

**A PROJECT REPORT ON** **HTTP CLIENT AND SERVER.**

**Computer Communications and Networks**  
**ITCS 6166/8166, Spring 2020**

**Project Members: Professor: Dr. Dewan T Ahmed**

**Prabhav A. Jani (801166353) TA: Divya Patel**

**Salma Roohi Khayum (801168027) Sahithi Priya Gutta**

**Abstract**

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a port at an IP, while another socket reaches out to the other to form a connection. Server forms the listener socket while the client reaches out to the server. The client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Also, that once a connection is established, both sides can send and receive information. A socket is one end of an inter process communication channel. The two processes each establish their own socket. Our goal is to implement inter process communication between client and server using HTTP. We implemented this functionality by using Java as a coding language. We have deployed two scripts, each for the Client and the Server. We have used multi-threading in our script to handle multiple requests simultaneously.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **1.1** | **Introduction** | **4** |
| **1.2** | **Sockets** | **4** |
| **1.3** | **Multithreading** | **4** |
| **1.4** | **Server Program** | **4** |
| **1.5** | **Client Program** | **5** |
| **1.6** | **Code for Server** | **5-9** |
| **1.7** | **Code for Client** | **9-12** |
| **1.8** | **Code for Closing Server** | **12-13** |
| **1.9** | **Execution of Program** | **13-18** |
| **2.1** | **UML Sequence Diagram** | **19-20** |
| **2.2** | **References** | **21** |

* 1. **Introduction**

We have implemented an HTTP client and server that runs a simplified version of HTTP/1.1. We have implemented two HTTP commands: GET and PUT. This project uses Java as the primary language. The multithreaded server used in the process, has an infinite loop to listen for different connections simultaneously. We have used an interrupt with a termination signal to shut down the server. Upon receiving the termination signal, the server shuts down gracefully, closing all sockets before exiting. Client establishes a connection with the server using a socket address. A socket address consists of IP address and Port Number. After the connection is established, the client can communicate with the server and vice versa. We have implemented this project in Apache NetBeans IDE 11.0 using Java.

* 1. **Sockets**

We are using TCP socket. Users can select port numbers greater than 5000. We are using a bufferedreader for reading input-streams from socket and print-writer for writing into the output-stream of the socket.

**1.3 Multithreading**

Multithreading is a process of executing multiple threads simultaneously in a single process. We have implemented multi-threading in Java to establish our goal of implementing the server in real time which can respond to multiple client requests at the same time without delay or loss. To establish connections and to send and receive files to and from the server we can use 1024 threads. On the Client side to send and receive data we have used only 1 thread.

* 1. **Server Program**

A server can open a connection and accept incoming client requests. The server program “myserver.java” uses multithreading to accept requests of multiple clients. User will specify port number in the argument on which server will Listens for incoming connections from clients. When a client connection is accepted, HTTP request is read. We have Constructed a valid HTTP response

* When the server receives a GET request, it would either construct a “200 OK” message followed by the requested object or a “404 File Not Found” message.
* When the server receives a PUT request, it saves the file locally.
* If the received file is successfully saved, the server constructs a “201 Created” response.

HTTP response is sent over the TCP connection to the client. Connection continues in loop to listen for an incoming connection. Multithreaded server has an infinite loop to listen for connections. We have implemented an interrupt(closeServer.java) with a termination signal in order to shut down the server. Upon receiving the termination signal, the server shuts down, closing all sockets before exiting and closes the client connection.

* 1. **Client Program**

Our client is designed to take the following command line arguments: server IPaddress, port on which to contact the server, HTTP command (GET or PUT), and the path of the requested object on the server. The client connects to the server via a TCP connection and submits a valid HTTP/1.1 GET or PUT request to the server, reads the server’s response and displays it.

* Connects to the server via a TCP connection
* Submits a valid HTTP/1.1 PUT request to the server
* Sends the file to the server
* Waits for the server’s reply
* Reads and displays the server’s response

