PROGRAMMING ASSIGNMENT – 3

Due: Tuesday, 11/17 11:59 pm

NOTE: This assignment is NOT a group assignment, but an individual assignment.

1. Objective:

In this programming assignment, you are required to build a simple web proxy server that is capable of accepting HTTP 1.0 requests from clients, pass them to HTTP server and handle returning traffic from the HTTP server back to clients.

2. Background:

Ordinarily, HTTP is a client-server protocol; the client usually connects directly with the server. But some time it is useful to introduce an intermediate entity called proxy. The proxy sits between HTTP client and HTTP server. When the HTTP client requests resource from HTTP server, it sends the request to the Proxy, the Proxy then forwards the request to the HTTP server, and then the Proxy receives reply from the HTTP server, and finally the Proxy sends this reply to the HTTP client.

Here are several advantages of utilizing a Proxy:

1. **Performance:** If we cache a page that frequently accessed by clients connected to this proxy, then this can reduce the extra need of creating new connections to the actual HTTP server every time.
2. **Content filtering and Transformation:** The Proxy can inspect URLs from the HTTP client requests, and then decide whether to block connections to some specific URLs. Or to reform web pages (e.g. image removal for some clients with limited processing ability).
3. **Privacy:** Some times when the HTTP server tries to log IP address information from HTTP client, it can only get the information of the Proxy but not the actual client.

3. Assignment Description:

a. The basics

In this assignment one Proxy application **webproxy** need to be created using ANSI C/C++. It should compile and run without error with the following command

webproxy 10001&

The first argument is the PORT number that is used to listen to HTTP client requests. Your application should handle any port numbers not just 10001.

b. Listening to the client

When your application starts, it will create a TCP socket connection to be used for listening for incoming HTTP client.

When a HTTP request comes in, your application should be able to figure out whether this request is properly- formatted by the following criteria:

1) Whether the HTTP method is valid (please refer to RFC1945 section8 and sectionD.1 for valid methods). Our web proxy only supports GET (other methods such as POST and CONNECT do not need to be supported)
2) Whether there exist requested URL

If the criteria are not meet, a corresponding error message will be send back to HTTP client. If the criteria are met, then the request will be forwarded. You only have to handle the error messages presented in PA #1.

c. Parsing the information from client

When a valid request comes in, the Proxy will need to parse the requested URL, it basically disassembles the content into the following 3 parts.

1) Requested host and port number (you should pick the correct remote port or use the default 80 port if none is specified).
2) Requested path, this is used to access correct resource at HTTP server.
3) Optional message body (this may not appear in every HTTP request)

For example, the message received by the Proxy is nothing different from the web client since you only need to implement the GET method. Below we set the browser to use the local proxy running on port 8000 and captured the GET request to www.yahoo.com.

```
--
PA#2$ nc -l 8000
GET http://www.yahoo.com/ HTTP/1.1
Host: www.yahoo.com
Proxy-Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36
DNT: 1
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: B=fgvem7ha9r7hv&b=3&s=0o; ucs=sfcTs=1425977267&sfc=1
--
```

d. Connecting to HTTP server and relaying data from HTTP server to HTTP client

After the Proxy parsed the information from the client, it constructs a new HTTP request and send it over a new socket connection to the HTTP server. Then the Proxy will send back the reply from the HTTP server back to the HTTP client using the original socket connection with the HTTP client.

e. Testing your proxy

First you need to run your proxy.

webproxy <port> &

Then try telnet into localhost (127.0.0.1) <port> and GET http://www.google.com HTTP/1.0

```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.google.com HTTP/1.0
```

Please press two Enter keys after this.

If your proxy is working correctly, the headers and HTML contents of the Google homepage should be displayed.

You can also test your Proxy with your web browsers; detailed steps are mentioned in Appendix A.

