University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 3 : Implementing File transfer using Socket Programming and HTTP GET/POST requests

Submitted By:

Name: Md Shamsur Rahman Sami
Roll No : 57

Name: Md Rakib Hossain
Roll No : 55

Submitted On :

February 08, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Contents

1 Introduction

1.1 Objectives

The primary objective of this lab experiment is to delve into the fundamentals of socket programming while emphasizing practical implementation. Through this exercise, participants aim to comprehend the intricacies of creating and managing network sockets, establishing robust connections between client and server applications, and orchestrating data transmission over a network.

A pivotal aspect of this experiment involves the implementation of a client-server communication model, whereby participants craft both client and server applications capable of seamless interaction. This entails mastering the intricacies of socket creation, binding, and listening on the server side, and connecting to the server from the client side.

Furthermore, the experiment delves into the intricacies of file transfer mechanisms, a quintessential aspect of network communication. Participants navigate the process of reading files from the server and transmitting them to clients, or receiving files from clients and storing them on the server. This involves handling binary data efficiently and ensuring data integrity throughout the transmission process.

An additional focus lies on error handling and robustness, where participants gain insights into implementing mechanisms to address potential issues that may arise during socket communication, such as network disruptions or file transfer errors. By instilling a proactive approach to error handling, participants enhance the reliability and resilience of their socket-based applications, thereby bolstering their proficiency in network programming.

2 Theory of File Transfer Using Socket Programming and HTTP GET/POST Requests

This section provides a theoretical overview of implementing file transfer using socket programming and HTTP GET/POST requests. It covers fundamental concepts, protocols, and considerations essential for understanding and implementing file transfer mechanisms.

2.1 Networking Basics

- **TCP/IP Protocol Suite:** TCP/IP protocols, including IP, TCP, and UDP, form the backbone of network communication. TCP provides reliable, stream-oriented data transfer, while UDP offers faster, datagram-based communication.

- **Sockets:** Sockets serve as endpoints for network communication, enabling processes to exchange data over a network. Server sockets listen for incoming connections, while client sockets initiate connections to servers.

2.2 HTTP Protocol

- **Fundamentals:** HTTP operates on a request-response model, with methods like GET and POST for retrieving and uploading files, respectively. Headers and status codes provide additional metadata and information about the request/response.
- **File Transfer with HTTP:** HTTP GET requests are commonly used to retrieve files from servers, while POST requests facilitate file uploads. Chunked Transfer Encoding allows efficient transfer of large files in smaller chunks.

2.3 Socket Programming Framework

- **Choice of Language:** Select a programming language with robust networking support, such as Python, Java, or C++.
- **Socket API:** Familiarize yourself with the language's socket API for creating, binding, listening, connecting, sending, and receiving data.
- **Error Handling:** Implement error handling mechanisms to handle connection failures, invalid requests, and other network issues gracefully.

2.4 Security Considerations

- **Authentication and Authorization:** Implement mechanisms to verify user identity and grant access to files based on permissions.
- **Encryption:** Encrypt data transfers to protect sensitive information, especially over public networks.
- **Vulnerability Management:** Stay updated on security vulnerabilities and take measures to mitigate risks.

2.5 Optimization Techniques

- **Buffering:** Use buffers to improve data transfer efficiency by reducing network calls.
- **Compression:** Compress files before transmission to reduce bandwidth usage.
- **Multithreading:** Implement thread-based concurrency for handling multiple file transfers simultaneously.

3 Methodology: File Transfer Using Socket Programming and HTTP

3.1 Understanding Requirements

- Define the requirements for the file transfer system, including supported file types, expected file sizes, security considerations, and performance requirements.

3.2 Designing the System Architecture

- Determine the overall architecture of the file transfer system, including client-server communication protocols, data formats, and error handling mechanisms.
- Choose between socket programming and HTTP for file transfer, considering factors such as network environment, scalability, and compatibility.

3.3 Implementing the File Server

- Develop the file server component responsible for handling incoming file transfer requests.
- For socket programming:
  - Create a server socket to listen for incoming connections.
  - Accept client connections and handle file transfer requests.
  - Read files from the server's file system and send them to clients over the network.

