



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2: Implementing File transfer using Socket
Programming and HTTP GET/POST requests

Submitted By:

Name : Abdullah Al Mahmud

Roll No : 15

Name : Zisan Mahmud

Roll No : 23

Submitted On:

February 7, 2024

Submitted To:

Dr. Md. Abdur Razzaque

Dr. Md Mamunur Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

In this lab, our main goal is to send files using socket programming and HTTP GET/POST requests. Operating within this regulated environment, our central aim is to acquire a detailed understanding of the principles and practices related to establishing connections between clients and servers. We will explore how these technologies can be leveraged to implement a file transfer system. We will delve into the intricacies of these techniques, understand their strengths and constraints, and discover how they can be synergistically applied for effective file transmission.

2 Objectives

The primary aim of this lab is to provide practical experience with socket programming and HTTP file transfer. In this lab, we will:

- Design and implement a Multi-threaded Chat System.
- Set Up an HTTP Server.
- Analyze the Use of GET and POST Methods.

3 Theory

Multithreading is a programming model that allows multiple threads to exist within the context of single process, sharing the process's resources but able to execute independently. In a multithreaded chat system, each client connection is handled in a separate thread. This allows server to handle multiple clients simultaneously. When a new connection is established, a new thread is spawned, and the communication with the client is moved to the thread. Socket programming is a method of communicating between server and clients using a network protocol, typically TCP/IP. It involves creating socket, which are endpoints for sending and receiving data across the network. File transfer using socket programming involves sending and receiving data over a network using socket. The server creates a socket and binds it with a specific port number, then it listens for incoming client connection. The client also creates a socket and connect to the server. Each client connection is handled in a separate thread, allowing the server to handle multiple clients simultaneously. After that, the clients send request (a simple request containing the name of the file) to the server. The server

receives the request, reads the requested file into a byte stream, and sends this byte stream over the socket to the client. The client receives the byte stream and writes it to a local file. This effectively completes the file transfer. HTTP (Hypertext Transfer Protocol) is a protocol used for transmitting hypertext over the Internet. It specifies the structure and transmission of messages as well as the responses that Web servers and browsers should give to different commands. It supports many methods to do any task on the server or to receive any data from a server. Among this methods, GET and POST methods are prominent methods of HTTP. GET method is used to appends form data to the URL in name or value pair. It limits the length of the URL and help user to submit the bookmark or result. It is useful for the data that does not require any security. It could be used to download objects from the server. POST is a method that is supported by HTTP and depicts that a web server accepts the data included in the body of the message. It could be used to upload objects to the server. These methods allow for the transfer of data between the client and server, facilitating the exchange of information.

4 Methodology

4.1 Socket Programming

- **Setting up the server:** First we have to set up the server that listens for incoming connections on a specific port. This involves creating socket, binding it to a specific IP address and port number, and then setting it to listen for incoming connections.
- **Setting up the client:** Here, we will create a socket and connect it with the server address and port.
- **File transfer:** If the connection is established, the clients send request for a file to the server. The server then reads the file name, locate the file on disk, read the file into a byte stream and send the byte stream to the client over the socket. The client will receive the byte stream and write it to a local file and save it.

4.2 File Transfer via HTTP

- **Setting up the HTTP server:** First, we will set up an HTTP server with few objects/files that can be accessed by GET and POST requests.

- **GET requests:** GET request is used to download a file from the server. The server will respond with the requested file in the body of the HTTP response.
- **POST requests:** POST request is used to upload a file to the server. The file will be included in the body of the HTTP POST request.

