**COMPUTER COMMUNICATION AND NETWORKS**

**ITCS 6166/8166**

**HTTP CLIENT AND SERVER**

**SUBMITTED BY:**

**Aarti Nimhan**

**IMPLEMENTATION OF HTTP CLIENT AND SERVER:**

We have implemented an HTTP Client and Server that runs a simplified version of HTTP/1.1. We have implemented this project using JAVA language and we have supported two HTTP methods: GET and PUT.

GET METHOD: In this method, the client requests data from the server. Various steps involved in it are listed below:

1. Firstly, the client establishes a TCP connection with the server.
2. After establishing a successful connection, a valid GET Request is being sent to the server.
3. Then the server checks if the requested file is present or not and would respond to the request accordingly. If the server finds the requested file it sends 200 OK in the response header and the file in the body. If the server does not find the requested file it sends 404 Not Found in the response. The client would read the response with the status 200 and save the file received in the response message locally. The client displays the response 200 or 404 on the console.

PUT METHOD: In this method, the client sends files to the server. Following steps are involved in this method:

1. Firstly, the connection is being established to the server via a TCP connection.
2. The client will send the file in the form of bytes in the body of the request message to the server. The server after receiving the request, extracts the file in the form of bytes from the request message and saves the file in its directory. The server then sends a 200 OK response to the client.

**IMPLEMENTATION IN USING JAVA LANGUAGE:**

To implement HTTP Client-Server project we created the following classes:

1. MyClient.java acts as the HTTP Client.
2. HTTPServer.java acts as the HTTP Server.
3. HttpClientHandler.java thread which handles the client request.
4. CommonUtil.java is a class for utility methods.

**Note:**

The server is multithreaded and on receiving the termination signal (Ctrl+C) , it shuts down gracefully and closes all the sockets before exiting.

**HTTP CLIENT IMPLEMENTATION:**

* We have created a JAVA file named “MyClient.java”.
* We are taking the parameters as command line arguments in the below mentioned format:

*MyClient HostName Port Method FileName*

* The program will check whether the input entered by the user is the given format or not and would display the error message in case it is not correct.
* Client then creates a new socket connection for the server by passing hostname and port as the arguments.
* If the method entered is GET, the client requests for a file from the server. When it receives a 200 OK response from the server it extracts the file from the response stream and creates a file and puts the contents from the response into this file in the form of bytes. It prints the response status from the server is on the console.
* The Client uses java.io.DataInputStream to read the file in the form of bytes and writes the file locally using java.io.FileOutputStream .
* The Client uses java.io.PrintStream to send the request to the server.
* While in case of PUT method, the client requests to save a file into the servers directory. This file is read and sent in the form of a byte stream along with the PUT request. The response from the server is printed on the console.
* The client reads the file in the form of bytes using Files.readAllBytes from java.nio.file.Files and writes it in the output stream using java.io.PrintStream.

**HTTP SERVER IMPLEMENTATION:**

* We have created a JAVA files named HTTPServer.java and HttpClientHandler.java
* Start the server by entering the command:

*HTTPServer Port*.

