Compsci 512: Distributed Systems
Assignment 1
Writing a Web Proxy

Instructor: Bruce Maggs

January 21, 2014

OVERVIEW

In this assignment, you will implement a web proxy that lies between your browser and the Internet. **Your program must be written in C and must use the pthreads library to create a separate thread to process each request that the proxy receives.** All HTTP and HTTPS requests sent from your browser to any Web site's origin server should go through your proxy, which will then forward any responses from origin servers back to the browser. If you implement your proxy correctly, then changing your browser settings to indicate that you would like to use your proxy as a web proxy, you should be able to browse the web and view web sites just as you could when you were not using your proxy.
The basic work flow of your proxy should be as follows:

1. From the command line (most likely on a linux computer), start your proxy so that it listens on a certain port given as a command-line argument. Port 8080 would be a reasonable choice.

2. In your browser, when you type a URL such as *http://www.cnn.com*, the request should be forwarded to your proxy. This can be done by going into your browser settings and indicating that you want to use a web proxy. You need to specify the host and the port of the proxy. Then your browser will then send all HTTP and HTTPS requests to your proxy.

3. Upon receiving a request for a new connection, the proxy should spawn a thread to process it. By reading from the connection, the thread learns whether the request is *GET* for HTTP or *CONNECT* for HTTPS.

4. In the case of GET, the request includes a URL that contains both a host/port and a path. The proxy uses the URL to fetch the content from the origin web server, and as it receives the content, it passes it on to the proxy.

5. In the case of CONNECT, the request includes a host/port, but no path. The proxy should open a connection to the origin server, and send an HTTP 200 status code to the browser. It should then begin to forward any data from the origin server to the browser and from the browser to the origin server without any further processing or examination of the data. Because you need to transfer data in both directions simultaneously, you should spawn a second thread to help process the request.

6. When there is no more data to be transfered, close the connection to the web server and the connection to the browser. Note that you are NOT required to maintain a persistent connection between client and the web server.

CODE BASE

You do not need to start from scratch. We provide you with some files that you should use as a starting point. You can find all these files at Sakai. The following files are provided.

1. *proxy.c* and *proxy.h*: *proxy.c* is the where the main function is and is only file that all you should work on. It provides a helper function that makes URL parsing a little easier.

2. *csapp.c* and *csapp.h*: *csapp.c* provides you a number of wrapper functions that deal with I/O and file descriptors. You should use functions from these files and there is no need to directly call Linux system calls such as *read*, *pthread_creat* etc. Something worth noting is that this implemented a robust I/O package(rio).

SOME IMPLEMENTATION DETAIL

Here are some hints that will help you get your proxy work correctly.

1. Many things can go wrong when establishing a network connection or in the midst of reading and writing across a network connection. Your proxy should check for errors and handle them gracefully. In particular, you proxy should not crash or exit just because something goes wrong with a network connection. Be careful about which library functions you use to ensure that you handle errors gracefully.

2. For HTTPS connections, you should NOT use the buffered read functions in RIO library, including: *rio_readin*, *rio_readnb*, *rio_readlineb*. Use *rio_readp* instead.

3. Make sure that your proxy ignores SIGPIPE signals by installing the *ignore* function as handler. This can be done by using *signal(SIGPIPE,SIG_IGN)*.

4. When EOF is detected while reading from the server socket, send EOF to client socket by calling shutdown(clientfd,1); (and vice versa).

5. The proxy should pass all request headers from client on to the server, while the following possible exception: You may want to strip off any request header of the form: "Proxy-Connection: Keep-Alive". Otherwise the server may expect the proxy to maintain a persistent connection, which your proxy should not do.

6. After passing the request headers on to the server, the proxy also MUST pass a line consisting only of "\r\n" (to tell the server that there will be no more request headers).