5 Experimental result

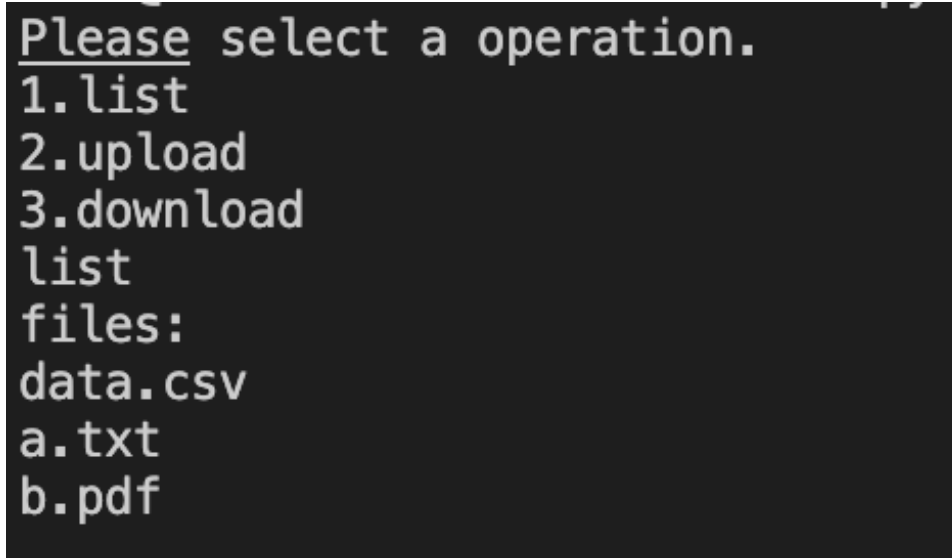
5.1 Task 1: File Transfer via Socket Programming

Server: The Server is responsible for showing the list of files to the clients, send or receive any file from the clients. Server can handle multiple clients at a time. The below image shows the different types of activities that the server can perform.

```
araf@Arafs-MacBook-Air Lab3 % python3 FileServer.py
Server is listening...
Accepted connection from ('127.0.0.1', 57654)
File list sent to ('127.0.0.1', 57654)
uploading from ('127.0.0.1', 57654)
File not found
uploading from ('127.0.0.1', 57654)
Saving file table.csv
File table.csv uploaded successfully from ('127.0.0.1', 57654)
Accepted connection from ('127.0.0.1', 57657)
downloading to ('127.0.0.1', 57657)
File a.txt downloaded successfully to ('127.0.0.1', 57657)
□
```

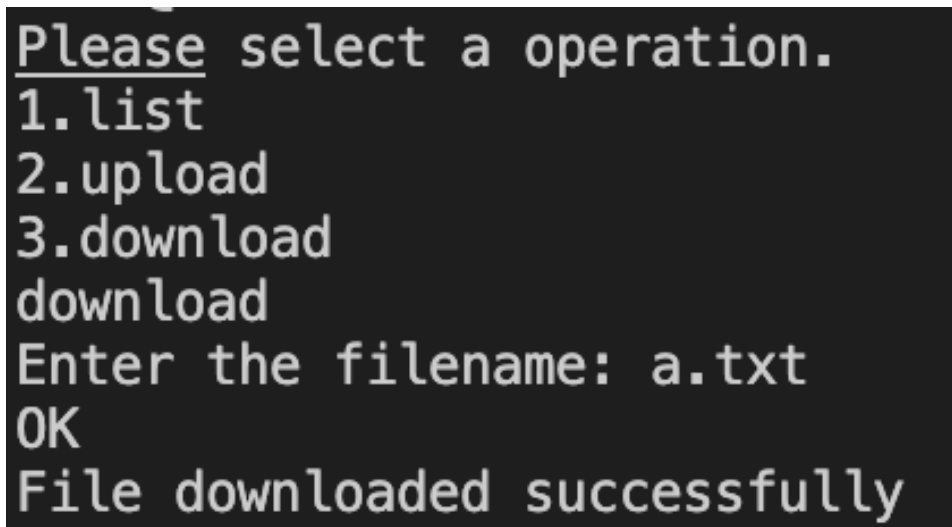
Figure 1: Responses from Server

Client: In the below images different types of operations by different clients are shown:

A terminal window with a black background and white text. The text shows a prompt to select an operation, followed by a list of operations (list, upload, download), and then a list of files (data.csv, a.txt, b.pdf).

```
Please select a operation.  
1.list  
2.upload  
3.download  
list  
files:  
data.csv  
a.txt  
b.pdf
```

