

CS 351 Fall 2009

Lab 6: A 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 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 an end server outside. The proxy may do translation on the page, for instance, to make it viewable on a Web-enabled cell phone. Proxies are also used as *anonymizers*. By stripping a request of all identifying information, a proxy can make the browser anonymous to the end 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 end server.

In this lab, you will write a Web proxy that logs requests. Your proxy will function in a purely sequential way: waiting for a request, forwarding the request to the end server, and returning the result back to the browser. A log of all requests will be saved in a disk file.

2 Hand Out Instructions

Start by updating your repository to the latest version of the lab distribution:

```
git pull
```

Besides a README and a Makefile, you will find the following three C source files in your tree:

- `proxy.c`: This is the only file you will be modifying and handing in. It contains the bulk of the logic for your proxy.
- `csapp.c`: This is the file of the same name that is described in the CS:APP textbook. It contains error handling wrappers and helper functions such as the RIO (Robust I/O) package (CS:APP 11.4), `open_clientfd` (CS:APP 12.4.4), and `open_listenfd` (CS:APP 12.4.7).
- `csapp.h`: This file contains a few constants, type definitions, and prototypes for the functions in `csapp.c`.

Your `proxy.c` file may call any function in the `csapp.c` file. However, since you are only handing in a single `proxy.c` file, please don't modify the `csapp.c` file. If you want different versions of functions in `csapp.c` (see the Hints section), define and write new functions in the `proxy.c` file.

3 Implementing Your Web 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.

Note that the only types of requests your proxy will service are HTTP GET requests – all other requests may be ignored. A request will look something like the following:

```
GET http://www.iit.edu/ HTTP/1.1
```

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.

3.1 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 (corresponding to the request above):

```
Fri 27 Nov 2009 09:52:03 CST: 127.0.0.1 http://www.iit.edu/ 13666
```

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 `csapp.c` to create a log entry in the required format.

3.2 Port Numbers

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

```
unix> ./proxy 8888
```

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 (including other students' proxies). See `/etc/services` for a list of the port numbers reserved by other system services.

4 Evaluation

Your implementation will be evaluated in the following areas:

- Basic proxy functionality (30 points). Your sequential proxy should correctly accept connections, forward the requests to the end server, and pass the response back to the browser, making a log entry for each request. Your program should be able to proxy browser requests to the following Web sites and correctly log the requests:

- `http://www.google.com`
- `http://www.aol.com`
- `http://www.cnn.com`

5 Submission

To submit your work, type the following in the root handout directory:

```
make handin
```

6 Hints

- The best way to get going on your proxy's server component is to start with the basic echo server (CS:APP 12.4.9) and then gradually add functionality that turns the server into a proxy.
- Initially, you should debug your proxy using telnet as the client (CS:APP 12.5.3). Later, you may test your proxy (locally) with a real browser. Explore the browser settings until you find "proxies", then enter the host and port where you're running yours. Just set your HTTP proxy, because that's all your code is going to be able to handle.
- Since we want you to focus on network programming issues for this lab, we have provided you with two additional helper routines: `parse_uri`, which extracts the hostname, path, and port components from a URI, and `format_log_entry`, which constructs an entry for the log file in the proper format.
- Be careful about memory leaks. When the processing for an HTTP request fails for any reason, all open descriptors must be closed and all associated memory resources freed.
- Use the RIO (Robust I/O) package (CS:APP 11.4) for all I/O on sockets. Do not use standard I/O on sockets. You will quickly run into problems if you do. However, standard I/O calls such as `fopen` and `fwrite` are fine for I/O on the log file.
- The `Rio_readn`, `Rio_readlineb`, and `Rio_writen` error checking wrappers in `csapp.c` are not appropriate for a realistic proxy because they terminate the process when they encounter an error. Instead, you should write new wrappers that simply return after printing a warning message when I/O fails. When either of the read wrappers detects an error, it should return 0, as though it encountered EOF on the socket.
- Reads and writes can fail for a variety of reasons. The most common read failure is an `errno == ECONNRESET` error caused by reading from a connection that has already been closed by the peer on the other end, typically an overloaded end server. The most common write failure is an `errno == EPIPE` error caused by writing to a connection that has been closed by its

peer on the other end. This can occur for example, when a user hits their browser's Stop button during a long transfer.

- Writing to connection that has been closed by the peer first elicits an error with `errno` set to `EPIPE`. Writing to such a connection a second time elicits a `SIGPIPE` signal whose default action is to terminate the process. To keep your proxy from crashing you can use the `SIG_IGN` argument to the `signal` function (CS:APP 8.5.3) to explicitly ignore these `SIGPIPE` signals