University of Dhaka

CSE:3111 Computer Networking Lab

Lab Report 3

Implementing File transfer using Socket Programming and HTTP GET/POST requests

Shariful Islam Rayhan
Roll: 41
Md Sakib Ur Rahman
Roll: 37

Submitted to:

Dr. Md. Abdur Razzaque
Dr. Muhammad Ibrahim
Dr. Md. Redwan Ahmed Rizvee
Dr. Md. Mamun Or Rashid

Submitted on: 2024-02-07

1 Introduction

In the previous lab,we get an broad introduction about socket programming.On that lab,we just make a client-server connection.On that,a server can only response one client at a time.But this lab extexded version of the previous on.The lab experiment focuses on the practical implementation of two fundamental networking concepts: socket programming and HTTP (Hypertext Transfer Protocol). With the overarching objective of understanding file transfer mechanisms, the experiment is divided into two main tasks.

Firstly, through multithreaded chat implementation using socket programming, the aim is to enable communication from multiple clients to a single server. The second task involves setting up an HTTP file server capable of handling concurrent client requests. Within this context, both HTTP GET and POST methods are utilized for downloading and uploading objects, respectively. This hands-on experience not only explores the intricacies of network communication but also provides practical insights into the crucial aspects of file transfer, emphasizing the significance of these skills in real-world applications and computer networking.

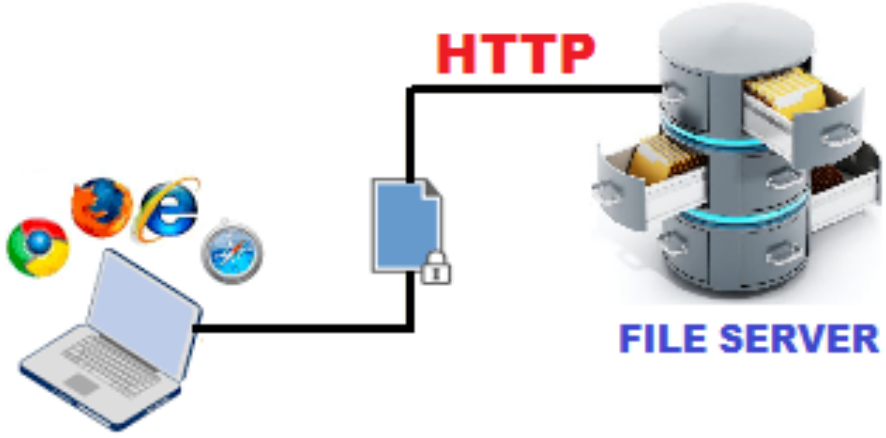

Figure 1: HTTP File Transfer

Figure 2: Handling Multiple Client in Socket

1.1 Objectives

- Enable the server to handle multiple clients simultaneously, implementing multithreading for concurrent communication.
- Prompt clients for the name of the file they wish to download upon connection.
- Implement a basic HTTP file server that listens for incoming connections on a specified port.
- Respond to client GET requests with the appropriate HTTP response headers, enabling the download of requested files.
- Process client POST requests, allowing the server to receive and save uploaded files.

2 Theory

In the context of socket programming, multithreading proves to be particularly advantageous as it enables a server to adeptly handle multiple client connections concurrently. This concurrency significantly enhances the server's responsiveness, allowing it to efficiently manage concurrent requests without impeding the execution of other processes. The primary purpose of incorporating multithreading in this scenario is to augment the scalability and overall performance of network applications. Consequently, the server becomes adept at simultaneously responding to numerous client requests, facilitating efficient file transfer to multiple users.

HTTP, known as Hypertext Transfer Protocol, forms the bedrock of communication on the World Wide Web. Adhering to a client-server model, wherein clients, typically web browsers, send requests to servers, HTTP ensures a seamless flow of information. Operating in a stateless manner, each HTTP request is treated independently, thereby simplifying the implementation and bolstering the reliability of web communication. The key advantages of HTTP encompass its inherent simplicity, platform independence, and versatility in transferring various types of data. This foundational protocol plays a pivotal role in facilitating communication between clients and servers, laying the groundwork for efficient and standardized data exchange on the web.

