Diary

I began the assignment by completing parts 1 and 2. I heavily relied on the sample code and worked through each line of code to ensure I was confident that I knew what the code was doing and why it was necessary. I copied the code into a slightly compressed format (for example, by removing comments that I felt were unnecessary and reducing the number of files) and made sure that the code compiled and worked exactly as the sample code. This took longer as expected as I made some small mistakes such as a few misplaced pointers and brackets, even though the code still compiled.

After the code for part 2 was fully functional I began to experiment with *service_client_socket* by customising what was sent back from the server, rather than just sending an exact copy of what the server received. Even though I only slightly altered what the server returned ("Server received this: [message]"), my lack of understanding of the networking functions and the overall structure of the code meant that this turned out to be much more difficult task than expected. However, once I quickly read through the libraries of functions like *write* and *send* I began to understand how these functions handled the data I was receiving and sending, and also the general structure my code had to follow.

When I was happy with the performance of part 2 and my understanding of the code I moved onto part 3. Here I spent some time playing with *wget* to find exactly how an HTTP request is structured and how the response should be structured. I found that the request had lots of information which was not relevant to the exercise and that the amount of information included in the headers varied through browsers, *wget* and *curl*. I also discovered that the only crucial part of the headers for this exercise was the resource location, for example "/" or "/test.txt". When looking into how the response should be structured, it became quite obvious that the only two lines I needed to include were the first line (HTTP version and status code) and, after a bit of confusion, the content length of the body.

Parsing the HTTP request posed another challenge. I went through many string search functions trying to find the right ones for the situation, and eventually settled on using *strstr* and *strtok* to get the host, resource location and request method. Once I was able to print out these key pieces of information, I realised that the rest of part 3 was not strictly necessary for the main assignment and that I would rather spend my time making progress on the main assignment.

I ran into many problems in the main assignment, some of them C-based and some of them networking based. Many of these problems I had no experience in, but a quick Stack Overflow search often allowed to me to progress in the exercise. These problems included: reading a file and placing the result into a string; reading directories; finding the attributes of files/directories; and multiple memory errors which I had never seen before.

One of the more interesting challenges in this exercise was returning some HTML code that could be displayed in a browser. Since I had very limited experience in HTML it took some time to find code that would correctly format what I wanted to send to the client, particularly when it came to providing the correct format for links on the webpage.

Perhaps what I spent the most time on in this final exercise was separating my code into functions and dealing with the resulting pointer problems and memory errors. These seem to be inevitable in C and there were a few errors which took a very long time to solve. For example, I had a lot of trouble moving the code from *read_file* into a separate function.

Due to my inexperience in C I ran into some basic problems such as not passing pointers to data into functions, but also some trickier problems where Stack Overflow could not save me (although these may also be simple mistakes). The main problem I faced was accessing the data from a *malloc* call in *read_file*, whereby I could not use *strlen* to find the length of the buffer, or change any part of the buffer itself (for example, adding a null terminator to the end). This is still the case and I am still unsure of the cause and/or solution.

If I were to repeat the exercise I would have spent more time initially researching the process of sending data via sockets, namely the *write* function and the requirements of an HTTP response. It took me some time to realise that I needed to include “Content-Length: “ as a header in my HTTP responses for the clients to finish waiting for data. It also took me quite some time to realise that I could send data using multiple *write*s as the client waits for all information before closing the connection. I believe my program would have been much simpler if I had designed it around using multiple *write*s to send a message, rather than adapting the code I had already to benefit from this discovery.