Programming Project #2: Building a Web Server

In this assignment you will develop a Web Server capable of processing HTTP request messages. When complete your solution should be able to serve a simple webpage to any standard web-browser. You will to implement a subset of the HTTP 1.0 standard, as defined in RFC 1945, where separate HTTP requests are sent for each component of the Web page.

As you are developing the code you can test your server from a web browser or a command line program like telnet. Remember that you will not be serving through the standard port 80, so you need to specify the port number within the URL that you give to your browser. For example, assuming you are working on Isengard and your server is listening to port 6789, and you want to retrieve the file index.html, then you would specify the following URL within the browser: http://isengard.mines.edu:6789/file1.html. If you omit ":6789", the browser will assume port 80 which most likely will not have a server listening on it.

Functional Requirements

At a minimum, your submission must do the following to receive full credit.

- Basic requirements:
  - The executable should be called web_server.
  - Your program must be written in C/C++, and must compile and run on Isengard.mines.edu
  - You must provide a verbose mode that provides basic debugging information at each step your program takes. The –v flag should enable the verbose mode.
  - After selecting an available TCP port number to be used to listen for new connections be sure to print the port number selected so the grader knows what port to use for testing. IMPORTANT: to avoid interference by Isengard's host firewall the port number you select must be between 5000 and 10,000.
  - In addition to functionality you will be graded on the quality of the code, including readability, comments and the use of proper programing practices.
- Accept and process HTTP 1.0 GET requests.
  - You are not required to process header lines, but they should not create errors (that is, you should gracefully ignore them).
  - Your program must serve only files located in the same directory as the server that match the pattern fileX.html or imageX.jpg. You must limit the file name to a single digit, 0-9. For example, file1.html or image3.jpg would be valid. It is important that you do not serve any other files. This is to protect your data, as well as the grader's. If you don't filter the filenames a malicious user could find your server and request any file they want.
- Respond to valid GET requests with a valid response. All responses should include at least:
  - The status line.
  - The Content-Length header line.
  - The Content-Type.
  - The file that was requested.
- Respond to invalid requests with the appropriate error.
  - If the request is a valid GET request but specifies a file other than those allowed, return a 404.
  - If the request is not a valid GET request, return a 400. Note that the request might be a valid HEAD or POST request, but since you don't have to implement those you should return a 400 to them as well.

- After responding to a request your code should return to the main loop and wait for the next request.
  - You can terminate your program with CTRL-C.

Hints and Suggestions.

- Since much of the underlying functionality is the same, you can use your echo server from assignment #1 as the starting point for this assignment. Most of the changes you will need to make will involve what you do with the data once you have a connection established and are reading data from the network.
- Note that it is best practice to catch the CTRL-C (i.e., SIGINT) and close your open socket before exiting. You are not required to do this for the program, but you should be aware that if you don't close the socket the port number will remain open for a brief period, which makes developing the code a hassle.
- There is a completed version of the assignment in my directory on Isengard that you can use to compare your behavior with what I'm looking for. The file is ~promig3/pub/bin/web_server
- The files we will use to test your program are in ~promig3/pub/webFiles You can copy those files to your directory to test your program.
- Remember that the image files are 8-bit binary. You must read them using an API that doesn't modify or interpret the data in any way. Be particularly wary of the **iostream** library. My recommendation is to use the very basic read() and write() system calls.

What to submit.

This assignment is due by the end of the day on Sunday, October 3rd. There will be a 5% per day penalty for late submissions with a maximum late penalty of 40% (but remember we need time to grade your submission before the end of the semester).

You should submit a single tarball of a single directory containing a makefile, a README.txt with your name and any information I need to compile the program and the source files needed to build your program. The single directory must be named with your username. The grader should not need to do anything other than untar your files, cd into your directory, type make and begin testing.

Do not include any core, object or binary files in the tarball. The Makefile provided with the sample code includes a target named submit that will create the tarball in the format we are looking for. All you need to do is:

```
$> make submit
```

In addition to functionality, you will be graded on the quality of the code, including readability, comments and the use of proper programing practices.