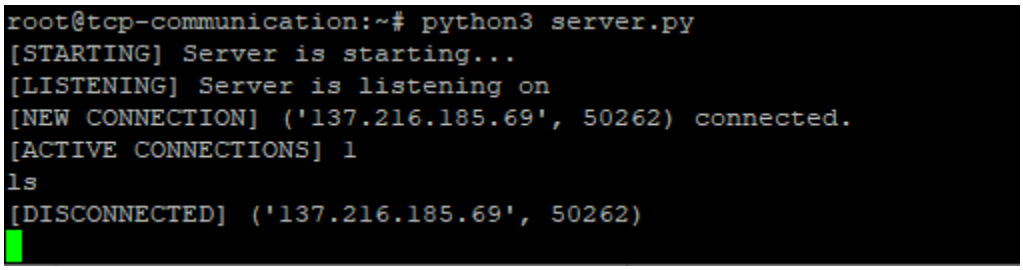
```
rm -r testfolder
[+] Command executed.

>>> [+] Command executed.

>>> ls
newfolder
server.py

>>>
```

Every operation reflected immediately both on the server and client ends.

### 5.4. Handling Disconnections

- Typed quit.
- 

- The client disconnected cleanly.




- The server removed the client thread properly

### 5.5 Connecting Multiple Clients

Each user who wants to connect must:

1. Have a copy of my client.py code.
2. Run the script: client.py

As long as the server is running and port 5050 is open, each client will connect and operate independently.

### 5.6. Server Behavior

Each client that connects is assigned a separate thread by the server. The server prints a message like:

[NEW CONNECTION] ('IP_ADDRESS', PORT) connected.

[ACTIVE CONNECTIONS] X

This confirms that each client is handled separately and concurrently.

## 6. Conclusion and Key Takeaways

This project gave me hands-on experience in:

- Setting up a cloud server manually
- Writing TCP socket programs in Python
- Handling message framing (64-byte header), file I/O operations, and threading.
- Designing simple network protocols (upload/download/end markers).
- Automating and securing remote file system operations.

By walking through each part myself from server setup to client interaction I deepened my understanding of how real-world cloud file systems operate under the hood.

Overall, the assignment challenged me in both networking concepts and software engineering skills, and the solution reflects a practical, working mini-cloud file system entirely built using Python's standard libraries.