**1.6 Code for Server**

|  |
| --- |
| **import java.io.BufferedReader;**  **import java.io.File;**  **import java.io.FileReader;**  **import java.io.FileWriter;**  **import java.io.IOException;**  **import java.io.InputStreamReader;**  **import java.io.PrintWriter;**  **import java.net.ServerSocket;**  **import java.net.Socket;**  **import java.util.concurrent.Executors;**  **import java.util.concurrent.ThreadPoolExecutor;**  **public class myserver {**  **static String[] temp = new String[3];**  **static ServerSocket ss;**  **static Socket s;**  **static BufferedReader br;**  **static String line = "";**  **static ThreadPoolExecutor pool = (ThreadPoolExecutor) Executors.newFixedThreadPool(1024);**  **static PrintWriter pw = null;**  **public static void main(String args[]) throws Exception {**  **int port;**  **System.out.println("starting Server");**  **try {**  **port = Integer.valueOf(args[0]);**  **if (port <= 5000) {**  **throw new Exception("advisable port number is greater than 5000\n");**  **}**  **} catch (Exception e) {**  **System.out.println("error with port number, provide proper port no.\n" + e);**  **return;**  **}**  **ss = new ServerSocket(port);**  **System.out.println("Server is listening on port " + port + " for clients\n");**  **while (!line.equals("close")) {**  **s = ss.accept();**  **br = new BufferedReader(new InputStreamReader(s.getInputStream()));**  **pw = new PrintWriter(s.getOutputStream(), true);**  **line = br.readLine();**  **if(line.equals("close")){**  **br.close();**  **pw.close();**  **s.close();**  **ss.close();**  **break;**  **}**  **System.out.println("\*\*\*Request-Header\*\*\*");**  **System.out.println(line);**  **temp = line.split(" ");**  **while (!line.equals("")) {**  **line = br.readLine();**  **System.out.println(line);**  **}**  **System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**  **if (temp[0].equals("GET")) {**  **MyServerThread thread = new MyServerThread(br, pw, temp[0], temp[1], temp[2]);**  **pool.execute(thread);**  **} else {**  **forPut(pw);**  **}**  **}**  **}**  **static void forPut(PrintWriter pw) {**  **String fileName = temp[1].substring(1), httpv = temp[2], status;**  **File f1 = new File(fileName);**  **String tempFileName = fileName.substring(0, fileName.indexOf(".")) + ".txt";**  **File f2 = new File(tempFileName);**  **PrintWriter pwforSocket = pw;**  **PrintWriter pwforFile = null;**  **if (f1.exists()) {**  **f1.delete();**  **}**  **try {**  **pwforFile = new PrintWriter(new FileWriter(f1));**  **System.out.println(br.readLine());**  **line = "just for temporary";**  **while (!line.equals("")) {**  **line = br.readLine();**  **pwforFile.println(line);**  **}**  **System.out.println("File written.\n");**  **br.close();**  **pwforFile.close();**  **//System.out.println("before changing name file name is: "+f2.getName());**  **f2 = new File(fileName);**  **f1.renameTo(f2);**  **} catch (IOException ex) {**  **System.out.println("error in writing file");**  **}**  **pwforSocket.flush();**  **pwforSocket.println(httpv + " 200 OK ");**  **pwforSocket.println("Content-Location : " + fileName);**  **pwforSocket.println("Connection: close");**  **pwforSocket.println();**  **try {**  **f1 = null;**  **f2 = null;**  **System.gc();**  **br.close();**  **pwforFile.close();**  **pwforSocket.close();**  **} catch (Exception ex) {**  **System.out.println("error in closing...");**  **}**  **}**  **static class MyServerThread implements Runnable {**  **File f1 = null;**  **PrintWriter pwforSocket = null, pwforFile = null;**  **String status, httpv, line, method, fileName;**  **BufferedReader brforFile = null, brforSocket = null;**  **MyServerThread(BufferedReader brforSocket, PrintWriter pw1, String command, String fileName, String httpversion) {**  **this.httpv = httpversion;**  **this.method = command;**  **this.pwforSocket = pw1;**  **if (fileName.equals("/")) {**  **this.fileName = fileName.substring(1);**  **} else {**  **this.fileName = "index.html";**  **}**  **this.brforSocket = brforSocket;**  **f1 = new File(fileName.substring(1));**  **}**  **@Override**  **public void run() {**  **forGet();**  **}**  **//Code to handle GET request**  **void forGet() {**  **if (f1.exists()) {**  **status = httpv + " 200 OK\r\n";**  **pwforSocket.write(status);**  **pwforSocket.write("Content-Type: text/html\r\nConnection: close\r\n\r\n");**  **try {**  **brforFile = new BufferedReader(new FileReader(f1));**  **while ((line = brforFile.readLine()) != null) {**  **pwforSocket.write(line + "\n");**  **}**  **pwforSocket.write("\r\n");**  **} catch (Exception ex) {**  **System.out.println("error :" + ex);**  **} finally {**  **try {**  **brforFile.close();**  **pwforSocket.close();**  **} catch (Exception ex) {**  **System.out.println("error in closing");**  **}**  **}**  **} else {**  **status = httpv + " 404 File Not Found";**  **pwforSocket.write(status);**  **}**  **pwforSocket.close();**  **}**  **}**  **}** |

