COM SCI 118 Computer Network Fundamentals

Project 1: Concurrent Web Server using BSD Sockets
Fall 2014

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 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 vi/pico/emacs 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.

Students should use Linux OS to finish the project. A virtual machine image of Ubuntu OS for your working environment is provided. (You can download it from *http://metro.cs.ucla.edu/cs118/CS118.rar*) The account information is cs118/cs118 and root/cs118. It is recommended that doing the projects or at least testing them in this provided environment, because:

- C/C++ compiler (gcc, g++) and BSD socket library on those workstations have been verified.
- Your submitted code will be tested and evaluated on them.

You need to install VirtualBox or VMware Player, which are freely available online.

Windows workstations are **NOT** recommended. They have their own idiosyncracies which may be too much for this simple project. Also, TAs will not be able to help you on any of these platforms.

3 Instructions

1. Read Chapter 2 of the textbook carefully. The textbook will help you to understand how HTTP works. However, there are some differences:
   - You must program in C/C++ (not Java). Examples of socket programming Client/Server code in C are available on the course Website. They are also put in the folder /home/cs118/workspace in the provided Ubuntu virtual machine image.
   - Instead of using port 80 or 8080 or 6789 for the listening socket, you should pick your own to avoid conflicts. It is suggested **NOT** to use port numbers 0-1024.
2. The project consists two parts:
   - Part A: Implement a "Web Server" that dumps request messages to the console. This is a good chance to observe how HTTP works. So start a browser, e.g. Mozilla Firefox (browser is the client side), request a file (e.g., a plain text document) to your server, record the request message, and find out what the fields in the message mean by looking up in the textbook or *RFC 1945*.
   - Part B: Based on the code you have done in Part A, add the following function to the "Web server": the "Web server" parses the HTTP request from the browser, creates an HTTP response message consisting of the requested file preceded by header lines, then sends the response directly to the client.

3. Pay attention to the following issues when you are implementing 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.
   - Make sure your *"Content-Type"* function recognizes at least html files. We will worry about other types of files later.

4. After you're done with every part (Part A and Part B), you need to test your server. You first put a html file in the directory of your server, or more exactly, where you run your server. Connect to your server from a browser with the URL of http://<machine name>:<port number>/<html file name> and see if it works. For your server in Part A, you should be able to see the HTTP request format 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 html file.

5. Back to the *"Content-Type"* function. Please add the support for binary files (e.g., images). Your browser should be able to correctly show these image files. Note that we will test **both plain text and binary files** for your server.

4 Materials to turn in

1. Your source code files (e.g. webserver.c)

2. A report of your project (pdf or word format) should not exceed 6 pages (except from source codes). In the report:

   - Give a high-level description of your server's design
   - What difficulties did you face and how did you solve them?
   - Include a brief manual on how to compile and run your source code (if TAs can't compile and run your source code by reading your manual, no credit would be given).
   - Show some simple test results from your program (e.g. in Part A you should be able to see an HTTP request)

   Please offer clear and adequate comments in our source code.

3. The cover page of the report should include the name of the course and project, your partner's name, student id, and SEASnet login name. **In each group there can be at most 2 students**.

4. Method of submission is described in Section 5.

5 Project Submission

1. Put all your files into a directory, named "project1_UID" where UID is the UCLA ID of one of the students in your team.

2. The directory "project1_UID" should include the following files:

   - Your Web Server code (e.g. webserver.c). The code of the server can be more than one files.
   - Your report (report.doc, report.pdf).
   - A Makefile. The TAs will only type "make" to compile your code, make sure your Makefile works under SEAS.
   - A README file which will contain Student names and Student IDs.

3. Each group just needs to submit one set of files.

4. Due date is Friday, October 24, 2014.