3 Methodology

3.1 Task 1: File Transfer via Socket Programming

By following the stpes neede to implement Server and Client for Task 1 we create the both file.The code snipets is given below

3.1.1 Server

```
import socket
import threading

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8081
server_socket=socket.socket()
server_socket.bind((SERVER_HOST, SERVER_PORT))
server_socket.listen()

def clienthandle(conn,addr):
        print(f"[*] Accepted connection from {addr[0]}:{addr
    [1]}")

        operation = conn.recv(1024).decode()
        filename = conn.recv(1024).decode()

        if operation == 'download':
            print(f"[*] Client requested file: {filename}")
            send_file(conn, filename,addr)
        elif operation == 'upload':
            print(f"[*] Client uploading file: {filename}")
            receive_file(conn, filename,addr)


def receive_file(conn, filename,addr):
    buffer_size = 4096   #Start with a reasonable buffer size
    received_data = b''  # Initialize an empty byte string to
    hold received data

    try:
        with open(filename, 'wb') as file:
            while True:
                data = conn.recv(buffer_size)
                if not data:
                    break
                received_data += data  # Append received data
    to the buffer
                buffer_size *= 2  # Double the buffer size for
    the next iteration
```

```
36
37              file.write(received_data)
38              # Write all received data to the file
39              print(f"[*] File received from {addr[0]}")
40      except FileNotFoundError:
41          print("[*] File not found")
42
43  def send_file(conn, filename,addr):
44      try:
45          with open(filename, 'rb') as file:
46              while True:
47                  data = file.read(4096)
48                  if not data:
49                      break
50                  conn.sendall(data)
51             # conn.sendall(b"")
52          print(f"[*] File sent to {addr[0]} ")
53      except FileNotFoundError:
54          print("[*] File not found")
55
56  def main():
57          print(f"[*] Listening on {SERVER_HOST}:{SERVER_PORT}")
58          while True:
59              conn, addr = server_socket.accept()
60              thread = threading.Thread(target=clienthandle, args
    =(conn,addr))
61              thread.start()
62
63
64  if __name__ == "__main__":
65      main()
```

3.1.2 Client

```
import socket

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8081


def receive_file(conn, filename):
    buffer_size = 4096
    received_data = b''

    try:
        with open(filename, 'wb') as file:
            while True:
                data = conn.recv(buffer_size)

                if not data or len(data) < buffer_size:
                   # print("break hosse na")
                    break


                received_data += data

            file.write(received_data)
            print("file is receives successfulyy")
            # Write all received data to the file
    except FileNotFoundError:
        print("[*] File not found")

def send_file(conn, filename):
    try:
        with open(filename, 'rb') as file:
            while True:
                data = file.read(4096)
                if not data:
                    break
                conn.sendall(data)
            print("[*] File sent")
    except FileNotFoundError:
        print("[*] File not found")


def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
    client_socket:
        client_socket.connect((SERVER_HOST, SERVER_PORT))
        operation = input("Enter 'download' to download file or
     'upload' to upload file: ")
```

```
        client_socket.sendall(operation.encode())

        if operation == 'download':
           # msg= client_socket.recv(1024).decode()
           # print(msg)

            filename = input("Enter the name of the file you want to download: ")
            client_socket.sendall(filename.encode())
            print(f"[*] Requesting file: {filename}")
            receive_file(client_socket, filename)
        elif operation == 'upload':
            filename = input("Enter the name of the file you want to upload: ")
            client_socket.sendall(filename.encode())
            print(f"[*] Uploading file: {filename}")
            send_file(client_socket, filename)


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

3.2 Task 2: File Transfer via HTTP Server

By following the stpes neede to implement Server and Client for Task 2 we create the both file.The code snipets is given below

3.2.1 Server

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import os


class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            if self.path == '/list_files':
                # Get a list of all files in the server directory
                files = os.listdir('.')
                files_str = "\n".join(files)

                # Send the list of files as the response
                self.send_response(200)
                self.send_header('Content-type', 'text/plain')
                self.end_headers()
                self.wfile.write(files_str.encode())
            else:
                # Extracting file path from the request
                file_path = self.path.strip("/")
                if not os.path.exists(file_path):
                    raise FileNotFoundError

                # Sending the file to the client
                with open(file_path, 'rb') as file:
                    self.send_response(200)
                    self.send_header('Content-type', 'application/octet-stream')
                    self.end_headers()
                    self.wfile.write(file.read())
        except FileNotFoundError:
            self.send_error(404, "File Not Found")

    def do_POST(self):
        try:
            content_length = int(self.headers['Content-Length'])
            # Extracting file name from the request path
            file_name = self.path.strip("/")
            if not file_name:
```

```
                raise ValueError("File name not provided in the request")
            # Reading file content from the request
            file_content = self.rfile.read(content_length)

            # Saving the received file
            with open(file_name, 'wb') as file:
                file.write(file_content)

            self.send_response(200)
            self.end_headers()
            self.wfile.write(b'File saved successfully')
        except Exception as e:
            self.send_error(500, str(e))


def run(server_class=HTTPServer, handler_class=SimpleHTTPRequestHandler, port=8005):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print(f"Server running on port {port}")
    httpd.serve_forever()


if __name__ == '__main__':
    run()
```

