

University of Science and Technology of Ha Noi



Practical Labwork 1

Distributed System - Le Nhu Chu Hiep

Report Labwork

Vu Trong Bach - 22BI13056

November 2024

Mục lục

1	Introduction	3
1.1	TCP and File Transfer over Sockets	3
2	Protocol Design	3
2.1	Client-Server Interaction Diagram	3
2.2	Logic of the Protocol	4
3	System Organization	4
3.1	System Architecture	4
3.2	Architecture Diagram	4
4	Implementation	4
4.1	Key Code Snippets	4
4.1.1	Server Code	4
4.1.2	Client Code	6
5	Responsibility Distribution	7
6	Conclusion	9

1 Introduction

The practical work aimed to implementation a 1-to-1 file transfer using TCP/IP via sockets. This will help us understand how data communication works at the transport layer using the TCP protocol.

1.1 TCP and File Transfer over Sockets

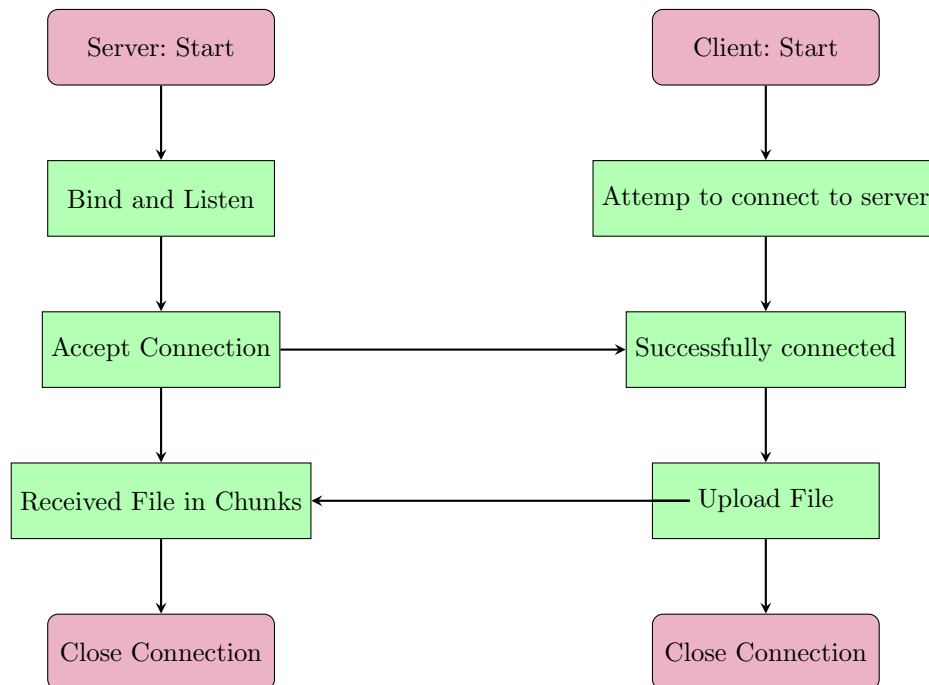
TCP-Transmission Control Protocol-guarantees reliable, ordered, and error-checked delivery of data. File transfer over sockets involves the following:

- Establish a connection between the client and the server.
- Data transfer from one machine to another.
- Properly handling connection closure and EOF (End of File).

2 Protocol Design

2.1 Client-Server Interaction Diagram

Below is the interaction diagram showing how the client and server communicate:



Hình 1: Client-Server Interaction for TCP File Transfer

2.2 Logic of the Protocol

1. The server sets up by binding to an IP address and port, then waits for connections from clients.
2. The client starts the process by connecting to the server.
3. Once connected, the client asks for a file. After we typing in the file that we want to transfer, the program will automatically transfer the file to the server.
4. Once the server received the file, it will store the file in a folder.
5. After successfully transfer, both the client and server will close the connection.

