HTTP Server Assignment

Diary 1522391

This diary details the progress I made throughout the assignment.

First steps

Initially I was very unsure how to go about starting the assignment, having very little knowledge of the HTTP standard in general, and with networking in C, which I did not know at all. The API lectures and example code provided by Ian (which eventually became the basis of the server code) provided a good starting point, I went through the code and figured out what each step was doing. Since the code solved the first two preparatory exercises, I set about working on the third: parsing the HTTP request. I wrote a poorly-implemented program using strchr() and pointer arithmetic to print the request URI, method and HTTP version. During this process I ran into several problems with C strings which has helped me a lot with my understanding of them: mainly the null byte at the end, which I was not always retaining. This caused problems when reading and printing these variables.

Responding to the request

Satisfied with this progress, I continued on to actually respond to the request. After being pointed to a particularly useful source (https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html), which helped me understand the HTTP response and request better, I finally managed to send a 404 back to the client regardless of input. Later that day I also managed to serve HTML files by reading them in. I experienced problems with this: a file would initially be served correctly, but then return a 404, and then perhaps work again several refreshes later. I found this was due to the aforementioned null byte bug: my URI string was not being terminated correctly and the program was reading it incorrectly. Unfortunately image files refused to work, a problem I would not solve until later. My initial method of parsing the request was also getting dangerously messy, with several nested if statements and exit points with failure.

Fixing the code

I decided that before I continued work, I would have to improve my code. I did this mainly by changing many malloc'd char arrays to stack-stored arrays to reduce the number of frees in the code. I also changed from using many strcats and strcpys to sprint and snprintf so I could combine many different variables at once (useful when writing the response header). I also tried to find out the image problem. Initially I discovered that reading binary files with the 'r' flag could lead to errors, so I changed this to the 'rb' flag, with no success. After studying the served images with a hex editor I discovered that the image was being served correctly, but with one extra null byte in the first position. I discovered that this was because I was sending an extra null byte with my header before sending the content.

Overhauling parsing

Having seen people talking about the strtok() method on various Canvas discussions, I decided to investigate it. Realising that it was a far superior method than my current parsing, I replaced a large bulk of my code with a shorter strtok() based method. I also implemented header field parsing, which I would use later for byte ranges. All I did at that moment was to print out the fields for debugging. Work on a similar networking Operating Systems assignment had introduced me to signal handling, so I implemented it to gracefully catch SIGINT and SIGTERM.

Directory listing

Reading the assignment specification (and prompts from friends) led me to realise I had missed something: listing directories. I studied online documentation and managed to create a function that iterated through items in a directory and wrote an html file containing these items (with different formatting if they were a directory or file). I soon ran into a problem where it would work once and then not work on subsequent clicks. After some investigation, I realised that the effects of chdir() were retained after every function call, so I stored the root directory prior to changing, so that I could return to it after reading from the directory. I also had trouble with implementing my 'up directory' button: I had believed that <a href = '..'> would work but this never complied, so I instead used a slightly 'hacky' method in which I parsed the directory address and removed everything following the final slash.

Byte ranges

With all the main features implemented, I turned to something slightly more advanced: serving byte ranges. I discovered there were several foibles to this: single and multiple ranges had to be handled differently, for example. Using my earlier header field parsing, I extracted the actual byte ranges using strtok() and stored them in arrays. While actually trying to extract the byte ranges (by using strncpy from a given index of the whole content), I ran into several memory errors causing segfaults. I discovered that this was because my char arrays were not actually being initialised, so I switched to using calloc rather than malloc for these allocations.

Tidying up

With the functionality I wanted implemented, I started to tidy up my code, by commenting and using better whitespace throughout. I also added several error checks for malloc which I had forgotten to when writing the code earlier. I also removed as many memory leaks as I could, although there is still one stubborn one revolving around pthread_create that will not disappear whatever I try (pthread_cancel, pthread_detach etc.).

What would I do differently

If I were to repeat the assignment, I would definitely try to keep my code a lot neater from the get-go, and actually solve problems rather than slapping 'quick fixes' on them (this was often used during my initial parsing strategy). Mid-way through the assignment my code was getting quite complex and it needed tidying so that I could actually understand what my own code was doing! Commenting on code throughout would also be a useful thing to do so that when I come back to it later I know where to apply fixes.