CptS/EE 455
Fall 2009
Project 4
Assigned: Wed Nov 11, 2009
Due: 11:59:59PM Fri Dec 4, 2009

Summary

In this project you will create an HTTP proxy server. A proxy server plays the "server" role with respect to a web browser and the "client" role with the respect to a web server, passing along requests from the client to the server. Proxy servers in common use typically add value by acquiring a set of pages in a *cache* which they can serve to their clients faster than by going to the real server (textbook section 2.2.5). Other uses include filtering out advertising, animations, images, etc. from web pages. Your proxy server will simply pass every request along to the appropriate server and pass every response back to the client; but it will also log all the HTTP headers that it sees in both requests and responses, making this part of the HTTP protocol more visible to you.

The purposes of this project: to become familiar with basic programming techniques and patterns used in a multi-process or multi-threaded web server; to further practice skills in creating TCP-based clients and servers; to develop your skills at creating programs that simultaneously play the "client" and "server" role.

Additional skills: you will also gain experience using string processing functions to parse and alter HTTP headers.

Logistics

You may write this program in C, C++, Python or Java. You must not use language-supplied or 3rd party libraries but must write all of the necessary string and protocol processing code yourself. If you want to use a different language, please ask me first.

What to turn in:

- Your code, your Makefile (if needed), the output from the test cases, and your COMMENTS and README files are be placed in proxy.tar or proxy.zip and submitted use on the online submission page. The README file should contain complete instructions for creating your server from source and for running it. You will need to include instructions for configuring a browser to use your server. In the COMMENTS file you will report on some experiments that you run using your proxy server. Details will be provided later.

During the week following the due date (finals week) you will each need to schedule an individual appointment with the TA to demonstrate your proxy server and to describe its operation.

Specifications

You are to create an HTTP proxy server that

1. accepts connections from web clients (i.e. Firefox or Internet Explorer) on a port of your choosing.
2. Upon accepting a connection creates a separate process (or thread) to handle communication for that request using the forking TCP Server pattern discussed in class.
3. Cleans up after any completed child processes using the wait3() system call. See the man page. You will want to use the WNOHANG option to clean up completed children without blocking in the case that none are complete.

The separate process for a particular request

1. reads an HTTP request from the socket connected to the client and returned by accept()
2. creates a new socket connected to the server specified in the client's request
3. passes a modified version of the client's request to the server
4. reads the server's response and passes a modified version of it to the client.

While doing steps 1 through 4, the process handling a request will print the request and response messages, along with all HTTP headers found in both the request and the response (before any alterations performed by your proxy server), on the standard output of the proxy server.

By a week before the project is due I will give you several URLs that you should visit with a browser connecting through your proxy server. I will ask you to explain in your COMMENTS file the meaning of each of the headers that your proxy server prints while connecting to these sites.

Header modifications

The specification calls for the proxy server to modify the request and response headers. The purpose for doing the modification is to simplify your implementation's task at finding boundaries in the data streams. As we have talked about, one of the problems facing a protocol designer or implementor is deciding how to delimit the boundaries between various parts of a message. In HTTP this delimiting is done in several ways. By altering the headers we can make most of these ways not happen.

We will alter the HTTP headers so that in almost all the cases that we care about, the boundaries will be marked by an "end-of-stream" condition. In order to accomplish this you need to make the following modifications to both the request and response headers:

1. delete any `Connection: xxx` headers (xxx can be any value).
2. delete any `Proxy-Connection: xxx` headers
3. Insert a `Connection: close` header.

Using `Connection: close` ensures that only one request and its response are ever passed over a single TCP connection, so the end-of-stream condition is the indicator you can follow to stop copying, close the connections, and exit the process (or thread) that is handing a particular request.

Another important boundary is the one between the headers and the content. This boundary is always marked by a blank line (look for a pair of carriage-return line-feeds in the input). The C escape sequence for this sequence is "\r\n\r\n".

Finally, in order to process POST requests you will need be able to read exactly the right amount of data following the POST header. The length that you need is contained in the `Content-Length:` header of the POST request.

Development Hints

I suggest starting by getting code working for a simpler problem first: develop a forking server that reads HTTP requests and prints them, sending back to the browser (client) a page that contains content of your choosing (a copy of the request might be interesting). Set up a web browser to use your server as its proxy server and try requesting some pages. Look at the output of your proxy to see what sort of input it is getting. That will help you with completing the assignment.

Once your basic server is working you will need to process GET, POST, and HEAD messages from the client to find out what web server to connect to. Remember that the format of these messages is

```
OP /path/to/file HTTP/1.x
```

A `host:` header will follow that identifies the server from which the file is to be retrieved. You will need to open a connection to the server and pass along the the request message on the new connection to the server.

Advice

1. The work YOU submit MUST be the result of your own efforts. Do not copy code from other students or from the Internet. Any violations of this rule will result penalties ranging from an automatic zero grade on the project to an F in the course.
2. Start early. This project can be quite time consuming. Remember that you are probably learning quite a number of new things in the course of the project such as use of wait() and fork(), how to organize programs that read data from one socket and write it to another, and so forth. As calibration points: it took me about 10 hours to do this project and while working out the details of the specification. It took a TA about 8 hours in a previous semester. You can expect to spend two or three times as long.
3. Ask questions EARLY. Vic and I are available to help you several hours per week, but we are not available for extensive help at the last minute.

Grading

- 65% of the grade will be for working code, defined as being able to use your proxy server with a browser to successfully handle web requests and responses. If everything except POST works you will lose 10%. Your demonstration of your server to the TAs will be part of this grade.
- 5% for the README file
- 5% for the Makefile

- If you program will not compile or core-dumps when run the maximum credit is 50% for the project.

The remaining 25% of the grade will be for your COMMENTS file including its analysis of the HTTP headers that you find. More details on what I expect in the COMMENTS file will be provided along with the list of web pages for you to test against.

Extra Credit

For up to 20% extra credit, eliminate the need to modify the `Connection:` headers. To accomplish this you will need to be able to process multiple requests and responses on a single connection. (This is something you would definitely want to be able to handle in real proxy server!)