

PA3 - Caching Proxy

Start Assignment

- Due Sunday by 11:59pm
- Points 100
- Submitting a file upload
- File Types c, h, and tar
- Available Mar 11 at 12pm - Apr 7 at 11:59pm

Programming Assignment 3 - HTTP Caching Proxy

Introduction

In this assignment, you will build, in C, a proxy server that is capable of relaying HTTP requests from clients to HTTP servers.

With the HTTP client-server protocol, the client usually connects directly to an HTTP server. Oftentimes it is useful to introduce an intermediate entity called a proxy. With a proxy, the browser instead sends an HTTP request to the proxy first. The proxy then forwards this request to the intended HTTP server, and then receives the reply from the HTTP server. The proxy finally forwards this reply to the requesting HTTP client.

There are several advantages to utilizing a Proxy:

- **Performance:** If we cache a page that is frequently accessed by clients connected to the proxy, we can reduce the need to create new connections to the origin HTTP server for every request.
- **Content Filtering & Transformation:** The Proxy can inspect URLs from the HTTP client requests and decide whether to block connections to some specific URLs. Or to reform web pages (e.g. image transcoding for clients with limited processing capability).
- **Privacy & Security:** When the origin HTTP server tries to log IP address information from the HTTP client, it can only get the information of the proxy but not the actual client. Forcing all web traffic through a proxy "chokepoint" can help network administrators centralize the detection of viruses and other threats.

The Proxy Server

The proxy must take two command line arguments: a port number for the server to use and a timeout value:

```
./proxy 8888 60 # Running your proxy with a port # of 8888
```

Your proxy must be able to handle multiple requests at the same time, such that multiple clients can access different servers via your proxy simultaneously. As with PA2, you may achieve this requirement through either a forking or multi-threaded implementation.

Your proxy should run in an infinite loop once it has started. You are expected to exit out of the proxy gracefully when pressing Ctrl + C. When an HTTP request comes in, your application should determine whether this request is properly formatted with the following criteria:

- The HTTP method is valid. Our web proxy only needs to support GET.

If the above criteria are not met, a corresponding error response will be sent to the HTTP client (e.g., `400 Bad Request`). Once a valid request is received, the proxy will parse the requested URL into the following 3 parts:

- Requested host and port number (you should use the default port 80 if no port number is specified).
- Requested path (used to access resources at the HTTP server).
- Optional message body (this may not appear in every HTTP request).

If the requested host is not resolved to any IP address, then the error response `404 Not Found` must be returned to the client. You can do this right after a `gethostbyname()` call as this will return an IP address only for valid hostnames.

The request received by the proxy will be identical to that received by an HTTP server (as in our previous programming assignment). If we were to set our browser to use a proxy running on port 8000 and capture the GET request to www.yahoo.com, `(http://www.yahoo.com,)` it would look something like the following:

```
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
```

Forwarding requests to the HTTP server and relaying data

After the proxy has parsed the information from the client, it constructs a new HTTP request and sends it over a second socket connection to the indicated HTTP server. The proxy will then forward the reply from the HTTP server back to the HTTP client. So you will need to manage two concurrent socket connections - one from the client to the proxy (`socket1`) and one from the proxy to the server (`socket2`). In summary:

- Your proxy listens on the specified port number and accepts a connection (`socket1`).
- You will parse the request received from this connection (`socket1`) and check if the request is properly formatted.
- If properly formatted, your proxy will create a second socket (`socket2`) and send a request to the specified HTTP server.
- Your proxy will then relay the results from the server (`socket2`) to the client (`socket1`).

Caching


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 can create more significant savings in the case of a more distant server or a remote server that is overloaded.

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 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 considerations that will affect the caching behavior of your web proxy:

1. Timeout setting: Your proxy will expect a timeout value on the command line as a runtime option. For example, `./proxy 8888 60` will run the proxy with a timeout value 60 seconds.
2. Page cache: You should check to see if a page exists in the cache 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. You will need to create local files to store the cached pages.
3. Expiration: You will need to 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 an expired page from the cache). To keep things simple, we are not handling the `Cache-Control` header, which is sent by the server and specifies how long the client or the proxy should cache the content. Instead, we will use the timeout specified by the command line argument.

4. Dynamic Content: Many webpages have parameters (generally anything after the question mark) embedded in the URL to specify that a page be dynamically generated. You must consider these parameters when determining whether or not to serve a previously cached page. You'll most likely want to not cache dynamic pages.

Tip: when you check to see if a specific URL exists in the cache, you might want to compare hash codes instead of the entire URL. For example, suppose that you store <http://www.yahoo.com/logo.jpg>  [\(http://www.yahoo.com/logo.jpg\)](http://www.yahoo.com/logo.jpg) with the hash key 0xAC10DE97073ACD81 using a hash function (like md5sum) of the URL. When you receive the same URL from the client, you can simply calculate the hash value of the requested URL and compare it with hash keys stored in the cache.

Please keep your cached files in the directory `./cache`

Extra Credit - Link Prefetch (10 extra points)

Building on top of your proxy and caching code, the last piece of functionality desired is called link prefetching. The idea behind 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. 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 a significant amount of time, especially for a page with a lot of links. In order to fetch multiple links simultaneously in background, you have to use multi-threading or forking. One thread/process 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/process, the proxy will parse a page and extract the HTTP links, request those links from the remote server, and add them to the cache.

Extra Credit - Regex Blocklist (10 extra points)

Create a file called `./blocklist` which has a list of websites and/or IP address regex's that must be blocked. Thus, for every incoming request, the proxy must compare the requested host/IP against the regex's in `./blocklist` and deny requests for these websites or IPs. When a client requests one of them, your proxy should return `403 Forbidden`. This file should be formatted with one hostname or IP address glob-style regex per line (ie. each line terminated with a newline):

```
*.google.com
mail.yahoo.com
157.240.28.[0-2][0-9][0-9]
```

Submission Requirements

1. Please submit your code as either individual `.c` and `.h` files or as a plain `.tar` file (not `.tgz` or `.zip`)
2. Support for simultaneous requests should include two or more clients accessing the proxy at the same time
3. The usage of any libraries or commands that implement an HTTP server or client is explicitly not allowed

References

<https://www.wikihow.com/Enter-Proxy-Settings-in-Firefox>  [\(https://www.digitalcitizen.life/how-set-proxy-server-all-major-internet-browsers-windows/\)](https://www.digitalcitizen.life/how-set-proxy-server-all-major-internet-browsers-windows/)

<https://www.digitalcitizen.life/how-set-proxy-server-all-major-internet-browsers-windows/>  [\(https://www.digitalcitizen.life/how-set-proxy-server-all-major-internet-browsers-windows/\)](https://www.digitalcitizen.life/how-set-proxy-server-all-major-internet-browsers-windows/)