Computer Network

Project #1 : TCPWebServer

Report

COMPUTER NETWORK PROJECT

Subject	Computer Network
Professor	Jin Seek Choi
Submission Date	October 23rd, 2021
University	Hanyang University
School	College of Engineering
Department	Department of Computer Science & Engineering
Student ID	**Name**
2019009261	최가온(CHOI GA ON)

Project Preview

The main goal of this project is to develop a multi-threaded web server that is capable of processing multiple simultaneous service request in parallel.

There are two stages for developing the server.

- implementing a multi-threaded server that simply displays the contents of the HTTP request message that it receives
- adding the code required to generate an appropriate response.

A WebServer is a server software, or hardware dedicated to running said software, that can serve contents to the World Wide Web. A web server processes incoming network requests over HTTP.

HTTP protocol provides a variety of status code for reliable data transfer.

In this project, we implemented the following status code.

1) OK
2) BAD_REQUEST
3) FORBIDDEN
4) NOT_FOUND
5) HTTP_VERSION_NOT_SUPPORTED
6) INTERNAL_SERVER_ERROR

Code Explanation

1. WebServer.java

Step 1.

Import necessary libraries for this project.

```
import java.io.* ;
import java.net.* ;
import java.net.http.HttpRequest;
import java.util.* ;
```

Step 2.

Create a class named “WebServer”. After creating the class, create a ServerSocket object with appropriate port number. In this example, the port number is 7000.

```
public final class WebServer {
    public static void main(String argv[]) throws Exception {

       try {
          // Could get the port number from the command line.
          // int port = (new Integer(argv[0])).intValue();

          // Establish the listen socket.
          ServerSocket serversocket = new ServerSocket(7000);    //
Mission 1
```

Step 3.

Create a socket from serversocket to listen for a TCP connection request. After creating a socket, construct HttpRequest_upgrade object to process the HTTP message. Finally, for the multi-threaded environment, create a thread from request object, and start.

```
// Process HTTP service requests in an infinite loop.
while (true) {
    // Listen for a TCP connection request.
```

```
   Socket socket = serversocket.accept(); // Mission 1

    // Construct an object to process the HTTP request message.
    HttpRequest_upgrade request = new HttpRequest_upgrade(socket);

    // Create a new thread to process the request.
    Thread thread = new Thread(request);   // Mission 2
    // Start the thread.
    thread.start();
}
```

Step 4.

Consider handling exceptions related to input/output.

```
}catch(IOException e) {
         System.out.print(e.getMessage());
      }catch(Exception e) {
         System.out.print(e.getMessage());
      }

   }
}
```

2. HttpRequest_upgrade.java

Step 1.

Import necessary libraries for this project. Declare an enum type object to illustrate the status code in HTTP.

```
import java.io.* ;
import java.net.* ;
import java.util.* ;

enum StatusCode{
   OK, BAD_REQUEST, FORBIDDEN, NOT_FOUND,
HTTP_VERSION_NOT_SUPPORTED, INTERNAL_SERVER_ERROR }
```

Step 2.

Declare necessary variables such as CRLF, HTTP version, buffer in/out size, content type, etc, in the HttpRequest_upgrade class.

```
final class HttpRequest_upgrade implements Runnable {
    final static String CRLF = "\r\n";
    final static String HTTP_VERSION = "1.1";
    final static String DEFAULT_CONTENT_TYPE = "application/octet-
stream";

    // final static string content Length "";
    final static int BUFFER_IN_SIZE = 2048; // 2048
    final static int BUFFER_OUT_SIZE = 2048; // 2048
    final static Properties CONTENT_TYPES = new Properties();
    final static EnumMap<StatusCode, String> SCODES = new
EnumMap<StatusCode, String> (StatusCode.class);

    static {
       CONTENT_TYPES.setProperty("html", "text/html");
      CONTENT_TYPES.setProperty("jpg", "image/jpg");

       SCODES.put(StatusCode.OK, "200");
       SCODES.put(StatusCode.BAD_REQUEST, "400");
       SCODES.put(StatusCode.FORBIDDEN, "403");
       SCODES.put(StatusCode.NOT_FOUND, "404");

       SCODES.put(StatusCode.HTTP_VERSION_NOT_SUPPORTED, "505");
    }
```

```
    StatusCode code;
    Socket socket;
    File requestedFile;
```

Step 3.