**1.7 Code for Client**

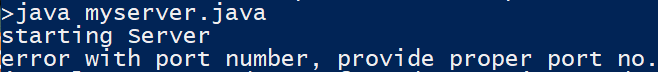
|  |
| --- |
| **package demoortry;**  **import java.io.BufferedReader;**  **import java.io.File;**  **import java.io.FileNotFoundException;**  **import java.io.FileReader;**  **import java.io.FileWriter;**  **import java.io.IOException;**  **import java.io.InputStreamReader;**  **import java.io.PrintWriter;**  **import java.net.Socket;**  **import java.net.URL;**  **import java.net.URLConnection;**  **public class myclient {**  **static String command, hostname, fileName, httpv = "HTTP/1.1", response = null, line = "";**  **static Socket s = null;**  **static PrintWriter pwforSocket = null;**  **static BufferedReader br = null;**  **static File f1, f2;**  **static int port;**  **static URLConnection con;**  **static URL url;**  **public static void main(String[] args) {**  **//variable**  **try {**  **command = args[2];**  **hostname = args[0];**  **fileName = args[3];**  **} catch (Exception e) {**  **System.out.println("it takes 4 arguments"**  **+ "\n1) IP address"**  **+ "\n2) port no."**  **+ "\n3) command \"GET\" or \"PUT\""**  **+ "\n4) filename to get ");**  **}**  **try {**  **port = Integer.valueOf(args[1]);**  **} catch (Exception e) {**  **System.out.println("port shoult be an integer");**  **return;**  **}**  **if (command.equals("GET")) {**  **try {**  **sendGetRequest();**  **} catch (Exception ex) {**  **System.out.println("error in sending request " + ex);**  **}**  **} else if (command.equals("PUT")) {**  **sendPutRequest();**  **} else {**  **System.out.println("command should be GET or PUT");**  **}**  **}**  **//Code to send GET request**  **private static void sendGetRequest() {**  **BufferedReader in = null;**  **PrintWriter pw = null;**  **try {**  **url = new URL("http://" + hostname + ":" + port + "/" + fileName);**  **con = url.openConnection();**  **in = new BufferedReader(**  **new InputStreamReader(**  **con.getInputStream()));**  **System.out.println("Response-Header..........");**  **System.out.println(con.getHeaderField(0));**  **System.out.println();**  **String line;**  **f1 = new File("index.txt");**  **f2 = new File(fileName);**  **if (f2.exists()) {**  **f2.delete();**  **}**  **pw = new PrintWriter(new FileWriter(f1));**  **System.out.println("Data.....................");**  **while ((line = in.readLine()) != null) {**  **System.out.println(line);**  **pw.println(line);**  **}**  **} catch (FileNotFoundException ex) {**  **System.out.println("Response-Header..........");**  **System.out.println(con.getHeaderField(0));**  **return;**  **} catch (Exception ex) {**  **System.out.println("error: " + ex);**  **}**  **try {**  **in.close();**  **pw.close();**  **f1.renameTo(f2);**  **} catch (Exception ex) {**  **System.out.println("error: " + ex);**  **}**  **}**  **//**  **private static void sendPutRequest() {**  **String endingHeader = "Connection: keep-alive\r\nPragma: no-cache\r\nCache-Control: no-cache\r\n\r\n";**  **f1 = new File(fileName);**  **if (f1.exists()) {**  **try {**  **s = new Socket(hostname, port);**  **pwforSocket = new PrintWriter(s.getOutputStream(), true);**  **br = new BufferedReader(new InputStreamReader(s.getInputStream()));**  **} catch (Exception ex) {**  **System.out.println("error occured during connection");**  **}**  **pwforSocket.flush();**  **pwforSocket.println(command + " /" + fileName + " " + httpv);**  **pwforSocket.println(endingHeader);**  **//pwforSocket.flush();**  **try {**  **//sleep(500);**  **BufferedReader brforFile = new BufferedReader(new FileReader(f1));**  **//System.out.println("sending lines after reading file");**  **while ((line = brforFile.readLine()) != null) {**  **pwforSocket.println(line);//write(line+"\n");**  **//System.out.println(line);**  **}**  **//System.out.println("file completed");**  **pwforSocket.println();**  **// System.out.println("writing completed");**  **brforFile.close();**  **} catch (Exception ex) {**  **System.out.println("file reading error" + ex);**  **}**  **try {**  **while (!line.equals("")) {**  **//System.out.println("in while reading response");**  **line = br.readLine();**  **//System.out.println(line);**  **}**  **} catch (Exception ex) {**  **System.out.println("Response-Header..........");**  **System.out.println("HTTP1.1 201 Created");**  **} finally {**  **pwforSocket.close();**  **try {**  **br.close();**  **} catch (IOException ex) {**  **System.out.println("error during in finally" + ex);**  **}**  **}**  **} else {**  **System.out.println("file does not exist");**  **}**  **}**  **}** |

