CPSC 441: Computer Networks

Professor Carey Williamson

Winter 2022

Assignment 1: Clown Proxy (40 marks)

Due: **Friday, January 28, 2022** (4:00pm)

Learning Objectives

The purpose of this assignment is to learn about the HyperText Transfer Protocol (HTTP) used by the World Wide Web. In particular, you will design and implement a Web proxy using HTTP to demonstrate your understanding of this application-layer protocol. Along the way, you will also learn a lot about socket programming, TCP/IP, network debugging, and more.

Preamble

With the New Year upon us, and the pandemic still a part of our lives, it is time for some frivolity to lighten things up. After all, April Fool's Day is less than three months away, and we need to be ready. Writing a Web proxy to brighten someone's day seems like a good strategy for this. To keep the assignment simple, we will restrict ourselves only to plain-text HTTP sites, not secure HTTP (HTTPS) sites.

Background

A **Web proxy** is a piece of software that functions as an intermediary between a Web client (browser) and a Web server. The Web proxy intercepts Web requests from clients and determines whether they should be transmitted to a Web server or not. If the request is blocked, the proxy informs the client directly. If the request is forwarded to the Web server, then any response that the proxy receives from the Web server is forwarded back to the client. From the server's point of view, the proxy **is** the client, since that is where the request comes from. Similarly, from the client's point of view, the proxy **is** the server, since that is where the response comes from. A Web proxy thus provides a single point of control to regulate Web access between clients and servers. A lot of Calgary schools use Web proxies to limit the types of Web sites that students are allowed to access. **Net Nanny** and **Barracuda** are examples of commercially available Web proxies.

Technical Requirements

In this assignment, you will implement your very own **clown proxy**, in either C or C++. The goals of the assignment are to build a properly functioning Web proxy for simple Web pages, and then use your proxy to alter certain content items before they are delivered to the browser.

There are two main pieces of functionality needed in your proxy. The first is the ability to handle HTTP requests and responses, while still forwarding them between client and server. This is called a **transparent** proxy. The second is the ability to parse, and possibly modify, HTTP requests and responses. This could involve: (a) rewriting some of the content in an HTTP response before that content is displayed by your Web browser; and (b) rewriting some of the HTTP requests so that they request a different object than is normally retrieved on a Web page.

In the spirit of silliness, your proxy should be able to do two specific things. First, it should replace all occurrences of the word "Happy" with the word "Silly" in an HTTP response. Second, it should replace all JPG image files on a given Web page with an image of a happy clown instead. (Please don't use an evil clown, since some people really do suffer from coulrophobia, which is the fear of clowns.)

The most important HTTP command for your Web proxy to handle is the "GET" request, which specifies the URL for an object to be retrieved. In the basic operation of your proxy, it should be able to parse, understand, and forward to the Web server a (possibly modified) version of the client HTTP request. Similarly, the proxy should be able to parse, understand, and return to the client a (possibly modified) version of the HTTP response that the Web server provided to the proxy. Please give some careful thought to how your proxy handles commonly occurring HTTP response codes, such as 200 (OK), 206 (Partial Content), 301 (Moved Permanently), 302 (Found), 304 (Not Modified), 403 (Forbidden), and 404 (Not Found).

You will need at least one TCP socket (i.e., SOCK_STREAM) for client-proxy communication, and at least one additional TCP socket for each Web server that your proxy talks to during proxy-server communication. If you want your proxy to support multiple concurrent HTTP transactions, you may need to fork child processes or create threads for request handling. Each child process or thread will use its own socket instances for its communications with the client and with the server.

When implementing your proxy, feel free to compile and run your Web proxy on any suitable department machine, or even your home machine or laptop, but please be aware that you will ultimately have to demo your proxy to your TA on campus at some point. You should try to access your proxy from your favourite Web browser (e.g., Edge, Firefox, Chrome, Safari), and computer (either on campus or at home). To test the proxy, you will have to configure your Web browser to use your specific Web proxy (e.g., look for menu selections like Tools, Internet Options, Proxies, Advanced, LAN Settings). Make sure that you only tamper with HTTP, and not HTTPS.

As you design and build your Web proxy, give careful consideration to how you will debug and test it. For example, you may want to print out information about requests and responses received, processed, forwarded, redirected, or altered. Once you become confident with the basic operation of your Web proxy, you can toggle off the verbose debugging output. If you are testing on your home network, you can also use tools like **WireShark** to collect network packet traces. By studying the HTTP messages and TCP/IP packets going to and from your proxy, you might be able to figure out what is working, what isn't working, and why.

