SS09, 2010

Lab Assignment 8: Logging Web Proxy

Assigned: Dec. 15

Due: Dec. 31, 19:59:59 PM

# Introduction

A web proxy is a program that acts as a middleman between a web server and browser. Instead of contacting the server directly to get a web page, the browser contacts the proxy, which forwards the request on to the server. When the server replies to the proxy, the proxy sends the reply on to the browser.

Proxies are used for many purposes. Sometimes proxies are used in firewalls, such that the proxy is the only way for a browser inside the firewall to contact a server outside. The proxy may do translation on the page, for instance, to make it viewable on a web-enabled cell phone. Proxies are used as anonymizers by stripping a request of all identifying information, a proxy can make the browser anonymous to the server. Proxies can even be used to cache web objects, by storing a copy of, say, 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 server. Squid, for example, (http://squid.nlanr.net) is a popular free proxy cache.

In this lab, you will write a simple web proxy that logs requests. In the first part of the lab, you will set up the proxy to accept a request (just an HTTP request), forward the request to the server, and return the result back to the browser, keeping a log of such requests in a disk file. In this part, you will learn how to write programs that interact with each other over a network (socket programming), as well as some basic HTTP.

In the second part of the lab, you will upgrade your proxy to deal with multiple open connections at once. You may implement this in two ways:

Your proxy can spawn a separate thread to deal with each request, or it may multiplex between the requests using the select(2) Unix system call. Either option will give you an introduction to dealing with concurrency, a crucial systems concept. Using threads for this lab is recommended.

# Requirements

You should finish this lab by yourself.

Please use SVN to get the proxy files from the ICS server 10.132.143.1

The files you will be modifying and turning in are proxy.c, csapp.c, and csapp.h. The proxy.c file contains the bulk of the logic for your proxy. The csapp.c and csapp.h files are described in your textbook. The csapp.c file contains error handling wrappers and helper functions such as the RIO functions (Section 11.4), the open clientfd function (Section 12.4.4), and the open listenfd function (Section 12.4.7). **Note: we have added several simple RIO functions that are not described in the textbook.** The csapp.h file contains the prototypes for the functions in csapp.c.

Part I: Implementing a web proxy

The first step is implementing a basic logging proxy. When started, your proxy should open a socket and listen for connections. When it gets a connection (from a browser), it should accept the connection, read the request, determine if it is a request for an HTTP connection, and parse it to determine the server that the request was meant for. It should then process the open a socket connection to that server, send it the request, receive the reply, and forward the reply to the browser if the request is not blocked.

Notice that, since your proxy is a middleman between client and server, it will have elements of both. It will act as a server to the web browser and as a client to the web server. Thus you will get experience with both client and server programming.

Processing HTTP GET requests (15 points)

When end user enters a URL such as http://www.yahoo.com/news.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.yahoo.com/news.html HTTP/1.0

In this case the proxy will parse the request, open a connection to www.yahoo.com, and then send a request of the form

GET /news.html HTTP/1.0

to the server www.yahoo.com. Since a port number was not specified in the browser's request, in this example the proxy connects to the default HTTP port (port 80) on the server. The proxy then simply forwards the response from the server on to the browser.

Logging (5 points)

Your proxy should also keep track of all requests in a log file named proxy.log. Each line should be 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:

Wed. 27 May 2008 02:51:02: 10.132.141.38 http://ics.fudan.edu.cn/ 34314

Note that size is essentially the number of bytes received from the server, from the time the connection is opened to the time it is closed. Only requests that are met by a response from a server (or cached response) should be logged. We have provided the function **format_log_entry** to create a log entry in the required format.

Web Page Cache(5 points)

Create a web page cache for buffering already loaded web pages. If request to the same web page is sent to proxy again, the cached web page should be returned to accelerate response time.

Port Numbers Configurable (5 points)

Since there will be many processes working on the same machine, all of them cannot use the same port to run. You should allow user select any non-privileged port for your proxy, as long as it is not taken by other system processes. Selecting a port in the upper thousands is suggested (i.e., 3070 or 8104).

Part II: Dealing with multiple requests (20 points)

Real proxies do not process requests sequentially. They deal with multiple requests in parallel. This is particularly important when handling a request can involve a lot of waiting (as it can when you are, for instance, contacting a remote web server). While your proxy is waiting for a remote server to respond to a request so that it can serve one browser, it could be working on a pending request from another browser. Thus, once you have a logging proxy, you should alter it to handle multiple requests simultaneously. There are two basic approaches to doing this:

Threads

A common way of dealing with concurrent requests is for a server to spawn a thread to deal with each request that comes in. In this architecture, the main server thread simply accepts connections and spawns off worker threads that actually deal with the requests (and terminate when they are done).

If you choose this method, however, you will have the problem that multiple peer threads will be trying to access the log file at once. If they do not somehow synchronize with each other, the log file will be corrupted (for instance, one line in the file might begin in the middle of another). You will need to use a semaphore to control access to the file, so that only one peer thread can modify it at a time.

Select

Another way to deal with concurrent requests is to multiplex between connections by hand using the select system call. As described in your text, the select function takes a set of file descriptors and waits until one of them is ready for reading or writing. You can use this call to simultaneously wait on all open socket connections, and process whichever one is ready first. For instance, your proxy might be waiting for a request to come from a client on one socket, for a response to come from a server on another socket, and at the same time listening for new connection requests on its listening socket. Your code would use select to wait for all of these things at once, handling whichever happened first.

With this architecture, you will not need to deal with synchronization among concurrent processes, since there is only one process running, but you will need to correctly multiplex on several connections, without blocking on any of them.

# Evaluation

Finish Part I (30 points).
List in the SUMMARY doc the web-site URLs you have tested that your proxy can successfully handle. This could help TA judge whether you have done your work correctly.

Finish Part II (20 points).

Style and Doc (10 points).
Make the code more understandable (5 points).
The SUMMARY doc (5 points, see the HANDIN part).

# Resources and Hints

1) Read Chapter 11 - 13 in your textbook. They contain useful information on system-level I/O, network programming, HTTP protocols, and concurrent programming.

2) Use the RIO (Robust I/O) package described in your textbook for all I/O on sockets. Do not use standard I/O on sockets. You will quickly run into problems if you do. You can start with the Tiny Web server described in your textbook. This is a fully functional Web server. You may copy any of the Tiny code and use it in your proxy.

3) Test your proxy with a real browser! Explore the browser settings until you find “proxies”, then enter the host and port where you're running yours. In Internet Explorer, choose Tools, then Options, then Connections, then LAN Settings. Check 'Use proxy server,' and click Advanced. Just set your HTTP proxy, because that's all your code is going to be able to handle.

4) In certain cases, a client closing a connection prematurely results in the kernel delivering a SIGPIPE signal to the proxy. This is particularly the case with Internet Explorer. To prevent your proxy from crashing, you should ignore this signal by adding the following line early in your code:
signal(SIGPIPE, SIG_IGN);
Also, in certain cases, writing to a socket whose connection has been closed prematurely results in the write system call returning a -1 and setting **errno** to **EPIPE**. Your proxy should not terminate when a write elicits an EPIPE error. Simply close the socket, update the display to reflect one less connection, and continue.
The easiest way to handle EPIPE errors is to modify the Rio_writen wrapper in csapp.c to test specifically for EPIPE errors and return some special value (say 0) when it

5) To test your proxy, you may use some simple website from the very beginning and move to some complicated website with different resources step by step. Some simple websites: www.ycul.org and www.google.cn, and some complicated websites may be www.sina.com.cn and so on.

6) **IMPORTANT**: If you use threads to handle connection requests, you must run them as detached, not joinable, to avoid memory leaks that will likely crash the machines. To run a thread detached, add the line pthread_detach(thread_id) in the parent after calling pthread_create().

# Handin

Check in your proxy files (**proxy.c**, **csapp.c**, and **csapp.h**) into the SVN server before the deadline together with a **SUMMARY doc** (summary.txt) including:
1. The parts of the requirement you have finished. If you proxy has some features beyond our expectation, you should tell us and you will get better grade.

2. Give a web-site URL list that can prove your proxy can work well.

3. Tell us how to change the port number your proxy is listening to.
4. Tell us how the web-page cache is organized.
5. Tell us how you handle multiple requests. (Using thread? Select?)
6. Other things that can help us understand your proxy better.