**1.8 Code For Closing Server**

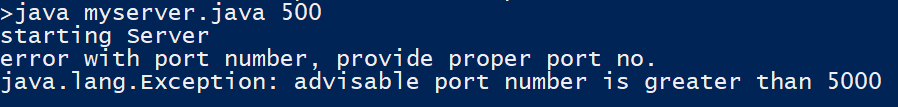
|  |
| --- |
| **import java.io.IOException;**  **import java.io.PrintWriter;**  **import java.net.Socket;**  **import java.util.logging.Level;**  **import java.util.logging.Logger;**  **public class closeServer {**  **static int port;**  **static Socket s;**  **static PrintWriter pw;**  **static String ip;**  **public static void main(String args[]){**  **if(args.length<2){**  **System.out.println("provide first argument IP address of server and second port number");**  **return;**  **}**  **ip=args[0];**  **port = Integer.valueOf(args[1]);**  **try {**  **s = new Socket(ip,port);**  **pw = new PrintWriter(s.getOutputStream(),true);**  **pw.println("close");**  **} catch (IOException ex) {**  **Logger.getLogger(closeServer.class.getName()).log(Level.SEVERE, null, ex);**  **}**    **}**  **}** |

**1.9 Execution of Program**

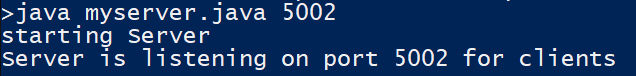
**1.9.1 Execution of Server Program**



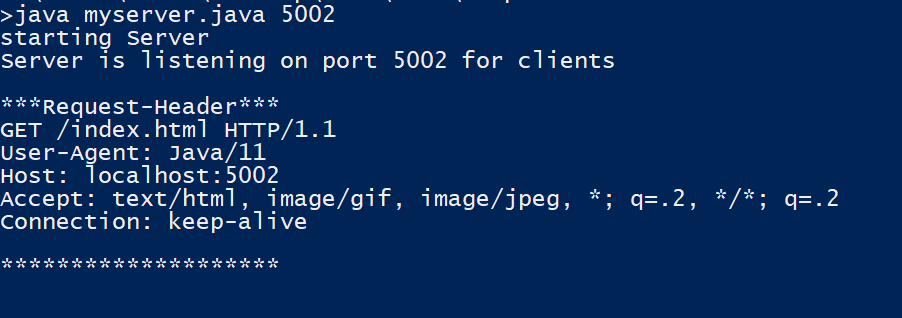
1.(a)Execution of server without arguments



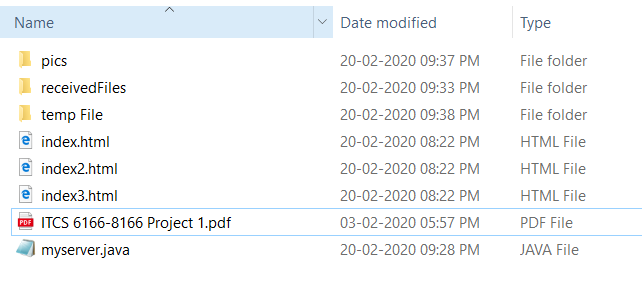
1.(b)Execution of server with wrong arguments