- For HTTP:
  - Set up an HTTP server using frameworks like Flask or Django.
  - Implement endpoints for handling file upload (POST) and download (GET) requests.
  - Receive files uploaded by clients and store them in a designated directory.
  - Serve files to clients upon receiving download requests.

3.4 Implementing the File Client

- Develop the file client component responsible for initiating file transfer requests and handling server responses.
- For socket programming:
  - Create a client socket to establish connections with the file server.
  - Send file transfer requests to the server and receive file contents in response.
  - Save received files to the client's file system.
- For HTTP:
  - Use HTTP client libraries (e.g., requests in Python) to send file upload and download requests to the server.
  - Handle server responses and process downloaded files as needed.

3.5 Testing and Validation

- Test the file transfer system under various scenarios, including different file sizes, concurrent transfers, and network conditions.
- Verify that file uploads and downloads are performed correctly and efficiently.
- Conduct security testing to ensure data integrity and confidentiality during file transfer operations.

3.6 Performance Optimization

- Identify performance bottlenecks and optimize the file transfer system for speed and efficiency.
- Implement techniques such as parallel processing, data compression, and caching to improve transfer speeds and reduce network latency.

3.7 Documentation and Maintenance

- Document the system architecture, implementation details, and usage instructions for future reference.
- Establish a maintenance plan to monitor and update the file transfer system as needed, including security patches and performance enhancements.

4 Experimental Result

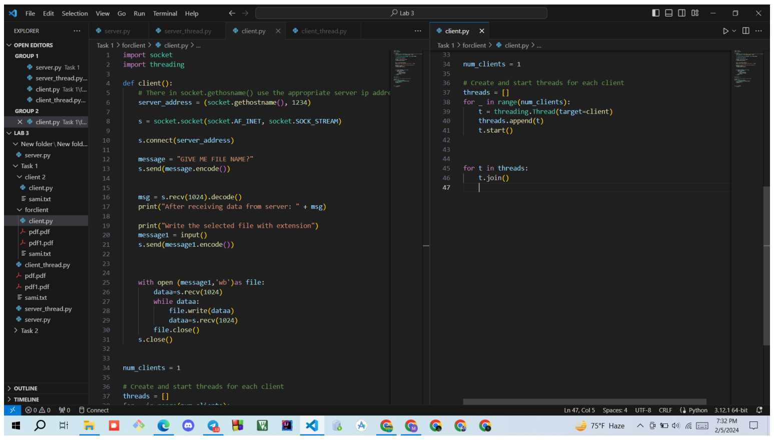
4.1 Task 1

- **File Transfer via Socket Programming handelling multiple clients**

```
    SERVER SIDE CODE

import socket
import threading
import os

def handle_client(client_socket,address):

    msg = client_socket.recv(1024).decode()
    print(msg)
    response = "This file are avaiable : pdf1.pdf,pdf.pdf,sami.txt"
    client_socket.send(response.encode())
    msg1 = client_socket.recv(1024)
    msg1 = msg1.decode()
    print(msg1)
    file_path = os.path.join(r"F:\Python\Python\Networking Lab\Lab 3\Task 1", ms

```

```python

    try:
        with open(file_path, 'rb') as f:
            while True:
                data = f.read(1024)
                if not data:
                    break
                client_socket.sendall(data)
        f.close()
    # except FileNotFoundError:
    #     print("File not found!")
    except Exception as e:
        print("Error occurred during file transmission:", e)
    finally:
        client_socket.close()





    # with open(file_path, 'rb') as file:
    # # Your file handling code here


    # # file =open("sami.txt","rb")
    #     dd=file.read()
    #     client_socket.sendall(dd)
    #     file.close()
    # client_socket.close()



def server():
    #server_address = (socket.gethostname(), 1234)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((socket.gethostname(),1234))
    s.listen()
    print("Server is listening for incoming connections...")

    while True:

        client_socket, address = s.accept()
        print("connection Successfull")
        client_thread = threading.Thread(target=handle_client, args=(client_sock
        client_thread.start()

if __name__ == "__main__":
    server()

