File Transfer Using TCP

Aryan Kamulugari
Department of Computer Science
and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, Kerala, India
amenu4aie20052@am.students.amrita.edu

Mamidipelly Srinivas
Department of Computer Science
and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, Kerala, India
amenu4aie20047@am.students.amrita.edu

Natte Sai Bharath
Department of Computer Science
and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, Kerala, India
amenu4aie20052@am.students.amrita.edu

Abstract: ***- we will learn how to do file transfers using TCP socket in python3 editing language. This is the most basic file transfer program we can do using client-server architecture. Here, we will create a client and server program file, where the client reads the data from the text file and sends it to the server. The server retrieves and stores the data in another text file.***
TCP stands for Transmission Control Protocol. It is a communication protocol designed to transfer data from end to end via a network. TCP is actually a "standard" communication protocol for data transfer over the Internet.
It is a very efficient and reliable communication method as it uses three-way handshake to connect a client with a server. It is a process that requires both the client and the server to switch sync (SYN) and approve (ACK) packets before data transfer can take place.

Introduction: -

In a network, a socket is a communications connection point (endpoint) that may be named and addressed. Socket programming demonstrates how to create communication channels between remote and local processes using socket APIs.

Socket-using processes might be on the same system or on other systems on multiple networks. Sockets are used in both single-user and network applications. Sockets make it possible to share data across processes on the same system or across a network, transfer work to the most efficient machine, and get quick access to centralised data. TCP/network IP's standard is socket application programme interfaces (APIs). Socket APIs are supported by a wide range of operating systems. Multiple transport and networking protocols are supported by i5/OS sockets. Thread safety is provided via socket system and socket network features.

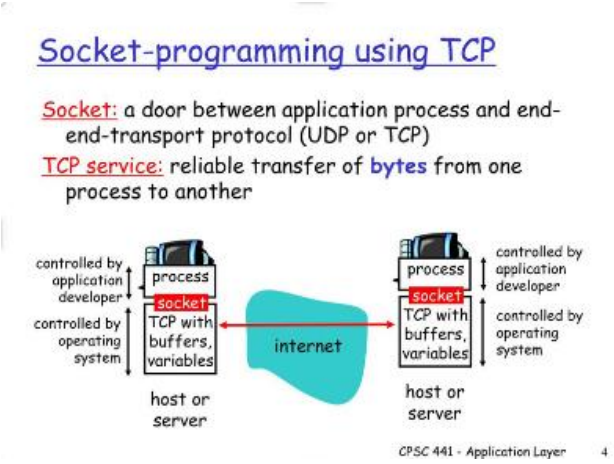
THE WORKING OF SOCKETS

Sockets are widely used for client-server communication. The server is often located on one machine, while the clients are located on different workstations. The clients establish a connection with the server, exchange data, and then disconnect.

The flow of events in a socket is normal. The socket on the server process in a connection-oriented client-to-server architecture listens for requests from a client. To do this, the server first creates (binds) an address that clients may use to locate it. The server then waits for customers to request a service after the address has been established. When a client connects to a server through a socket, data is exchanged between the client and the server. The server processes the client's request and responds to the client.

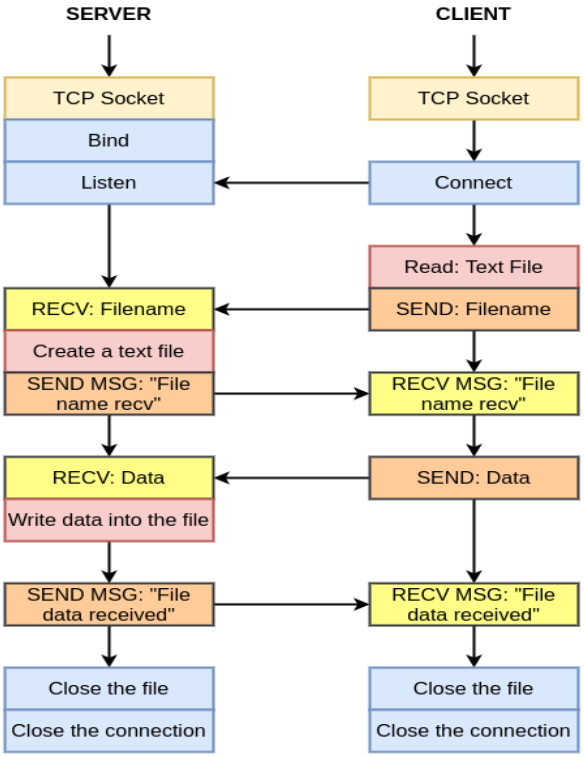
1. The socket() API creates an endpoint for communications and returns a socket descriptor that represents the endpoint.

