Programming Project #1: Building an HTTP 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 netcat (nc) or curl. Linux restricts the use of port numbers less than 1023 to processes that run with root permissions. It is a bad idea to test programs as root, so you should use port numbers larger than 1023 when running your program. When testing with a browser you will need to specify the port number within the URL that you give to your browser. For example, assuming you are working on your VDI 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://localhost: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 (CSCI 471 and 598-J)

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 Ubuntu 24.04 LTS. We will test your submission in the virtual lab.
  - You must support an optional debug flag, setting the verbosity of debugging messages.
    - -d <num> # Number between 0 and 6, with 0 the least verbose.
  - 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.
  - In addition to functionality, you will be graded on the quality of the code, including readability, comments and the use of proper programing practices.
  - Files are severed relative to a directory named data, which is located in the CWD when you launch your web server.

- 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 subdirectory *data* 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.
  - While not required, it is best practice to catch the SIGINT sent by CTRL-C so you can cleanly close the listening socket. There is a document on Canvas with instructions for catching signals.

Additional Requirements (CSCI 598-J Only)

Students in CSCI 598-J must implement the other two methods that are part of the HTTP 1.0 standard, which are the HEAD and the POST methods.

- Respond to valid HEAD requests with a valid response.
    - A HEAD response is identical to the response to a GET request, except you don't send the file.
- Respond to a valid POST request by saving the transmitted file.
    - Check to see if the requested filename already exists.
        - If the file does not exist, create it.
        - If the file does exit, overwrite it.
    - Save the body of the request.
    - Send the appropriate response.
        - Return a 200 OK if the file already existed and you overwrote it.
        - Return a 201 Created if the file did not exist and you had to create it.
- Unlike the 471 version of this assignment, 598-J submissions are required to catch the SIGINT and cleanly close the listening socket.

Hints and Suggestions.

- There is skeleton to get you started on github.
- 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.

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 project 1's sample code includes a target named submit that will create the tarball in the format we are looking for.

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