System Programming - Proxy Lab Report

2019 Spring, 2017-18570 Sungchan Yi

1. Proxy Lab

In this assignment, we write a simple HTTP proxy that caches web objects. Our objective in this assignment is to set up the proxy to accept incoming connections, read/parse requests, forward requests to web servers, read responses, forward responses to the corresponding clients. We learn about basic HTTP operation and how to use sockets to write programs communicating over the network. Also, upgrade the proxy to handle multiple concurrent connections, and learn about concurrency. Finally, add cache to the proxy to speed up the request handling.

2. Implementation

The implementation details are in the comments.

Part 1: Sequential Web Proxy

For this part, implement a sequential proxy that handles GET requests. The proxy listens for connection on a port number given in `argv[1]`. After accepting the connection, read the request from the client, parse the request. Then determine if the client sent a valid request. Then, the proxy will establish its own connection to the appropriate web server and request the object that the client specified, which will be forwarded to the client.

Part 2: Concurrent Proxy

After part 1, we upgrade the program and implement concurrent request handling for the proxy. To implement this, we simply create a new thread each time a connection is established. To avoid memory leaks, we detach the thread and make it run. This part could be done almost immediately after part 1; there's nothing much to modify.

Part 3: Caching Web Objects

A proxy would be inefficient if it had to make requests to the actual server every time a client asked for it. Instead, the proxy will cache web content inside its own cache, and speed up the forwarding process.
For this lab, we cache web contents in memory and transmit it to the client. Moreover, huge amount of data will not be cached (Specifically, contents of size larger than the maximum cache size will not be cached) and we adopt a LRU replacement policy to control the cache.
Lastly, we have to consider synchronization. Since our proxy is concurrent we should consider the fact that read/writes to the cache should not be interrupted by other threads. So we use

semaphores (mutex) to control this and prevent the proxy from sending or writing incorrect information. Also, the assignment requires us to prefer the readers. Only one thread should be allowed to write to the cache, but readers can read from the cache anytime if no writer is writing to it.

Testing

For testing the proxy, run the proxy by

```
$ ./proxy 15213
```

and test the output for some url through

```
$ curl --proxy localhost:15213 http://www.snu.ac.kr/index.html
```

The result will be compared to

```
$ curl http://www.snu.ac.kr/index.html
```

3. Conclusion

Through this lab, we wrote a proxy that can handle concurrent requests and cache requests for future uses. This was personally the most interesting assignment. Although considering the concurrency for the reader favoring cache was quite difficult, it seems ok. I think I might want to work more on this if I have the time. (implement POST etc.)