Technical Report: TCP-Based/UDP-Based File Transfer System

1. Introduction

The purpose of this project is to design and implement a TCP-based, and UDP-based, file transfer system that allows simple file sharing between a client and a server. The system uses Python's socket module that allows us to transfer files and supports functionalities such as multiple file transfers, connection persistence, and logging transfers. This report outlines the project, implementation, and challenges.

The system aims to provide an efficient method for file transfer within a local or controlled network environment. Key use cases include:

File sharing between devices in an organization. Secure file storage and retrieval with a server. Logging and tracking file transfer activities for security purposes.

The project follows a client-server model:

Client: Initiates file transfers and interacts with the server using CMD prompt. Server: Receives files, stores them in a predefined directory, and logs transfer events for monitoring and debugging.

The communication is established using Transmission Control Protocol (TCP), which ensures reliable data delivery and error checking.

1. Project Overview

Server (server\_tcp.py) features include connection handling with persistent connections using multithreading, allowing multiple clients to interact with the server simultaneously. Files are saved in a dedicated transfer directory. Significant events such as new connections, file transfers, and errors are logged in a serverlog.txt file.

Key functions in the server include handle\_client(conn, addr), which processes requests from a connected client and ensures proper cleanup of resources after connection closure. The open\_server(HOST, PORT) function initializes the server socket, listens for incoming connections, and manages client handling by delegating to a new thread for each connection.

Challenges addressed include ensuring thread safety and resource cleanup, dynamically creating the transfer directory if it is absent, and implementing connection persistence to allow clients to send multiple files without reconnecting.

Client (client\_tcp.py) features include prompting the user for file paths, reading files, and transferring them to the server. The client maintains a live connection with the server until the user chooses to terminate it and provides error handling to notify the user if a file is missing or if the transfer fails.

Key functions in the client include connect\_server(HOST, PORT), which establishes a connection to the server and facilitates file transfers, and terminates the session when the user inputs 'quit()'. Challenges addressed include managing file not found errors gracefully and ensuring data integrity during file transfer with chunk-based sending and a termination signal (FILE\_TRANSFER\_COMPLETE).

1. Results and Testing

Functional testing confirmed successful individual file transfers, the ability to send multiple files in a single session without disconnecting, and persistent connections until the client requests closure. Load testing simulated concurrent connections from multiple clients to test the server’s multithreading capability. The server handled up to 5 simultaneous connections without any issues and logged each connection and transfer event correctly.

1. Challenges and Solutions

Handling multiple clients was addressed by introducing multithreading on the server side, ensuring simultaneous processing of requests. Directory management challenges were solved by automatically creating the transfer directory if it did not exist, preventing file-saving errors. Persistent connections were implemented by modifying the handle\_client function to use an infinite loop that listens for client commands and transfers files until the client explicitly disconnects.

1. Conclusion

The TCP-based file transfer system successfully demonstrates reliable file sharing in a client-server architecture. By incorporating multithreading, persistent connections, and logging, the project meets essential requirements for practical usage. With further enhancements, the system can be extended for use in real-world scenarios requiring secure and scalable file transfer solutions.

6. READ ME file and first time run images

READ ME:

**# How to use**

download the file and open your CMD prompt application.

open multiple instances of the CMD

run this for server:

```python3 server\_tcp.py ```

run this for client

```python3 client\_tcp.py ```

you can run multiple cmd windows for multiple clients. this simulates a multi-client environment which is a requirement of the project.

in the client\_tcp.py file you can add the path to the file you want to send and press enter, then the file is sent to the repository's "transfer" folder.

example run client:

C:\Users\User\Desktop\tcp\_sockets\_to\_transfer\_files-main>python3 client\_tcp.py\_tcp.py

File Path: C:\Users\User\Desktop\test1.txt

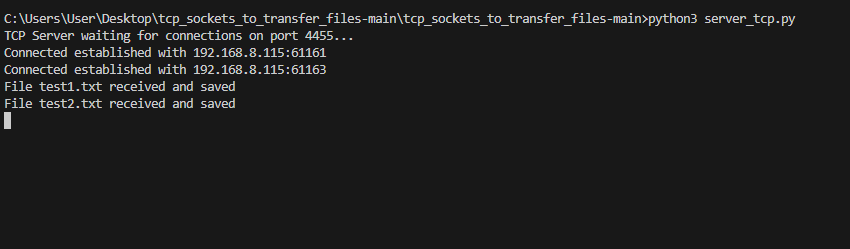
File Path: C:\Users\User\Desktop\test2.txt

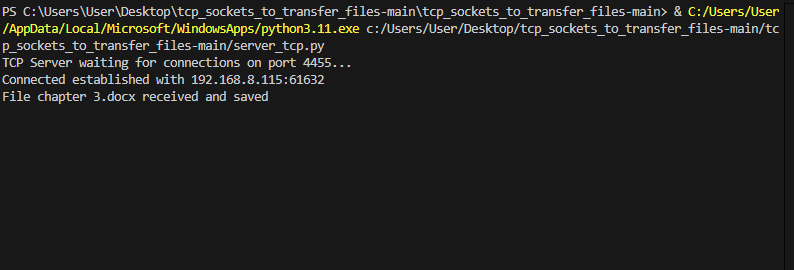
File Path: quit()

finally you can kill the connection with the quit() command.

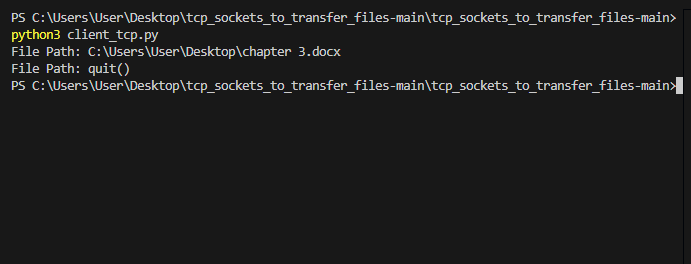
These are images of the code running through vscode terminal:

server:





Client:



File uploaded:

