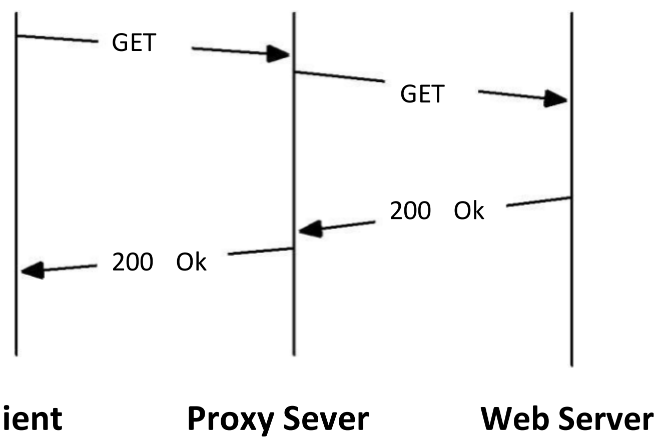
Network Programming Assignment #4:

Simple HTTP Proxy (100 points + 50 points Bonus)



In this assignment, you will implement a simple HTTP proxy server and HTTP command line client. This implementation will use HTTP/1.0, which is specified in RFC 1945 (*Note*: this is the older version of HTTP, which does not support Cache-Control HTTP Headers). HTTP operates over TCP. Your proxy server will start on a user specified port and listen for incoming HTTP requests.

When the proxy server receives a client request, it will first check its cached data in an attempt to serve the request. If there is not a valid cache entry, however, (1) the request will be proxied to the intended destination, (2) the response will be sent by the proxy to the client, and (3) the response will also be cached by the proxy for later use. Your proxy cache should maintain at least 10 document entries in the cache. Entries should be replaced in a Least Recently Used (LRU) fashion.

Entries should not be used if their timestamp indicates they are "stale." HTTP/1.0 supports the `Expires` header, which specifies the date/time after which the entity should be considered "stale." The `Expires` header is not a required header, however. A simple, but not foolproof, criterion for data to be considered "fresh" if the `Expires` header is missing is that the cache has accessed the document from the Web server recently and the document was last modified a relatively long time ago (there is an optional HTTP/1.0 `Last-Modified` header that most modern Web servers include in the header, which will allow you to determine when the document was last modified).

For this assignment (the Bonus does this a different way), you can assume that a document missing an `Expires` header is "fresh" if it was last accessed from a Web server in the preceding 24 hours and it was last modified one month ago or earlier. The HTTP/1.0 `Date` header represents the date/time the message was sent from the Web server, where the time is in Coordinated University Time (UTC), i.e., Greenwich Mean Time (GMT), and the date format is described in RFC 1945. You can use the value in the `Date` header to determine the 24 hour period. If both the `Expires` and `Last-Modified` headers are missing, the proxy should *not* cache the page.

Your proxy server should provide useful logging messages to `stdout` during its operation.

Your client will take input from the command line as to the web proxy address and port as well as the URL to retrieve. The client will then store the retrieved document in the directory where the client is executed. Your client should provide useful logging messages and error messages to `stdout` while downloading the file.

Usage Syntax:

```
/proxy <ip to bind> <port to bind>

/client <proxy address> <proxy port> <URL to retrieve>
```

Bonus (50 points)

A more common way to determine if a cache page is "fresh" is to use the `date` in the `Expires`, `Last-Modified`, or `Date` header in conjunction with an `If-Modified-Since` message header line in a `GET` request, which is known as a *Conditional Get*, to ask the Web server if the page has changed:

```
GET /doc HTTP/1.0
Host: www.tamu.edu
If-Modified-Since: date
```

You can use this technique even if the `Expires` header indicates a document has expired since the content may not have changed. You can also cache a document in the proxy that is missing both `Expires` and

`Last-Modifed` headers since you will be verifying with the Web server that the document is “fresh.” The *Conditional Get* technique can save message bandwidth and/or time if the page has not changed.

