Assignment 3: Concurrent Web Proxy

Due: April 22th, 11:55PM

1. Introduction

In this assignment, you are required to upgrade your sequential Basic Web Proxy of Assignment 2 so that it uses threads or processes to deal with multiple clients concurrently. That is, if your proxy is waiting for some event on behalf of one connection, it should be able to able to handle events for other connections as well.

This part will give you some experience with concurrency and synchronization, which are crucial computer systems concepts. You will be required to implement **two** versions of the concurrent web proxy: **a threaded version and a multi-process version**.

2. Handout Instruction

There is no handout for this assignment as the code written for the sequential Basic Web Proxy serves as the code base for the Concurrent Web Proxy. You may either use your own implementation of the Web Proxy or the *proxy.c* (a sample solution to Basic Web Proxy in proxy.zip) as your code base. You will have to reorganize the code base and write additional code to make it work for Concurrent proxy. Maintain the same project directory structure as in Assignment 2.

3. Implementation

Real proxies do not process requests sequentially. They deal with multiple requests concurrently. Once you have a working sequential logging proxy (Assignment 2), you should alter it to handle multiple requests concurrently. Following are two approaches to achieve concurrency;

1. Process based using file locking for synchronization: Create a new process to deal with each new connection request that arrives.
2. Thread based using mutexes for synchronization: Create a new thread to deal with each new connection request that arrives.

You are required to implement both the versions. Additionally, you need to ensure the following:

- The proxy should log all client requests to a shared log file proxy.log
- Access to shared resources like the log file is synchronized. Logging format and requirements is same as that of Assignment 2. For example, in the multithreading approach, it is possible for multiple peer threads to access the log file concurrently. Thus, you will need to synchronize access to the file such that only one peer thread can modify it at a time. If you do not synchronize the threads, the log file might be corrupted. For instance, one line in the file might begin in the middle of another
- Your multithreaded version should be **thread-safe**. Protect calls to thread unsafe functions such as *gethostbyaddr* and *gethostbyname* to keep these functions thread safe. (A piece of code is thread-safe if it only manipulates shared data structures in a manner that guarantees safe execution by multiple threads at the same time)

Please note that the concurrent proxy is an extension of the Basic Web proxy and thus contains all the functionalities and features mandated by Assignment 2.

4. Testing your Proxy

For this assignment, there is no script to check your implementation. You will have to come up with your own tests to help you debug your code and decide when you have a correct implementation.

5. Submission

You will need to submit your complete project directory on Canvas.

- Create a gzipped tar ball that includes all your source code.
- The filename for the tarball must be **username-concurrent-proxy.tar.gz** where username is your NETID.
- All your files in the tarball **must** be in a directory, named **username-concurrent-proxy**.
- Please include a README file to introduce how to compile/test the two versions of the proxy implementations.

The submission will be due on the date mentioned on Canvas.