COM SCI 118 Spring 2020
Computer Network Fundamentals

Project 1: Web Server Implementation using BSD Sockets
Due date: April 24th, 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

You need a plain-text editor to create 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 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++11 extensions, for parts that are not directly related to networking, such as string parsing.

We recommend you develop and test your code in Ubuntu 16.04.3 LTS (Xenial Xerus). You may obtain a copy of Ubuntu 16.04.3 from its official website (http://releases.ubuntu.com/16.04/, 64-bit PC AMD64 desktop image) and install in VirtualBox (https://www.virtualbox.org) or other similar VM platforms. Note the OS you use must have a Web browser that ships with a graphical interface for testing. We do not support Windows workstations for Project 1 because of its different socket programming convention and behaviors. Use of macOS is allowed but not encouraged, since TAs may not provide support for OS-related issues other than Linux.

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 go over carefully on 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 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 local disk. The minimum supported file types must include the four types listed below:

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

   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 for files in any subdirectories.
   - The client request a file that does not exist.
   - The client request a file named with special characters other than alphabets and period.

4. Pay attention to the following issues when you are 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, you should pick your own to avoid conflicts. It is suggested not to use port numbers 0 – 1024 (these are reserved ports).

   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:5000/test.html`) and see if it works. Your browser should be able to show the content of the requested file (or displaying 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 just need to add the header fields required. 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 Student. Your code will be graded based on several testing rubrics. We list the main grading criteria below; more details will be announced via CCLE.

- 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.

5 Project Submission

Project due date is 11:59 p.m. on Friday, April 24th on CCLE. Late submission is allowed by Sunday, April 26th (20% deduction on Saturday, 40% deduction on Sunday). Put all your files into a directory, compress the directory and generate a UID.tar.gz where UID is your UCLA ID. Your submission should include the following files:

- Your source codes (e.g. webserver.c). The code of the server can be more than one file.
- A Makefile. The TAs will only type make to compile your code.
- A README file which will contain following information
  - Your name, email, and UCLA ID
  - The high level design of your server (one paragraph)
  - The problems you ran into and how you solved the problems (up to three paragraphs)
  - List of any additional libraries used Acknowledgement of any online tutorials or code example (except class website) you have been using.