National University of Singapore
School of Computing
CS2105: Introduction to Computer Networks
Semester 1, 2018/2019

Assignment 1
Web Server

Due: 23rd September 2018, 23:59
Total marks: 7

Overview

In this assignment, you will implement a simple Web server that is able to server static Web pages and objects to a client. This exercise will give you a chance to learn more about the most popular protocol on the Internet: the Hyper-text Transfer Protocol (HTTP).

This programming assignment is worth 7 marks. All work in this assignment should be done **individually**.

Pair Work

This assignment should be solved individually. However, if you are struggling, you may pair up with another student and work on the assignment together. The maximum number of students working together on a submission is 2. Pair work will incur a **1 mark penalty** for each student.

Under no circumstances should you solve it as a pair and then submit it as individual work. This is considered plagiarism and will be dealt with according to the university's protocol.

Submission

You will submit your program on IVLE. Zip your files into a single zip file and name it "`a1-<student number>.zip`" and upload to the Assignment 1 folder. For example, if your student number is A0123456X, then you should name your file "`a1-a0123456x.zip`". The required files for submission should not be in any subdirectory upon unzipping.

The dateline for submission is at 23rd September 2018, 23:59. Late submissions will be penalized 1 mark per day.

Special Instructions for Pair Work

If you are submitting this assignment as a pair, you are to include special comments at the top of your submitted source code as follow:

```
// PAIR WORK
// A0123456X
// A0654321X
```

where each lines begins with the comment character for the language, line 1 would be `PAIR WORK` and lines 2 and 3 are the student numbers of both students.

There will be no allowance for students who forget to declare that they solved the assignment as a pair. We advise you update the header of your source file as soon as you decide to work as a pair.

One and only one designated student should upload the zip file to IVLE. The other team member should never submit. It is up to you to decide who the designated submitter is.

Please do not change the designated submitter after submission. The same student should re-upload any update to the submission. Under no circumstance should the other student upload any zip file.

General Submission

Java

Your program main class will be `WebServer`, and following Java requirements, will be in the file `WebServer.java`. Any extra files that you use should be properly referenced and zipped in the proper directory structure.

We will use the Java 9 on `sunfire` to run your code. The location of the Java executables on `sunfire` is at `/usr/local/java/jdk/bin/`.

Python

Your program should be named `WebServer.py`. Any extra files that you use should be properly referenced and zipped in the proper directory structure.

We will use Python 3.7 on `sunfire` to run your code. The location of the Python executables on `sunfire` is at `/usr/local/Python-3.7/bin/python3`. Note this is NOT the default `python` that is on the path.

Plagiarism Warning

While you are free to discuss this assignment with your friends, you should submit code designed and written individually by yourself. We employ a zero-tolerance policy against plagiarism. Confirmed breach will result in zero mark for the assignment and further disciplinary action from the school.

DO NOT copy from the Web (or anywhere else). While there are plenty of web server implementation on GitHub, StackOverflow, etc., this assignment is rather straightforward so there is no need to look for reference. The only reference you can use is the official documentation/API for the programming language, i.e. Java or Python. Instead, if you are stuck, please consult the teaching staff.

But if you really must use some reference, you should try to understand it. After understanding the code, put it away, do not look at it, and write your own code. Subsequent exercises will build on the knowledge that you gain during this exercise. Possibly also the exam.

We have tools to check for plagiarism. Please be extra careful and do not share solutions with your friends.

Good practices include:

- Discussion of general approaches to solve the problem excluding detailed design discussions and code reviews.
- Hints about which classes to use.

- High level UML diagrams.

Unacceptable practices include (but are not limited to)

- Passing your solution to your friends.
- Uploading your solution to the Internet including public repositories.
- Passing almost complete skeleton codes to your friends.
- Coding the solution for your friend.
- Sharing the screen with a friend during coding.
- Sharing notes.

Testing

Using `sunfire`

You can test your web server on `sunfire` by uploading your program and then running it on an available port. For example

```
$ javac WebServer.java
$ java WebServer 8888
```

If you wish to access your web server while it is running on `sunfire`, you need to bind the TCP server socket to the public IP address. To simplify things, just bind to `0.0.0.0` and you should be able to access your web server from your PC.

You can use your web browser to access your web server using
`http://sunfire.comp.nus.edu.sg:8888/demo.html` We recommend using cURL so you can control the request headers as well as examine the exact output. Also, you can run Wireshark on your PC to examine the exchange of data/segments

You can also login to `sunfire` from another SSH session and use cURL to test your web server:

```
$ curl -0 localhost:8888/demo.html
```

The `-0` option tells cURL to use HTTP/1.0. You may also wish to use the verbose (`-v`) option to see what cURL is doing behind the scenes, and the `-i` option to output the response headers.

To get images, you can use the `-o` option to specify an output file:

```
$ curl -0 -i -o doge.jpg localhost:8888/img/doge.jpg
```

Test Suite

The autotest script on sunfire is up. Just like in assignment 0, transfer your zip file to sunfire, and run the following command:

```
~cs2105/autotest/a1 a1-a0123456x.zip
```

The Web Server

You are given a skeleton template for your WebServer implementation. Use the template and **do not start from scratch**. You may however, modify the signature and return type of the methods as you deem necessary.

