Department of
Computer Science and Engineering

University of Dhaka

Title: Implementing File Transfer using Socket Programming and HTTP GET/POST requests

CSE 3111: Computer Networking Lab

Batch: 29/3rd Year 1st Semester 2025

1 Objective(s)

- To give hands-on experience with socket programming and HTTP file transfer.
- To set up an HTTP server process with a few objects.
- Use GET and POST methods to upload and download objects between HTTP clients and a server.

2 Background Theory

2.1 File Transfer Using Socket Programming

A socket is mainly the door between the application process and the end-to-end transport protocol. Java Socket programming can be connection-oriented or connectionless. The main problem of the simple two-way socket programming is that it can not handle multiple client requests at the same time. A server can only provide service to the client that comes first to connect with the server. The other clients can not connect to that server. To resolve this problem, the server opens different threads for each client, and every client communicates with the server using that thread.

File transfer can be implemented by reading the file into a byte stream at the sender side and writing it to disk at the receiver side.

1. TCP is used for reliable, ordered, and error-checked delivery of data.
2. The Server Socket listens for incoming client connections.
3. The Client Socket connects to the server's IP and port.
4. Files are sent as binary streams using Input and Output streams.

A visual flowchart or diagram for the file transfer process using Socket programming is as follows

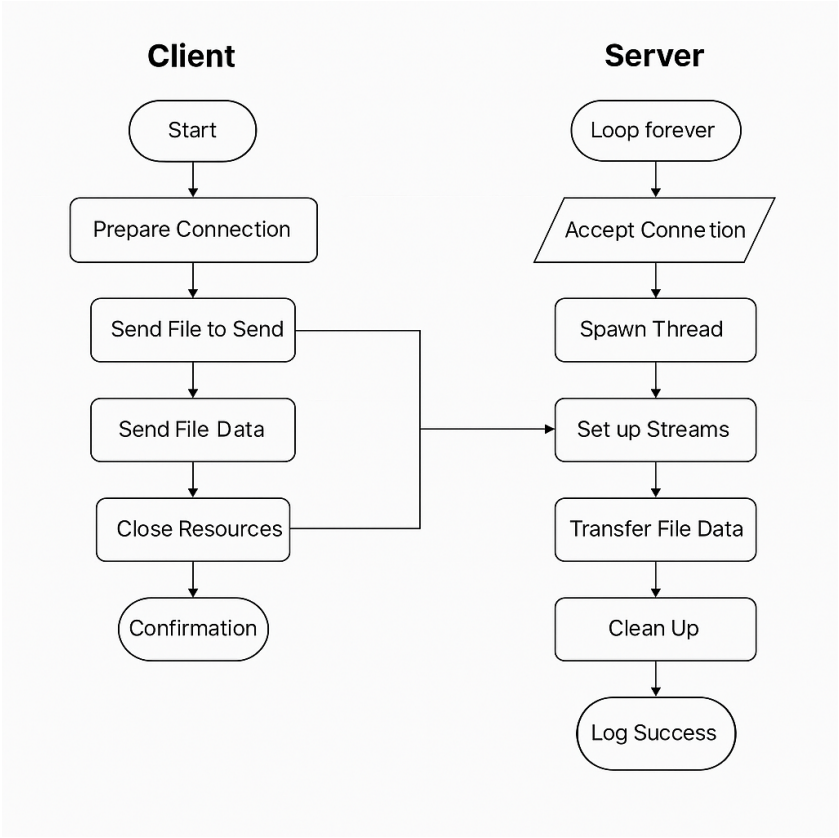

Figure 1: Visual Flow chart for the file transfer process

2.2 HTTP File Transfer

HTTP (HyperText Transfer Protocol) is a request-response protocol used between clients and servers. In the context of file transfer:

- HTTP GET is used to download files from a server.
- HTTP POST is used to upload files to a server.

HTTP File Transfer is used because it is Platform and language-independent and has built-in support in Java (via *HttpURLConnection* and *HttpServer*). It is also easy to integrate with web clients or REST APIs.

3 Lab Task 1 (Please implement yourself and show the output to the instructor)

1. Implement a simple file server that listens for incoming connections on a specified port. The server should be able to handle multiple clients simultaneously.
2. When a client connects to the server, the server should prompt the client for the name of the file they want to download.
3. The server should then locate the file on disk and send it to the client over the socket.
4. Implement a simple file client that can connect to the server and request a file. The client should save the file to disk once it has been received.

3.1 Problem analysis

The detailed algorithms of the server-side and client-side connections are given in Algorithm 1 and 2.