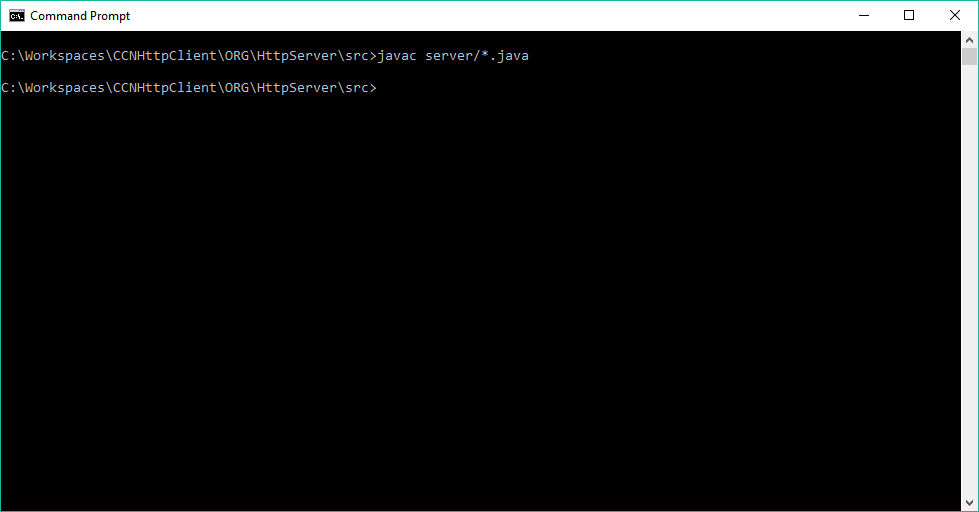
* Creating socket with specified port number, the server starts listening for requests from clients.
* Calling serverSocket.accept() the server accepts a request from the client and creates a new thread using HttpClientHandler for each request. This thread is responsible for processing the client request.
* The HTTP Server continuously listens for connections from clients.
* Once a request is received at the HttpClientHandler, it first checks if the Server Directory “ServerDir” is created, if not it will create it.
* It then reads the DataInputStream and extracts the httpRequestMethod and path of the file requested.
* If the Client requests a GET method, file is searched in the "ServerDir" directory and all its child directories. If tile is present the server reads the file in the form of bytes using Files.readAllBytes from java.nio.file.Files and writes it in the output stream using java.io.PrintStream. It sends 200 OK and file in response.
* If the file is not found it sends 404 Not Found in response.
* If the Client requests a PUT, the server will use java.io.DataInputStream to read the file in the form of bytes and writes the file into the “ServerDir” using java.io.FileOutputStream method. It then sends 200 OK File created successfully to the client in response.
* Any request other than this is replied with 301 Bad Request.
* The client connection is closed.
* On termination with Ctrl-C from the Command Line the HTTP Server gracefully closes all the sockets and Shuts down. It uses addShutdownHook to do this.

**Note:**

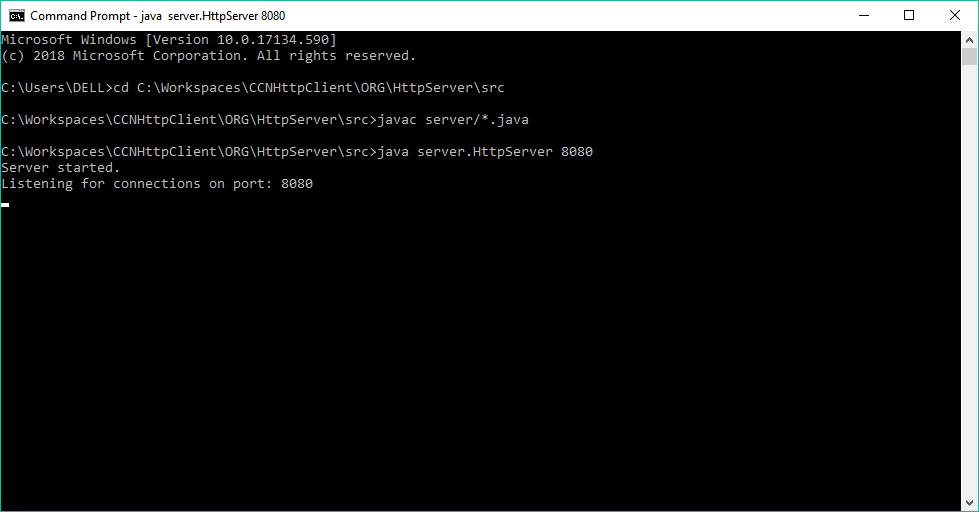
* This application works with client and server running on different machines.
* It works for text, html and .jpg images as well.
* It works with relative paths as well.