When you are finished, please submit your solution in electronic form to your TA via D2L. Your submission should include the source code for your Web proxy, a brief user manual describing how to compile and use your proxy, and a description of the testing done with your proxy. Please remember that assignments are to be done individually, and submitted to your assigned TA on time. You should also plan to give a brief demo of your proxy to your TA during a tutorial time slot just after the assignment deadline.

Testing

During your demo, your proxy will be tested on the following test cases:

- **a very simple Web page**
- **a Web page with song lyrics**
- **a Web page with a link to a photo of Floppy**
- **a Web page with an embedded image of Floppy**
- **a Web page with several embedded images**
- **Carey's home page with multiple embedded images**

Good luck, and have fun!

Grading Rubric

The grading scheme for the assignment is as follows:

- **12 marks** for the design and implementation of a functional Web proxy that can handle simple HTTP GET interactions between client and server, using either HTTP/1.0 or HTTP/1.1. This basic proxy should be able to deliver Web pages in unaltered form, and be able to handle HTTP redirection when it occurs. Your implementation should include proper use of TCP/IP socket programming in C or C++, and reasonably commented code.
- **6 marks** for that part of your Web proxy that can parse HTTP responses, detect the word "Happy", and substitute the word "Silly" instead.
- **8 marks** for that part of your Web proxy that can parse HTTP requests, identify what content is being requested, and make proper substitutions of a clown image for the original image when a JPG file is requested.
- **4 marks** for a clear and concise user manual (at most 1 page) that describes how to compile, configure, and use your Web proxy. Make sure to indicate the required features and optional features (if any) that the proxy supports. Make sure to clarify where and how the testing was done (e.g., home, university, office), what works, and what does not. Be honest!
- **10 marks** for a suitable demonstration of your proxy to your TA in your tutorial section, or to your professor at a mutually convenient time. A successful demo will include marks for the test cases given above, as well as clear answers to questions asked during your code walk-through.

Bonus (optional)

Up to **4 bonus marks** will be given for a clown proxy that can randomly choose from a small set of different clown images when doing image substitution, resulting in a page that varies every time you load it. Make sure to show this bonus feature to your TA during the demo.

Tips

- This is a **very challenging** assignment, so please **get started early**. You will likely need 7-14 days of thinking/coding/debugging time to get it fully working.
- If you have never done socket programming in C/C++ before, you should make sure to get to your CPSC 441 tutorials on this topic. Don't miss them!

- Make sure that you have completed and understood **Assignment 0**, which provides a basic introduction to client-server socket programming.
- If you don't speak HTTP already, make sure to get to your CPSC 441 tutorial on this topic. For example, the "Content-Type" header in an HTTP response is useful for determining the object type and the "Content-Length" header will tell you how big an object is (in bytes). (Hint: you don't want to print binary JPG images onto your screen in debugging messages!)
- Focus on the basic HTTP proxy functionality first, by simply forwarding everything that you receive from the client directly to the server, and everything you receive from the server directly back to the client. Then add more functionality, such as request parsing, URL rewriting, or HTTP redirection.
- Your proxy will need one socket for talking to the client, and another socket for talking to the server. Make sure to keep track of which one is which. This is very important to understand!
- Your proxy will likely need to dynamically create a socket for every new server that it talks to. Most of the examples above involve only one server, which is easier. But you will likely need to generalize this to multiple servers. If so, make sure to manage these sockets properly.
- Start with **very simple** Web pages, such as those indicated above. Once you have these working, then you can try more complicated Web pages with lots of embedded objects, possibly from multiple servers.
- Be aware that most Web browsers like to cache content locally (i.e., browser cache), rather than retrieving it from the server each time it is requested. You will want to force a reload from the server each time so that you don't get fooled during your testing.
- You may find that network firewalls block certain ports, which may make configuration and use of your proxy tricky. For example, it might be easier to do all of your testing using machines within the CPSC network, rather than external ones. A good Wireshark trace can help show you what is actually happening on the network.
- Try to avoid servers that automatically redirect HTTP to HTTPS, since TLS handshakes and encrypted content are well beyond the intended scope of the assignment. Let's keep things simple with HTTP only.
- The assignment is structured into incremental pieces that are each worth partial marks. As you get the different pieces of functionality working, make sure to save working versions of your code somewhere. Adding new functionality sometimes breaks things in a bad way, and you don't want this happening just prior to the assignment deadline. Expressed another way, it is probably better to have a fully-working

partial solution for your demo, rather than a partially-working full solution. If your code does not even compile, there is not much that you can demo, so please be careful if/when making last-minute code changes.

- Here is a lengthy **debugging checklist** for socket programming that you might find helpful. Good luck!