



# 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: Meherun Farzana

Roll No : 05

Name: Mohd. Jamal Uddin Mallick

Roll No : 07

**Submitted On :**

February 8, 2024

**Submitted To :**

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Redwan Ahmed Rizvee

# 1 Introduction

File transfer is a critical aspect of modern networking applications, facilitating the exchange of data between systems. This lab report focuses on implementing file transfer using two essential techniques: socket programming and HTTP GET/POST requests. Socket programming provides a foundational framework for establishing communication channels between client and server applications, while HTTP protocols offer standardized methods for data exchange over the web.

## 1.1 Objectives

- **Socket Programming:** Develop a file transfer mechanism using socket programming to establish communication between client and server applications.
- **HTTP GET/POST Requests** Implement HTTP GET and POST requests to facilitate file retrieval and submission over the network.

# 2 Theory

## 2.1 File Transfer via Socket

Socket programming is the foundation of network communication, facilitating the establishment of connections between client and server applications for data exchange. File transfer via sockets refers to the process of transferring a file from one system to another over a network connection using socket programming. File transfer via sockets offers more control over the transfer process compared to higher-level protocols like HTTP, but also requires manual handling of error handling, data chunking, and other aspects of the transfer.

## 2.2 File Transfer via HTTP

HTTP (Hypertext Transfer Protocol) serves as the backbone of web communication, enabling clients to request resources from servers using methods such as GET and POST. HTTP is a standard protocol for transmitting data over the web, and is widely used for web communication between a client (e.g. a web browser) and a server (e.g. a web server). In file transfer via HTTP, the client sends a request to the server using either an HTTP GET or POST request. Overall, file transfer via HTTP is a simple and convenient way to transfer files over a network, but may not be as fast or efficient as other methods, such as socket programming.

# 3 Methodology

## 3.1 Server

The server is initialized on a specific port and it listens for incoming requests. Whenever a client requests to connect, the server accepts the connection and provides necessary services.

### 3.1.1 File Transfer via Socket

In case of sockets, the server lets the client choose whether they want to get the list of available files, upload or download a file. The server can handle any type of file including text, audio, image, video etc. of all formats.

The server is able to handle multiple clients parallelly because of multi-threading.

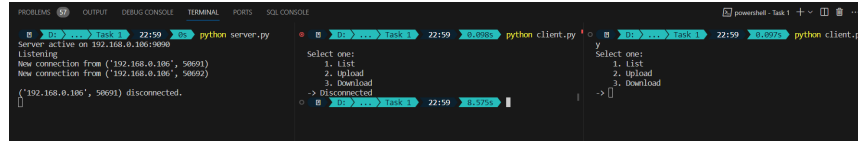


Figure 1: Multi-threading in socket

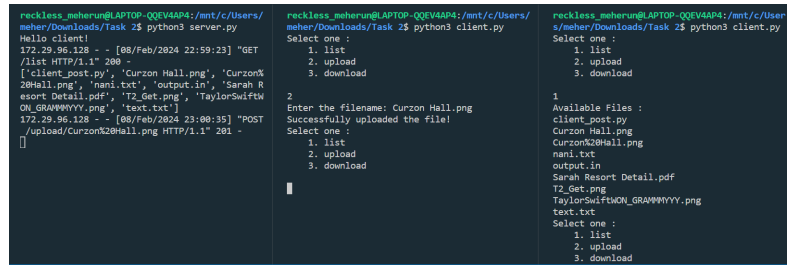


Figure 2: Multi-threading in HTTP

### 3.1.2 File Transfer via HTTP

In case of HTTP, the server is an HTTP server, serving GET and POST requests from clients. The client receives a response from the server indicating the success or failure of the transfer.

## 3.2 Client

### 3.2.1 File Transfer via Socket

The client side is a program requesting service from the server. It tries to connect to the server address mentioning the particular port on which the server is serving. When the server accepts the connection, it is provided with the options of service the client can avail. The client selects whether they want to see the list of available files, upload or download a file.

### 3.2.2 File Transfer via HTTP

In case of HTTP, the client can make a *GET* request to the server for a specific resource and avail it. Also it can make a *POST* request for uploading new files into the server.

