

University of Science and Technology of Ha Noi



Practical Labwork 1

Distributed System - Le Nhu Chu Hiep

Report Labwork

Tran Trung Hieu - 22BI13162

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	8

1 Introduction

The following practical work is aimed at the implementation of 1-to-1 file transfer using TCP/IP via sockets. This task helps to understand how data communication works at the transport layer using the TCP protocol.

1.1 TCP and File Transfer over Sockets

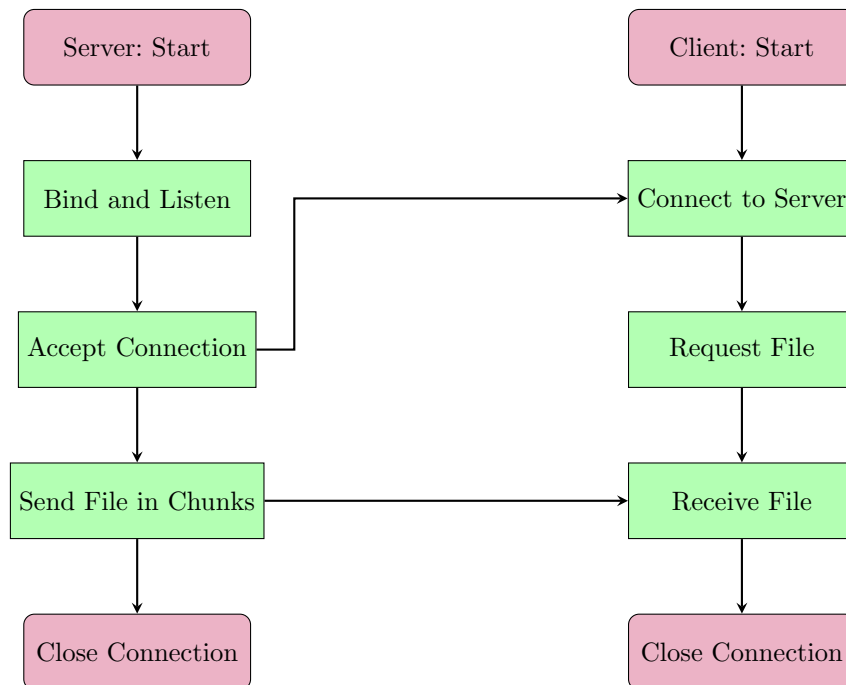
TCP-Transmission Control Protocol-guarantees reliable, ordered, and error-checked delivery of data. File transfer over sockets involves the following:

- 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, and the server responds by sending the requested data.
4. After the file is fully transferred, both the client and server close the connection properly.

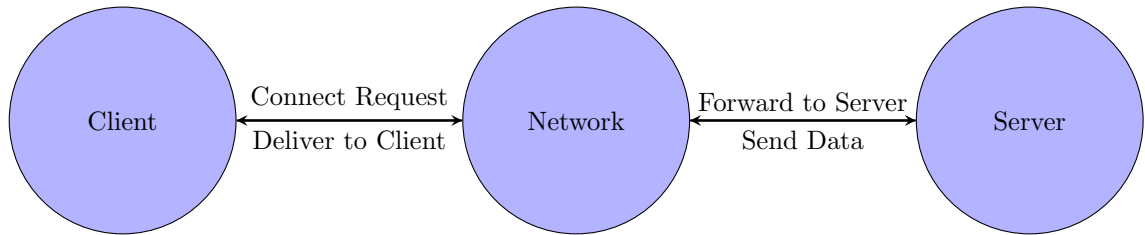
3 System Organization

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 downloads 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 = '192.168.157.167' # Replace with your server's IP address
PORT = 8386 # Port for communication

def start_server():
    # 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}")

        # 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}")

        # Open the file with the received filename to save the incoming data
        with open(filename, 'wb') as file:
            while True:
                data = conn.recv(1024) # Receive file data in chunks (1KB)
                if not data: # No more data means the file transfer is complete
                    break
                file.write(data)

        print(f"File '{filename}' received and saved in the server folder.")
        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:_") # 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.")

        # Send the filename to the server
        client_socket.send(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_{filename}_to_the_server...")
            while chunk := file.read(1024): # Read in chunks of 1KB
                client_socket.send(chunk)

        print(f"File_{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:

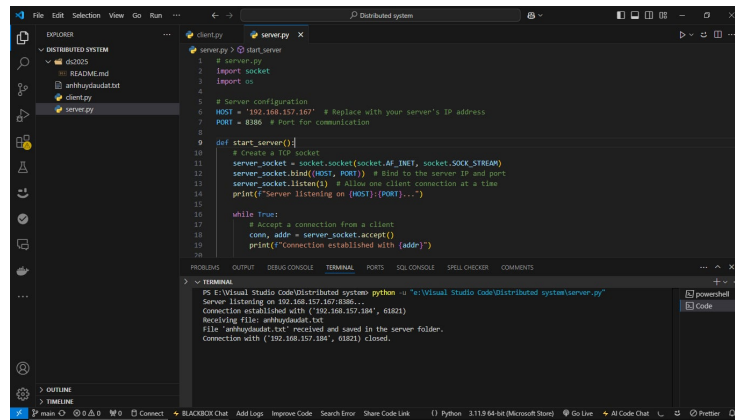
- **Ducnm:** Run the client file to connect to my server.
- **Hieutt(me):** Send file "huydaudat" from server to client.
- **Hieutt(me):** Designed the protocol and documented the report.

```

C:\Users\user> python client.py
Enter the server's IP address: 192.168.1.102
Attempting to connect to 192.168.1.102...
Connected to server at 192.168.1.102:8080
Enter the full file name (with extension): vishnuvrat
File not found. Please try again.
Enter the full file name (with extension): vishnuvrat.txt
Enter the full file name (with extension): vishnuvrat.txt
Sending file 'vishnuvrat.txt' to the server...
File 'vishnuvrat.txt' sent successfully to the server.

```

Hình 3: Client connects to server and sends file



The screenshot shows a Visual Studio Code window with a file explorer on the left containing 'distributed_system', 'client.py', and 'server.py'. The main editor displays the 'server.py' file with the following code:

```
1 # server.py
2 import socket
3 import os
4
5 # Server configuration
6 HOST = "192.168.157.167" # Replace with your server's IP address
7 PORT = 8388 # Port for communication
8
9 def start_server():
10     # Create a TCP socket
11     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     server_socket.bind((HOST, PORT)) # Bind to the server IP and port
13     server_socket.listen(1) # Allow one client connection at a time
14     print(f"Server listening on {HOST}:{PORT}...")
15
16     while True:
17         # Accept a connection from a client
18         conn, addr = server_socket.accept()
19         print(f"Connection established with {addr}")
20
21         # Receive file data
22         data = conn.recv(1024)
23         while data:
24             # Save the file
25             filename = os.path.basename(data.decode())
26             with open(filename, 'wb') as f:
27                 f.write(data)
28             data = conn.recv(1024)
29         conn.close()
30
31 if __name__ == '__main__':
32     start_server()
```

The terminal at the bottom shows the execution of the server script:

```
PS E:\Visual Studio Code\distributed system> python -u "E:\Visual Studio Code\distributed system\server.py"
Server listening on 192.168.157.167:8388...
Connection established with ("192.168.157.164", 61821)
Receiving file: anhhayduat.txt
File "anhayduat.txt" received and saved in the server folder.
Connection with ("192.168.157.164", 61821) 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.