1.(c) Server started listening on provided port number

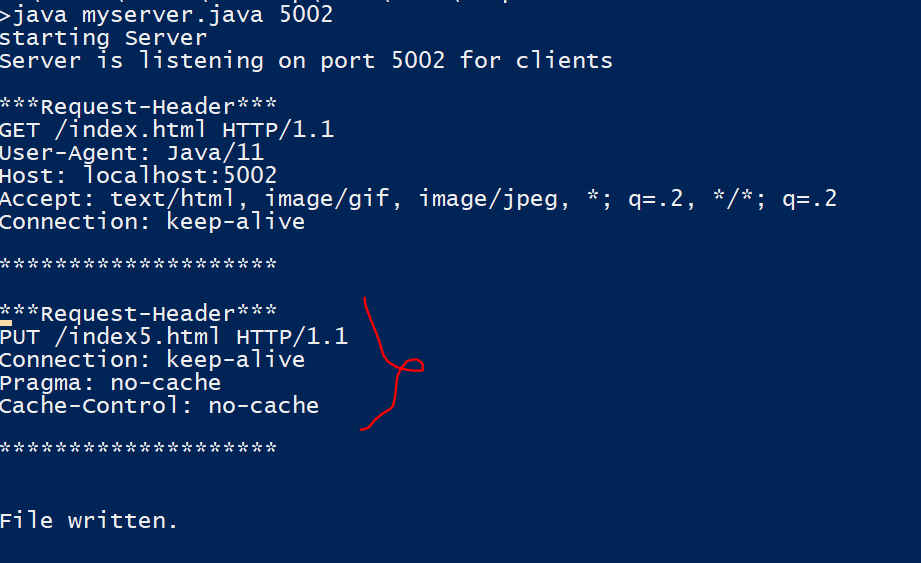


1.(d) Server received GET request from client



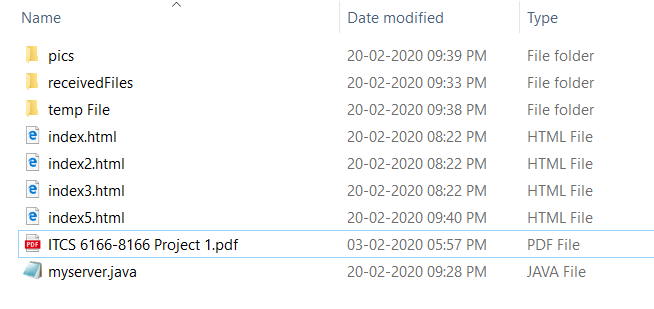
1.(e)Server Side before PUT request



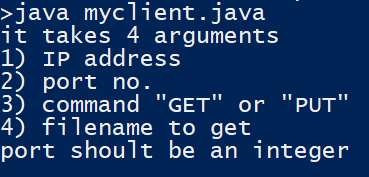


1.(f) Server on receiving PUT Request



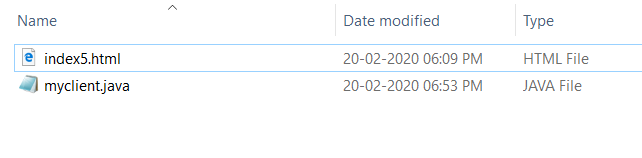
1.(g) Server Side after PUT request

**1.9.2 Execution of Client program**



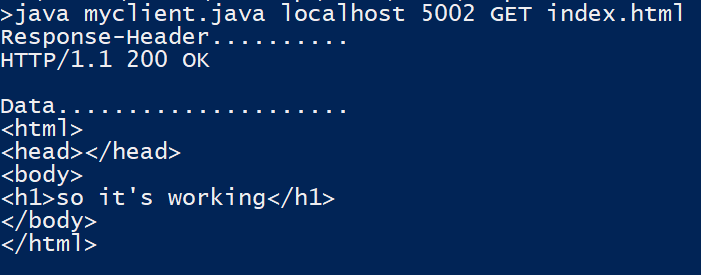
2.(a) Client without arguments

Justification:- If a user does not provide any argument, the program will provide description of arguments needed.



2(b) Client Side before GET request

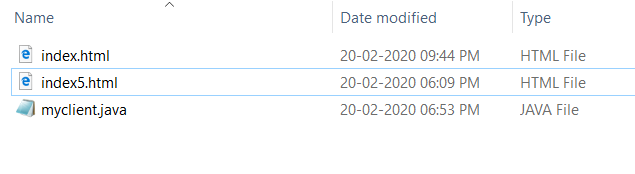




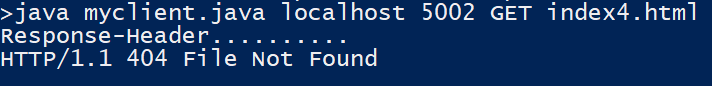
2(c) Client GET example

Justification:- client is requesting for file “index.html” using GET method from Localhost(IPaddress of local machine)

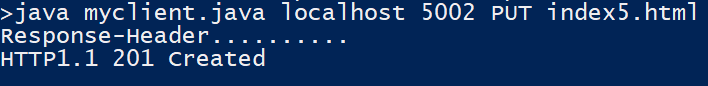




2.(d) Client Side after GET request



2.(e) Client is requesting for file “index4.html” which is not present with the server.



