HTTP Server Journal

A.K.A - A misadventure in time management, discipline and caffeine dependency.

There's not much use in denying to myself that I had a dissappointing work ethic towards this project. It was consistently left near the back of my priority queue. It's no surprise that before I can realise it the deadline is looming upon me like a bounty on my head.

The Goal

My goal was to fulfill the core functionality; parse HTTP requests and respond accordingly, where the client's experience is clean, bug free and protected against tech-illiterate users getting something they didn't expect.
I wanted to implement it in C and do it right. I wanted to implement SSL but I didn't have the time.

A skim read of RFC 2616 made me almost decide I needn't cater to half of what the HTTP standard prescribes. For the purpose of the assignment, there'd be no urgent need for most of the standard. I implemented GET and would return a HTTP 501 error on any other request.
Implementing POST would be the next stretch goal.

The Process

Thanks to the Operating Systems module's C based server-architecture assignment introducing me to the field, I felt familiar enough with pthreading and sockets such that I pretty swiftly made a simple threaded server that can read data from whoever connects to it.

Parsing the HTTP requests was more painful than it should've been.
Working with strings in C is more work than it's worth! I have no doubts that trying to perform operations on strings took the vast majority of my time on this project.
I like to think of myself as a a very confident programmer... in Java. I'm comfortable with solving problems with code but doing what I wanted in the C language was a fun learning experience.

Given that I was somewhat hurrying the code to production, the first draft (and first submission) of my code was full of really horrible programming practices, it was truly shocking.

After achieving what I felt like was the primary goal of the assignment, I turned to cleaning up my code a bit, so that I could submit something I actually felt pride for.
Needless to say, it's a lot less ugly now. I split functionality into easier to read chunks and I have much fewer ambiguous temporary variables. It's still in one file, and thus is not a daemon. If I had more time I'd simply write a fork() routing that detaches my logic-processes from the TTY.

With HTTP GET requests implemented and the c code tidied up, I wanted to make quality of life improvements.

I created a www folder for all file requests to be directed do, so a HTTP request can't request the src code.
I also expanded on the HTTP responses. Originally the HTTP response used a fixed buffer size of 1024 bytes to write to the socket. This caused seg faults when I served any files of decent size. I made a dynamic buffer that is stack-allocated dynamically depending on the size of the file needing to be served, plus the HTTP header.

If I were to do this assignment again, I'd give myself a sensible time to do it in. This'd let me actually develop the features of the program in a more well rounded manner. I was quite interested in implementing SSL but alas, I didn't give myself a realistic time frame within which to build the system.