2. When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
3. The listen() API indicates a willingness to accept client connection requests. When a listen() API is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() API and the bind() API binds a name to the socket. A listen() API must be issued before an accept() API is issued.
4. The client application uses a connect() API on a stream socket to establish a connection to the server.
5. The server application uses the accept() API to accept a client connection request. The server must issue the bind() and listen() APIs successfully before it can issue an accept() API.
6. When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer APIs. Clients and servers have many data transfer APIs from which to choose, such as send(), recv(), read(), write(), and others.
7. When a server or client wants to stop operations, it issues a close() API to release any system resources acquired by the socket.



FILE TRANSFER: -

This is the most basic file transfer program that we can do using a client-server architecture. Here, we are going to do the build a client and a server program file, where the client read the data from a text file and send it to the server. The server receives it.



Project Structure

The project is divided into two files:

1. server.py
2. client.py

The client.py contains the code, used to read a text file and send its data to the server. While the server.py file receives the data sent by the client and save it to a text file.

LARGE FILE TRANSFER: -

Here we are going to built a client server program where the client is going to transfer a large text file. We have to import tqdm to check the progress of our exported data. We have to encode and then decode as we are dealing with the txt file. For this we use "utf-8" format. The bigger the txt file is the more time it will take.

File Transfer: **SERVER**
The server performs the following functions:

1. Create a TCP socket.
2. Bind the IP address and PORT to the server socket.
3. Listening for the clients.
4. Accept the connection from the client.
5. Receive the filename from the client and create a text file.
6. Send a response back to the client.
7. Receive the text data from the client.
8. Write (save) the data into the text file.
9. Send a response message back to the client.
10. Close the text file.
11. Close the connection.

File Transfer: **CLIENT**
The client performs the following functions:

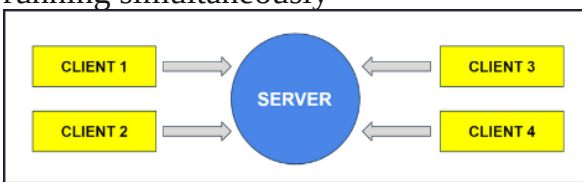
1. Create a TCP socket for the client.
2. Connect to the server.
3. Read the data from the text file.
4. Send the filename to the server.
5. Receive the response from the server.
6. Send the text file data to the server.
7. Receive the response from the server.
8. Close the file.
9. Close the connection.

Multithreaded File Transfer

Multithreaded File Transfer using TCP Socket in Python

The creation of a multi-threaded file-server file transfer system using python editing language. The server is capable of handling multiple clients simultaneously through the cable. The server assigns each client a series of hosting to work for that client.
A thread is a light-weight process that does not require much memory overhead, they are cheaper than processes. Multithreading is a process of executing multiple threads simultaneously in a single process. Multithreading is a technique where multiple threads are spawned by a process to do different tasks, at about the same time, just one after the other. This gives you the illusion that the threads are running in parallel, but they are actually run in a concurrent manner. In Python, the Global Interpreter Lock (GIL) prevents the threads from