Create a constructor for HttpRequest_upgrade class and implement "run" method for the multi-threaded environment.

```
// Constructor
  public HttpRequest_upgrade(Socket socket) throws Exception {
    this.socket = socket;
    this.code = null;
    this.requestedFile = null;
  }

  // Implement the run() method of the Runnable interface.
  public void run() {
    try {
        processRequest();
    } catch (Exception e) {
       System.out.println(e);
    }
  }
```

Step 4.

Create processRequest method for processing requests. Parse the input request string by using parseRequestLine method.

```
private void processRequest() throws Exception {

    // Get a reference to the socket's input and output streams.
    InputStream is = null;
    DataOutputStream os = null;
    FileInputStream fis = null;

    // Set up input stream filters.
    BufferedReader br = null;

    try {
       is = socket.getInputStream();
       os = new DataOutputStream(socket.getOutputStream());
       br = new BufferedReader(new InputStreamReader(is),
BUFFER_IN_SIZE);
```

```
    // Get the request line of the HTTP request message.
    String requestLine = br.readLine();
    String errorMsg = parseRequestLine(requestLine);
```

Step 5.

Print headerline if it not null, and open file from requestedFile. There needs to be done about exception handling for file I/O.

```
String headerLine = null;
      while ((headerLine = br.readLine()).length() != 0) {
      System.out.println(headerLine);
   }

      if(errorMsg == null) {
         try {
            fis = new FileInputStream(requestedFile);
         } catch(FileNotFoundException e){
            System.out.println("FileNotFoundException while opening
file inputStream");
            e.printStackTrace();
            code = StatusCode.NOT_FOUND;
         }
      } else {
         System.out.println();
         System.out.println(errorMsg);
      }
      sendResponseMessage(fis, os);

} finally {
   if(os != null)
      os.close();
   if(br != null)
      br.close();
   if(fis != null)
      fis.close();
   socket.close();
}
  }
```

Step 6.

In sendHeaderLines method, we implemented to specify the file name and the content length(fixed as 1024).

```
private void sendHeaderLines(DataOutputStream os) throws Exception{
(...)
switch (code) {
case OK:
   contentTypeLine +=
this.contentType(this.requestedFile.getName()) + CRLF;     //
Mission 8
   contentTypeLine += contentLength + "1024" + CRLF;  // Mission 6
   break;

default:
   contentTypeLine += this.contentType("text.html") + CRLF;   //
Mission 7
}
(...)
```

Step 7.

In parseRequestLine, it checks for requestLine which is created from the user. First, it checks the input format. If its count of tokens are not 3, it sends NOT_FOUND status code.

Now, we need to specify the file path. In this project, the .java file and .html, .jpg files are in the same folder.

After checking the validity of input, we need to clarify that the HTTP version is 1.1, not 1.0. If the version is 1.0, this program sends BAD_REQUEST status code to the user.