3.2.2 Client

```
import requests
import os
# Server URL
SERVER_URL = 'http://localhost:8005'


def list_files():
    # Send a GET request to list files
    response = requests.get(f"{SERVER_URL}/list_files")
    if response.status_code == 200:
        print("Files available on the server:")
        print(response.text)
    else:
        print(f"Failed to retrieve file list. Status code: {
    response.status_code}")


def download_file(file_name):
    # Send a GET request to download a file
    response = requests.get(f"{SERVER_URL}/{file_name}")
    if response.status_code == 200:
        with open(file_name, 'wb') as f:
            f.write(response.content)
        print(f"File '{file_name}' downloaded successfully")
    else:
        print(f"Failed to download file '{file_name}'. Status
    code: {response.status_code}")


def upload_file(file_path):
    try:
        # Extracting the filename from the file path
        file_name = os.path.basename(file_path)

        # Send a POST request to upload the file
        files = {'file': (file_name, open(file_path, 'rb'))}
        response = requests.post(f"{SERVER_URL}/{file_name}",
    files=files)

        if response.status_code == 200:
            print(f"File '{file_path}' uploaded successfully")
        else:
            print(f"Failed to upload file '{file_path}'. Status
     code: {response.status_code}")
    except FileNotFoundError:
        print(f"File '{file_path}' not found.")
    except Exception as e:
```

```
        print(f"An error occurred while uploading file '{file_path}': {e}")


def main():
    while True:
        print("1. List files\n2. Download file\n3. Upload file\n4.Exit")
        choice = input("Enter your choice: ")

        if choice.isdigit():  # Check if the input is a valid integer
            choice = int(choice)
            if choice == 1:
                list_files()
            elif choice == 2:
                file_name = input("Enter the name of the file to download: ")
                download_file(file_name)
            elif choice == 3:
                file_path = input("Enter the path of the file to upload: ")
                upload_file(file_path)
            elif choice == 4:
                break
            else:
                print("Invalid choice. Please enter a valid option.")
        else:
            print("Invalid input. Please enter a number.")


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

4 Experimental result

Some Snapshots of Task 1 and Task 2 are shown here with detailed explanation:

4.1 Task 1: File Transfer Via Socket

4.1.1 Starting Server and Multiple Client

Form the snapshot we can server accepted multiple client.Two clients are connected to server.It has benn possible because of thereading.Threading causes to start a specific connection to specific client.

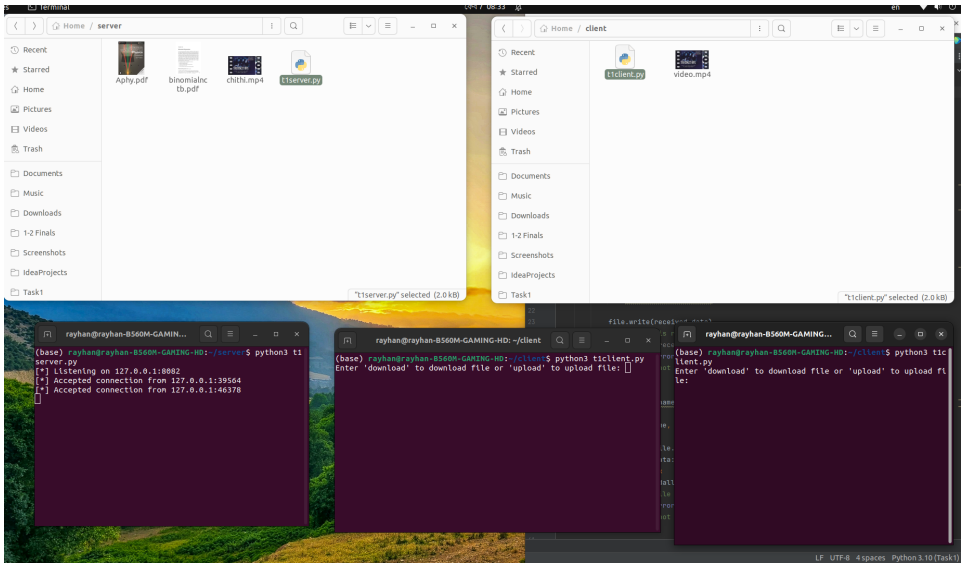

Figure 3: Multiple Client Connected to server

4.1.2 Download And Upload

Multiple clients can upload and download file to and form server at a time.Here video.mp4 was uploaded to server from client and chithi.mp4 was downloaded from the server.

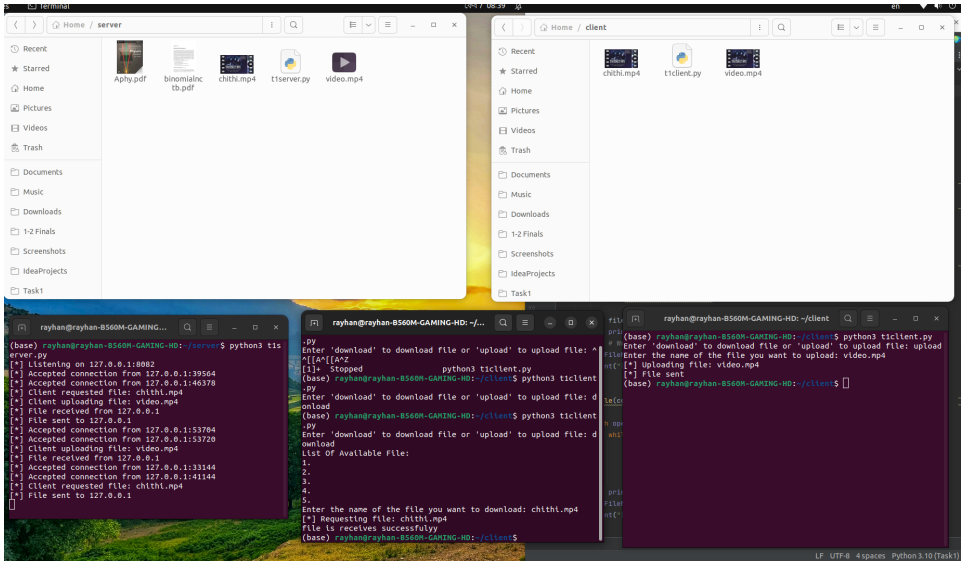

Figure 4: Multiple clients are uploading and downloading

4.2 Task 2

4.2.1 Starting Server and Client

Here client connected to server and see the list of available file in server to download.

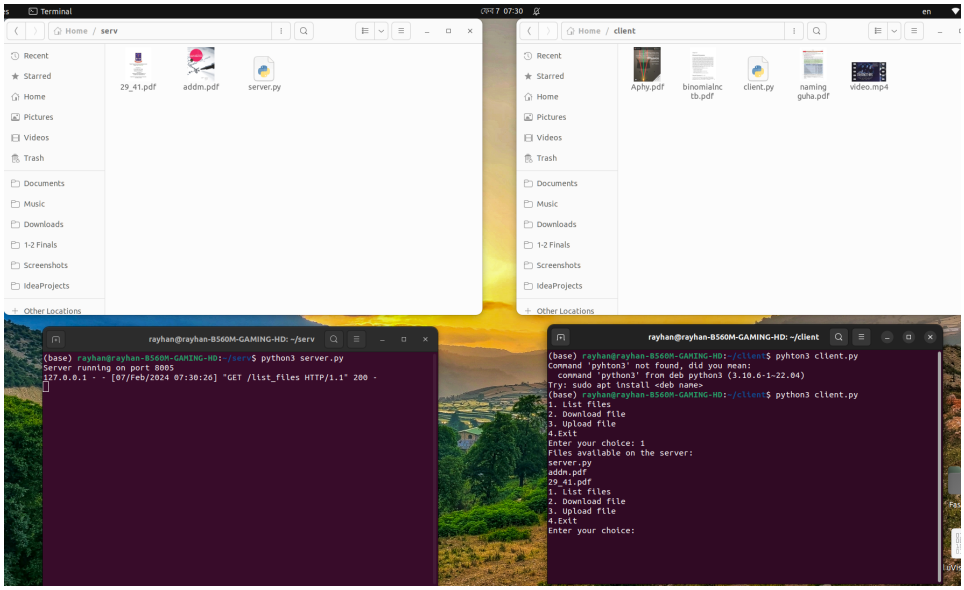