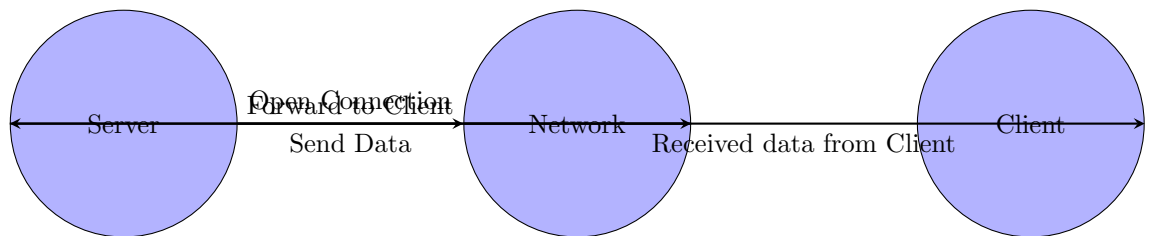
3 System Organization

3.1 System Architecture

The system consists of two main components: the client and the server. Their roles are described in the following:

- **Server:** Handles incoming connections and sends requested files to clients.
- **Client:** Connects to the server and upload files.

3.2 Architecture Diagram



Hình 2: System Architecture for TCP File Transfer

4 Implementation

4.1 Key Code Snippets

4.1.1 Server Code

The server code is responsible for listening to connections and sending files:

Listing 1: Server Code

```
# server.py
import socket
import os

# Server configuration
HOST = '172.27.242.41' # Replace with your server's IP address
PORT = 8386 # Port for communication
SAVE_DIR = "uploaded_files" # Directory to save received files

def start_server():
    # Ensure the save directory exists
    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)

    # Create a TCP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((HOST, PORT)) # Bind to the server IP and port
    server_socket.listen(1) # Allow one client connection at a time
    print(f"Server listening on {HOST}:{PORT}...")

    while True:
        # Accept a connection from a client
        conn, addr = server_socket.accept()
        print(f"Connection established with {addr}")

        try:
            # Receive the filename first, up to the newline character
            filename = b""
            while True:
                byte = conn.recv(1)
                if byte == b'\n': # Delimiter indicates the end of the filename
                    break
                filename += byte
            filename = filename.decode('utf-8') # Decode bytes to string
            print(f"Receiving file: {filename}")

            # Define full path to save the file
            save_path = os.path.join(SAVE_DIR, filename)

            # Open the file to save the incoming data
            with open(save_path, 'wb') as file:
                while True:
                    data = conn.recv(1024) # Receive file data in chunks (1KB)
                    if not data: # Condition to check if the file transfer is c
```

```

        break
    file.write(data)

    print(f"File_{filename}_received_and_saved_in_{SAVE_DIR}/'")
except Exception as e:
    print(f"Error_receiving_file:{e}")
finally:
    conn.close() # Close the client connection
    print(f"Connection_with_{addr}_closed.")

if __name__ == "__main__":
    start_server()

```

4.1.2 Client Code

The client code connects to the server and receives the file:

Listing 2: Client Code

```

# client.py
import socket
import os

# Server configuration
SERVER_IP = input("Enter_the_server's_IP_address:_") # Enter Server's IP address
PORT = 8386 # Server's port

def send_file():
    try:
        # Create a TCP socket
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print(f"Attempting_to_connect_to_{SERVER_IP}:{PORT}...")
        client_socket.connect((SERVER_IP, PORT))
        print(f"Connected_to_server_at_{SERVER_IP}:{PORT}")

        # Get the filename and ensure it exists
        while True:
            filename = input("Enter_the_full_file_name_(with_extension):_")
            if os.path.isfile(filename): # Ensure the file exists
                break
            print("File_not_found._Please_try_again.")

        # Extract just the file name for sending (without the directory path)
        base_filename = os.path.basename(filename)

        # Send the filename to the server