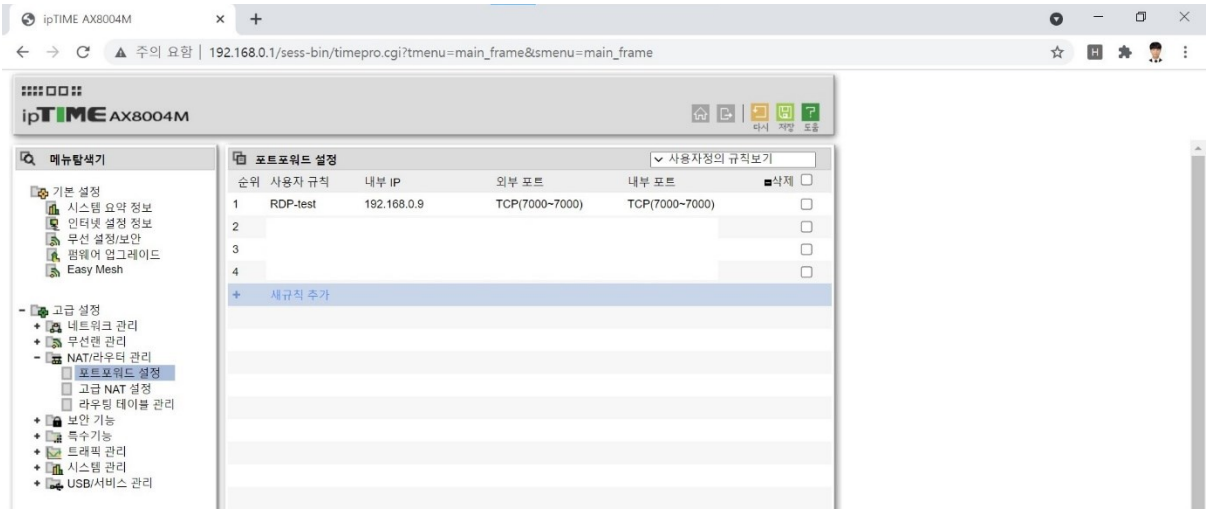
After checking all things, send OK status code if there are no any problem.

```
private String parseRequestLine(java.lang.String requestLine) {
   // TODO Auto-generated method stub
(...)
      if(tokens.countTokens() != 3) {
     // the input doesn't match expectations(format)
         code = StatusCode.NOT_FOUND;   // Mission 4
```

```
            return "Request line malformed. Returning BAD NOT FOUND.";
        }

        String method = tokens.nextToken().toUpperCase();
        String fileName = tokens.nextToken();

        System.out.println("file name: " + fileName);

        // Modify to your path
    // fileName = "." + fileName ;
        fileName = "." + fileName;
        File file = new File(fileName);

        if(!file.exists()) {
            code = StatusCode.NOT_FOUND;
            return "File not FOUND";
        }
        if(!file.canRead()) {
            code = StatusCode.FORBIDDEN;
            return "File is not Readdable";
        }

(...)

        requestedFile = file;
        String version = tokens.nextToken().toUpperCase();
        if(version.equals("HTTP/1.0")) {
            code = StatusCode.BAD_REQUEST; // Mission 5
            return "HTTP Version String is malformed.";
        }
        if(!version.matches("HTTP/([1-9][0-9.]*)")) {
            code = StatusCode.BAD_REQUEST;
            return "HTTP Version String is malformed.";
        }
        if(!version.equals("HTTP/1.0")
&& !version.equals("HTTP/1.1") ) {
            code = StatusCode.HTTP_VERSION_NOT_SUPPORTED;
            return "HTTP Version is not supported.";
        }

        code = StatusCode.OK;   // Mission 3
        return null;
}
```

Instructions

Configure port forward settings to start this program.



As the port number is used as 7000 in the source code above, the outer port and the inner port is set as 7000.