4. Grading Criteria:

a. Multi-threaded Proxy (Mandatory for CSCI4273/5273 and ECE 5023 students)

The basic implementation does not require multi-threading. Here you should implement your application so that multiple clients can access the different servers via your Proxy simultaneously.

b. Caching (Mandatory for CSCI 4273/5273 and ECEN 5023 students and Extra Credits for CSCI 4273 students)

Caching is one of the most common performance enhancements that web proxies implement. Caching takes advantage of the fact that most pages on the web don't change that often, and that any page that you visit once you (or someone else using the same proxy) are likely to visit again. A caching proxy server saves a copy of the files that it retrieves from remote servers. When another request comes in for the same resource, it returns the saved (or cached) copy instead of creating a new connection to a remote server. This saves a modest amount of time and CPU if the remote server is nearby and lightly trafficked, but can create more significant savings in the case of a more distant server or a remote server that is overloaded (it can also help reduce the load on heavily trafficked servers).

Caching introduces a few new complexities as well. First of all, a great deal of web content is dynamically generated, and as such shouldn't really be cached. Second, we need to decide how long to keep pages around in our cache. If the timeout is set too short, we negate most of the advantages of having a caching proxy. If the timeout is set too long, the client may end up looking at pages that are outdated or irrelevant.

There are a few steps to implementing caching behavior for your web proxy:

1. First, alter your proxy so that you can specify a timeout value (probably in seconds) on the command line.
2. Second, you'll need to alter how your proxy retrieves pages. It should now check to see if a page exists in the proxy before retrieving a page from a remote server. If there is a valid cached copy of the page, that should be presented to the client instead of creating a new server connection.
3. Finally, you will need to somehow implement cache expiration. The timing does not need to be exact (i.e. it's okay if a page is still in your cache after the timeout has expired, but it's not okay to serve a cached page after its timeout has expired), but you want to ensure that pages that are older than the user-set timeout are not served from the cache.

Tip: when you check if the specific URL exists in the cache, you had better compare hash codes instead of the entire URL. For example, suppose that you store http://www.yahoo.com/logo.jpg with the hash key 0xAC10DE97073ACD81 using your own hash function or md5sum. When you receive the same URL from the client, you can simply calculate the hash value of the requested URL and compares it with hash keys stored in the cache.

c. Link Prefetch (Extra Credits for CSCI4273/5273 and ECEN 5023 students)

Building on top of your caching and content transformation code, the last piece of functionality that you will implement is called link prefetching. The idea behind link prefetching is simple: if a user asks for a particular page, the odds are that he or she will next request a page linked from that page. Link prefetching uses this information to attempt to speed up browsing by parsing requested pages for links, and then fetching the linked pages in the background. The pages fetched from the links are stored in the cache, ready to be served to the client when they are requested without the client having to wait around for the remote server to be contacted.

Parsing and fetching links can take an appreciable amount of time, especially for a page with a lot of links. For this reason, if you haven't already, at this stage you should make your proxy into a multi-threaded application. One thread should remain dedicated to the tasks that you have already implemented: reading requests from the client and serving pages from either the cache or a remote server. In a separate thread, the proxy

will parse a page and extract the HTTP links, request those links from the remote server, and add them to the cache.

d. HTTP 1.1 (Keep-alive connection) support (Extra Credits for CSCI4273/5273 and ECEN 5023 students)

5. Submission and Grading

You should submit your completed proxy by the date posted on the course website to Blackboard. Remember to submit after uploading. You will need to submit a tarball file containing the following:

1. All of the source code for your proxy
2. A README file describing your code and the design decisions that you made (will be graded).
3. Your tarball should be named <identikey>.tar.gz where <identikey> is your identikey.
4. Your code should compile without errors or warnings.
5. Your proxy should work with Firefox.

Appendix A.

Configuring Firefox to Use a Proxy

Because Firefox defaults to using HTTP/1.1 and your proxy speaks HTTP/1.0, there are a couple of minor changes that need to be made to Firefox's configuration. Fortunately, Firefox is smart enough to know when it is connecting through a proxy, and has a few special configuration keys that can be used to tweak the browser's behavior.

1. Type 'about:config' in the title bar.
2. In the search/filter bar, type 'network.http.proxy'
3. You should see three keys:
   network.http.proxy.keepalive(you may not have this),
   network.http.proxy.pipelining,
   network.http.proxy.version.
4. Set **keepalive** to false. Set **version** to 1.0. Make sure that **pipelining** is set to false.

HTTP Basics:
https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html