

COM SCI 118 Spring 2018

Computer Network Fundamentals

Project 1: Web Server Implementation using BSD Sockets
Due date: April 27th, 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 can 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.

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. SEASnet linux servers are also accepted for testing. However, 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 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 Chapter 2 of the textbook carefully. The textbook will help you understand how HTTP works. You can also use the socket programming materials covered by your TA to learn more on the C/C++ socket programming.

You must program in C/C++ (not Java). To help you get familiar with socket programming, we provide a demo client-server code at <http://metro.cs.ucla.edu/courses/cs118/project1-example/>.

2. Part A: Implement a “Web Server” that dumps HTTP request messages to the console. This is a good chance to observe how HTTP works. 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. Your Web server should print out the HTTP request header it received. You should be able to explain what do the fields in the message mean by looking up in the textbook or RFC 1945. Do not provide extra debugging output in your submitted code.
3. Part B: Complete your “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.

Your server should correctly transmit file content as binary data. Whatever data received by an HTTP client should be identical to the data requested, which can be checked by the `diff` program. If the requested file does not exist, your server should display an HTML page showing 404 not found error.

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 if the file format is not supported. These supported file types include plain text files (i.e. `*.html`, `*.htm`, and `*.txt`), JPEG and GIF images (i.e. `*.jpg`, `*.jpeg`, and `*.gif`).

A Web client may request a file with or without file extension. File names are handled in a case-insensitive fashion (i.e. `ABC.jpg` and `abc.JPG` are referring to the same file), therefore an HTTP request for `ABC.jpg` should get a valid response if `abc.JPG` exists in the requested path on the server. Your server code also needs to handle the case where file name contains space. For example, a file named `new image.jpg` on the server should be served correctly if it is requested. Your server does not need to handle requests for files in any subdirectories. All files are supposed to be accessed in the same level as the actual server program.

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

If you are running the browser and server 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 both parts, you need to test your server. 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>` and see if it works. For your server in Part A, you should be able to see HTTP requests in the console of your server machine. For your server in Part B, your browser should be able to show the content of the requested file (or displaying image).

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 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 can print out the correct HTTP request messages to the console.
- The server program transmits a small binary file (up to 512 bytes) correctly.
- The server program transmits a large binary file (up to 100 MB) correctly.
- The server program serves an HTML 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 which file name contains space correctly in the browser.

5 Project Submission

Project due date is 11:59 p.m. on Friday, April 27th on CCLE. Late submission is allowed by Sunday, April 29th (10% deduction on Saturday, 20% 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. Only one member of your team is required to submit your project to CCLE. 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.
- Your report in PDF format only (report.pdf).
- A README file which will contain student names and student IDs.

Your report should not exceed 6 pages. In your report:

1. Give a high-level description of your servers design.
2. What difficulties did you face and how did you solve them?
3. Include a brief manual in the report to explain how to compile and run your source code (if TAs cant compile and run your source code by reading your manual, the project is considered not to be finished).
4. Include and briefly explain some sample outputs of your client-server (e.g. in Part A you should be able to see an HTTP request). You do not have to write a lot of details about the code, but just adequately comment your source code. The cover page of the report should include the name of the course and project, your partners name, and student id.