CSE3027: Computer Networks

Project 2:
Web Proxy
Spring 2018
Instructor: Prof. Suk-Bok Lee

Due: 11:59 PM, Friday, June 8, 2018

1 Introduction

A Web proxy is a program that acts as a middleman between a Web browser and an end server. Instead of contacting the end server directly to get a Web page, the browser contacts the proxy, which forwards the request on to the end server. When the end server replies to the proxy, the proxy sends the reply on to the browser. The proxy can also be used to cache Web objects, by storing a copy of, for instance, an image when a request for it is first made, and then serving that image in response to future requests rather than going to the end server. In this project, you will write a Web proxy that logs requests. In the first part of the project, you will write a simple sequential proxy that repeatedly waits for a request, forwards the request to the end server, and returns the result back to the browser, keeping a log of such requests in a disk file. This part will help you understand basics about network programming and the HTTP protocol. In the second part of the project, you will turn your proxy into a proxy cache by adding a simple main memory cache of recently accessed web pages. Finally, you will upgrade your proxy so that it uses threads to deal with multiple clients concurrently. This part will give you some experience with concurrency and synchronization, which are crucial computer systems concepts. Your proxy should spawn a separate thread to deal with each request.

2 Getting Started

Reading Chapter 2 of the textbook will help you to understand how HTTP works, and this project requires:

- You must program in C (not Java).
- Your program should run on a linux machine. Your submitted code will be tested and evaluated on linux machines.

3 Part I: Implementing a Sequential Web Proxy

In this part you will implement a sequential logging proxy. Your proxy should open a socket and listen for a connection request. When it receives a connection request, it should accept the connection, read the HTTP request, and parse it to determine the name of the end server. It should then open a connection to the end server, send it the request, receive the reply, and forward the reply to the browser if the request is not blocked. Since your proxy is a middleman between client and end server, it will have elements of both. It will act as a server to the web browser, and as a client to the end server. Thus you will get experience with both client and server programming.

Processing HTTP Requests

When an end user enters a URL such as

```
http://www.example.com/index.html
```

into the address bar of the browser, the browser sends an HTTP request to the proxy that begins with a line looking something like this:

```
GET http://www.example.com/index.html HTTP/1.0
```

In this case the proxy will parse the request, open a connection to www.example.com, and then send an HTTP request starting with a line of the form: GET /index.html HTTP/1.0 to the server www.example.com. Please note that all lines end with a carriage return ‘\r’ followed by a line feed ‘\n’, and that HTTP request headers are terminated with an empty line. Since a port number was not specified in the browsers request, in this example the proxy connects to the default HTTP port (port 80) on the server. The web browser may specify a port that the web server is listening on, if it is different from the default of 80. This is encoded in a URL as follows: http://www.example.com:8080/index.html. The proxy, on seeing this URL in a request, should connect to the server www.example.com on port 8080. The proxy then simply forwards the response from the server on to the browser. Please read some articles about HTTP requests to better understand the format of the HTTP requests your proxy should send to a server.

Logging

Your proxy should keep track of all requests in a log file named proxy.log. Each log file entry should be a file of the form:

```
Date:  browserIP URL size
```

where browserIP is the IP address of the browser, URL is the URL asked for, size is the size in bytes of the object that was returned. For instance:

```
Sun 20 May 2018 02:51:02 EST: 166.104.231.100 http://cnlab.hanyang.ac.kr/ 34314
```

Note that size is essentially the number of bytes received from the end server, from the time the connection is opened to the time it is closed. Only requests that are met by a response from an end server should be logged.

Port Numbers

Your proxy should listen for its connection requests on the port number passed in on the command line:

```
./proxy 15213
```

You may use any port number p, where $1024 \leq p \leq 65536$, and where p is not currently being used by any other system or user services. See /etc/services for a list of the port numbers reserved by other system services.