Algorithm 1: Algorithm of server-side socket connection for sending requested files.

1: Create a ServerSocket object, namely handshaking socket, which takes the port number as input
2: Create a plain Socket, namely a communication socket object that accepts client requests
3: When a connection is received, create a new instance of a custom Thread class (e.g., ClientHandler) with the client Socket.
4: Call start() method on the thread to begin execution
5: Inside ClientHandler.run()– a) Create a BufferedReader to receive text data (file name) from the client. b) Create a BufferedWriter to send text responses (messages) to the client: c) Create a DataOutputStream to send the file size and binary file content.
6: Send a prompt to the client: "Enter the file name you want to download:"
7: Read File Name from Client
8: Check if File exists on Server a) Create a File object with the received file name. b) Check if the file exists and is a regular file.
9: IF File is found, then a) send confirmation to the client. b) Send the file size to the client (to help with download). c) Open a FileInputStream to read the file bytes. d) Read file content in chunks using a buffer and send it through the DataOutputStream:
10: If the File is not found, then send a "NOT_FOUND" message.
11: Close streams and all types of connections. Print a log message indicating the file was successfully received.

Algorithm 2: Algorithm of Client Side Socket Connection

1: Create a Socket object which takes the IP address and port number as input.
2: Receive the server prompt asking for the file name.
3: Input the desired file name and send it to the server.
4: If the received response from the server is "FOUND"– a) Read the file size. b) Prepare to save the file locally using FileOutputStream and save it as downloaded_<filename>. c) Read the file content from the socket and
5: If response is "NOT_FOUND", then display a message that the file doesn't exist on the server.
6: Close the connection and exit.

4 Lab Task 2 (Please implement yourself and submit as a lab report)

1. Implement a simple HTTP file server that listens for incoming connections on a specified port. The server should be able to handle multiple clients simultaneously.
2. When a client sends a GET request to the server with the path of the file they want to download, the server should locate the file on disk and send it to the client with the appropriate HTTP response headers. The client should save the file to disk once it has been received.
3. When a client sends a POST request to the server with a file, the server saves it.

4.1 Problem Analysis

4.1.1 SERVER-SIDE ALGORITHM (HTTP File Server)

1. Initialize the Server
   - Set a port (e.g., 8080).
   - Create an instance of HttpServer.
   - Define two contexts:
     - /download → for serving files.
     - /upload → for receiving files.
   - Assign each context a handler.
   - Set a thread pool executor to handle multiple clients.
   - Start the server.
2. Handle GET Requests – File Download
   - Check if the method is GET. If not, respond with HTTP 405 (Method Not Allowed).
   - Extract filename from query parameters (e.g., /download?filename=example.txt).
   - Check if the file exists on the server disk:
     - If not:
       * Set status to 404 (Not Found).
       * Send "File Not Found" response.
     - If found:
       * Set response headers as follows: Content-Type: application/octet-stream and Content-Disposition: attachment; filename="filename"
       * Send HTTP 200 OK with file size as content length.
   - Open a FileInputStream for the file.
   - Write the file bytes to the response output stream in chunks.
3. Handle POST Requests – File Upload
   - Check if the method is POST. If not, respond with HTTP 405.
   - Create a new file to save the uploaded data. Name it upload_<timestamp> or use another naming convention.

- Open:
  - InputStream from HttpExchange (client request body).
  - FileOutputStream for the new file.
- Read data in chunks from the input and write to the file output.
- After saving:
  - Send HTTP 200 OK.
  - Respond with a confirmation message including the filename.

4.1.2 CLIENT-SIDE ALGORITHM (HTTP File Client)

1. Upload File (HTTP POST)
   - Ask the user to input the file path to upload.
   - Create a URL object for http://server_ip:8080/upload.
   - Open a HttpURLConnection.
     - Set method to POST.
     - Set doOutput(true) to enable writing to the output stream.
   - Open:
     - FileInputStream for the local file.
     - Output stream from the connection.
   - Read from the file and write to the connection output stream in chunks.
   - Close streams.
   - Read response from server and display upload confirmation.
2. Download File (HTTP GET)
   - Ask the user to input the filename to download.
   - Create a URL object for http://server_ip:8080/download?filename=<filename>.
   - Open a HttpURLConnection and Set method to GET.
   - If response code is 200:
     - Open Input stream from the connection.
     - Open a FileOutputStream to save the file locally.
   - Read from input and write to output in chunks.
   - Close all streams.
   - Notify user of successful download.
   - If response code is 404–Display "File not found" message.

5 Policy

Copying from the Internet, classmates, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.