COMP3310 2021 - Assignment 2: An annoying web-proxy

Background:

- This assignment is worth 15% of the final mark
- **It is due by 23:59 Sunday 9 May AEST - note: CANBERRA TIME (gmt+10)**
- Late submissions will not be accepted, except in special circumstances
    - Extensions must be requested well before the due date, via the course convenor, with appropriate evidence.

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.

窥探一下交通状况

For this assignment, you need to write a web proxy in C or Java, **without** the use of any external web/http-related libraries (html-parsing support is ok). ENGN students with limited C/Java backgrounds should talk with their tutors as we have other options there, though the requirements will be the same and more closely considered. As most networking server code is written in C, with other languages a distant second, it is worth learning it.

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 HTTP headers.

Wireshark will be helpful for debugging purposes, compare it to a direct web-browser transaction. 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.

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

1. Act as a proxy against a website we name. You must allow that name to be specified either as a command line argument or read from a file.
2. Rewrite (simple) html links that originally pointed to the website to now point to your proxy, so all subsequent requests 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 content, by replacing **displayed** Australian capital city names with random city names of your choosing, but be consistent. Be careful not to break the website access (e.g. where a page is called Sydney.html, it still has to work, don't rename that link - only modify the displayed text).
    a. You can do more, e.g. rotating/replacing images. More fun but no extra marks here.
4. Logs (prints to STDOUT):
    a. Timestamp of each request
    b. Each request that comes into your proxy, as received ('GET / HTTP/1.0', etc.)
        i. Don't log the other headers.
    c. Each status response that comes back (200 OK, 404 Not found, etc.)
        i. Don't log the other headers
    d. A count of the modifications made to the page by your proxy, counting text changes and link rewrites separately (i.e. return two labelled numbers)

We will test this against the Bureau of Meteorology (BoM) website, by opening our web browser or telnet, making a top-level ('/') page request to your running proxy as if it were a server and we should get back the BoM homepage, modified suitably. Any (simple) links we click on that page should take us back to your proxy and again through to the BoM site for that next page, and so on. We're not going to go too deep, there are some overly complex pages on the sites, but we will pick 5-10 pages. There will be only one client at a time running against your client.

The reason for being flexible about the website to run against is that you can also daisy-chain proxies, i.e. to connect one proxies' output to another's input. This is one way of testing new protocol developments before they are accepted as IETF RFC's, to see that everyone agrees with the protocol syntax. You can test this with classmates in tutorials or outside. It's also used to federate a hierarchy of caches, so that the most popular content for a given network radius is more likely to be cached closer to the consumers, on potentially smaller caches.

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 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.

Your code will be assessed on

1. Output correctness (the http queries it sends, the modified BoM pages, the log of requests),
2. Performance (a great proxy should be perfectly transparent, not causing any delays),
3. Code correctness, clarity, and style, and
4. Documentation (i.e. comments and any README - how easily can somebody new pick this up and modify it).

Marks are allocated roughly 50% for 1-2 and 50% for 3-4.

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.