## 4 Experimental result

Some Snapshots of the Client Side queries can be seen in the following figures:

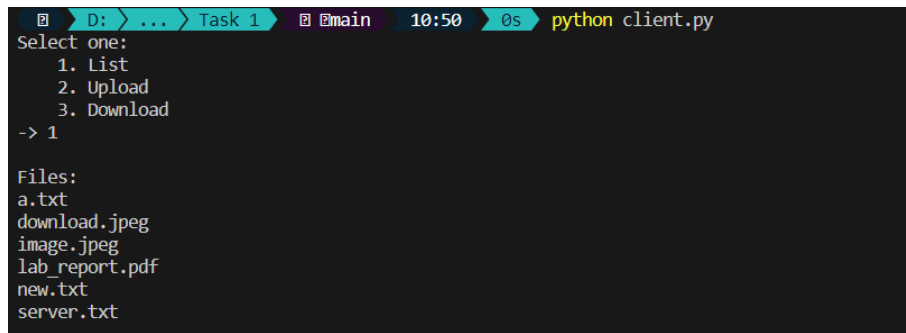
## 4.1 Task 1: File Transfer via Socket Programming

### 4.1.1 Server:

Server creates a socket and binds it to one of its ports. And it starts listening to incoming requests. When a client connection arrives, it accepts the connection and serves accordingly.

### 4.1.2 Client:

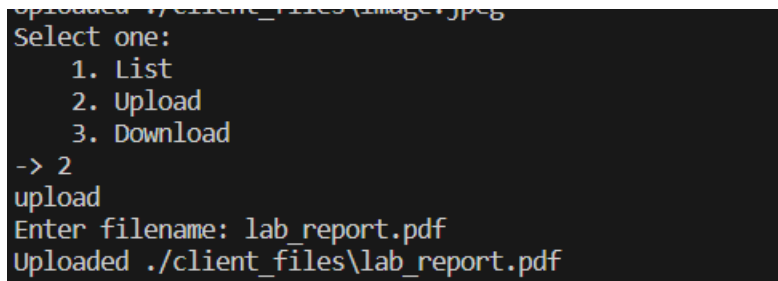
The client requests to connect to the server in the mentioned port. After being accepted, the client is asked what request they want to make. The client sends the request using TCP protocol and waits to receive a response.



```
D:\Task 1 > python client.py
Select one:
  1. List
  2. Upload
  3. Download
-> 1

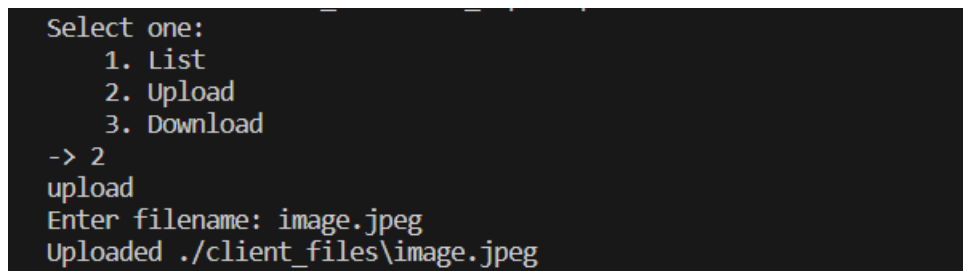
Files:
a.txt
download.jpeg
image.jpeg
lab_report.pdf
new.txt
server.txt
```

Figure 3: Output for Option 1 (Showing all the files available in the directory)



```
Uploaded ./client_files/image.jpeg
Select one:
  1. List
  2. Upload
  3. Download
-> 2
upload
Enter filename: lab_report.pdf
Uploaded ./client_files/lab_report.pdf
```

Figure 4: Output for Option 2 (Client uploading a pdf)



```
Select one:
  1. List
  2. Upload
  3. Download
-> 2
upload
Enter filename: image.jpeg
Uploaded ./client_files/image.jpeg
```

Figure 5: Output for Option 2 (Client uploading a jpeg file)



```

172.29.96.128 - - [08/Feb/2024 21:24:51] "GET /list HTTP/1.1" 200 -
['Curzon Hall.png', 'Sarah Resort Detail.pdf', 'TaylorSwiftWON_GRAMMMYY
Y.png', 'text.txt']
172.29.96.128 - - [08/Feb/2024 21:25:07] "POST /upload/Curzon%20Hall.pn
g HTTP/1.1" 201 -

