UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

DISTRIBUTED SYSTEMS

PRACTICAL WORK 1 - TCP FILE TRANSFER

HA DINH TUAN
BA12-183
tuanhd.ba12-183@st.usth.edu.vn
CYBER SECURITY

Lecturer: Le Nhu Chu Hiep

Hanoi, November 2024

1 Introduction

The objective of this practical work was to develop a file transfer system using TCP/IP with a client-server architecture. The system is made up of a server and a client that communicate through sockets.

2 Protocol design

2.1 Overview of the protocol

The File Transfer Protocol (FTP) is created to provide a reliable, efficient, and secure method for transferring files between a client and a server over a TCP/IP network. It utilizes two distinct connections: one for control commands (the control connection) and another for transferring file data (the data connection).

2.2 Steps in protocol design

The file transfer protocol is organized as follows:

1. **Connection establishment:** The client connects to the server using a specific IP address and port, while the server listens to and accepts the incoming connection.

2. **File metadata:** The client sends the file name and size to the server, which then acknowledges that it is prepared to receive the file.

3. **File transfer:** The client transmits the file in chunks, and the server receives and writes the data, sending an acknowledgment ("ACK") for each received chunk.

4. **Acknowledgment:** After each chunk is received, the server sends a confirmation back to the client to indicate successful receipt.

5. **Completion:** Once all chunks have been transferred, the server notifies the client that the transfer is complete.

6. **Connection closure:** After the transfer is completed, both the client and the server close the control and data connections.

2.3 Figure design protocol

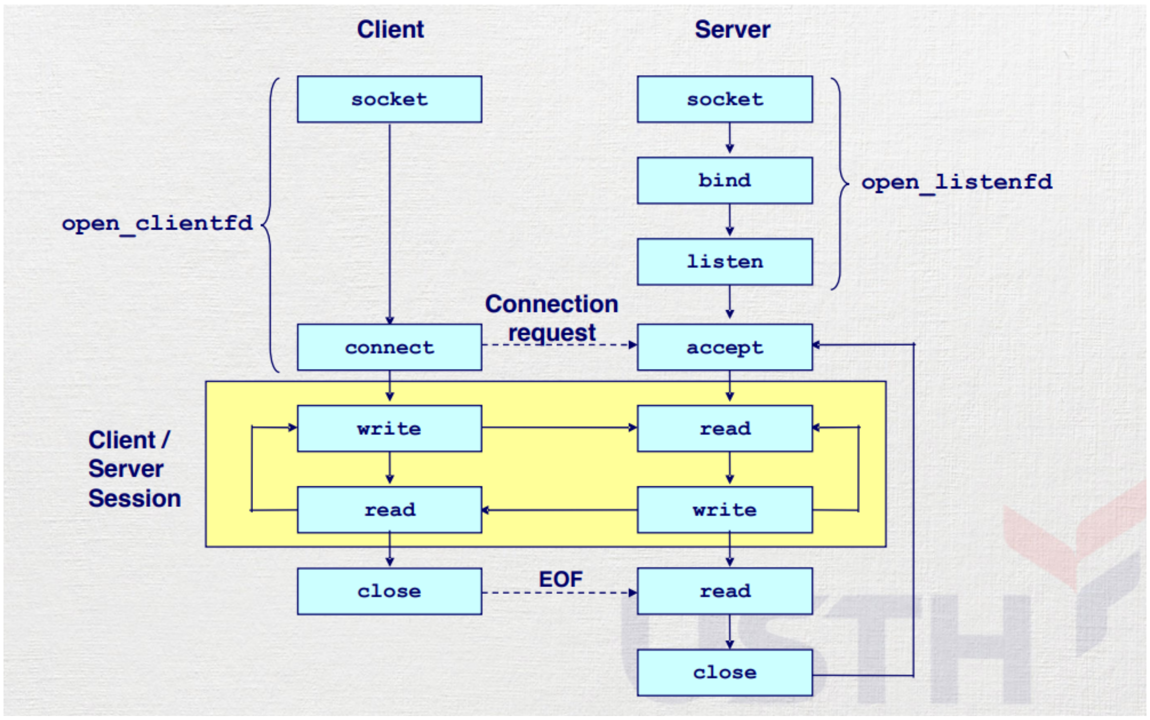


Figure 1: FTP design

3 Systems organization

3.1 System architecture

The system is designed using a "Client-Server" architecture, where the "Client" initiates the connection and sends a file to the "Server." The two primary components of the system are the "Client" and the "Server," which communicate via a TCP connection. Below is a detailed description of the system architecture:

- Client:
  - The client begins the file transfer by establishing a TCP connection with the server.