```

```

        client_socket.send(base_filename.encode('utf-8') + b'\n')
# Send filename with newline delimiter

        # Open the file and send its content in chunks
        with open(filename, 'rb') as file:
            print(f"Sending_file_{base_filename}_to_the_server...")
            while chunk := file.read(1024): # Read in chunks of 1KB
                client_socket.send(chunk)

        print(f"File_{base_filename}_sent_successfully_to_the_server.")
        client_socket.close()

    except ConnectionRefusedError:
        print("Connection_failed:_Ensure_the_server_is_running_and_reachable.")
    except socket.timeout:
        print("Connection_timed_out:_Server_is_taking_too_long_to_respond.")
    except Exception as e:
        print(f"An_unexpected_error_occurred:{e}")

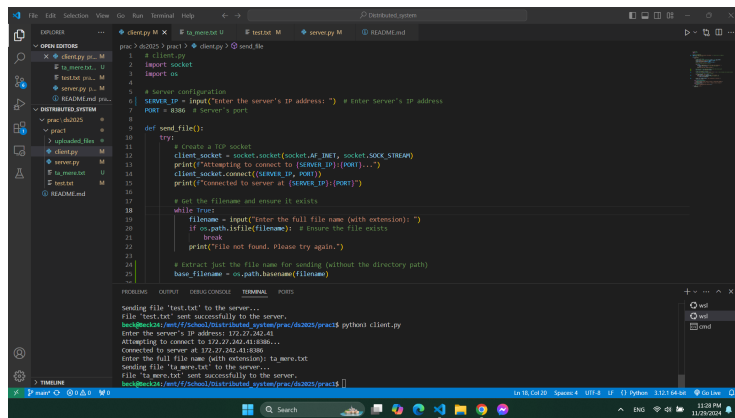
if __name__ == "__main__":
    send_file()

```

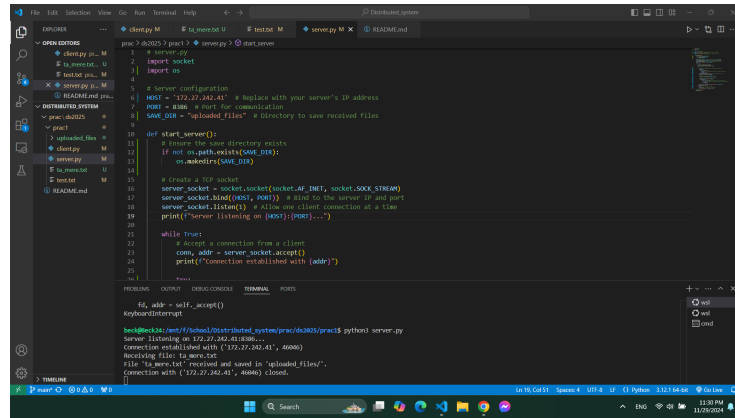
5 Responsibility Distribution

The project tasks were distributed among the members of the group as follows:

- **Huy:** Run the client file to connect to my server.
- **Bach(me):** Send file "test" from server to client.
- **Bach(also me):** Designed the protocol and documented the report.



Hình 3: Client connects to server and sends file



```
1 # server.py
2 import socket
3 import os
4
5 # server configuration
6 HOST = '172.27.242.41' # Replace with your server's IP address
7 PORT = 8080 # Port for communication
8 SAVE_DIR = "uploaded_files" # Directory to save received files
9
10 def start_server():
11     # Ensure the save directory exists
12     if not os.path.exists(SAVE_DIR):
13         os.makedirs(SAVE_DIR)
14
15     # Create a TCP socket
16     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17     server_socket.bind((HOST, PORT)) # Bind to the server IP and port
18     server_socket.listen(1) # Allow one client connection at a time
19     print(f"Server listening on {HOST}:{PORT}...")
20
21     while True:
22         # Accept a connection from a client
23         (conn, addr) = server_socket.accept()
24         print(f"Connection established with {addr}")
25
26         # Receive data from the client
27         data = conn.recv(1024)
28         if not data:
29             break
30
31         # Save the received data to a file
32         filename = os.path.join(SAVE_DIR, f"file_{addr}.txt")
33         with open(filename, "wb") as f:
34             f.write(data)
35
36         print(f"File '{filename}' received and saved in '{SAVE_DIR}'")
37         conn.close()
```

```
python3 server.py
Server listening on 172.27.242.41:8080...
Connection established with ('172.27.242.41', 40000)
Receiving file to server-test
File 'ta_server.txt' received and saved in 'uploaded_files/'.
Connection with ('172.27.242.41', 40000) closed.
```

Hình 4: Server connects to client and receives file

6 Conclusion

This labwork offered practical insights into TCP/IP and socket programming, demonstrating how file transfer can be implemented in a client-server architecture. During the implementation, challenges such as managing EOF signals and handling connection errors were encountered and effectively addressed using robust exception handling and thorough testing. Future enhancements could include implementing encryption to ensure secure file transfers and extending the system to handle multiple clients simultaneously, improving scalability and security.