COMP3310 2022 - Assignment 2: An annoying web-proxy

Background:

- This assignment is worth 15% of the final mark
- **It is due by 23:55 Friday 22 April AEST - note: CANBERRA TIME (gmt+10)**
- Late submissions will not be accepted, except in special circumstances
    - Extensions must be requested as early as possible before the due date, via the course convenor, with suitable evidence or justification.
- ***If you would like feedback on particular aspects of your submission, please note that in the README file within your submission.***

This is a coding assignment, to enhance and check your network programming skills. The main focus is on native socket programming, and your ability to understand and implement the key elements of an application protocol from its RFC specification.

Assignment 2 outline

A web-proxy is a simple web-client and web-server wrapped in a single application. It receives requests from one or more clients (web-browsers) for particular content URLs, and forwards them on to the intended server, then returns the result to your web-browser - in some form. How is this useful?

- It can cache content, so the second and later clients to make the same request get a more rapid response, and free up network capacity.
- It can filter content, to ensure that content coming back is 'safe', e.g. for children or your home, or for staff/their computers inside an organisation.
- It can filter requests, to ensure that people don't access things they shouldn't, for whatever policy reasons one might have.
- It can listen to requests/responses and learn things, i.e. snoop on the traffic. Getting people to use your proxy though is a different challenge...
    - And of course it can listen to and modify requests/responses, for fun or profit.

For this assignment, you need to write a web proxy in C, Java or Python[1], **without** the use of any external web/http-related libraries (html-parsing support is ok though).

Your code MUST open sockets in the standard socket() API way, as per the tutorial exercises. Your code MUST make appropriate and correctly-formed HTTP/1.0 (RFC1945) or HTTP/1.1 enhanced requests (to a web-server, as a client) and responses (to a web-browser, as a server) on its own, and capture/interpret the results on its own in both directions. You will be handcrafting HTTP packets, so you'll need to understand the structures of requests/responses and key HTTP headers.

Wireshark will be helpful for debugging purposes. The most common trap is not getting your line-ending '\n\n' right on requests, and this is rather OS and language-specific. Remember to be conservative in what you send and reasonably liberal in what you accept.

[1] As most high-performance networking servers, and kernel networking modules, are written in C with other languages a distant second, it is worth learning it. But, time is short. If you want to use another language (outside of C/Java/Python), discuss with your tutor – it has to have native socket access, and somebody has to be able to mark it.

What your successful and highly-rated proxy will need to do:

1. Act as a proxy against a famous website, http://comp3310.ddns.net/
    a. *That website is not yet fully operational, an announcement will be made when it is. It will be an approximate mirror site of the Australian National Botanic Gardens site.*

2. Rewrite (simple) absolute URL links that originally pointed to the website to now point to your proxy, so all subsequent requests (to our website) also go via your proxy.
    a. *Sometimes links are not written in pure `<a href="…">` style, e.g. they are calculated within javascript, and we will accept those breaking, after checking.*

3. Modifies the text content, by replacing every instance of the word "the" in the body text with the word "eht" in bold (i.e. in html you write `<b>eht</b>`

4. Logs (prints to STDOUT):
    a. The timestamp of each request
    b. Each client-request that comes into your proxy, as received ('GET / HTTP/1.0', etc.)
        i. Don't log other headers
    c. Each server-status-response that comes back (200 OK, 404 Not found, etc.)
        i. Don't log other headers
    d. A count of the modifications made to that page by your proxy, counting text changes and link rewrites separately (i.e. return two labelled numbers)

We will test this against the specified website, by running your code, opening our web browser, making a top-level ('/') page request to your running proxy as if it were the server and we should get back our remote homepage, modified suitably[2]. Any (simple) links we click on that page in our browser should take us back to your proxy and again through to the site for that next page, and so on. We're not going to go too deep, there are some overly complex pages, we will just pick a few. There will be only one client-browser at a time running against your proxy. Note, this is an interactive process, you're not caching or otherwise storing modified pages.

You should only need to manage the HTML pages (check the Content-Type header) for modifications. Any non-HTML content (e.g. images, JS, CSS, etc.) from the site can be passed through unchanged. Don't forget to capture and passthrough all the headers in the request, as the site may require at least the HTTP/1.1 'Host:' header.

For efficiency you can also use persistent http/tcp connections, but beware of connection-timeouts.

Submission and Assessment

You need to submit your source code, and an executable (where appropriate). If it needs instructions to run, please provide those in a README file. Your submission must be a zip file, packaging everything as needed, and submitted through the appropriate link on wattle.

There are many existing web-proxying/caching tools and libraries out there, many of them with source. While perhaps educational for you, the assessors know they exist and they will be checking your code against them, and against other submissions from this class.

[2] Most browsers support the direct configuration of a proxy address, but the behaviour can be a bit inconsistent, so we're trying to avoid that.

Your code will be assessed on [with marks% available]

1. Output correctness [40%]
    - The http queries it sends to the server on behalf of the client-browser
    - The modified server-content and http packaging it returns to the client-browser
    - The ability for users to follow links in their browser
    - The log of requests/responses as above.
2. Performance [20%]
    - A great proxy should be perfectly transparent, not causing any significant delays.
    - How easy the code is to run, using a standard Linux environment (like the CS Labs, WSL)
3. Code correctness, clarity, and style [40%]
    - Use of native sockets, writing own HTTP sender/receiver messages
    - Documentation, i.e. comments and any README - how easily can somebody new pick this up and modify it.

There can be gnarly html pages, with embedded scripts and links in image-maps, and other tricks, as well as many references to other websites. We'll be relaxed about broken links in some cases, and you should not proxy links or requests that are not on the comp3310.ddns.net website.

You should be able to test your code against any HTTP-based website you like, although a lot of sites use HTTPS now, or have complex html/js pages that can make parsing harder. Wireshark is very helpful to check behaviours of your code against browsers or command line tools like wget/curl. Your tutors can help you with advice (direct or via the forum) as can fellow students.