Figure 2: List of files shown on Client

A terminal window with a black background and white text. The text shows the same operation selection menu as Figure 2, but with 'download' selected instead of 'list'. It then prompts for a filename, which is 'a.txt', and shows a confirmation message 'File downloaded successfully'.

```
Please select a operation.  
1.list  
2.upload  
3.download  
download  
Enter the filename: a.txt  
OK  
File downloaded successfully
```

Figure 3: Successful File download from Client

```
Please select a operation.  
1.list  
2.upload  
3.download  
download  
Enter the filename: table.exe  
File does not exist
```

Figure 4: File is not available on Server

```
Please select a operation.  
1.list  
2.upload  
3.download  
upload  
Enter the filename: table.csv  
OK  
File uploaded successfully
```

Figure 5: Client successfully uploaded a file on Server

```
Please select a operation.
1.list
2.upload
3.download
upload
Enter the filename: table.txt
OK
File not found
```

Figure 6: File is not available on Client so can't be uploaded

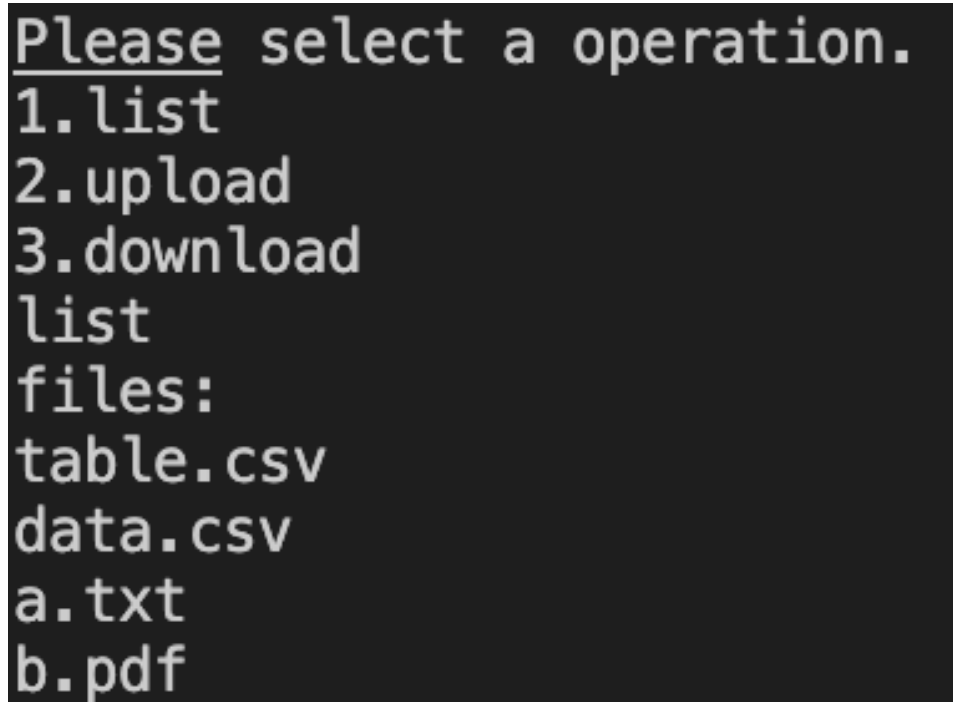
5.2 Task 2: File Transfer via HTTP

Server: The HTTP server is capable of handling GET requests to display a list of files, retrieve specific files, and POST requests to upload new files. GET requests to the `/list` endpoint return a list of files available in the server's directory. GET requests to `/download/` followed by a file name retrieve the specified file if it exists, otherwise, it returns a 404 error. POST requests on `/upload/` endpoint allow clients to upload files to the server. The server saves the uploaded file to its directory and responds with a 200 OK status code upon successful upload.

```
Serving on port 12349
127.0.0.1 - - [06/Feb/2024 21:43:05] "GET /list HTTP/1.1" 200 -
['table.csv', 'data.csv', 'a.txt', 'b.pdf']
127.0.0.1 - - [06/Feb/2024 21:43:39] "POST /upload/c.pdf HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2024 21:44:28] "GET /download/5.pdf HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2024 21:44:28] code 404, message File not found
127.0.0.1 - - [06/Feb/2024 21:44:28] "GET /download/5.pdf HTTP/1.1" 404 -
□
```

Figure 7: HTTP Server

Client: Client can do HTTP request to server to see the list of the files, can download a file through GET request and can upload a file using POST request.