**Screenshots:**

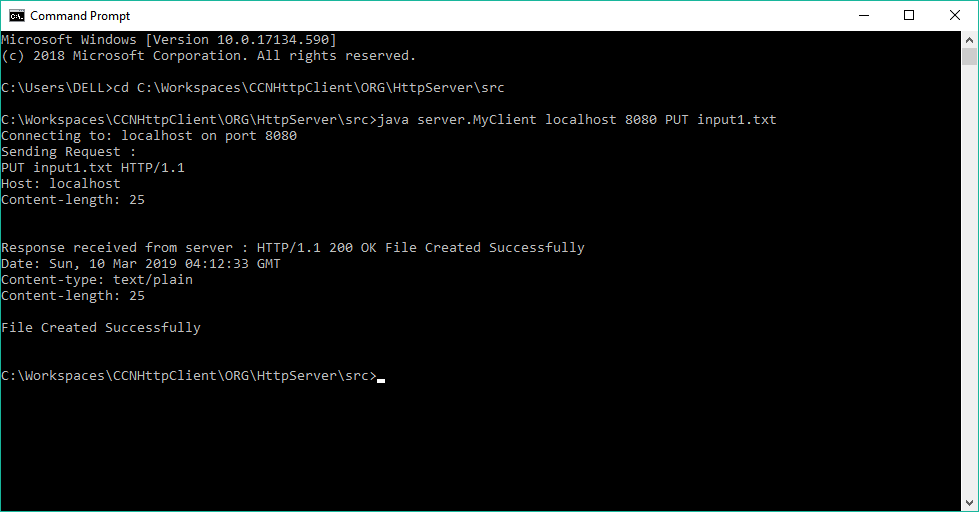
**1.Compiling the server**

****

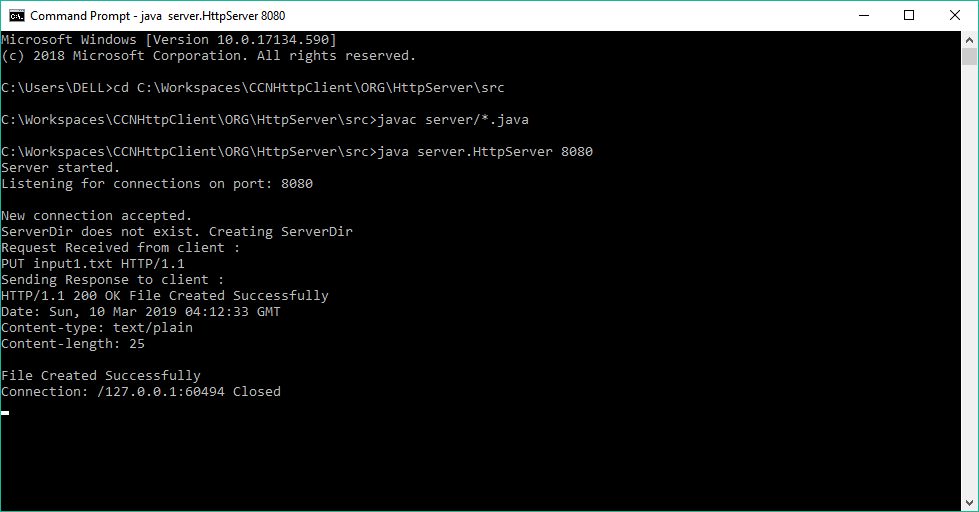
**2.Starting the Server**

****

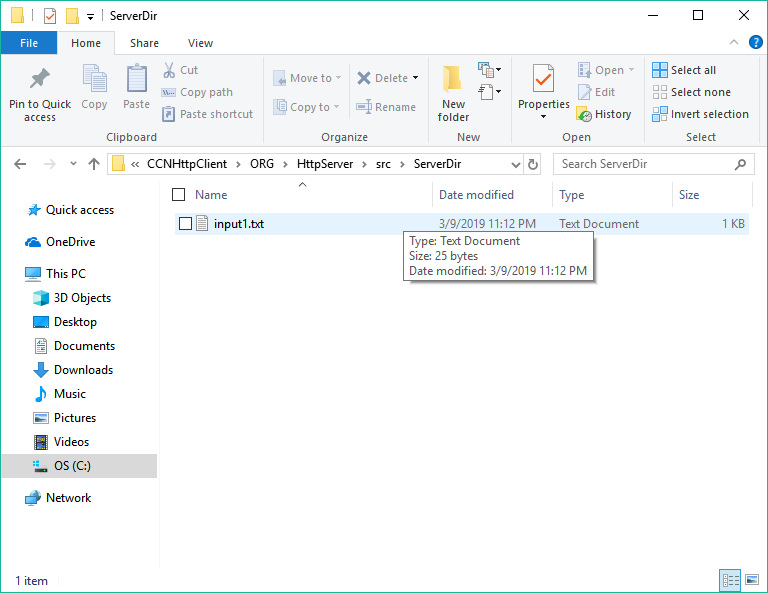
1. **Sending PUT Request to Server from Client. Response received at Client**