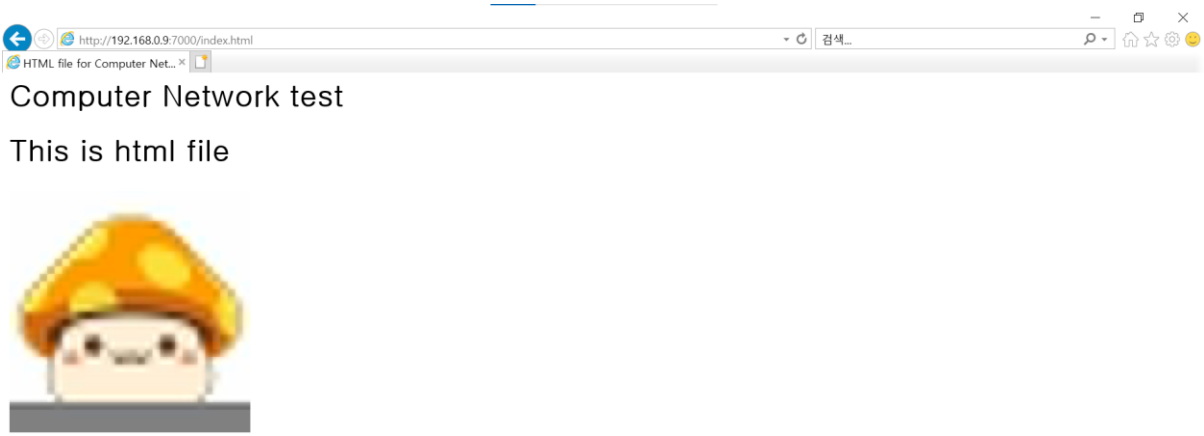
Open a commercial Internet client software(Chrome, Safari, Internet Explorer …).
Go to : http://192.168.0.9:7000/index.html

This means that

- IP address: 192.168.0.9
- Port number: 7000
- Accessed File: index.html

Execution & Results

1) The result of browser (env: Internet Explorer)



2) The result of webserver (env: Intellij)

```
PS C:\Users\USER\Documents\TCPWebServer> cd src
PS C:\Users\USER\Documents\TCPWebServer\src> java WebServer

Received HTTP request:
GET /index.html HTTP/1.1
file name: /index.html
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: ko
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 192.168.0.9:7000
Connection: Keep-Alive
StatusLine : HTTP/1.1 200 OK
entityBody :
<HTML>
 <HEAD><TITLE>?</TITLE></HEAD>
 <BODY>?</BODY>

code OK
type : text/html
sending request file to Client...

Received HTTP request:
GET /image.jpg HTTP/1.1
file name: /image.jpg
Accept: image/png, image/svg+xml, image/jxr, image/*;q=0.8, */*;q=0.5
Referer: http://192.168.0.9:7000/index.html
Accept-Language: ko
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 192.168.0.9:7000
Connection: Keep-Alive
StatusLine : HTTP/1.1 200 OK
entityBody :
<HTML>
 <HEAD><TITLE>?</TITLE></HEAD>
 <BODY>?</BODY>

code OK
type : image/jpg
sending request file to Client...
```

3) Test result from MIR lab page

* This test is held in the MIR laboratory with the help of the teaching assistant of Computer Network class.

Your Test Result

Student INFO

Student Number	Student Name	WebServer IP	WebServer PORT	ACCESS TIME	SCORE
2019009261	GAON CHOI	166.104.28.166	7000	2021-01-21 06:01:50	100

List of Test Items

WEB SERVER SOCKET :	TRUE
Multi Thread :	TRUE
STATUS CODE : 200 OK	TRUE
STATUS CODE : 404 NOT FOUND(EXCEPTION HANDLING)	TRUE
STATUS CODE : 400 BAD REQUEST(HTTP PROTOCOL VERSION)	TRUE
CONTENT LENGTH	TRUE
CONTENT TYPE TEXT/HTML	TRUE
CONTENT TYPE IMAGE/JPEG	TRUE

Opinions

Preparing for this project, I realized the concept of the client and server. In this project, my program was a server for providing specific html file. And the people who accesses my server with their web application was the clients.

I'm working as an undergraduate researcher in Visual Computing Lab. In there, I work for the pose estimation based on a website, so I could learn some basics of HTML. With them, I made my own .html file and used it for this project.

In addition, this project was implemented in an environment of multi-thread. With the knowledge about multi-thread that I learned in OS class, I could have understood and applied thread to my project. It was interested.