A terminal window with a dark background and light gray text. The text is as follows:

```
Please select a operation.  
1.list  
2.upload  
3.download  
list  
files:  
table.csv  
data.csv  
a.txt  
b.pdf
```

Figure 8: File list from HTTP Server


```
Please select a operation.  
1.list  
2.upload  
3.download  
upload  
Enter the filename: c.pdf  
File uploaded successfully
```

Figure 9: Uploading file to HTTP Server

```
Please select a operation.  
1.list  
2.upload  
3.download  
download  
Enter the filename: 5.pdf  
File downloaded successfully
```

Figure 10: File download from HTTP Server

6 Experience

1. We learned how the file is transferred using socket and HTTP
2. We faced problem while listing the files in the server, then we used beautiful soap which is a Python library for pulling data out of HTML and XML files.
3. We encountered some real world problems during this experiment which provide an opportunity to debug and troubleshoot issues.
4. Finally, we gained insights into the complexities involved in file transferring between clients and servers.

References

- [1] GeeksforGeeks, *Introducing Threads in Socket Programming*, Available at: <https://www.geeksforgeeks.org/introducing-threads-socket-programming-java/>
- [2] DataCamp, *Python GET and POST request*, Available at: <https://www.datacamp.com/tutorial/making-http-requests-in-python>
- [3] Python Documentation, *HTTP server*, Available at: <https://docs.python.org/3/library/http.server.html>
- [4] Codexpedia, *Python web server for GET and POST*, Available at: <https://www.codexpedia.com/python/python-web-server-for-get-and-post-requests/>
- [5] GeeksforGeeks, *File Transfer using TCP Socket in Python*, Available at: <https://www.geeksforgeeks.org/file-transfer-using-tcp-socket-in-python/>

A Code

A.1 Task 1

File Transfer via Socket Programming

Server

```
import socket
import threading
import os
import time

# Set the server's IP address and port
server_ip = 'localhost'
server_port = 12347

server_path = '/files'

cwp_path = os.getcwd()

os.makedirs(cwp_path+server_path, exist_ok=True)

# Create a socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the server's IP address and port
server_socket.bind((server_ip, server_port))

# Listen for a connection
server_socket.listen(1)
print('Server is listening...')

def handle_client(client_socket, client_address):
    while True:
        # Receive the operation
        operation = client_socket.recv(1024).decode()

        if operation == 'list':
            # Send the list of files
            files = os.listdir(cwp_path+server_path)
            if len(files) == 0:
                client_socket.send('There is no files in the server'.encode())
            else:
                send_data = 'files:\n'
                for file in files:
                    send_data += file + '\n'
                client_socket.send(send_data.encode())
                print(f"File list sent to {client_address}")
        elif operation == 'upload':
            # Receive the filename
            filename = client_socket.recv(1024).decode()

            if os.path.exists(cwp_path+server_path+'/'+filename):
                client_socket.send('File already exists'.encode())
            else:
                client_socket.send('OK'.encode())
                print(f"uploading from {client_address}")

                res = client_socket.recv(1024).decode()
                if res == 'File not found':
                    print('File not found')
                    continue
            # Receive the file size
            file_size = int(res)
```

```

        with open(cwpath+server_path+'/'+filename, 'wb') as file:
            print(f'Saving file {filename}')
            received_size = 0
            while received_size < file_size:
                file_data = client_socket.recv(1024)
                received_size += len(file_data)
                file.write(file_data)

            print(f"File {filename} uploaded successfully from {client_address}")

            client_socket.send('File uploaded successfully'.encode())

    elif operation == 'download':
        # Receive the filename
        filename = client_socket.recv(1024).decode()

        if (os.path.exists(cwpath+server_path+'/'+filename)):
            client_socket.send('OK'.encode())
            print(f"downloading to {client_address}")
            file_size = os.path.getsize(cwpath+server_path+'/'+filename)
            client_socket.send(str(file_size).encode())
            with open(cwpath+server_path+'/'+filename, 'rb') as file:
                while True:
                    file_data = file.read(1024)
                    if not file_data:
                        break
                    client_socket.send(file_data)
            print(f"File {filename} downloaded successfully to {client_address}")
        else:
            client_socket.send('File does not exist'.encode())
    else:
        client_socket.send('Invalid operation'.encode())

    time.sleep(0.5)

while True:
    # Accept a connection
    try:
        client_socket, client_address = server_socket.accept()
        print(f'Accepted connection from {client_address}')

        # Create a new thread to handle this client
        threading.Thread(target=handle_client, args=(client_socket, client_address,)).start()
    except KeyboardInterrupt:
        server_socket.close()
        break

