CSE3461 Lab1: Concurrent Web Server using BSD Sockets

Due Date for Electronic submission: Tue, Feb 14, 2017 (23:59:59 PM)

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 can use the same environment in miniLab.

In the assignment package, several files are provided to test with your web server. Of course, you are free to create and use your own test files.

- text.html: a html file which contains text only.
- picture.html: a html file which contains text and a small picture ($<$ 200KB).
- bigpicture.html a html file which contains text and a big picture ($>$ 1MB).
- *.jpg and *.jpeg: picture source files.

3 Instructions

1. Read Chapter 2 of the textbook carefully. The textbook will help you to understand how HTTP works. You likely learn how to do a socket programming. Here is one difference. 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 of three tasks, which are specified as following:

   - Task 1: 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. Microsoft Internet Explore (browser is the client side), connect 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*.
   - Task 2: Based on the code you have done in Task 1, add one more function to the "Web server", that is, 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.
   - Task 3: Back to the *"Content-Type"* function. Add in the support for GIF and JPEG images. Your browser should be able to correctly show these MIME files. You can use any images to test your program. You can use "diff" or "md" to ensure that the transferred file is the same as the original one.

   Finally, this simple web server needs to implement the following features:

   - When the client sends GET to request for one .html which exists, it should return this .html file.
   - When the client sends GET to request for one .html which **doesn't** exist, it should send the response as specified in HTTP 1.0.
   - When the client sends GET to request for one picture which exists, it should return the picture which will be displayed on the web browser.

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 both tasks, you need to test your server.

   - You can put all the html and picture files in the directory of your server, or more exactly, wherever 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 Task 1, you should be able to see the HTTP request format in the console of your server machine.
   - For your server in Task 2 and Task 3, your browser should be able to show the content of the requested html file.

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 2 pages (except from source codes). In the report:

   - Include a brief manual in the report to explain how to compile and run your source code (if the grader can't compile and run your source code by reading your manual, the project is considered not to be finished). A makefile is highly recommended (by default).
   - Give a high-level description of your server's design, especially highlighting what are unique in this project.
   - Include and briefly explain some sample outputs of your client-server (e.g. in Task 1, you should be able to see an HTTP request), as well as the results of Tasks 2 and 3 (esp. if not all tasks done).
   - Describe what difficulties did you face and how did you solve them, if applicable.

   You do not have to write a lot of details about the code, but just adequately comment your source code.
3. The first page of the report should include the name of the course and project, your name and student ID at the top.
4. Method of submission is described in Section **??**.

To submit,

1. Put all your files into one directory.
2. Zip or tar all the files in this directory and get one submission file, which is named "lab1_name_SID.zip", where name and SID are your name and student ID.
3. Submit this zipped file to Carmen.