Lab 2: Building a Web Server

There are two programs you need to create in this assignment. You need to turn in a single PDF file with four items:

1. **A brief description of the basic logic of your program**
2. **Sample runs and screenshots as instructed to demonstrate the correctness of your program**
3. **Answers to questions raised in the assignment**
4. **Appendix that contains full listing of your source code – preferably pretty-print from your development platform rather than cut-and-paste to your word document.**

As a prerequisite, please work through the practice codes in the Application Layer and Socket Programming module on Canvas.

1. Web Server Program (40 points)
   In this problem, you will learn the basics of socket programming for TCP connections: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

   You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "`404 Not Found`" message back to the client. It is important to remember that in the HTTP header, only the first line and the last line before the data are mandatory.

   You will be provided the skeleton code for the Web server in the file `WebServerPROB.c`. You are to complete the skeleton code. The places where you need to fill in code are marked with

   ```
   # FILL IN START
   ```
   and
   ```
   # FILL IN END
   ```

   Each place may require one or more lines of code. Your code should be able to handle both text and final files. Two HTTP objects (`helloworld.html` and `image.jpg`) will be provided to you for testing.

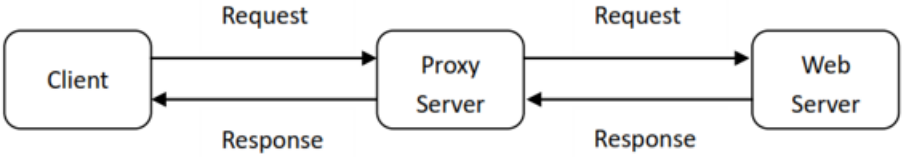
   Tasks:
   1. Use my colab testing script (Step 1-4) to test your web server using the loopback address (localhost). Make sure you test both HTTP objects and screenshot the results.
   2. Test your web server and client browser running on two different computers outside of colab. You can use your laptop and the server at school, or between the two team members' computers. Make sure to provide the screenshots.

2. (60%) Web Proxy Program

   In this problem, you will learn how web proxy servers work and one of their basic functionalities – caching. Your task is to modify your web server into a web proxy server which can cache web pages. It is a very simple proxy server which only understands simple GET-requests, but is able to handle all kinds of objects – not just HTML pages, but also images.

   Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance, we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server

will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.



You should use your solution from problem 1 as a starting point to complete this problem.

Tasks:

3. Continue to use my colab testing script (Step 5 onwards) to test both your webserver and proxy programs using the loopback address. A complete test should involve the client retrieving the same object twice, with the proxy server requesting the object from the server the first time and retrieving from the cache the second. You can demonstrate that based on the output log from your proxy server. Make sure you test both HTTP objects and screenshot the results.
4. Test your code again with two computers – the proxy server and the client can be run in the same computer while the web server should be on a different one. Make sure to provide the screenshots.
5. If you are successful at step 4, flush the cache at the proxy server. Fire up Wireshark at the router to show that consecutive retrievals of the same file through the proxy server would have the anticipated behavior at the proxy. Report the time required for the two retrievals. Explain your answers.
6. In this last test, replace your webserver program with an actual external Webserver of your choice. Comment on the performance of your proxy server.