A Web Server is just a TCP server that understands HTTP. It will 1) create a TCP socket, 2) listen to the socket and accept a new TCP connection, 3) read HTTP request(s) from TCP connection and handle it/them, and finally, 4) send HTTP response(s) to the client through TCP.

Following good OOP practise, we will separate various functionalities into different classes/methods:

1. The `start` method of the `WebServer` class should create a welcome socket and listen to new connection requests. Once a TCP connection is established, this method should pass the connection socket to `handleClientSocket` method (please refer to lecture 3 notes for a demonstration of TCP ServerSocket and Socket).

2. `handleClientSocket` will then take all necessary steps to handle a HTTP request from a client. It should make use of the `HttpRequest` class. Firstly, you have to read a HTTP request from the socket. Next, you should call `parseRequest` of the `HttpRequest` class to parse the request. `parseRequest` has not been defined for you; you are free to define it with any parameters you deem necessary.

3. An instantiated object of the `HttpRequest` class represents a single incoming HTTP request. Thus, it is a good idea to create a new `HttpRequest` object for each request.

4. Once the request is parsed, Web Server should call `formHttpResponse`. This method inspects the `HttpRequest` and returns a byte (array for Java, string for Python) which can be sent to the client using `sendHttpResponse`.

HTTP Specification

In this assignment, your Web Server should support both HTTP/1.0 and HTTP/1.1. Do not despair. You only have to implement a very small subset of the two protocols as shown below. They are extracted from RFC 2616 https://tools.ietf.org/html/rfc2616.

Your Web Server only has to support the following syntax for HTTP described using EBNF:

```
request        ::=  request-line { header-line } CRLF
request-line   ::=  "GET" SP request-url SP protocol CRLF
request-url    ::=  <absolute path of the object>
                    e.g. /demo.html or /img/doge.jpg, etc. Note that this path is absolute relative to
                    the current directory of the Web Server. Thus /demo.html does not refer to
                    /demo.html on the file system, but ./demo.html in the current directory.

header-line    ::=  field-name ":" SP field-value CRLF
field-name     ::=  <any alphanumeric character and "_" and "-">
field-value    ::=  <a string>

protocol       ::=  "HTTP/1.0" | "HTTP/1.1"

SP             ::=  <US-ASCII SP, space (32)>
CR             ::=  <US-ASCII CR, carriage return (13)>
LF             ::=  <US-ASCII LF, linefeed (10)>
CRLF           ::=  CR LF
```

```
response      ::=  status-line { header-line } CRLF
status-line   ::=  protocol SP status-code CRLF
status-code   ::=  "200␣OK"
```

Note: You only need to support `GET` request. There is also actually no requirement to include any response headers as long as your web server follows conventions that an HTTP client like cURL or Chrome can interpret.

Details and Grading

The score for this assignment is 7 marks. If you obtain above 7 marks, 50% of the extra marks can be used as bonus top-up for the assignment CA component.

You will be awarded the stated marks if your web server complies with the followin:

1 mark: Compiles and runs, and accepts an incoming TCP connection on the given port.

1 mark: Responds with status code “`404 Not Found`” when the requested URL does not exists.

1 mark: Can handle a single small web object (≈5 MB) like an HTML file or an image file. The `Content-Length` header can be used to inform the client of the size of the object in the case of persistent connections.

2 mark: Can handle a web page that contains multiple static objects.

2 marks: Implements **both** HTTP/1.0 and HTTP/1.1 correctly. For HTTP/1.1, the server supports persistent connections with pipelining. If you use a timer, you can set the time out value to 2 seconds.

The following criterias can be quite challenging and can give you bonus marks.

2 marks: Can handle very large Web objects (several GBs) such that you cannot read the entire file into memory, in which case, you cannot set the `Content-Length` field. This poses no problem with HTTP/1.0 as the server simply closes the connection once the object is transferred.

But HTTP/1.1 requires keeping the connection persistent, thus the client needs to know when it is done receiving the object. The `Transfer-Encoding` header set to `chunked` can be used. In this case, the object is transferred in multiple chunks of know length, and a chunk with zero length indicates the end of the transfer.

Since the response data is too large to fit into memory, you will be unable to write it to a byte array. To support this feature, you should modify the methods where appropriate.

A time limit of 15 s is imposed for every 100 MB of data returned by your web server. This is to prevent inefficient code from choking up when trying to transfer a large file.

1 mark will be given for supporting this in HTTP/1.0 and 1 mark for HTTP/1.1.

1 mark: Includes the `Last-Modified` header in the response based on the last modified date/time of the file given by the OS. The server supports a conditional get request with the header `If-Modified-Since` and responds accordingly with a `304 Not Modified` status code where needed.

Your web server should conform to the HTTP date format specified in `https://tools.ietf.org/html/rfc7231#section-7.1.1.1`. There are pre-defined libraries which can help you with the date parsing and formatting.

1 mark: Handles multiple HTTP/1.1 client connections simultaneously by writing a multi-thread server. You will have to modify the skeleton class to introduce multi-threading.