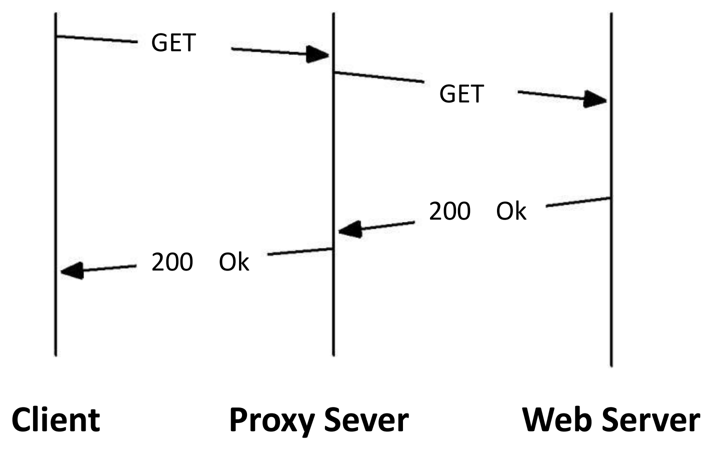
Network Programming Assignment #3

ECEN 602, Fall 2015

Due Oct. 27, 2015 NLT 5:00 pm

Simple HTTP Proxy (100 points)



In this assignment, you will implement a simple HTTP proxy server and HTTP command line client. This implementation will work for 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). Your proxy server will start on a user specified port and listen for incoming HTTP requests. When the proxy server receives a request, it will first check its cached data in an attempt to serve the request. If there are no valid cache entries, however, the request will be proxied to the intended destination, and the results will be cached 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, and 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. Another commonly used criterion for data to be considered “fresh” is that the cache has seen the document lately and it was modified a relatively long time ago (there is an optional `Last-Modified` header). For this assignment, you can assume that a document accessed in the last 24 hours is “fresh” if it was last modified one month ago or earlier. 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. It 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)

Even though a document might be expired in the cache, its content might not have been updated at the source. There are several ways for the proxy to determine if the document at the source matches the cache, which can save message bandwidth or time if the document has not changed. Implement either a conditional 'GET' or the 'HEAD' method to determine if the document is new. If it is not new, then update your cache and serve the existing data. If it is new, then provide the new document back to the client.

Notes:

1. You will need to use the same select loop architecture as in the previous project to handle concurrent client access.
2. Keep in mind HTTP/1.0 uses TCP as its transport protocol.
3. The client only needs to generate a "GET" message.
4. You will need to learn how to parse timestamps and make time comparisons in order to manage your cache properly.
5. You can access the HTTP/1.0 protocol (RFC1945) specification using the link http://tools.ietf.org/html/rfc1945

Submission Guidelines:

1. The Network Programming Assignment should be uploaded to the Google Drive folder before the beginning of class on the date that it is due. The Google drive folder (teamxx) have been shared to each team. A NPA3 folder with demo files will be added shortly.
2. When submitting, delete all the demo files and upload only these files: makefile, readme.txt, proxy.c and client.c

3. The readme.txt should contain a summary of your code: architecture, usage, and errata, etc.
4. Make sure all binaries and object code have been cleaned from your project before you submit.
5. Your project must compile on a standard linux development system. Your code will be graded on a linux testbed.
6. Make sure to check operation of the proxy server and client across a network. Use two different computers on the same network.
7. Should you have any doubts, please ask questions.
8. All submissions will go through Stanford Moss to detect plagiarism. Don’t copy code from somewhere else.