  - It first sends the file metadata (such as name and size), followed by the file data in chunks.
  - The client waits for an acknowledgment (ACK) from the server after each chunk to ensure the data is transferred reliably.
  - Once the entire file has been successfully sent, the client closes the connection.

- Server:
  - The server listens for incoming connections on a pre-defined port.
  - After the connection is established, it receives the file metadata and prepares to receive the file.
  - The server receives the file data in chunks and writes it to a file.
  - After receiving each chunk, the server sends an acknowledgment (ACK) to the client to confirm successful receipt.
  - When the entire file has been received, the server indicates the completion of the transfer and closes the connection.

4 File transfer implementation

4.1 Client code

This Python program implements a simple client that can upload and download files to/from a server using TCP/IP. The client connects to the server, allows the user to choose between uploading or downloading a file, and handles file transfers in chunks. The client interacts with the server via socket communication and follows a menu-based interface.

```
import socket
import os

def connect_to_server():
    SERVER_IP = input("Enter IP address of server: ")
    PORT = int(input("Enter port of server: "))

    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_S
        client_socket.connect((SERVER_IP, PORT))
        print(f"Connected to server at {SERVER_IP}:{PORT}")

        while True:
            print("\nOptions:")
            print("1. Upload a file")
            print("2. Download a file")
            print("3. Exit")
            choice = input("Enter your choice: ")

            if choice == "1":
                client_socket.send(b"UPLOAD")
                upload_file(client_socket)
            elif choice == "2":
                client_socket.send(b"DOWNLOAD")
                download_file(client_socket)
            elif choice == "3":
                print("Exiting...")
                break
            else:
                print("Invalid choice. Try again.")

        client_socket.close()

    except Exception as e:
        print(f"An error occurred: {e}")
```

```
def upload_file(client_socket):
    filename = input("Enter full filename to upload: ")
    if os.path.isfile(filename):
        client_socket.send(filename.encode())  # send file name to
        with open(filename, 'rb') as f:
            while chunk := f.read(1024):
                client_socket.send(chunk)  # send file data in bloc
        print(f"File '{filename}' uploaded successfully.")
    else:
        print("File not found. Please try again.")

def download_file(client_socket):
    filename = input("Enter full filename to download: ")
    client_socket.send(filename.encode())  # request filename from

    response = client_socket.recv(1024).decode()
    if response == "OK":
        # To avoid overwriting, append a suffix to the downloaded f
        downloaded_filename = f"downloaded_{filename}"
        with open(downloaded_filename, 'wb') as f:
            while True:
                data = client_socket.recv(1024)
                if not data:
                    break
                f.write(data)  # save file received from server
        print(f"File '{filename}' downloaded successfully as '{dow
    else:
        print(f"Error: {response}")

if __name__ == "__main__":
    connect_to_server()
```

4.2 Server code

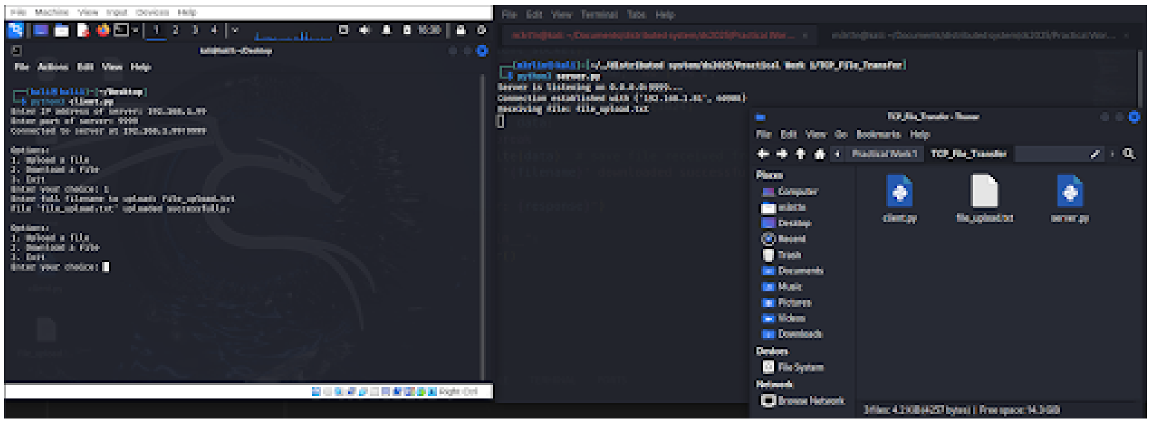
This Python code implements a simple TCP server for file transfer. It listens for incoming connections from clients and handles file uploads and downloads. Let's go through the key sections of the code to explain how it works.

