Programming Assignment 3: HTTP Proxy with Caching

Due: December 5th, 2017 23:59:59pm

This assignment is worth 12% of your total score in this course. In this assignment, you will write an (non-standard-compliant) HTTP proxy that accepts HTTP GET requests from clients, fetches the desired content, and returns this content to the client. Your HTTP proxy must also cache data locally in order to reduce the response time.

You must **work by yourself** on this assignment and use **C or C++**.

1 Background

HTTP GET requests are used to request data from the specified source. For example, if we want to access a webpage at `http://www.foo.com/bar.html`, our browser will send something similar to the following HTTP GET request.

```
GET /bar.html HTTP/1.1
Host: www.foo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

In the above example, our browser is the client, and `Host: www.foo.com` is the server host name. The client is requesting server resource located at `/bar.html`. Usually, HTTP servers run on port 80. So, by default, the server port number is 80. If, for example, the HTTP server runs on port 8080, then the URL would be `http://www.foo.com:8080/bar.html`, and the `Host` field in the HTTP request will also contain the port number: `www.foo.com:8080`.

The HTTP server responds with HTTP response. A valid HTTP response includes: (1) the status line, (2) the response header, and (3) the entity body. For example, the text below shows an example HTTP response. Notice that this response header includes a field: `Content-Length`, which tells the client how much data to expect.

```
HTTP/1.1 200 OK
Date: Thu, 02 Apr 2015 01:51:49 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Tue, 10 Feb 2015 17:56:15 GMT
ETag: "56638402b-a85-50ebf9a5ecdc0"
Accept-Ranges: bytes
Content-Length: 2693
Content-Type: text/html; charset=ISO-8859-1

... ... <content of the bar.html>
```

An **HTTP proxy** is an application that resides between the HTTP client and server. The HTTP proxy relays the client's HTTP requests to the HTTP server, and forwards the server's response back to the client. Proxies are often

used when the client cannot directly connect with the HTTP server. If a client is configured to use an HTTP proxy, the HTTP GET request it sends out will not only include path to the resource at the server, but also the server host name. For example, the request would look like:

```
GET http://www.foo.com/bar.html HTTP/1.1
Host: www.foo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

To request the resource for the client, the HTTP proxy starts a new TCP connection with the server host `www.foo.com` at port number `80` and ask for file `bar.html`. When the HTTP proxy relays this request from the client, it will have to remove the host name, `http://www.foo.com` from the request line of the GET request, so that the request line specifies only the local resource path at the destination server. When the HTTP response arrives at the proxy, the proxy will forward it to the client. If the proxy is correctly implemented, a client web-browser should be able to display the requested resource.

2 Task I: Multi-threaded HTTP Proxy

Your HTTP proxy should support a subset of the HTTP standard. It only needs to forward HTTP GET requests and should only support basic portions of the standard. That is, you do not need to support persistent connections, request pipelining or other, advanced portions of the standard. Your proxy must only correctly handle HTTP requests where a single request/response pair is sent over a single TCP connection. Specifically, your proxy should operate as follows:

- It should create a TCP server socket to listen for incoming TCP connections on an unused port, and output the host name and port number the proxy is running on.
- When a new connection request comes in, the proxy accepts the connection, establising a TCP connection with the client.
- The proxy reads HTTP request sent from the client and prepares an HTTP request to send to the HTTP server.
- The proxy starts a new connection with the HTTP server, and sends its prepared HTTP request to the server.
- The proxy reads the HTTP response and forwards the HTTP response (the status line, the response header, and the entity body).
- The proxy closes the TCP connection with the server.
- The proxy sends the server's response back to the client via its TCP connection with the client. This TCP connection with the client should have remained open during its communication with the server. Also note that the proxy should use `Content-Length` to determine how much response content to read from the server and forward to the client.
- The proxy closes the connection socket to the client.

To maintain a high level of throughput even under multiple simultaneous client requests, different requests should be handled in different threads, with a new thread created for each client request. In this assignment, you are required to use the **pthreads** (POSIX threads) library for the multi-threaded proxy implementation.

2.1 How to test your implementation

You first download a resource located at a URL using the `wget` command without the proxy. This is the correct resource. Then you download the same resource at the same URL with your proxy. Use the `diff` command to check if the two downloaded resources matches.