```

Figure 9: POST request server side

#### 4.2.2 Client:

The client side in an HTTP communication refers to the entity that initiates the request for data from the server. The client can be a web browser, a mobile app, or any other software that needs to retrieve data from an HTTP server. On the client side, an HTTP request message is generated and sent to the server. This message includes information such as the URL of the requested resource, the HTTP method (e.g. GET or POST), and any necessary data or headers. In conclusion, the client side in an HTTP communication is responsible for initiating requests to the server, processing the response from the server, and rendering or further processing the data as necessary.

```

reckless_meherun@LAPTOP-QQEV4AP4:/mnt/c/Users/meher/Downloads/Folder$
python3 client.py
Select one :
    1. list
    2. upload
    3. download

1
Available Files :
Curzon Hall.png
Sarah Resort Detail.pdf
TaylorSwiftWON_GRAMMMYYY.png
text.txt
Select one :
    1. list
    2. upload
    3. download


```

Figure 10: Showing available services

```
3
Enter the filename: Sarah Resort Detail.pdf
Successfully downloaded the file!
Select one :
  1. list
  2. upload
  3. download
```

Figure 11: Output for Get

```
3. download

2
Enter the filename: Curzon Hall.png
Successfully uploaded the file!
Select one :
  1. list
  2. upload
  3. download
```

Figure 12: Output for Post

## 5 Experience

1. We had to learn how to enable the server to handle multiple client at once.
2. We had to see some examples of how to use HttpServer package in Python

## 6 Appendix

### 6.1 Task 1

#### 6.1.1 Server Side:

```
import socket
import threading
import pickle
import os

HOST = socket.gethostbyname(socket.gethostname())
PORT = 9090

def server_init():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    print(f'Server active on {HOST}:{PORT}')
    print('Listening')
    server.listen()

    while True:
        conn, addr = server.accept()
        client_thread = threading.Thread(target=handle_client, args=(conn, addr))
        client_thread.start()

def list(conn):
    print('List')
    file_list = os.listdir('./server_files')
    print(file_list)
    if len(file_list)==0:
        response = 'No files available.'
    else:
        response = '\n'.join(file_list)

    conn.send(response.encode())

def upload(conn):
```



```

print('Upload')
conn.send('upload'.encode())

length = conn.recv(1024).decode()
conn.send('OK'.encode())

response = b''
got_len=0
while got_len<int(length):
    data = conn.recv(1024)
    response+=data
    got_len+=len(data)

response = pickle.loads(response)

if response['status'] == 'ERROR': return

filepath = os.path.join('./server_files',response['filename'])

with open(filepath,'wb') as f:
    f.write(response['content'])

print('Received')

def download(conn):
    print('Download')
    conn.send('download'.encode())
    filename = conn.recv(1024).decode()
    filepath = os.path.join('./server_files',filename)

    try:
        with open(filepath,'rb') as f:
            content = f.read()
            response = {
                'status': 'OK',
                'filename': filename,
                'content': content
            }
    except FileNotFoundError:
        print('File does not exist')
        response = {
            'status': 'ERROR',
        }

response = pickle.dumps(response)

```

```

conn.send(f'{len(response)}'.encode())
conn.recv(1024).decode()
conn.send(response)
conn.recv(1024).decode()

def handle_client(conn, addr):
    print(f'New connection from {addr}')

    options = '''Select one:
    1. List
    2. Upload
    3. Download'''
    try:
        while True:
            conn.send(options.encode())
            selected = conn.recv(1024).decode()
            print(selected)

            match selected:
                case '1':
                    list(conn)
                case '2':
                    upload(conn)
                case '3':
                    download(conn)
    except ConnectionAbortedError:
        print(f'{addr} disconnected.')
        # conn.recv(1024)

if __name__ == '__main__':
    server_init()

```

### 6.1.2 Client Side:

```

import socket
import pickle
import os
import tqdm

HOST = socket.gethostbyname(socket.gethostname())
PORT = 9090
sock: socket.socket