```

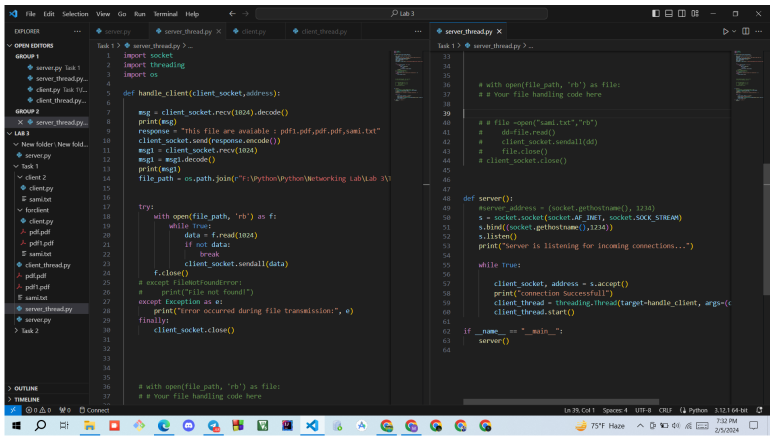
```
    CLIENT SIDE CODE
import socket
import threading

def client():
    # There in socket.gethosname() use the appropriate server ip address.
    server_address = (socket.gethostname(), 1234)

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.connect(server_address)

    message = "GIVE ME FILE NAME?"
    s.send(message.encode())


    msg = s.recv(1024).decode()
    print("After receiving data from server: " + msg)

    print("Write the selected file with extension")
    message1 = input()
    s.send(message1.encode())



    with open (message1,'wb')as file:
        dataa=s.recv(1024)
        while dataa:
            file.write(dataa)
            dataa=s.recv(1024)
        file.close()
    s.close()


num_clients = 1

# Create and start threads for each client
threads = []
for _ in range(num_clients):
    t = threading.Thread(target=client)
    threads.append(t)
    t.start()



for t in threads:
    t.join()

```



```
import socket
import threading

def client():
    # There in socket.gethosname() use the appropriate server ip addre
    server_address = (socket.gethostname(), 1234)

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.connect(server_address)

    message = "GIVE ME FILE NAME?"
    s.send(message.encode())


    msg = s.recv(1024).decode()
    print("After receiving data from server: " + msg)

    print("Write the selected file with extension")
    message1 = input()
    s.send(message1.encode())



    with open (message1,'wb')as file:
        dataa=s.recv(1024)
        while dataa:
            file.write(dataa)
            dataa=s.recv(1024)
        file.close()
    s.close()


num_clients = 1

# Create and start threads for each client
threads = []
```

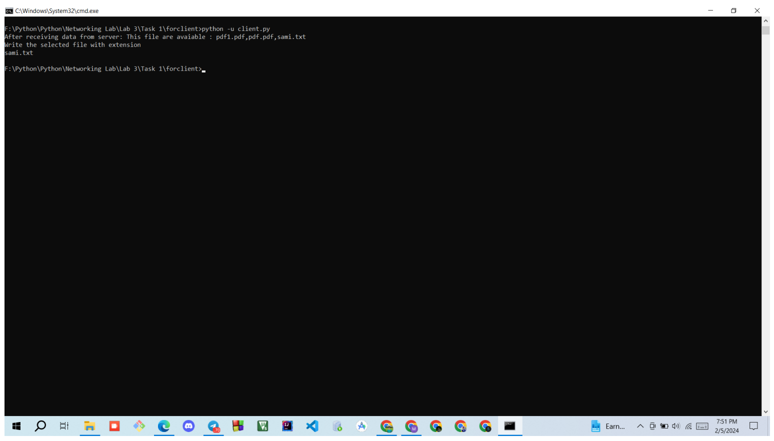
```

num_clients = 1

# Create and start threads for each client
threads = []
for _ in range(num_clients):
    t = threading.Thread(target=client)
    threads.append(t)
    t.start()



for t in threads:
    t.join()
    
```