Suppose your HTTP proxy is started on `remote02.cs.binghamton.edu` port `47590`. You can run the following Bash shell command in an **EMPTY** directory on a **remote.cs computer** to download web resource via your proxy.

```
$> bash
$> export http_proxy=http://remote02.cs.binghamton.edu:47590 && wget
http://www.foo.com/bar.html
```

To download without the HTTP proxy, use the following command:

```
$> export http_proxy="" && wget http://www.foo.com/bar.html
```

Note:

- You MUST replace `http://www.foo.com/bar.html` with a valid URL.
- Since this is only an HTTP proxy, not an HTTPS proxy, your URL MUST use HTTP, not HTTPS.
- You may also run your HTTP proxy on your own machine and use Wireshark for debugging.
- Your web browser also has a cache. A second request for the same URL may be directly served by your browser cache without going through the proxy. Therefore, you are recommended to use `wget` for debugging and testing.

3 Task II: Caching

Your HTTP proxy must also implement caching. With caching, the proxy stores the responses of past requests in its local storage. If an incoming request matches an entry in the cache, the proxy returns its cached data in its local storage to the client directly. This can greatly reduce the response time and reduce the network bandwidth.

The cache size is set as a command line input to the proxy executable. For example, if the maximum cache size is 320,000 bytes, then your proxy should be started as:

```
./proxy 320000
```

You can assume that the cache size is set to a value bigger than the size of any single object that will be requested.

If the cache becomes full, Least Recently Used (LRU) replacement policy[1] will be used for selecting an entry in the cache to evict.

Upon successfully serving a request, your proxy must write to standard output (stdout) the following items:

client IP client's IP address in dotted decimal representation

requested URL

cache status can only be one of the two values: **CACHE_HIT** or **CACHE_MISS**

content length is the size of the entity body in bytes

[1] `https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_Recently_Used_.28LRU.29`

request time is request processing time in milliseconds resolution from when the connection from the client is accepted to when the connection to the client is closed (i.e., the last bytes were sent to the client).

These five items of a same request should be printed to a same line and seperated by the "|" character. Below is an example:

```
128.226.118.20|http://www.foo.com/bar.html|CACHE_MISS|2693|300
128.226.118.26|http://www.foo.com/bar.html|CACHE_HIT|2693|2
```

The first request for `http://www.foo.com/bar.html` was not found in the cache. So the resource was fetched from the remote server. Total request time was 300 ms. When a different client with a different IP address requested the same URL, it was a cache hit, and the request time was much smaller, only 2 ms.

4 How to submit

To submit the project, you should first create a directory whose name is identical to your BU email ID. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-p3`.

You should put the following files into this directory:

1. All of your source code that implements the HTTP proxy. Note that your proxy executable should take exactly one command line input – cache size in bytes – and must output the host name and port number it is running on.
2. A `Makefile` to compile your source code into one executable, which should be named `proxy`.
3. (Optional) A `Readme` file if there is anything you want the TA to be aware of when grading.
4. A `STATEMENT` file, containing the following statement followed by the student's full name:

   "I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment and my grade will be reduced by one level (e.g., from A to A- or from B+ to B) for my first offense, and that I will receive a grade of **"F" for the course** for any additional offense of any kind."

Compress the directory (e.g., `tar czvf jdoe-p3.tgz jdoe-p3`) and submit the tarball (in `tar.gz` or `tgz` format) to myCourses. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-p3.tar.gz` or `jdoe-p3.tgz`. Failure to use the right file name will result in a 5% point deduction.

Your project will be graded on the CS Department computers `remote.cs.binghamton.edu`. It is your responsibility to make sure that your code compiles and runs correctly on these remoteXX computers.

Your project must be your original work. We will use MOSS[2] to detect plagiarism in the projects.

Appendix

Below is a list of URLs you can use for testing your proxy and caching implementation.

```
http://www.cs.binghamton.edu/~yaoliu/courses/cs528/cs528.html
http://www.cs.binghamton.edu/~yaoliu/courses/cs528/course.css
http://www.cs.binghamton.edu/~yaoliu/courses/cs528/syllabus.pdf
http://www.cs.binghamton.edu/~yaoliu/courses/cs528/lena_std.tif
http://www.cs.binghamton.edu/~yaoliu/courses/cs528/skype-ubuntu-precise_4.3.0.37-1_i386.deb
```

[2] `https://theory.stanford.edu/~aiken/moss/`

The list below includes URLs from servers located outside the computer science department. **Please do not test your proxy on these URLs unless you have successfully tested on the URLs above.** You should also limit the number of requests you send to these websites. If the volume is too high, the external server may consider you are performing a denial-of-service (DoS) attack and may block your IP.

```
http://www.binghamton.edu:8080/asf-logo.png
http://www.binghamton.edu:8080/docs/setup.html
http://portquiz.net:10000/portquizm.png
http://httpbin.org/html
http://httpbin.org/image/png
http://httpbin.org/image/jpeg
```