```

```

def list(sock):
    print('\nFiles:')
    response = sock.recv(1024).decode()
    print(f'{response}\n')

def upload(sock):
    response = sock.recv(1024).decode()
    print(response)
    filename = input('Enter filename: ')
    filepath = os.path.join('./client_files',filename)

    try:
        with open(filepath,'rb') as f:
            content = f.read()
        response = {
            'status': 'OK',
            'filename': filename,
            'content': content
        }
        success = True

    except FileNotFoundError:
        print('File does not exist')
        response = {
            'status': 'ERROR',
        }
        success = False

    response = pickle.dumps(response)
    sock.send(f'{len(response)}'.encode())
    sock.recv(1024).decode()
    sock.send(response)

    if success: print(f'Uploaded {filepath}')
    else: print('Try again.')

def download(sock):
    response = sock.recv(1024).decode()
    print(response)
    filename = input('Enter filename: ')
    sock.send(filename.encode())

    length = sock.recv(1024).decode()
    sock.send('OK'.encode())

```

```

response = b''
got_len = 0
with tqdm.tqdm(total=int(length), unit='B', unit_scale=True, unit_divisor=1024) as p:
    while got_len < int(length):
        data = sock.recv(1024)
        response += data
        got_len += len(data)
        progress_bar.update(len(data))

response = pickle.loads(response)
sock.send('RECV'.encode())

if response['status'] == 'ERROR':
    print('File does not exist')
    return

filepath = os.path.join('./client_files', filename)

with open(filepath, 'wb') as f:
    f.write(response['content'])

print(f'Downloaded {filepath}')

def client_init():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))
    try:
        while True:
            options = sock.recv(1024).decode()
            print(options)

            selected = input('-> ')
            sock.send(selected.encode())

            match selected:
                case '1':
                    list(sock)
                case '2':
                    upload(sock)
                case '3':
                    download(sock)
    except KeyboardInterrupt:
        print('Disconnected')

client_init()

```

## 6.2 Task 2

### 6.2.1 Server Side

```
import http.server
import socketserver
import os
import json

filepath = os.getcwd()+"/files"

PORT = 12349

def list(self):
    self.send_response(200)
    self.send_header('Content-type', 'application/json')
    self.end_headers()
    files = os.listdir(filepath)
    print(files)
    self.wfile.write(json.dumps(files).encode())

def download(self):
    filename = self.path[10:]
    try:
        self.send_response(200)
        self.send_header('Content-type', 'application/octet-stream')
        self.end_headers()
        with open(os.path.join('files', filename), 'rb') as file:
            self.wfile.write(file.read())
    except FileNotFoundError:
        self.send_error(404, "Sorry! Did not find the file!")

def upload(self):
    filename = self.path[8:]
    content_length = int(self.headers['Content-Length'])
    file_content = self.rfile.read(content_length)
    with open(os.path.join('files', filename), 'wb') as file:
        file.write(file_content)

    self.send_response(201)
    self.send_header('Content-type', 'text/plain')
    self.end_headers()
    self.wfile.write(b'Successfully uploaded the file!')

class FileHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/list':
```

```

        list(self)
    elif self.path.startswith('/download/'):
        download(self)
    else:
        super().do_GET()

    def do_POST(self):
        if self.path.startswith('/upload/'):
            upload(self)

if __name__ == "__main__":
    os.makedirs('files', exist_ok=True)
    handler = FileHandler

    with socketserver.TCPServer("", PORT), handler) as httpd:
        print(f"Hello client!")
        httpd.serve_forever()

```

### 6.2.2 Client Side:

```

import requests
import os

server_url = 'http://172.29.96.128:12349'

while True:
    options = '''Select one :
    1. list
    2. upload
    3. download
    '''

    print(options)

    selected_operation = input()

    match selected_operation:
        case '1': #list
            response = requests.get(f'{server_url}/list')
            files = response.json()
            print("Available Files : ")
            for file in files:
                print(file)

        case '2': #upload

```

```

filename = input('Enter the filename: ')
with open(filename, 'rb') as file:
    response = requests.post(f'{server_url}/upload/{filename}', data=file)
print(response.text)

case '3': # download
    filename = input('Enter the filename: ')
    response = requests.get(f'{server_url}/download/{filename}')
    if response.status_code == 200:
        with open(filename, 'wb') as file:
            file.write(response.content)
        print('Successfully downloaded the file!')
    else:
        print('Sorry! Failed to download file!')

```

## References

- [1] Multithreading in python. *GeeksforGeeks*, aug 3 2022. [Online; accessed 2023-01-25].
- [2] File transfer using tcp socket in python. *GeeksforGeeks*, apr 17 2023. [Online; accessed 2023-01-25].
- [3] Python Basics. Create a python web server. aug 31 2021.