Server Program:

```
import socket

HOST = 'localhost'  # Server IP address
PORT = 12345  # Server port number

def receive_file():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((HOST, PORT))
        server_socket.listen(1)
        print('Waiting for connection...')
        client_socket, addr = server_socket.accept()
        print('Connected to:', addr)

        with open('received_file', 'wb') as file:
            while True:
                data = client_socket.recv(1024)
                if not data:
                    break
                file.write(data)

        print('File received successfully.')

receive_file()
```

Client Program:

```
import socket

HOST = 'localhost'  # Server IP address
PORT = 12345  # Server port number
FILE_PATH = 'file_to_send.txt'  # Path of the file to be sent

def send_file():
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
client_socket.connect((HOST, PORT))

with open(FILE_PATH, 'rb') as file:
for data in file:
client_socket.sendall(data)

print('File sent successfully.')

send_file()
```

Make sure to change the HOST, PORT, and FILE_PATH variables according to your specific setup. The server will save the received file as 'received_file' in the same directory.

Remember to handle exceptions, perform error checking, and add additional features as needed.

Explaination for code: Server side

Import the socket module: This module provides the necessary functions and classes for creating and interacting with sockets.

Define the HOST and PORT variables: These variables specify the server's IP address (in this case, 'localhost' refers to the current machine) and the port number on which the server will listen for incoming connections.

Define the receive_file() function: This function handles the server-side file receiving logic.

Create a socket object: Using socket.socket(), we create a TCP socket for the server. The AF_INET parameter indicates that we are using IPv4 addressing, and SOCK_STREAM specifies that we want to use TCP

Bind the socket to the server address: Using server_socket.bind((HOST, PORT)), we associate the socket with the server address and port number. This step is necessary before listening for incoming connections.

Start listening for connections: Using server_socket.listen(1), the server socket begins listening for incoming connections. The argument '1' specifies the maximum number of queued connections.

Accept incoming connection: When a client attempts to connect, server_socket.accept() blocks the execution until a client successfully establishes a connection. It returns a new socket object (client_socket) representing the connection and the address (addr) of the client.

Open a file to save the received data: Using open('received_file', 'wb'), we open a new file in binary write mode with the name 'received_file'. This file will store the received data.

Receive and write data to the file: Inside the while loop, we continuously receive data from the client using client_socket.recv(1024). The received data is stored in the data variable. If no data is received (empty data), the loop breaks.

Write data to the file: We write the received data to the file using file.write(data).

Print a success message: Once the loop completes, we print a success message indicating that the file was received successfully.

Call the receive_file() function: At the end of the code, we call the receive_file() function to start the server and wait for a client to connect.

For client:

The provided code is the client-side portion of the file transfer program using TCP in Python. Let's break down the program:

Import the socket module: This module provides the necessary functions and classes for creating and interacting with sockets.

Define the HOST, PORT, and FILE_PATH variables: These variables specify the server's IP address (in this case, 'localhost' refers to the current machine), the server port number, and the path of the file to be sent.

Define the send_file() function: This function handles the client-side file sending logic.

Create a socket object: Using socket.socket(socket.AF_INET, socket.SOCK_STREAM), we create a TCP socket for the client. The AF_INET parameter indicates that we are using IPv4 addressing, and SOCK_STREAM specifies that we want to use TCP.

Connect to the server: Using client_socket.connect((HOST, PORT)), the client socket attempts to establish a connection with the server specified by the HOST and PORT variables.

Open the file to be sent: Using open(FILE_PATH, 'rb'), we open the file specified by the FILE_PATH variable in binary read mode.

Iterate over the file data: Using a for loop, we iterate over the file data line by line.

Send data to the server: Inside the loop, we use client_socket.sendall(data) to send each line of data to the server.

Print a success message: Once the loop completes, we print a success message indicating that the file was sent successfully.

Call the send_file() function: At the end of the code, we call the send_file() function to initiate the file transfer process.

Make sure to have the specified file available in the correct location before running the code. Additionally, ensure that the server is running and configured with the correct HOST and PORT