****

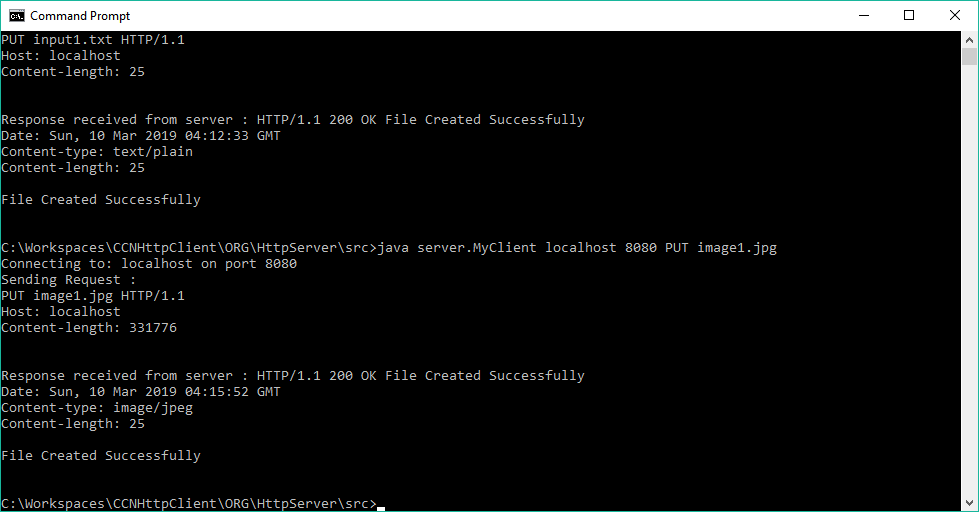
**4. Server Sending response to PUT Request. The server is still in Listening mode to accept more requests.**



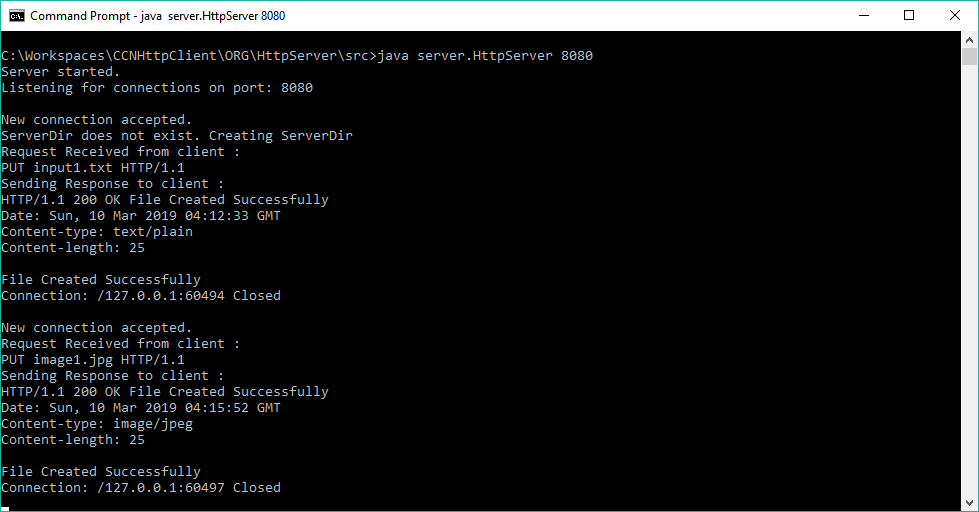
5. File Received and Stored locally by Server.



