Programming Assignment 1:
Building a Multi-Threaded Web Server[1]

In this assignment you will develop a multi-threaded Web server that is capable of processing multiple simultaneous service requests in parallel. You should be able to demonstrate that your Web server is capable of delivering your home page to a Web browser.

We are going to implement version 1.0 of HTTP, as defined in RFC 1945, where separate HTTP requests are sent for each component of the Web page. The server will be able to handle multiple simultaneous service requests in parallel. This means that the Web server is multi-threaded. In the main thread, the server listens to a fixed port. When it receives a TCP connection request, it sets up a TCP connection through another port and services the request in a separate thread. To simplify this programming task, we will develop the code in two stages. In the first stage, you will write a multi-threaded server that simply displays the contents of the HTTP request message that it receives. After this program is running properly, you will add the code required to generate an appropriate response.

As you are developing the code, you can test your server from a Web browser. But remember that you are not 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/index.html. If you omit ":6789", the browser will assume port 80 which most likely will not have a server listening on it.

The textbook authors recommend you develop your code in two phases. In the first phase you create the code needed to listen for incoming requests. When a request arrives you display it rather than responding to it. That ensures you get the socket() API setup correctly before you try to parse the HTTP protocol. In the second phase you can add the code that responses to the HTTP requests.

Web Server in C/C++: Part A

Download the skeleton code I created from Blackboard, unpack it and ensure that it will compile and link on the system you want to use for development. Remember that you may develop on any system you like but what you submit must compile and run on isengard.mines.edu. If you have any concerns about the portability of your code you should develop directly on Isengard.

Once you have the skeleton setup and running you should start filling in the code. You may of course do this however you like, but I would recommend the following:

1. Fill in the sections of main() required to create, bind, establish a listening queue and then close a stream type socket. Running the program shouldn't show any new output but it shouldn't produce errors either.

2. Add code to accept new connection requests and as requests are received call the processRequest() function in a new thread. Be sure the create the thread detached or we have to clean up after it later. Once this is complete when you run the program it should wait at the accept call. You can then

[1] This assignment is based off and borrows heavily from a similar assignment provided on the textbook's companion website. The text of the assignment is almost identical to that assignment, my primary change has been to modify it to use C++ rather than Java. The original assignment can be viewed at the URL: http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html

connect to the program with a web browser (or telnet). When you connect the processRequest() function should print a message but your web browser might complain or just hang since it's not getting any response.

3. Modify the processRequest() function so that it reads the full contents of the request and echos it to stdout. You may also want to fill in the send404() function so that your web browser gets something meaningful back from your program.

At this point you should have the functioning skeleton of a TCP server application. Up to this point the structure of all server applications are almost identical, regardless of the application protocol they are going to implement.

Web Server in C/C++: Part B

Instead of simply terminating the thread after displaying the browser's HTTP request message, you should now analyze the request and send an appropriate response. For this assignment we are going to make several simplifying assumptions: (1) you may assume that the only method sent will be a GET, (2) you may assume that the browser sends only non-persistent connection requests and (3) you may ignore the information in the header lines, and use only the file name contained in the request line.

1. Modify the processRequest() function so that rather than simply echo'ing the input onto stdout it passes the file descriptor to readGetRequest() then add code to the readGetRequest() function so that it reads the request from the file descriptor looking for the line with the GET request on it. For simplicities sake you may assume that the GET method will be properly formatted with GET as the first three characters of the line. readGetRequest() should return the string containing the GET keyword and the path to the resource being requested.

2. Modify the findFilename() function so that it can parse the GET request into the three standard components; the GET keyword, the filename and path and the version of HTTP used. Since HTTP requests should always be relative to a particular directory (so you don't accidentally expose the whole filesystem) you should prepend the path to the directory you do want to expose. In this case prepend "." to make the request relative to the current directory.

   Note that a real webserver will include other protections to keep requests from traversing up the path, including but not limited to using chroot. Since we can't do that you must remember that as long as your program is running anyone who knows, or can figure out, what port you are using can get any of your files. Because of this I recommend that while testing you check to see what file the request is looking for and compare it to a whitelist of filenames hard-coded into your program.

3. Once findFilename() has parsed the GET request and ensured the filename is reasonably safe it can return the full path to processRequest(). Now that processRequest has the filename it should confirm the file exists. If it does not exit send back a 404 error (by calling send404()) and exit.

4. Modify sendHeader() so that it builds the reply header per the HTTP standard. At a minimum your reply header should include:
   4.1. The status line
   4.2. The Content-Length header line.
   4.3. The Content-Type header line. For this assignment you may assume that the file is either a plain text file with the extension .txt, an html file with the extension .html or an image with

the extension .jpg. If you want to challenge yourself a little try expanding this list using one of the standard mime-type databases available on Isengard.

4.4. Remember to terminate the header with a blank line (return/linefeed combination).

5. Modify the sendFile() function so that it sends the file.

6. After sending the response the processRequest() function should close the file descriptor and exit. Since we created the thread with the **detached** state there is no need to clean up after the thread.

This completes the code for the second phase of development of your Web server. Try running the server and using a web browser to look some pages. Remember to include a port specifier in the URL of your home page, so that your browser doesn't try to connect to the default port 80.

What to submit.

This assignment is due by the end of the day on Sunday, September 22nd. There will be a 5% per day penalty for late submissions (the maximum late penalty is 20%, but remember we need time to grade you 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.

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