National University of Singapore

School of Computing

CS2105 **Assignment 1** Semester 2 AY18/19

Learning Objectives

In this assignment, you will implement a HTTP-like web server two functions: serving static files and providing a simple key-value store service. After completing this assignment, you should be able to (1) do simple network programming, and (2) have a good understanding of the mechanisms of HTTP.

Group Work

This assignment should be solved individually. However, you are allowed to form a team with another student (max. team size is 2) if you find it too difficult. Group submissions are subject to **1 mark penalty** for each students.

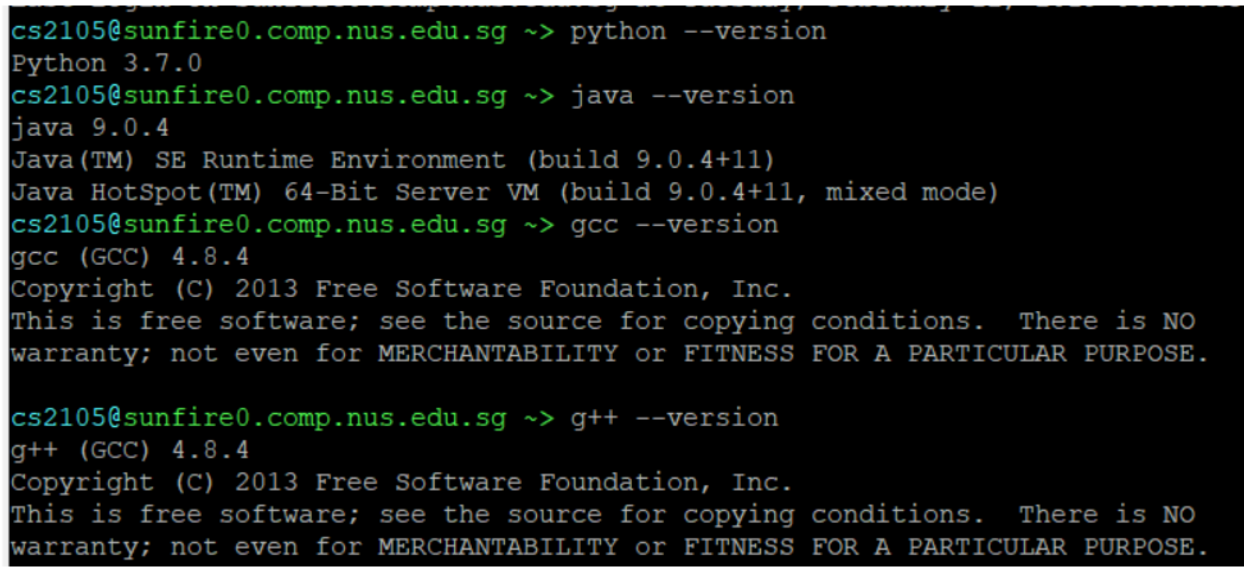
Under no circumstances should you solve it in a group and then submit it as an individual solution. This is considered plagiarism. Please also refer to the detailed explanations below for more information about plagiarism. Also refer to the special instruction below for submission of group work.

We allow group work because some students might struggle to solve this assignment alone, especially when you are new to network programming. Do not give up, do not copy. Instead, form groups of two and solve the assignment together.

Program Submission

For individual submission, please name your single source file as "**`WebServer-<Student number>.py`**" and submit it to the "Assignment_1_student_submission" folder of IVLE workbin. Here, the **`<Student number>`** is your matriculation number which starts with letter A. An example file name would be "**`WebServer-A0165432X.py`**". If you use Java, C, or C++ to implement the web server, please use ".java", ".c", or ".cpp" respectively as the extension name. Note that the first and last letters of your student number should be capital letters.