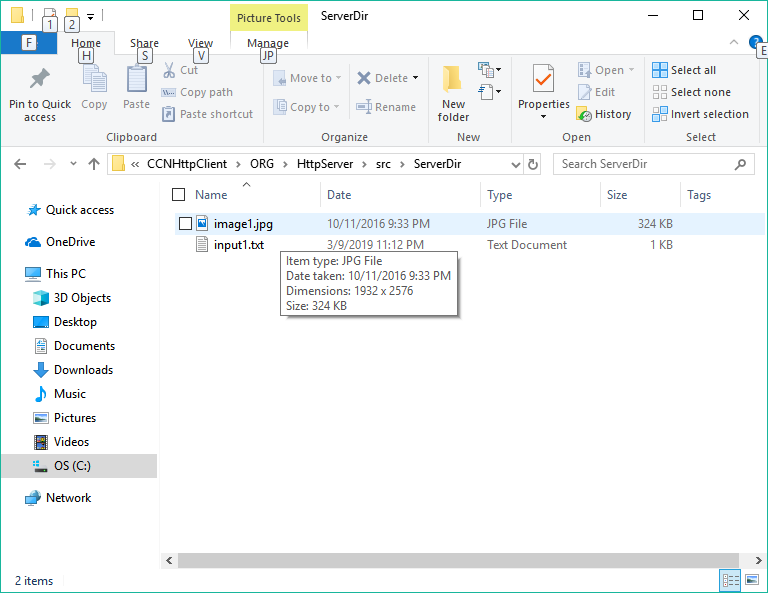
1. **Client Sending PUT request to server With an Image file and receiving successful response.**



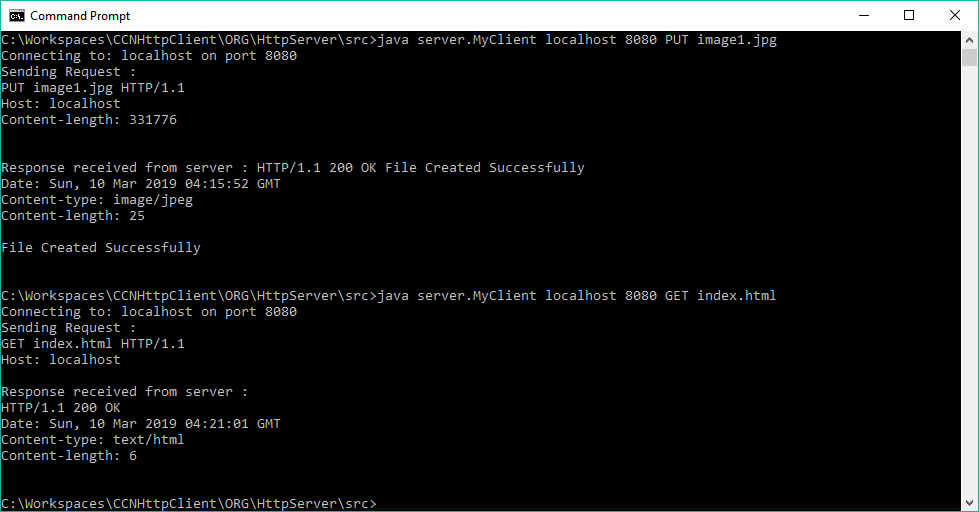
1. **Server Side Response . Server continues to listen for more requests.**