The logic for implementing the Bonus is as follows: If the document exists in the proxy cache but it is “stale,” i.e. the `Expires` timestamp has passed or is not present, then the proxy will issue a *Conditional GET* to the web server using an `If-Modified-Since` header in its request. The `If-Modified-Since` header will contain the `date` from the `Expires` field. If the `Expires` header is absent, then the `Last-Modified` or the `Date` timestamps will be used, in that order, to infer whether the document has expired or not.

Implementation Notes:

1. There is a short introduction to the TCP-based HTTP protocol in our textbook in section 9.1.2, pp. 708-717. In addition, the level of detail in *Headers for Dummies* is probably sufficient to understand what you need to know about HTTP headers for this assignment: https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039
2. Your proxy server needs to use a `select` loop architecture, as in assignment #2, to handle a listening socket for new clients, connected clients, and web servers. In this assignment, you can have multiple client and web server reads and writes outstanding. In assignment #2 you did not have to use the `select` function to notify you when the socket send buffer was *writable*, but you will need do that in this assignment because you are basically writing files rather than a short message (review the section on when a socket is ready for writing on p. 7 of assignment #2).
3. You proxy server will need to parse the Uniform Resource Locator (URL) to get the hostname, e.g., a URL of www.tamu.edu/index.html has a hostname of www.tamu.edu and a document name of /index.html. The proxy server will need to convert the hostname to an IPv4 address using either `getaddrinfo()` or the older, IPv4-only `gethostbyname()`. Both functions are described in Beej’s Guide.
4. HTTP/1.0 establishes a separate TCP connection for each document retrieved from the Web server or the proxy server. The Web or proxy server closes the TCP connection to signal end-of-file (Aside: Modern Web servers and clients do not work this way. We will discuss in class how modern Web servers and clients use persistent connections).
5. Make sure to check operation of the proxy server and client across a network. Use two different computers on the same network.

6. The client only needs to generate a `GET` message.
7. You will need to learn how to parse timestamps and make time comparisons in order to manage your cache properly.
8. You can access the HTTP/1.0 protocol (RFC1945) specification using the link http://tools.ietf.org/html/rfc1945
9. You can view HTTP headers for a web page in a variety of ways, e.g., download Firefox Quantum: Developer Edition: the headers can be viewed in the Network Monitor section and click on Headers. Viewing headers might be useful for picking web pages with which to test your proxy server.

Submission Guidelines:

1. My expectation is that each team member will contribute equally to the network programming assignments. Please include a statement in your README that describes the role of each team member in completing the assignment.

2. Test cases – Please develop a set of test cases to demonstrate that your HTTP proxy server works correctly. Submit a short report describing how you tested your final implementation, and a description of your test cases along with screen captures of the test cases to document correct operation. At a minimum, include the following test cases: (1) a cache hit returns the saved data to the requester; (2) a request that is not in the cache is proxied, saved in the cache, and returned to the requester; (3) a cache miss with 10 items already in the cache is proxied, saved in the LRU location in cache, and the data is returned to the requester; (4) a stale `Expires` header in the cache is accessed, the cache entry is replaced with a fresh copy, and the fresh data is delivered to the requester; (5) a stale entry in the cache without an `Expires` header is determined based on the last Web server access time and last modification time, the stale cache entry is replaced with fresh data, and the fresh data is delivered to the requester; (6) a cache entry without an `Expires` header that has been previously accessed from the Web server in the last 24 hours and was last modified more than one month ago is returned to the requester; (7) three clients can simultaneously access the proxy server and get the correct data (you will need to access large documents to show this), and (8) if you implement the Bonus feature, develop test cases to show that it works properly; you should

modify (5) – (6) above to match the operation of the Bonus.

3. Network programming assignments are due by 11:59 pm on the due date.

4. Your source code must be submitted to Turnitin.com using Canvas by 11:59 pm on the due date. Turnitin.com is plagiarism detection software that will compare your code to files on the Internet as well as your peers' code. Additional details on how to submit your code will be provided prior to the submission date.

5. All programming assignments must include the following: makefile, README, and the code.

6. The README should contain a description of your code: architecture, usage, errata, etc.

7. Make sure all binaries and object code have been cleaned from your project before you submit.

8. Your project must compile on a standard Linux development system. Your code will be graded on a Linux testbed.

9. Explanation on the submission procedure will be provided by the TA prior to the submission date.