Figure 5: Before get() and post() condition of server and client

4.2.2 Download

Here client downloaded addm.pdf file from the server by get() method.

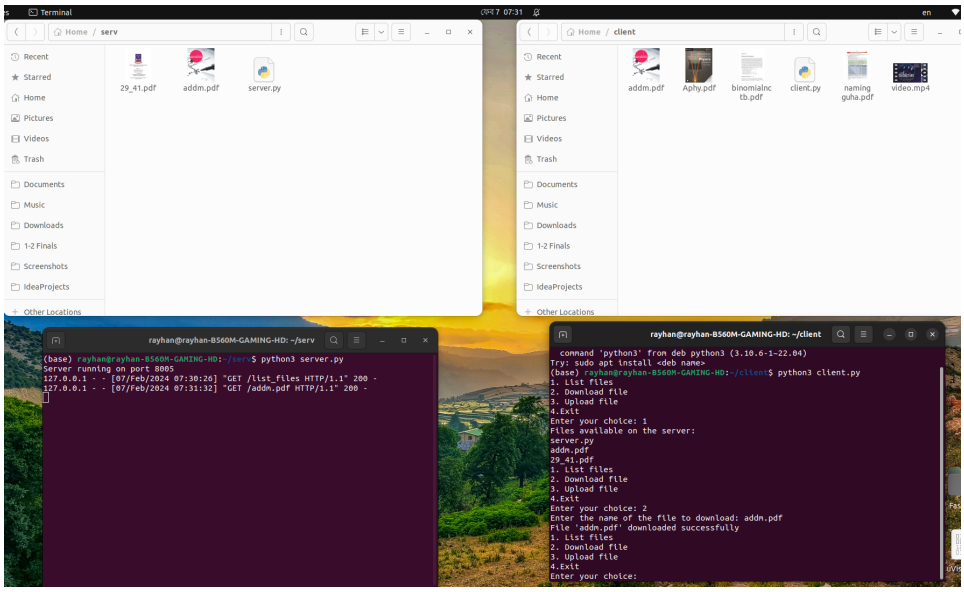

Figure 6: After Download From Server

4.2.3 Upload

Here client uploaded video.mp4 to the server by post() method.

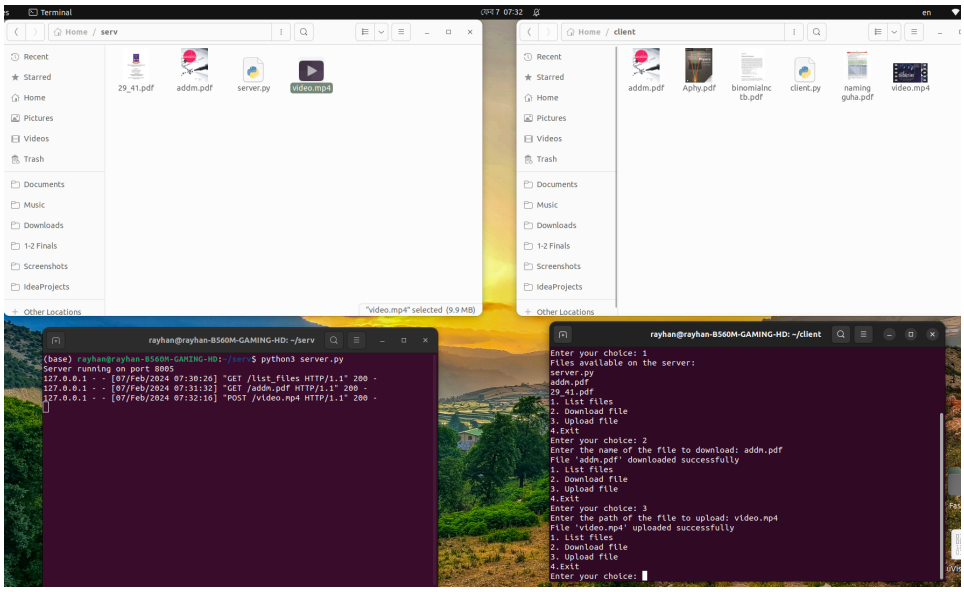

Figure 7: Client uploading video.mp4

4.2.4 Multiple Client

Here we can see multiple client can connect to server and can use post() and get() method also.

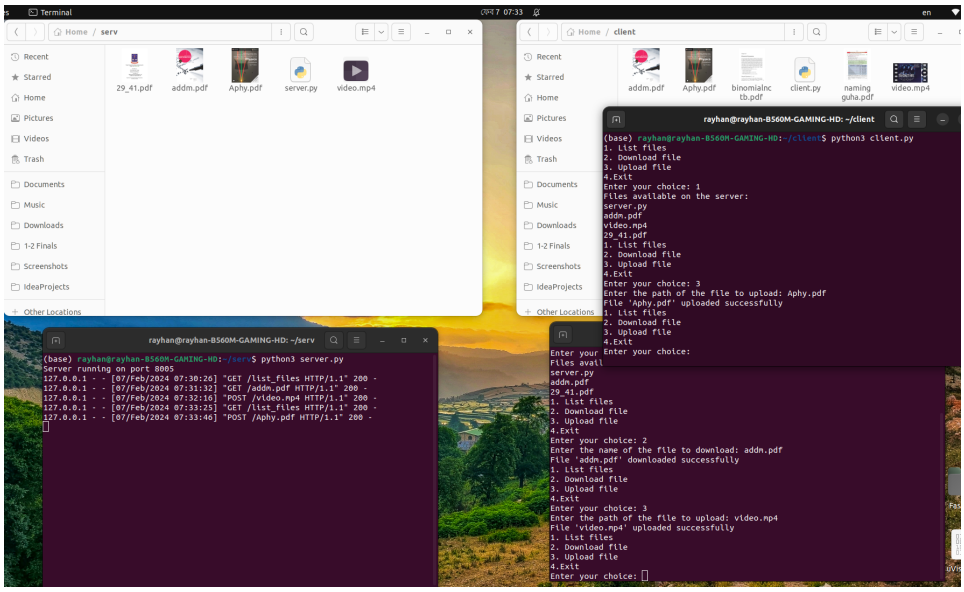

Figure 8: Multiple Client Can access the Server

5 Experience

1. Used multithreading to connect multiple client
2. Learn How Flie Transfer Server can be implemented in Socket
3. Learn How Flie Transfer Server can be implemented in HTTP
4. It was nice to implement server and client and make sense

References

[1] HTTP : https://www.youtube.com/watch?v=DeFST8tvtuI&t=60s

[2] Socket: https://www.youtube.com/watch?v=qFVoMo6OMsQ&t=612s

[3] SendingGetandPost:https://www.digitalocean.com/community/tutorials/java-httpurlconnection-example-java-http-request-get-post