

## **Part 1 : Socket Programming**

### **HTTP Client and Server:**

The client accepts the command line arguments as per the below syntax:

**httpclient hostname port command filename**

In response to a GET/PUT command from the client, the below sequence of actions are performed on Client and Server:

Client initiates a connection to the server using TCP on any port mentioned by the user and passes the necessary arguments to GET/PUT a required file from/on the server as mentioned in the above syntax.

The server then establishes connection to the client on the incoming port and socket and identifies the authenticity of the type of request. Based on the command (GET/PUT) executed in the client, performs the necessary steps and then closes the connection.

In case of GET operation, any file that is being fetched by the client from the server is read and displayed on the console on the client end. A screenshot (fig.a) has been attached depicting the same process on the client and server.

In case of PUT operation, any file that is being put on the server by the client is read by the server and locally copied in the local path of server (Serverfiles in this case). A screenshot (fig.b) has been attached depicting the same process on the client and server.

### **Steps How to Execute:**

1. Compile the Httpclient and Httpserver using commands: `javac httpclient.java; javac httpserver.java`.
2. Start the server using the command: `java httpserver 5678` (any port number as required).
3. Execute the command to get/put from the client end as below:

```
java httpclient <hostserver> 5678 GET test.txt
```

Here the executable is HttpSocketClient and HttpSocketServer.

4. The server and client display the necessary responses based on the success or failure to execute the command (200 OK or 404 Not Found messages).

5. Once the command is successful, check for necessary files in the local path (clientfiles/serverfiles) to verify if the files are created successfully.

(fig.a) GET request:

The screenshot shows a Notepad++ editor with a Java file named `HttpSocketServer.java`. The code is a simple HTTP server that listens on port 5678. It handles GET requests by sending a 200 OK response and the content of a file named `test.txt`. The Command Prompt window shows the server running and receiving a GET request for `test.txt`, returning a 200 OK response and the content of the file.

```

1 import java.io.*;
2 import java.net.*;
3
4 public class HttpSocketServer {
5     public static void main(String[] args) {
6         try {
7             ServerSocket serverSocket = new ServerSocket(5678);
8             System.out.println("The server is running.");
9             while (true) {
10                 Socket socket = serverSocket.accept();
11                 System.out.println("New connection with client# 0 at Socket[addr=/127.0.0.1,port=58907,localport=5678]");
12                 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
13                 String request = in.readLine();
14                 System.out.println("Type of Request: GET /test.txt HTTP/1.1");
15                 File file = new File("test.txt");
16                 if (file.exists()) {
17                     FileInputStream fis = new FileInputStream(file);
18                     DataInputStream dis = new DataInputStream(fis);
19                     byte[] buffer = new byte[1024];
20                     int read;
21                     while ((read = dis.read(buffer)) != -1) {
22                         socket.getOutputStream().write(buffer, 0, read);
23                     }
24                     socket.getOutputStream().flush();
25                     System.out.println("HTTP/1.1 200 OK");
26                     System.out.println("this is the test for CN socket programming!!!");
27                 } else {
28                     System.out.println("File not found");
29                 }
30                 in.close();
31                 socket.close();
32             }
33         } catch (IOException e) {
34             e.printStackTrace();
35         }
36     }
37 }
38
39 /*
40  * In response to a GET command,
41  * 2. submit a valid HTTP/1.1 GET
42  */
43
44 pw.print("GET /" + filename +
45         " HTTP/1.1 200 OK\r\n");
46 pw.print("Accept: text/plain,\r\n");
47 pw.print("\r\n");
48 pw.flush();
49
50 //Create a file on client to
51 String path = System.getProperty("user.dir") + "\\serverfiles\\test.txt";
52 File file = new File(path);
53 BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file));
54
55 /*
56  * In response to a GET command, the client must:
57  */
58 }
59
60 }

```

Command Prompt Output:

```

C:\GoogleDrive\UNCC\S16_CNN\project\1\socket_prog>java HttpSocketServer localhost 5678 get test.txt
TCP Connection Established to localhost:5678
HTTP/1.1 200 OK
this is the test for CN socket programming!!!
C:\GoogleDrive\UNCC\S16_CNN\project\1\socket_prog>

```

(fig.b) PUT request:

The screenshot shows two Command Prompt windows. The left window shows the execution of `httpclient.java` to send a PUT request to a server on port 5455. The right window shows the execution of `httpserver.java` which receives the PUT request and saves the file `test.txt` in the `serverfiles` directory.

```

F:\Spring 16\CN\Project\Bkp\socket_prog>javac httpclient.java
F:\Spring 16\CN\Project\Bkp\socket_prog>java httpclient localhost 5455 put test.txt
TCP Connection Established to localhost:5455
HTTP/1.1 200 OK File Created
F:\Spring 16\CN\Project\Bkp\socket_prog>

```

```

F:\Spring 16\CN\Project\Bkp\socket_prog>javac httpserver.java
F:\Spring 16\CN\Project\Bkp\socket_prog>java httpserver 5455
The server is running.
New connection with client# 0 at Socket[addr=/127.0.0.1,port=55161,localport=5455]
Type of Request: PUT /test.txt HTTP/1.1
Accept: text/plain, text/html, text/*
this is the test for CN socket programming!!!
File Copied at F:\Spring 16\CN\Project\Bkp\socket_prog\serverfiles\test.txt
Closing Socket...
Connection with client# 0 closed

```