Figure 1: Client side code



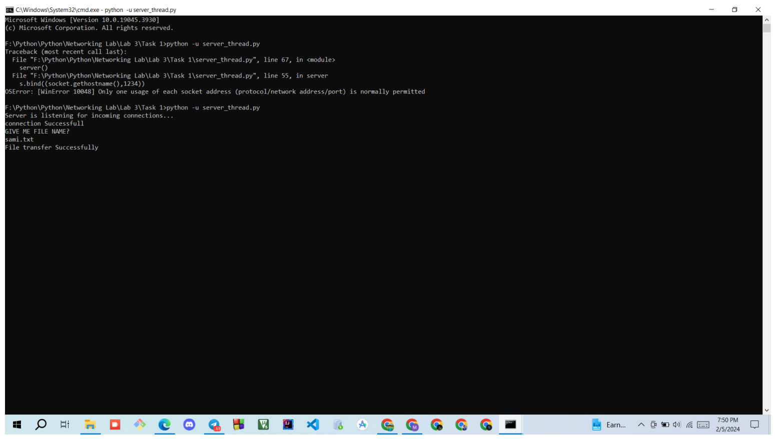
```python
import socket
import threading
import os

def handle_client(client_socket,address):

    msg = client_socket.recv(1024).decode()
    print(msg)
    response = "This file are avaiable : pdf1.pdf,pdf.pdf,sami.txt"
    client_socket.send(response.encode())
    msg1 = client_socket.recv(1024)
    msg1 = msg1.decode()
    print(msg1)
    file_path = os.path.join(r"F:\Python\Python\Networking Lab\Lab 3\T


    try:
        with open(file_path, 'rb') as f:
            while True:
                data = f.read(1024)
                if not data:
                    break
                client_socket.sendall(data)
        f.close()
    # except FileNotFoundError:
    #     print("File not found!")
    except Exception as e:
        print("Error occurred during file transmission:", e)
    finally:
        client_socket.close()





    # with open(file_path, 'rb') as file:
    # # Your file handling code here


    # # file =open("sami.txt","rb")
    #     dd=file.read()
    #     client_socket.sendall(dd)
    #     file.close()
    # client_socket.close()



def server():
    #server_address = (socket.gethostname(), 1234)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((socket.gethostname(),1234))
    s.listen()
    print("Server is listening for incoming connections...")

    while True:

        client_socket, address = s.accept()
        print("connection Successfull")
        client_thread = threading.Thread(target=handle_client, args=(c
        client_thread.start()

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

Figure 2: Server Side Code



```
F:\Python\Python\Networking Lab\Lab 3\Task 1\forclient>python -u client.py
After receiving data from server: This file are avaiable : pdf1.pdf,pdf.pdf,sami.txt
Write the selected file with extension
sami.txt

F:\Python\Python\Networking Lab\Lab 3\Task 1\forclient>
```

Figure 3: Client Side result



```
C:\Windows\System32\cmd.exe - python  -u server_thread.py
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

F:\Python\Python\Networking Lab\Lab 3\Task 1>python -u server_thread.py
Traceback (most recent call last):
  File "F:\Python\Python\Networking Lab\Lab 3\Task 1\server_thread.py", line 67, in <module>
    server()
  File "F:\Python\Python\Networking Lab\Lab 3\Task 1\server_thread.py", line 55, in server
    s.bind((socket.gethostname(),1234))
OSError: [WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted

F:\Python\Python\Networking Lab\Lab 3\Task 1>python -u server_thread.py
Server is listening for incoming connections...
connection Successfull
GIVE ME FILE NAME?
sami.txt
File transfer Successfully
```

Figure 4: Server Side result

4.2 Task 2

- **File Transfer via HTTP handelling multiple clients simultaneously.**

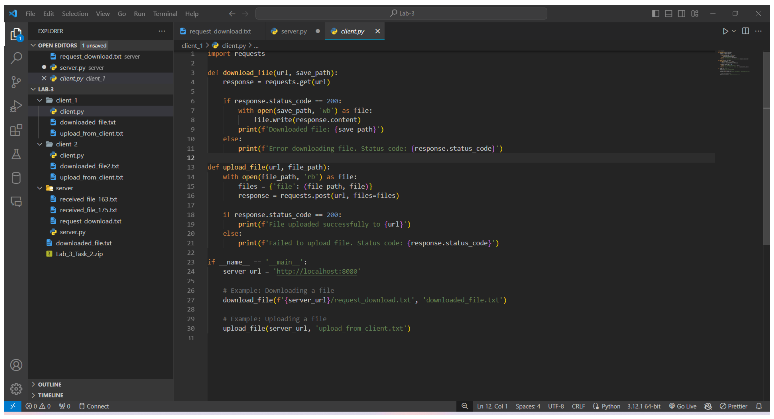
```
    SERVER SIDE CODE
    from http.server import BaseHTTPRequestHandler, HTTPServer
import os

class FileServerHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:

            if self.path == '/':

                file_path = 'index.html'
            else:

                file_path = self.path[1:]


            if os.path.exists(file_path) and os.path.isfile(file_path):
                with open(file_path, 'rb') as file:
                    self.send_response(200)
                    self.send_header('Content-type', 'application/octet-stream')
                    self.send_header('Content-Disposition', f'attachment; filena
                    self.end_headers()
                    self.wfile.write(file.read())
                    print("Client Get Request ")

            else:
                self.send_response(404)
                self.end_headers()
                self.wfile.write(b'File not found')

        except Exception as e:
            self.send_response(500)
            self.end_headers()
            self.wfile.write(str(e).encode())

    def do_POST(self):
        try:
            content_length = int(self.headers['Content-Length'])
            file_content = self.rfile.read(content_length)


            unique_filename = f'received_file_{content_length}.txt'

            with open(unique_filename, 'wb') as file:
                file.write(file_content)

            self.send_response(200)
            self.end_headers()
            print("Client Post Request ")
            print("Save it in server disk ")
            self.wfile.write(b'File saved successfully')

        except Exception as e:
            self.send_response(500)
            self.end_headers()
            self.wfile.write(str(e).encode())

def run_server(port=8080):
    try:
        server_address = ('', port)
        httpd = HTTPServer(server_address, FileServerHandler)
        print(f'Starting server on port {port}')
        httpd.serve_forever()

    except KeyboardInterrupt:
        print('Server stopped')

if __name__ == '__main__':
    run_server(port=8080)



```

```
    CLIENT SIDE CODE
   import requests

def download_file(url, save_path):
    response = requests.get(url)

    if response.status_code == 200:
        with open(save_path, 'wb') as file:
            file.write(response.content)
        print(f'Downloaded file: {save_path}')
    else:
        print(f'Error downloading file. Status code: {response.status_code}')

def upload_file(url, file_path):
    with open(file_path, 'rb') as file:
        files = {'file': (file_path, file)}
        response = requests.post(url, files=files)

    if response.status_code == 200:
        print(f'File uploaded successfully to {url}')
    else:
        print(f'Failed to upload file. Status code: {response.status_code}')

if __name__ == '__main__':
    server_url = 'http://localhost:8080'

    # Example: Downloading a file
```

```python
    download_file(f'{server_url}/request_download.txt', 'downloaded_file.txt')