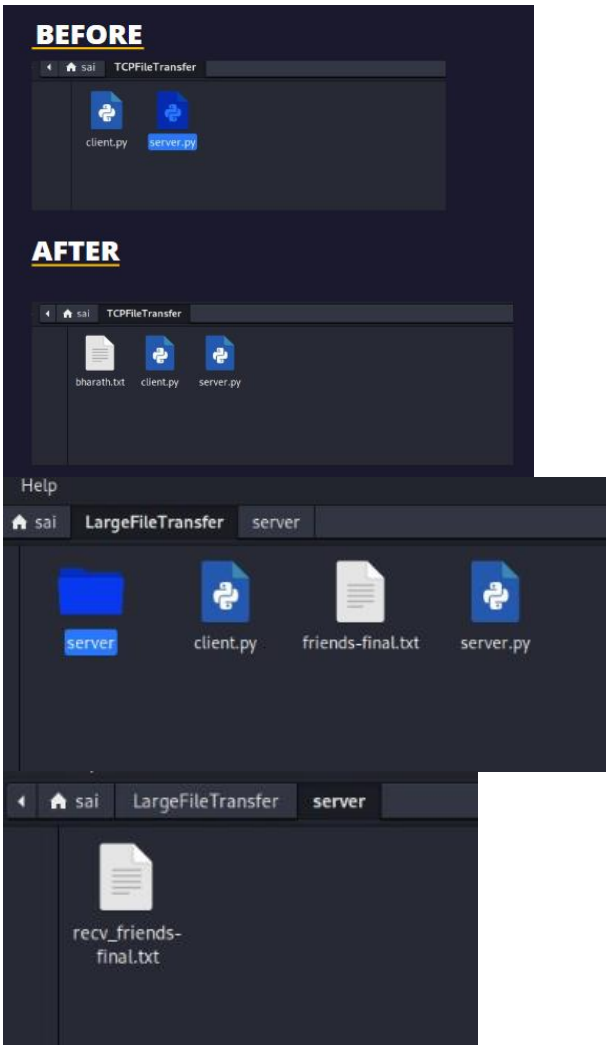
running simultaneously

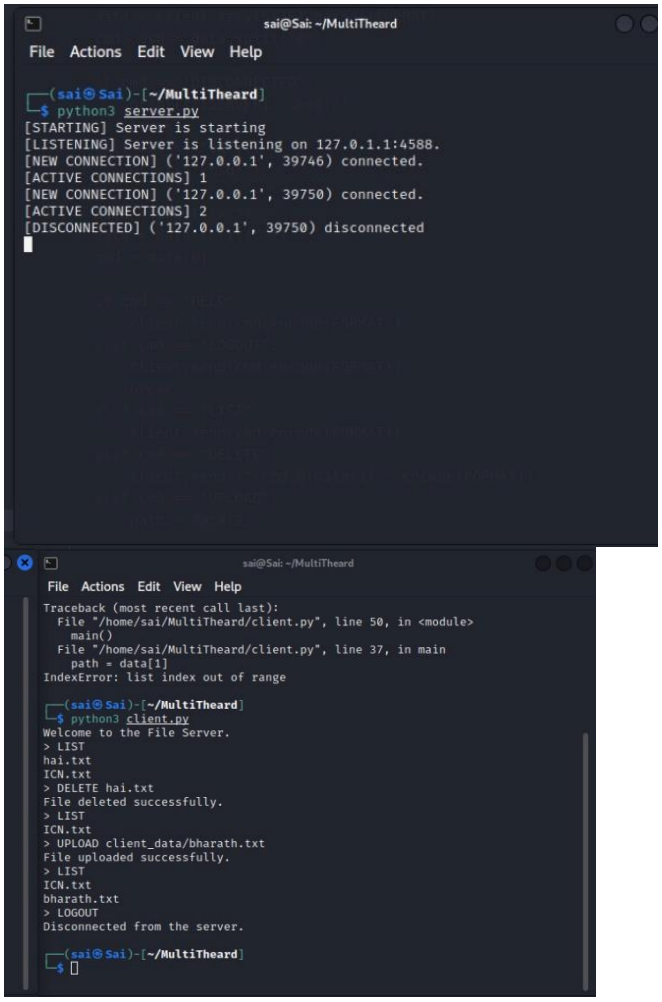


The server supports the following functions:
LIST: List all files that appear on the server.
UPLOAD method: Upload file to server
DELETE file name: Delete file from server
LOGOUT: Disconnect from server
HELP: List all the instructions

Multithreaded File Transfer

RESULT AND ANALYSIS





CONCLUSION -

These three processes are highly efficient and reliable communication protocol as it uses a three-way handshake to connect the client and the server. It is a process that requires both the client and the server to exchange synchronization (SYN), TCP protocol running on the receiving host. The receiving TCP returns a segment called an ACK to acknowledge the successful receipt of the segment. The sending TCP sends another ACK segment, then proceeds to send the data.