- 10th October
    - Started working on the assignment from scratch, initially following the broad outline (3 smaller tasks: single-threaded, multithreaded & parsing HTTP reqs)
    - I've got something that I think should work (for single-threaded), but the server doesn't seem to be accepting connections
    - The problem was that I was passing arg[0] not arg[1] as the port, which (whilst it worked), means that the socket was bound to a **completely** different port than expected because I was using *stoi* rather than *strtoul*...
- 11th October
    - Added some multithreading, which seems to be working. When a client disconnects, it doesn't work properly though.
    - I was closing clisockfd rather than *clisockfd, but I'm passing a pointer to the file descriptor to the handleConnection method - Now I've fixed this.
    - I'm going to focus on the OS assignment and my FYP for a bit and then come back to this - I feel like I'm making good progress.
- 17th October
    - Added structs and method stubs that will be used in methods to handle everything to do with HTTP responses - my plan is to initially respond with a successful request (containing the index page) regardless of the incoming request so I can test that it would successfully be displayed.
- 18th October
    - I've added the functionality to timestamp requests, and have implemented most of the methods required for creating/deleting/constructing/sending an HTTP response
    - The server now correctly serves a 200 response with the index page (stored as a char * in the program), which is displayed by the browser. I've ran into a couple of memory errors that I need to diagnose though...
    - The memory errors were caused because strlen doesn't count the null terminator of the string in the returned length. I've now fixed this.
    - I've added the appropriate method stubs and structs for handling HTTP requests - I plan to start handling GET requests soon, after I implement the *parse_http_request* method.
    - Whilst working, I noticed a potentially insecure piece of code where I'd erroneously used strcpy rather than strncpy on the body of the request that's provided by the user. This could have been used for a buffer overflow attack.
- 24th October
    - I've started work on actually handling HTTP requests when they're sent to the server
    - Namely, I've started implementing the *parse_http_request* method. I was originally going to use strtok(2) for this, but because it stores state, it isn't threadsafe. Instead, I'm using strtok_r(3), which is threadsafe.
    - For now, I'm just going to parse the request line of the request - I can get enough information from this to fulfil at least GET requests.

- 25th October
    - I've written a ReadFile method which reads a given file on the system into a char * - I'm going to use this to serve files from the server.
    - Because I want to restrict access to files requested, I've created a simple whitelist system and *servable* method, that just checks through a list (files.txt) of whitelisted files.
    - I'm now correctly handling 404 requests too - If a file isn't servable (i.e. it isn't whitelisted), the server will send back a 404 request.
    - If the http_verb isn't recognised, the server now sends back a 400 error.
    - I've tried reading in image files, but it doesn't seem to read or send the whole file, just the header...
- 26th October
    - Continuing work on the issues with images - the problem seems to be down to the fact that images can contain null bytes that aren't terminating
    - The first problem seems to be that I'm using strlen to get the length of the contents of the file once it has been read into a char * - I've changed how I'm doing this now. This still doesn't seem to have fixed the overarching issue though
    - Another problem is that I've been using *strncpy* in the *add_http_response_body* method, but it only copies up to **either** the specified length, or when a null character is reached. I've switched to using *memcpy* here instead.
    - The last piece to the puzzle is that I was writing to the socket using *dprintf* (to write to the fd), and supplying the body of the response as part of the format string - but using the %s token. This meant that only the characters up to the first null character were ever sent. I've changed this around to write everything except the body using dprintf, and then doing a second *write* with the body.
    - Extended the server to handle (most) other MIME types.
    - I've also added a system to keep track of allocated memory so that in the case of early termination etc. it's all cleaned up prior to exit.
    - Compiled everything with pedantic and made sure it's all compiling properly with the Makefile I've provided.
- 27th October
    - I've started doing some more extensive testing of the server itself - there are a few cases that caused it to crash (such as trying to access a non-servable (or non-existent) resource) - I've now sorted these out and the server seems relatively robust now.
    - If I get chance, my next aim is to implement HEAD requests as these aren't too different to GET requests.