****

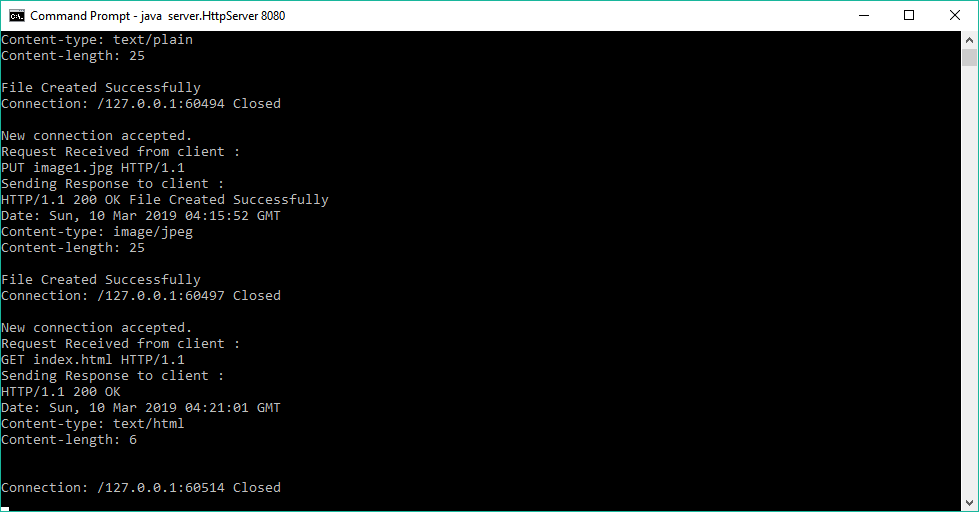
1. **Server saves the image in its ServerDir**

****

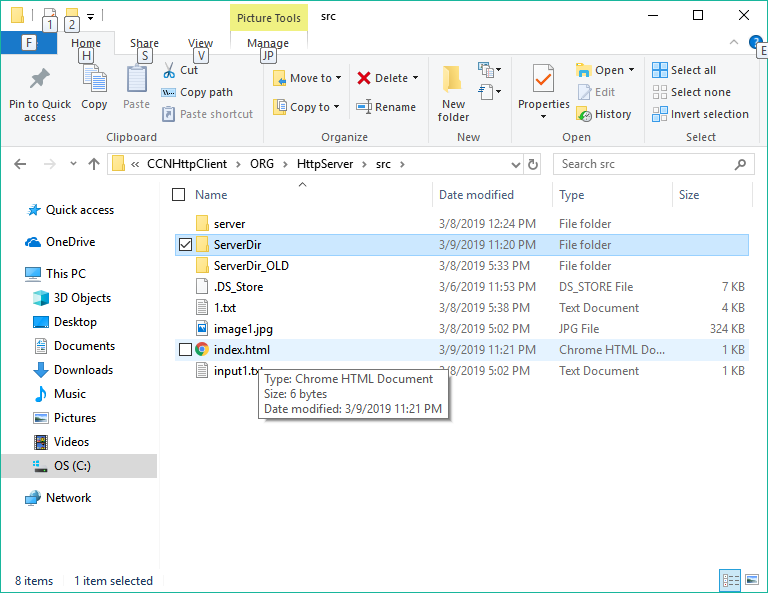
1. **Client sending GET request to server and receiving 200 OK response.**

