Assignment 2: HTTP (15%)

Fall 2023 – CPSC 441

Due at 23:59, Oct. 15 on D2L

Assignment policy:

- This is an **individual** assignment, so the work you hand in must be your own. Any external sources used must be properly cited (see below).
- Extensions will not be granted to individual students. Requests on behalf of the entire class will only be considered if made more than 24h before the original deadline.
- Some tips to avoid plagiarism in your programming assignments:
  1. Cite all sources of code that you hand in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:
     `# the following code is from https://www.quackit.com/hello_world.cfm.`
     Use the complete URL so that the marker can check the source.
  2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
  3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
  4. Collaborative coding is strictly prohibited. Your assignment submission must be entirely your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. You cannot use another student's code, even with citation.
  5. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task.
  6. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

Background:

So far, there haven't been enough frogs in this course, so this is our chance to fix that.

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.

The purpose of this assignment is to learn about the HyperText Transfer Protocol (HTTP) used by the World Wide Web. To do this, 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 about socket programming, TCP/IP, network debugging, and more. To keep the assignment manageable, we will restrict ourselves to only plain-text HTTP sites, and not HTTPS sites.

Instructions:

In this assignment, you will implement your very own **frog 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 a proxy. The first is the ability to handle HTTP requests and responses by 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.

Your proxy should be able to do two specific things. First, it should replace all occurrences of the word "Frog" (regardless of capitalization) with the word "Fred" in an HTTP response. Second, it should replace all JPG image files on a given Web page with an image of a frog instead (you can choose any one you like, or randomly choose a photo from a collection).

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 to the dropbox on 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. 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 – instructions for this will be provided on Piazza.

Sample websites:

I've provided a selection of websites for you to test your assignment on. All of them can be accessed through my homepage, `http://pages.cpsc.ucalgary.ca/~jcleahy/`. There are five webpages you can use, arranged in increasing order of difficulty. The TAs will ask you to demonstrate on these web pages, so make sure everything is working properly.

Rubric (40 points total):

- **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 clear and commented code.
- **6 marks** for the part of your Web proxy that can parse HTTP responses, detect the word "Frog", and substitute the word "Fred" instead.
- **8 marks** for the part of your Web proxy that can parse HTTP requests, identify what content is being requested, and make proper substitutions of a frog 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 registered tutorial section. 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.

Tips:

- This is a challenging assignment, so get started early. Give yourself at least a week of coding/thinking/debugging time, and plan your questions for the available tutorials.
- A handy debugging checklist is included under Assignments on D2L.
- If you have never done socket programming in C/C++ before, make sure to attend your tutorials on this topic. Check the D2L tutorial schedule for dates.
- If you don't speak HTTP already, there is a tutorial on this topic as well. 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 writing a basic transparent proxy 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.
- 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 (refresh page) 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, or on virtual machines, rather than external ones. Wireshark 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. As mentioned, you assignment only needs to work on HTTP.

- Some browsers will also force you to use HTTPS instead of HTTP. This can be diagnosed with telnet and changed in the browser settings if necessary. Firefox doesn't seem to have this issue by default, so would I recommend it if this aspect becomes a problem for you.
- 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. It is probably better to have a fully-working partial solution for your demo 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 about making last-minute code changes.

author: jcleahy@ucalgary.ca