    # Example: Uploading a file
    upload_file(server_url, 'upload_from_client.txt')

```

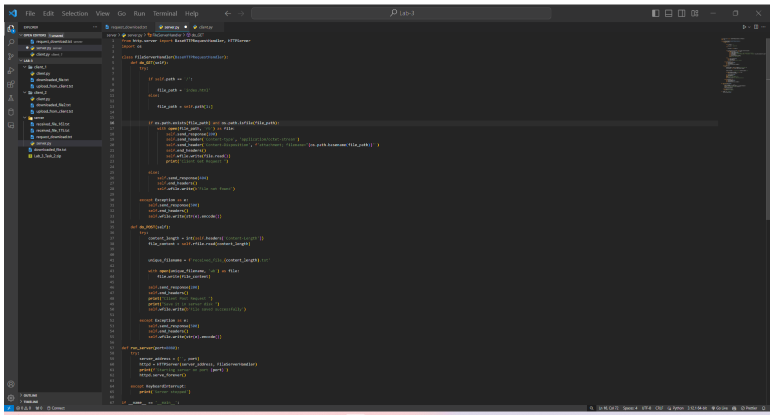
—

.



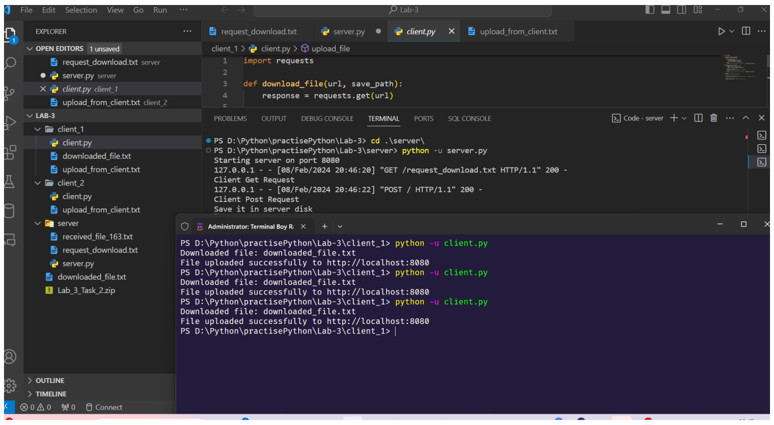
```python
import requests

def download_file(url, save_path):
    response = requests.get(url)

    if response.status_code == 200:
        with open(save_path, 'wb') as file:
            file.write(response.content)
        print(f'Downloaded file: {save_path}')
    else:
        print(f'Error downloading file. Status code: {response.status_code}')

def upload_file(url, file_path):
    with open(file_path, 'rb') as file:
        files = {'file': (file_path, file)}
        response = requests.post(url, files=files)

    if response.status_code == 200:
        print(f'File uploaded successfully to {url}')
    else:
        print(f'Failed to upload file. Status code: {response.status_code}')

if __name__ == '__main__':
    server_url = 'http://localhost:8080'

    # Example: Downloading a file
    download_file(f'{server_url}/request_download.txt', 'downloaded_file.txt')

    # Example: Uploading a file
    upload_file(server_url, 'upload_from_client.txt')
```

Figure 5: Client side code



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

class FileServerHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:

            if self.path == '/':

                file_path = 'index.html'
            else:

                file_path = self.path[1:]


            if os.path.exists(file_path) and os.path.isfile(file_path):
                with open(file_path, 'rb') as file:
                    self.send_response(200)
                    self.send_header('Content-type', 'application/octet-stream')
                    self.send_header('Content-Disposition', f'attachment; filename="{os.path.basename(file_path)}"')
                    self.end_headers()
                    self.wfile.write(file.read())
                    print("Client Get Request ")

            else:
                self.send_response(404)
                self.end_headers()
                self.wfile.write(b'File not found')

        except Exception as e:
            self.send_response(500)
            self.end_headers()
            self.wfile.write(str(e).encode())

    def do_POST(self):
        try:
            content_length = int(self.headers['Content-Length'])
            file_content = self.rfile.read(content_length)


            unique_filename = f'received_file_{content_length}.txt'

            with open(unique_filename, 'wb') as file:
                file.write(file_content)

            self.send_response(200)
            self.end_headers()
            print("Client Post Request ")
            print("Save it in server disk ")
            self.wfile.write(b'File saved successfully')

        except Exception as e:
            self.send_response(500)
            self.end_headers()
            self.wfile.write(str(e).encode())

def run_server(port=8080):
    try:
        server_address = ('', port)
        httpd = HTTPServer(server_address, FileServerHandler)
        print(f'Starting server on port {port}')
        httpd.serve_forever()

    except KeyboardInterrupt:
        print('Server stopped')

if __name__ == '__main__':
```