****

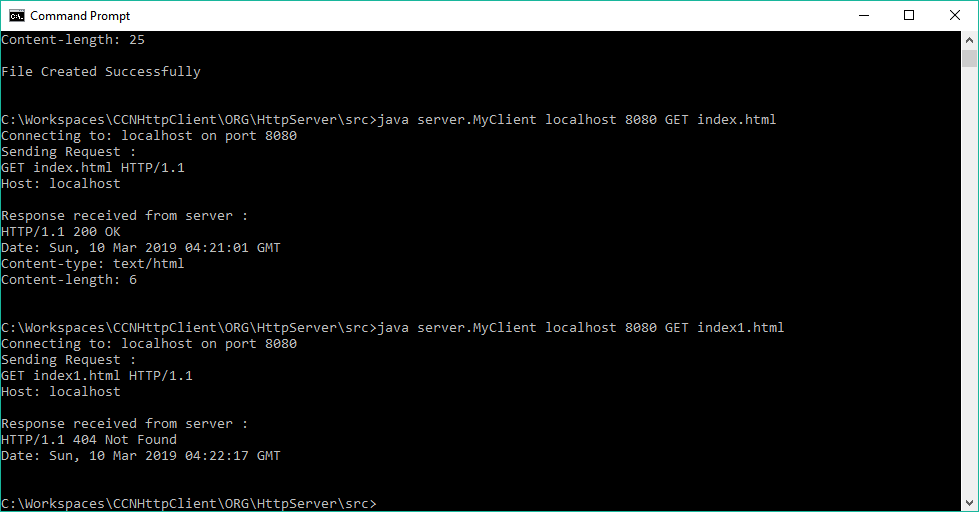
1. **Server Sending Response to GET**



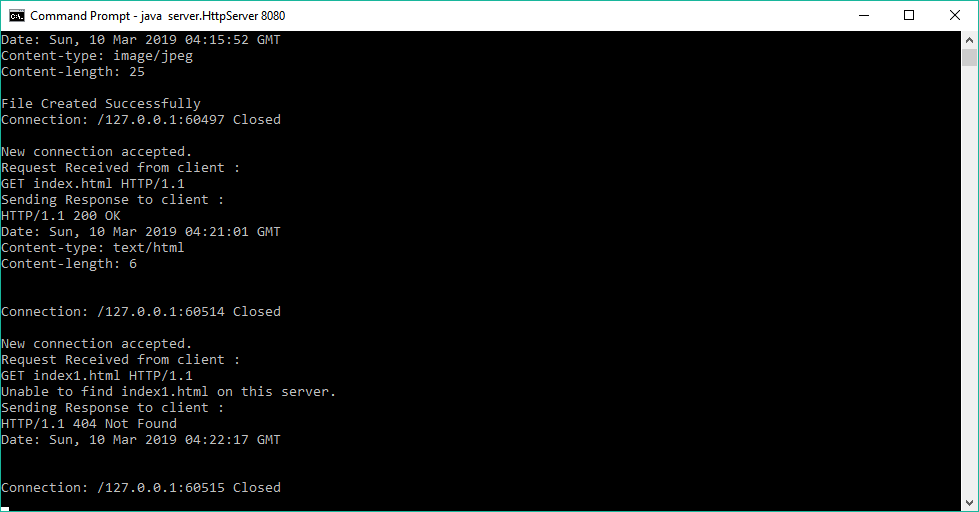
1. **Index file received in response saved locally by client**

****

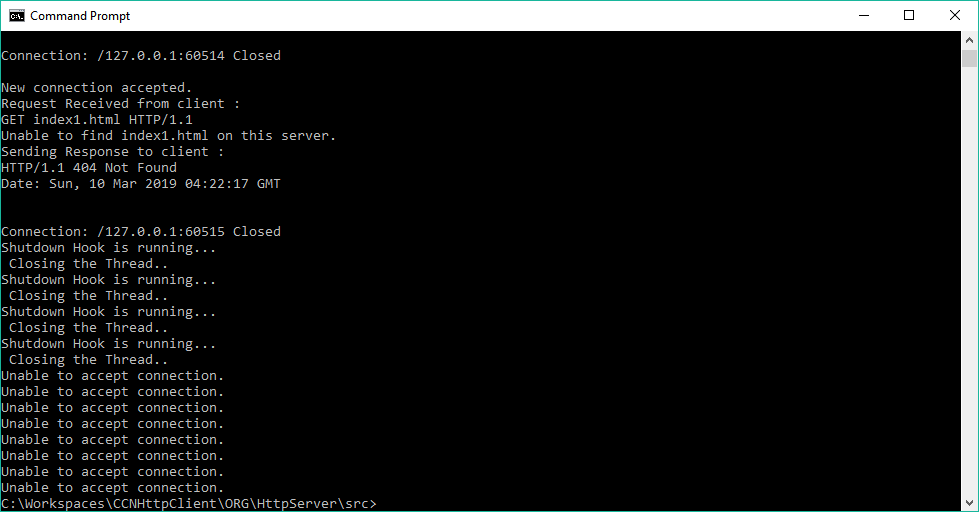
1. **Client requests with GET for a file which doesn’t exist at the server. Client gets 404 Not found in response.**

****

1. **Server Side Response building**

****

1. **Terminating Server with Ctrl-C, it shuts down gracefully closing all sockets**

****