Listing 1: Server code for file transfer using TCP/IP

```
import socket
import os

HOST = '0.0.0.0'  # Listen on all interfaces
PORT = 9999

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM
    server_socket.bind((HOST, PORT))
    server_socket.listen(5)  # Accept max 5 devices
    print(f"Server is listening on {HOST}:{PORT}...")

    while True:
        conn, addr = server_socket.accept()
        print(f"Connection established with {addr}")

        operation = conn.recv(1024).decode()
        if operation == "UPLOAD":
            handle_upload(conn)
        elif operation == "DOWNLOAD":
            handle_download(conn)
        else:
            conn.send(b"ERROR: Invalid operation")

        conn.close()  # Ensure connection is closed after each oper

def handle_upload(conn):
    filename = conn.recv(1024).decode()  # Get filename from client
    print(f"Receiving file: {filename}")
    # Check if file already exists and create a new name if necessa
    if os.path.isfile(filename):
        # If the file exists, create a new name by appending a numb
        base_name, ext = os.path.splitext(filename)
        counter = 1
        new_filename = f"{base_name}_{counter}{ext}"
        while os.path.isfile(new_filename):  # Check if the new nam
            counter += 1
            new_filename = f"{base_name}_{counter}{ext}"
        filename = new_filename
        print(f"File already exists. Renaming to {filename}")

    # Write the received file data directly to the new file
    with open(filename, 'wb') as f:
        while True:
            data = conn.recv(1024)
            if not data:
                break  # If no more data, stop writing
            f.write(data)  # Write data to file
    print(f"File '{filename}' received and saved.")

def handle_download(conn):
    filename = conn.recv(1024).decode()  # Get filename from client
    print(f"Client requested file: {filename}")

    if os.path.isfile(filename):  # Check if file exists
        conn.send(b"OK")
        with open(filename, 'rb') as f:
            while chunk := f.read(1024):  # Read file in chunks
                conn.send(chunk)
        print(f"File '{filename}' sent successfully.")
    else:
        conn.send(b"ERROR: File not found")  # Error if file doesn'
        print(f"File '{filename}' not found")
```

```
if __name__ == "__main__":
    start_server()
```

4.3 Testing code snippet



```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 client.py
Enter IP address of server: 192.168.1.99
Enter port of server: 9999
Connected to server at 192.168.1.99:9999

Options:
1. Upload a file
2. Download a file
3. Exit
Enter your choice: 1
Enter full filename to upload: file_upload.txt
File 'file_upload.txt' uploaded successfully.

Options:
1. Upload a file
2. Download a file
3. Exit
Enter your choice:
```

```
┌──(m3rlin@kali)-[~/…/distributed system/ds2025/Practical Work 1/TCP_File_Transfer]
└─$ python3 server.py
Server is listening on 0.0.0.0:9999...
Connection established with ('192.168.1.81', 40000)
Receiving file: file_upload.txt
```

Figure 2: Server command line

5 Conclusion

In this project, we successfully implemented a file transfer system using the TCP/IP protocol with a client-server architecture. The communication between the "Client" and "Server" components was carried out efficiently over a socket connection. The client sent the file in chunks, and the server received and stored each chunk.

The key features of the system include:

- Reliable file transfer with acknowledgment after each chunk.
- Handling of file metadata, such as the file's name and size, before the transfer begins.
- A clear separation of responsibilities between the client (sending the file) and the server (receiving and saving the file).
- Simple error handling to handle connection issues and ensure robust transfer.

This project provided a solid understanding of how to implement a basic file transfer system using TCP/IP sockets. It gave valuable insights into the functioning of such systems and how they can be further enhanced for more complex real-world applications.

6 References

References

[1] *socket — Low-level networking interface*. Retrieved from `https://docs.python.org/3/library/socket.html`

[2] *File Transfer Protocol*. Retrieved from `https://en.wikipedia.org/wiki/File_Transfer_Protocol`

[3] *File Transfer Using TCP*. Retrieved from `https://www.hackingarticles.in/how-to-transfer-file-using-tcp-ip-in-python/`