We recommend that you use Python 3 for the assignments. If you use Java or C/C++, we will compile and run your program for grading, using the default compilers on sunfire (g++ 4.8.4 or java 9.0.4). The used programming language is inferred from the file extension during grading. For a Java program, the class name should be consistent with the source file name, and please implement the **public static main()** method so that the program can be executed as a standalone process after compilation. Below screenshot shows respective language/compiler version on sunfire.



```
cs2105@sunfire0.comp.nus.edu.sg ~> python --version
Python 3.7.0
cs2105@sunfire0.comp.nus.edu.sg ~> java --version
java 9.0.4
Java(TM) SE Runtime Environment (build 9.0.4+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11, mixed mode)
cs2105@sunfire0.comp.nus.edu.sg ~> gcc --version
gcc (GCC) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

cs2105@sunfire0.comp.nus.edu.sg ~> g++ --version
g++ (GCC) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

The deadline of submission is **10th March 2019 (Sunday), 6:00pm sharp**. Please start early and submit early. You can always upload a newer version before the deadline. We strongly discourage students to submit their assignments only a few minutes before the submission deadline. The "Assignment_1_student_submission" folder will be closed exactly at the deadline, and all late submission should be made into a different folder named "Assignment_1_late_submission". **1** mark of penalty will be imposed on late submissions. The later submission folder will be closed on **24th Mar 6:00pm sharp** and there's no acceptance of late submission afterwards.

This programming assignment is worth **10** marks.

Special Instructions for Group Submission

For group submission, please include both student numbers and name your source file as "**`WebServer-<Student number1>-<Student number2>.py`**" and submit it to the same folder as with individual submission. An example file name would be "**`WebServer-A0165432X-A0123456Y.py`**". For each group, there should only be one designated member who submits the file (either one), in order to avoid problems caused by multiple branches within a group. **Do not change the designated submitter!** If the group needs to upload a new version, it should be done by the same designated submitter as well.

Do not solve the assignment in a group and submit it as individual work. This is considered plagiarism. All submission will be subject to a thorough plagiarism check. Plagiarism cases will receive 0 mark. There will be no acceptance for excuses like forgetting to declare that they solved the assignment in a group.

You are not allowed to post your solutions to any publicly accessible site on the Internet.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, you should **design and write** your own code. 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.

You should refrain from having detailed discussions with other students. Additionally, you should not share notes. You should not allow anyone to look at your code. If you want to solve this assignment together, please do so **but declare it as group work**.

The Web Server

In this assignment, you are required to implement three components to form a HTTP-like web server. The first component is a simplified protocol modified from HTTP 1.1, which consists of the basic request-response mechanism and the persistent connection feature from HTTP 1.1, but with different header formats. On top of this protocol, a static file server and an interactive key-

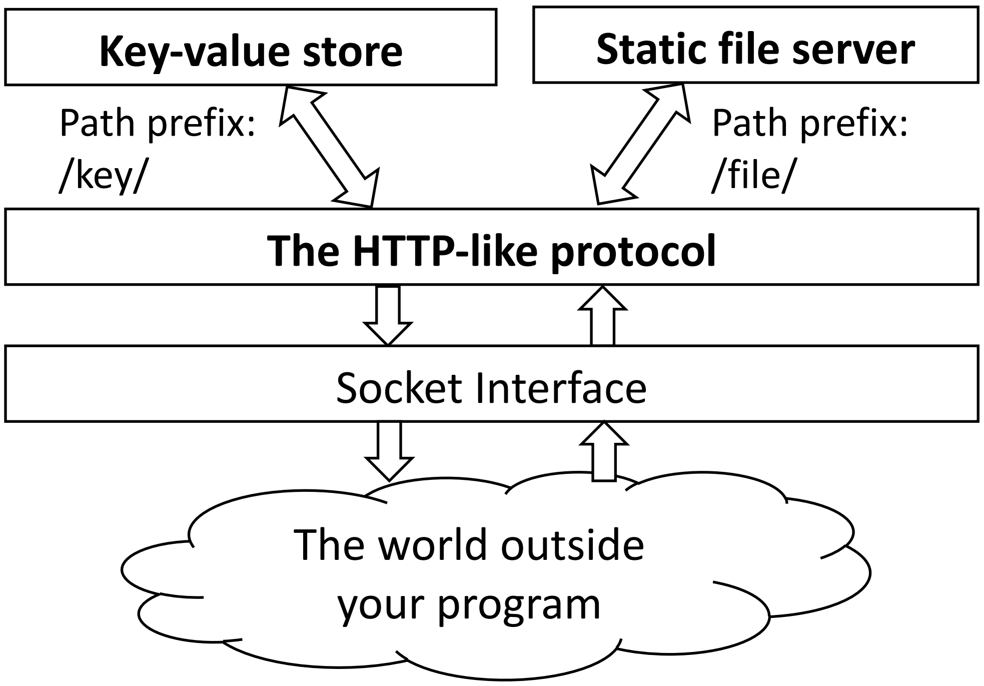


Figure 1: A layered architecture of the server. You need to implement the three components highlighted with bold fonts.

value store is to be implemented under two different path prefixes. Figure 1 shows the overall architecture of the web server.

The HTTP-like Protocol

The basic mechanism of this HTTP-like protocol is the same as that of HTTP 1.1. That is, when the server accepts a new TCP connection from a client, it reads the request headers and an optional request body sent by the client, and then sends valid response headers and optionally a response body. After such a request-response cycle, the connection is persisted (not closing immediately), and the server waits for a new request from the client or a notification for closing the connection.

For the simple scenarios in this assignment, the server only needs to handle the basic request and response information along with the **"Content-Length"** header field. Therefore, we simplify the header format of this protocol from that of the actual HTTP 1.1 as follows:

1. All new-line (i.e. "\r\n") and colon delimiters are replaced by single-space delimiters, as we do not have header components that contain spaces. Here, space is defined as the character with ASCII code 32. This means that the header is now a string consisting of non-empty substrings concatenated with a space in between, and that two consecutive spaces mark the end of header. The header string should contain only ASCII characters.
2. Each header field consists of two substrings. The first is the case-insensitive name of the header field, and the second is the value of this field.
3. For a request, the first two substrings are the case-insensitive HTTP method followed by the case-sensitive path. If there is a content body after the header, a "Content-Length" header and its value follow the first two substrings. Finally, the header is ended with two consecutive spaces. For example, the entire request could be
   a. `GET␣/file/cs2105.txt␣␣`
   b. `GET␣/file/CS2105.txt␣␣`
   c. `POST␣/key/ModuleCode␣Content-Length␣6␣␣`**`CS2105`**

   Here, space is denoted by ␣ and content body is denoted by bold fonts.
4. For a response. The first two substrings are the HTTP status code followed by a non-empty description of the status. The description has no real effects and is kept as a convention as in HTTP. If the response has a content body, a "Content-Length" header should similarly be included. Corresponding to the three sample requests above, the responses could be
   a. `404␣NotFound␣␣`
   b. `200␣OK␣content-length␣27␣␣`**`Intro␣To␣Computer␣Networks!`**
   c. `200␣Okay␣␣`

The server can assume that the client only sends correctly formatted requests. The client may send unknown headers which the server should ignore.

The Key-value Store

A key-value store can be understood as a dictionary data structure that supports insertion, update, retrieval, and deletion of key-value pairs, with keys functioning as pointers to the values. In this assignment, we restrict keys to be case-sensitive strings for simplicity. Due to the use of "Content-Length" header, values can safely be any binary string. The server should keep all data in memory only and avoid accessing the disk.

The key-value store should be accessible to the client through paths with prefix "/key/". To operate on a specific key, the complete HTTP path is "/key/" appended with the actual key string. For example, a request to operate on key `ModuleCode` sends `/key/ModuleCode` as the HTTP path. With a path that identifies a key, the server should support the following key-value operations over the HTTP-like protocol:

1. Insertion and update of a value
    a. For the request, the HTTP method is "POST", and the value to be inserted or updated should follow the header as the content body. There should also be a "Content-Length" header indicating the number of bytes of the value string.
    b. The server should respond with a "200" status code after inserting or updating the value. In this case, the client always expects the insertion or update to be successful.
2. Retrieval
    a. For the request, the HTTP method is "GET", and there is no content body.
    b. For the response, if the key does not exist in the store, the server returns a "404" code. Otherwise, the server should return a "200" code, send the correct "Content-Length" header and then the value string as the content body.
3. Deletion
    a. The HTTP method of the request is "DELETE", and there is no content body.
    b. For the response, if the key does not exist, the server returns a "404" code. Otherwise, it should delete the key-value pair from the store and respond with a "200" code and the deleted value string as the content body. The "Content-Length" header should also be sent accordingly.

Static File Serving

The web server should serve static files in the current working directory under the path prefix "/file/". For example, when a client sends a request to get the path "/file/CS2105.txt", the server should try to open and read the file with relative path "CS2105.txt". If the entire file can be successfully opened and read, the server returns code "200" and the file content as the content body, with "Content-Length" header properly sent. If the file does not exist, the server should return code "404". Otherwise (e.g. the server fails to open the file for reading due to lack of permission), the server should return code "403".

Your Task

You are given a folder containing multiple files for testing purpose. Write your program into only one file named WebServer.xxx where xxx is the corresponding file extension for respective language. Refer to Program Submission Section earlier for more details.

To run the Web Server program on sunfire, use the following command:

```
python3 WebServer.py <port>
```

For example:

```
python3 WebServer.py 8000
```

You should choose a port number greater than 1023 because operating systems restrict the access to ports lower than 1024. If you get a `BindException: Address already in use` (or similar), try a different port number. Remember to configure your python3 alias and refer to assignment 0 if you forgot how to do it. **Note: regardless of language you are using, the one and only one compulsory argument passed to the program should be the port number, and there's no standard input.**

To solve the task, you have to develop a web server program. 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.

Detailed specification is described in previous sections named **The Web Server, The HTTP-like Protocol, The Key-value Store and Static File Serving**.

Following good OOP practise, suggested method/class names to be implemented are shown in bold font below. Students can choose different names for their methods as well.

1. The **`start`** method of the **`WebServer`** class should create a **`ServerSocket`** 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. First, you have to read a HTTP request from the socket. Next, you should call **`parseRequest`** of the **`HttpRequest`** class to parse the request.
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 decide whether to send a 200 OK or a 404 response. **`formHttpResponse`** returns a byte array which can be sent to the client using **`sendHttpResponse`**.

Testing Your Solution

We will grade your submission on **`sunfire`**. Thus, it is a good idea to test your submission on **`sunfire`** before submission. In the past we had submissions that do not work on **`sunfire`**, e.g. because **`sunfire`** might have a different Java/python versions installed. Unless stated otherwise for individual tasks, you are allowed to use libraries installed in public folders of sunfire (e.g. /usr/lib) only.

You can test your Web Server in many different ways. For example, using the command line, or by manually sending requests to the Web Server. We also provide a simplified test case.

Test case: We will use carefully crafted scripts to test the functionality of your submission. For your convenience, we provide a simplified and condensed version of our test cases for your own testing. We strongly recommend you to execute this test case before submission. Passing this test case does not guarantee full marks but failing it means something is wrong. To execute the test case, you need to

(1) open two SSH windows to sunfire

(2) run your Web Server in one window (refer to previous section)

(3) execute the following command in another window with port number the same as the one passed to the WebServer:

```
bash test.sh <port>
```

For example,

```
bash test.sh 8000
```

Grading Rubric

1. The HTTP-like protocol (4 marks)
    a) The server correctly understands all valid requests and sends valid responses only
    b) The server supports persistent connection with pipelining as specified
    c) The server returns correct status codes according to the specifications
    d) The server supports different functions under different prefixes of the HTTP path
2. The key-value store (4 marks)
    a) The server supports insertion and update of key-value pairs through the POST method
    b) The server supports retrieval of value by key through the GET method
    c) The server supports removal of key-value pairs through the DELETE method
3. Serving static files (2 marks)
    a) The server serves static files with the path relative to the directory where the server program is started

During grading, we will only transfer files that are sufficiently small (< 5 MB). Thus, it is safe to load the entire file into memory. Please refer to assignment 0 exercise 2 if you are unsure how to do it.

Question & Answer

If you have any doubts on this assignment, please post your questions on IVLE forum or consult the teaching team. We are not supposed to debug programs for you, and we provide support for language-specific questions on a best-effort basis only. The intention of Q&A is to help clarify misconceptions or give you necessary directions.