4 Part II: Caching Web Objects

In this part you will add a cache to your proxy that will cache recently accessed content in main memory. HTTP actually defines a fairly complex caching model where web servers can give instructions as to how the objects they serve should be cached and clients can specify how caches are used on their behalf. In this project, however, we will adapt a somewhat simplified approach. When your proxy queries a web server on behalf of one of your clients, you should save the object in memory as you transmit it back to the client. This way if another client requests the same object at some later time, your proxy does not need to connect to the server again. It can simply resend the cached object. Obviously, if your proxy stored every object that was ever requested, it would require an unlimited amount of memory. To avoid this (and to simplify our testing) we will establish a maximum cache size of

```
MAX CACHE SIZE = 5MB
```

and evict objects from the cache when the size exceeds this maximum. We will require a simple least recently-used (LRU) cache replacement policy when deciding which objects to evict. One way to achieve this is to mark each cached object with a time-stamp every time it is used. When you need to evict one, choose the one with the oldest timestamp. Note that reads and writes of a cached object both count as using it. Another problem is that some web objects are much larger than others. It is probably not a good idea to delete all the objects in your cache in order to store one giant one, therefore we will establish a maximum object size of

```
MAX OBJECT SIZE = 512KB
```

You should stop trying to cache an object once its size grows above this maximum. The easiest way to implement a correct cache is to allocate a buffer for each active connection and accumulate data as you receive it from the server. If your buffer ever exceeds MAX OBJECT SIZE, then you can delete the buffer and just finish sending the object to the client. After you receive all the data you can then put the object into the cache.

5 Part III (OPTIONAL): Concurrent Web Proxy

Real proxies do not process requests sequentially. They deal with multiple requests concurrently. Once you have a working sequential logging proxy, you should alter it to handle multiple requests concurrently. The simplest approach is to create a new thread to deal with each new connection request that arrives. With this approach, it is possible for multiple peer threads to access the log file concurrently. Thus, you will need to use a semaphore to synchronize access to the file such that only one peer thread can modify it at a time. If you do not synchronize the threads, the log file might be corrupted. For instance, one line in the file might begin in the middle of another.

6 Hints

- Test your proxy with a real browser! It is very exciting to see your code serving content from a real server to a real browser such as Mozilla Firefox; if you can test with other browsers, you are encouraged to do so. To setup Firefox to use a proxy, open the Settings window. In the Advanced pane, there is an option to Configure how Firefox connects to the Internet. Click the Settings button and set only your HTTP proxy (using manual configuration). The server will be whatever shark machine your proxy is running on, and the port is the same as the one you passed to the proxy when you ran it.

- Here is how you should forward browser requests to servers so as to achieve the simplest and most predictable behavior from the servers:
  - Forward all requests to servers as version HTTP/1.0, even if the original request was HTTP/1.1. Since HTTP/1.1 supports persistent connections by default, the server won't close the connection after it responds to an HTTP/1.1 request. If you forward the request as HTTP/1.0, you are asking the server to close the connection after it sends the response. Thus your proxy can reliably use EOF on the server connection to determine the end of the response.
  - Replace any Connection/Proxy-Connection: [connection-token] request headers with Connection/Proxy-Connection: close. Also remove any Keep-Alive: [timeout-interval] request headers. The reason for this is that some misbehaving servers will sometimes use persistent connections even for HTTP/1.0 requests. You can force the server to close the connection after it has sent the response by sending the Connection: close header.

7 Project Submission

Note that the programming language that you use should be C and your program should run on a linux machine.

1. Put all your files into a directory, named "project2_UID_YourName" where UID is your student ID and YourName is your name (e.g., Hong Kildong → project2_xxx_Hong_Kildong).
2. In the directory that contains "project2_UID_YourName", type the following command in UNIX shell

   ```
   tar cvf project2_UID_YourName.tar project2_UID_YourName
   ```

   For example, if you have a "cse3027" directory in your home directory, and "project2_UID_YourName" directory within "cse3027". Once you login from ssh, you need to

   ```
   cd cse3027
   tar cvf project2_UID_YourName.tar project2_UID_YourName
   ```

3. Submit the file "project2_UID_YourName.tar" via online submission in course webpage.
4. The directory "project2_UID_YourName" should include the following files:
   - All **commented** source files.
   - A Makefile.
     - Graders will only type "make" to compile your code; make sure your Makefile works!
   - Short report (report.doc, report.pdf). 1-3 pages.
     - You need to include the following items in your report:
       - Give a high-level description of your proxy's design
       - What difficulties did you face and how did you solve them?

The project due date is June 8, 2018.