ECE 14:332:456, NETWORK CENTRIC PROGRAMMING
Proxy Lab: Writing a Caching Web Proxy

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.

Proxies are useful for many purposes. Sometimes proxies are used in firewalls, so that browsers behind a firewall can only contact a server beyond the firewall via the proxy. Proxies can also act as anonymizers: by stripping requests of all identifying information, a proxy can make the browser anonymous to Web servers. Proxies can even be used to cache web objects by storing local copies of objects from servers then responding to future requests by reading them out of its cache rather than by communicating again with remote servers.

In this lab, you will write a simple HTTP proxy that caches web objects. For the first part of the lab, you will set up the proxy to accept incoming connections, read and parse requests, forward requests to web servers, read the servers' responses, and forward those responses to the corresponding clients. This first part will involve learning about basic HTTP operation and how to use sockets to write programs that communicate over network connections. In the second part, you will upgrade your proxy to deal with multiple concurrent connections. This will introduce you to dealing with concurrency, a crucial systems concept. In the third and last part, you will add caching to your proxy using a simple main memory cache of recently accessed web content.

2 Logistics

This is an individual project.

3 Handout instructions

Start by downloading proxy-handout.tar from Sakai to a directory in which you plan to do your work. The tar file is a file archive that can be extracted on UNIX machines with the command:

tar xvf proxy-handout.tar

This will cause a number of files to be unpacked in the directory. The three files you will be modifying and turning in are proxy.c, csapp.c, and csapp.h. You may add any files you wish to this directory as you will be submitting the entire directory (Make sure you update the Makefile as you add any new files). The proxy.c file should eventually contain the bulk of the logic for your proxy. The csapp.c file contains error handling wrappers and helper functions such as the RIO functions, the open clientfd function, and the open listenfd function. It is not mandatory to use the helper functions in csapp.c. You can write yours if you feel so. Include in your report if you make any changes in the csapp.h and/or csapp.c file.

4 Implementation

The basic sequential web proxy will handle requests one at a time. When started, your proxy should open a socket and listen for connection requests on the port number that is passed in on the command line. (See the section Port Numbers below.)

When the proxy receives a connection request from a client (typically a web browser), the proxy should accept the connection, read the request, verify that it is a valid HTTP request, and parse it to determine the server that the request was meant for. It should then open a connection to that server, send it the request, receive the reply, and forward the reply to the browser.

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.

4.1 Processing HTTP Requests

When an 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 an HTTP request starting with a line of the form:

GET /news.html HTTP/1.0

to the server www.yahoo.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.comon port 8080. The proxy then simply forwards the response from the server on to the browser.

NOTE: Be sure to parse out the port number from the URL. We will be testing this. If the port is not explicitly stated, use the default port of port 80.

4.2 Port numbers

Every server on the Internet waits for client connections on a well-known port. The exact port number varies from Internet service to service. The clients of your proxy (your browser for example), will need to be told not just the hostname of the machine running the proxy, but also the port number on which it is listening for connections.

Your proxy should accept a command-line argument that gives the port number on which it should listen for connection requests. For example, the following command runs a proxy listening on port 15213:

```
unix> ./proxy 15213
```

You will need to specify a port each time you wish to test the code youve written.

4.3 Logging

You will need to log each request to a log file. Your proxy should keep track of all requests in a log file named proxy.log. Each log file entry should be in 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 27 Oct 2002 02:51:02 EST: 128.2.111.38 http://www.yahoo.com/news.html 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. We have provided the function format_log_entry in proxy.c to create a log entry in the required format.

5 Testing your Proxy

For this assignment, there is no script to check your implementation. You will have to come up with your own tests to help you debug your code and decide when you have a correct implementation. Below is way by which you can debug your proxy, to help you get started. Be sure to exercise all code paths and test a representative set of inputs, including base cases, typical cases, and edge cases.

- You can test the working by any internet browser. You will need to configure the HTTP proxy settings for that browser. For example in Mozilla, you can set the appropriate proxy setting at $Options->Advanced->Network->Connection->Configure$ how Firefox connects to the Internet. Click the Settings button and set only your HTTP proxy (using manual configuration). The server will be whatever machine your proxy is running on, and the port is the same as the one you passed to the proxy when you ran it.
- Use print statements to trace the flow of information between the client, proxy and the server

6 Tips

We strongly recommend that you tackle this proxy problem in the following steps. This should help you break the problem down into more manageable pieces that you can test individually. Working versions of these will also count for partial credit if the complete proxy is not finished.

1. Write a proxy server that accepts a connection from a web browser and simply prints the entire request message from the web browser on the screen.
2. Extend the server from (1) to extract the browser IP and URL information from the request message. Open a log file and write this information into a file, instead of printing the message on the screen. Use the provided function to format the output. Set size to 0, since it is unknown at this point.
3. Write a separate client that takes as input a URL and prints out the web server response for that URL as well as the size of the response in bytes.
4. Combine the server and client into a single program. Instead of having the client print the output on the screen, feed it back to the web browser.

7 Submission and Demo Instructions

To receive full credit, your solution needs to be well-commented (in your own words, of course). At the top of your source code file, please include a comment of about 2 paragraphs explaining how your program solves this assignment. At the start of every function in your program explain what this function does in 1-5 sentences, depending on the complexity of the function. Include a short comment every 3-5 program lines explaining what the next sequence of code will accomplish. Add additional comments for non-obvious lines of code or parameters to explain how this works. Note that deductions for completely uncommented code can be substantial.

You will need to submit your complete project directory on Sakai.

- Create a gzipped tar ball that includes all your source code.
- The filename for the tarball must be **username-proxy.tar.gz** where username is your login name on the EE lab computer system
- All your files in the tarball **must** be in a directory, named **username-proxy**

The submission will be due on the date mentioned on Sakai. **After the submission date, you will have to give a demo, of the exact implementation that you submitted on Sakai, at a time that we will announce later on Sakai.**