Figure 6: Server Side Code

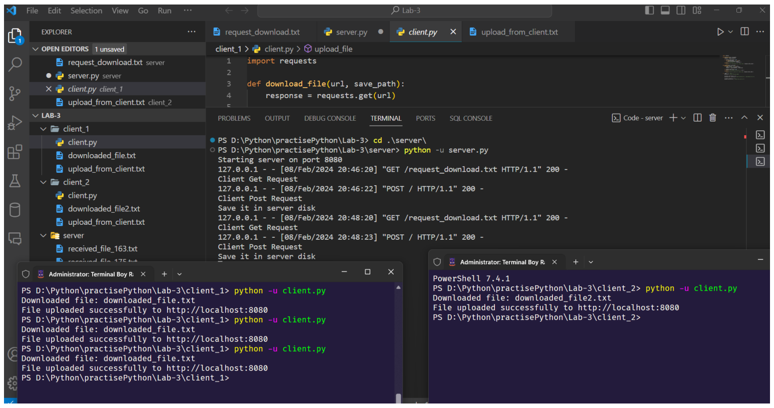


```python
import requests

def download_file(url, save_path):
    response = requests.get(url)
```

```
PS D:\Python\practisePython\Lab-3> cd .\server\
PS D:\Python\practisePython\Lab-3\server> python -u server.py
Starting server on port 8080
127.0.0.1 - - [08/Feb/2024 20:46:20] "GET /request_download.txt HTTP/1.1" 200 -
Client Get Request
127.0.0.1 - - [08/Feb/2024 20:46:22] "POST / HTTP/1.1" 200 -
Client Post Request
Save it in server disk
```

```
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1>
```

Figure 7: Result for Single client



```python
import requests

def download_file(url, save_path):
    response = requests.get(url)
```

```
PS D:\Python\practisePython\Lab-3> cd .\server\
PS D:\Python\practisePython\Lab-3\server> python -u server.py
Starting server on port 8080
127.0.0.1 - - [08/Feb/2024 20:46:20] "GET /request_download.txt HTTP/1.1" 200 -
Client Get Request
127.0.0.1 - - [08/Feb/2024 20:46:22] "POST / HTTP/1.1" 200 -
Client Post Request
Save it in server disk
127.0.0.1 - - [08/Feb/2024 20:48:20] "GET /request_download.txt HTTP/1.1" 200 -
Client Get Request
127.0.0.1 - - [08/Feb/2024 20:48:23] "POST / HTTP/1.1" 200 -
Client Post Request
Save it in server disk
```

```
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1> python -u client.py
Downloaded file: downloaded_file.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_1>
```

```
PowerShell 7.4.1
PS D:\Python\practisePython\Lab-3\client_2> python -u client.py
Downloaded file: downloaded_file2.txt
File uploaded successfully to http://localhost:8080
PS D:\Python\practisePython\Lab-3\client_2>
```

Figure 8: Result for multiple client

5 Experience

1. Ensuring that the server and client handle user inputs safely and validate file paths and names.
2. We have create multiple clients , thats why we needs more than one terminal for clients side and one for server sides
3. First we run server side program and then client request server for specific task
4. Server handle multiple clients request simultaneously, using treads programming
5. Handling different HTTP methods: Supporting both GET and POST methods for file transfer.
6. Identifying and resolving issues related to socket communication, file handling, or HTTP protocol implementation.

References

[1] Socket Programming in Python: `https://www.geeksforgeeks.org/socket-programming-python/`

[2] Socket Programming in Python: `https://realpython.com/python-sockets/`

[3] File Transfer via HTTP: `https://www.freecodecamp.org/news/simplehttpserver-explained-how-to-send-files-using-python/#:~:text=Open%20browser%20and%20type%20in,settings%20on%20the%20first%20computer.`

[4] File Transfer via HTTP: `https://www.youtube.com/watch?v=DeFST8tvtuI&t=630s`