2.(f) Client is using the PUT method to upload file “index.html” to server.



2.(g)User specified wrong file name when using put method

**2.1 UML Sequence Diagram:**

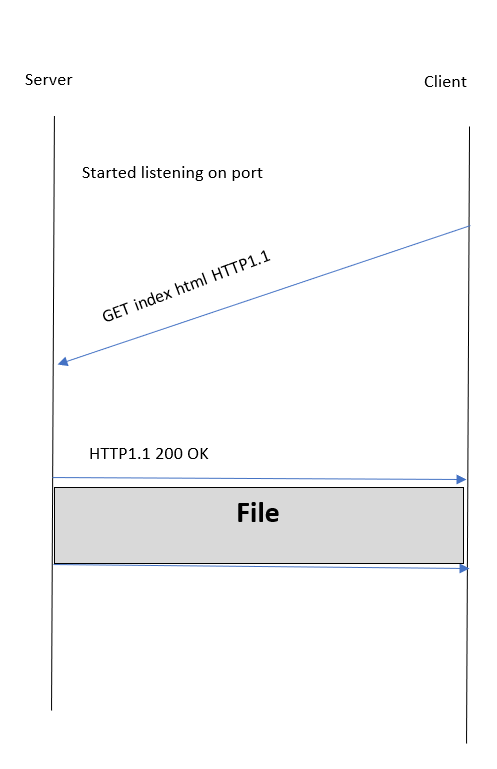
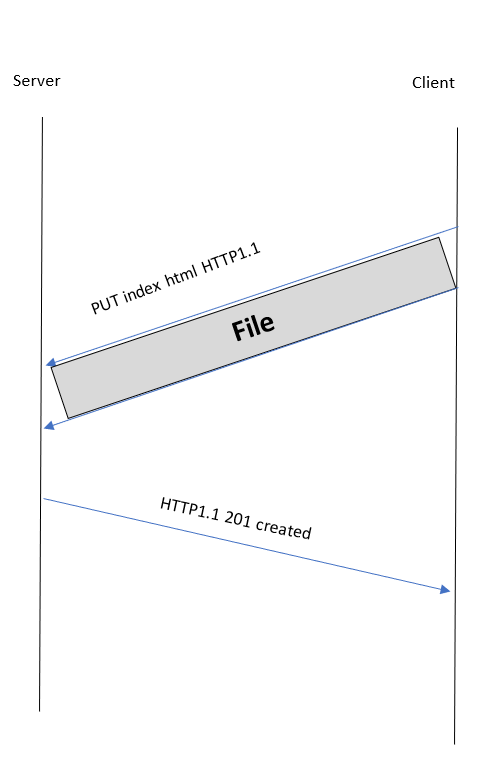
** **

fig 1. UML diagram for GET request fig.2 UML diagram for PUT request

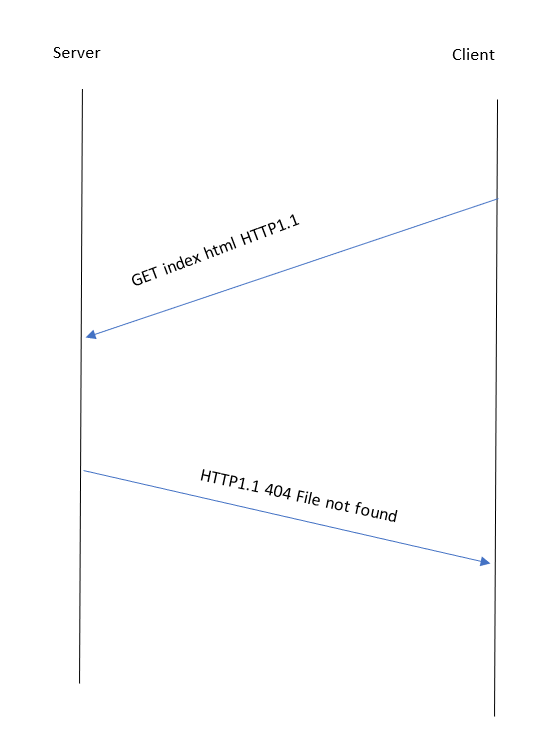


fig.3 diagram for file not found response

**2.2 References:**

<https://www.geeksforgeeks.org/socket-programming-cc/>

<https://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html>

<https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html>