```

Client

```

import socket
import os

# Set the server's IP address and port
server_ip = 'localhost'
server_port = 12347

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect((server_ip, server_port))

while True:
    # Receive the operation prompt from the server
    operation_prompt = "Please select a operation.\n1. list\n2. upload\n3. download"
    print(operation_prompt)

```

```

# Send the operation to the server
operation = input()
client_socket.send(operation.encode())

if operation == 'list':
    # Receive and print the list of files
    files = client_socket.recv(1024).decode()
    print(files)
elif operation == 'upload':
    # Send the filename to the server
    filename = input('Enter the filename: ')
    client_socket.send(filename.encode())

    response = client_socket.recv(1024).decode()
    print(response)

    if response == 'OK':
        # Send the file size to the server
        if os.path.exists(os.getcwd()+ '/' + filename):
            file_size = os.path.getsize(filename)
            client_socket.send(str(file_size).encode())
            # Send the file to the server
            with open(filename, 'rb') as file:
                while True:
                    bytes_to_send = file.read(1024)
                    if not bytes_to_send:
                        break
                    client_socket.send(bytes_to_send)
            print(client_socket.recv(1024).decode())
        else:
            print('File not found')
            client_socket.send('File not found'.encode())
elif operation == 'download':
    # Send the filename to the server
    filename = input('Enter the filename: ')
    client_socket.send(filename.encode())

    response = client_socket.recv(1024).decode()
    print(response)

    if response == 'OK':
        # Receive the file from the server

        file_size = int(client_socket.recv(1024).decode())

        with open(filename, 'wb') as file:
            received_size = 0
            while received_size < file_size:
                bytes_received = client_socket.recv(1024)

                received_size += len(bytes_received)

                file.write(bytes_received)
            print('File downloaded successfully')
    else:
        msg = client_socket.recv(1024).decode()
        print(msg)

# Close the socket
client_socket.close()

```

A.2 Task 2

File Transfer via HTTP

Server

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

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

PORT = 12349

class FileHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/list ':
            self.send_response(200)
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            files = os.listdir(PATH)
            print(files)
            self.wfile.write(json.dumps(files).encode())
        elif self.path.startswith('/download/'):
            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, "File not found")
        else:
            super().do_GET()

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

if __name__ == "__main__":
    os.makedirs('files', exist_ok=True)
    handler = FileHandler
    with socketserver.TCPServer(("", PORT), handler) as httpd:
        print(f"Serving on port {PORT}")
        httpd.serve_forever()
```

Client

```
import requests
import os
from bs4 import BeautifulSoup

server_url = 'http://localhost:12349'

while True:
    operation_prompt = "Please select a operation.\n1.list\n2.upload\n3.download"
    print(operation_prompt)

    operation = input()

    if operation == 'list':
        response = requests.get(f'{server_url}/list ')
        files = response.json()
        print("files:")
        for file in files:
            print(file)
    elif operation == '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)
    elif operation == '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('File downloaded successfully ')
        else:
            print('Failed to download file ')
```