COM SCI 118 Winter 2023
Computer Network Fundamentals

Project 1: Web Server Implementation using BSD Sockets
Due date: Feb 3rd, 11:59 p.m. PT

1 Goal

In this project, we are going to develop a Web server in C/C++. Also, we will take the chance to learn a little bit more about how the Web browser and server work behind the scene.

2 Lab Working Environment

Please use the template from `https://github.com/luckiday/CS118-W23-Project1`. You need a plain-text editor to edit the source code. You may use vim/emacs/nano in Unix/Linux systems. Since C/C++ is a cross-platform language, you should be able to compile and run your code on any machine with a C/C++ compiler and BSD socket library installed. No high-level network-layer abstractions (like httplib, Boost.Asio or similar) are allowed in this project. You are allowed to use some high-level abstractions, including C++14 extensions, for parts that are not directly related to networking, such as string parsing.

This project is graded on Ubuntu 22.04 LTS (Jammy Jellyfish). You may develop the project on compatible a Unix environment (including macOS) but we highly recommend testing under the same Ubuntu environment before testing. You may obtain a copy of Ubuntu 22.04 from its official website (`https://releases.ubuntu.com/22.04/`, 64-bit PC AMD64 desktop image) and install it in VirtualBox (`https://www.virtualbox.org`) or other similar VM platforms. Note that the OS you use must have a Web browser that ships with a graphical interface for testing. We do not support Windows for Project 1 because of its different socket programming conventions and behaviors. You may use Windows Subsystems for Linux (WSL) or a virtual machine. For macOS users, please also test your code on Ubuntu before submission.

3 Instructions

1. Read the HTTP sections in Chapter 2 of the textbook carefully. The textbook will help you understand how HTTP works. For details of HTTP, you can refer to RFC 1945 (`https://tools.ietf.org/html/rfc1945`). You should carefully go over the socket programming slides posted and discussed by the TAs. Note that, you must program in C/C++ rather than in Java as the textbook shows.

2. Implement a "Web Server" by responding to the client's HTTP request. The "Web Server" should parse the HTTP request from the client, creates an HTTP response message consisting of the requested file preceded by header lines, then sends the response directly to the client. In order to test it, you should first start your Web server, and then initiate a Web client. For example, you may open Mozilla Firefox and connect to your Web server.

3. Your server should support several common file formats so that a standard Web browser can display such files directly instead of prompting to save to the local disk. The minimum supported file types must include following types:

   - plain text files encoded in UTF-8: *.html and *.txt
   - static image files: *.jpg and *.png
   - document files: *.pdf

   Your server should also correctly transmit file content as binary data if the filename does not contain any file extension. Whatever data received by an HTTP client should be identical to the data requested, which can be checked by the `diff` program. (Hint: you can set MIME type of the HTTP response as application/octet-stream for binary file requests.)

   Your server does **NOT** need to handle the following scenarios:

   - The client requests files in any subdirectories.
   - The client requests a file that does not exist.
   - The client requests a file named with special characters other than alphabets, periods, spaces or % signs.

4. Pay attention to the following issues when implementing and testing the project.

   If you run the server and a Web browser on the same machine, you may use `localhost` or `127.0.0.1` as the name of the machine. **Instead of using port 80 or 8080 for the listening socket, we use port 15635 to avoid conflicts. We will also use this port in the grading.**

   After you are done with implementing the web server, you need to test it. You can first put an HTML file in the directory of your server program. Then, you should connect to the server from a browser using the URL `http://<machinename>:<port>/<filename>` (e.g., `http://localhost:15635/test.html`) and see if it works. Your browser should be able to show the content of the requested file (or display image).

5. Useful tips:

   - It would be helpful for debugging if you print out the HTTP request header that the server received.
   - When constructing the HTTP header response, you only need to implement the required header fields. For example, HTTP version, status code, content type, etc.
   - If the client does not recognize the body data, please check whether you put a blank line after the last line of your HTTP header, i.e., \r\n\r\n .
   - If the browser can successfully receive the response of a text file but fails to display the content. This means that your server probably only sends null bytes to the browser. You can verify this by changing the content type to application/octet-stream to download this file and use hexdump to examine the file.

4 Grading Criteria

Your code will be first checked by a software plagiarism-detecting tool. If we find any plagiarism, you will not get any credit, and we are obliged to report the case to the Dean of Students. Your code will be graded based on several testing rubrics. We list the main grading criteria below; more details will be announced via Bruin Learn.

- The server program compiles and runs normally.
- The server program transmits a binary file (up to 1 MB) correctly. We will test binary file transmission by requesting a filename with no extension. You can test this function with the following commands.

```
$ cat /dev/urandom | head -c 1000000 > binaryfile # generate a 1MB file
$ curl -o downloadfile <machinename>:<port>/binaryfile # request the file
$ diff downloadfile binaryfile # check if the downloaded file is intact
```

- The server program serves a plain text file correctly, and it can show in the browser.
- The server program serves an image file correctly, and it can show in the browser.
- The server program serves a file with a file name containing white spaces (e.g., ucla icon.jpg).
- The server program serves a file with a file name containing % signs (e.g., ucla%icon.jpg).

5 Project Submission

The project due date is 11:59 p.m. on Friday, Feb. 3rd on Gradescope. Late submission is allowed by Sunday, Feb. 5th (20% deduction on Saturday, 40% deduction on Sunday).

You need to submit a zip archive that contains all the source code and a `Makefile` that can compile your server program. The `Makefile` should be located in the root directory within your submission. The `make` command will compile the program and **produce a single standalone executable named `server`**.

The server program will be graded by an autograder on Gradescope. You will be able to see part of the test result after submission. We don't limit the number of submission attempts, and the final grading is determined by your *last* submission.