Linsey Szabo
Computer Networks
Fall 2023

Project 4 Design Document

Server Design, Model, and Components

To make my HTTP server, I used the typical server design where the server is always listening for a client and the client initiates communication. To establish this in code, my server is continuously listening for incoming connections, and upon establishing such a connection, it spawns a thread to correctly respond to that client.

To respond, my server first parses the request. Through string methods, it determines the file of interest and a given “Range” parameter if it’s been provided. Next, it determines which of the following codes—“OK”, “Forbidden”, “Partial Content”, “Not Found”—is applicable to a particular request. This is accomplished by extracting the given file and searching for it under the “content” directory. Then, the server builds the correct response.

To make the following status codes easy to accomplish, I made macro variables pertaining to reused headers like date, content-length, content-range, and so on. I also made a dictionary, mapping possible file extensions to their content type according to the IANA official list of media types.

Forbidden

This case pertained to a filename with “confidential” in it. This case only included the content type and HTTP header (e.g. HTTP/1.1 Forbidden) followed by the HTML to display the forbidden status code. As with the rest of the status codes, I ensured that each header ended with a carriage return and that I added an additional carriage return before the HTML code. Lastly, the entire message was encoded and sent to the client.

OK

This status code pertained to the case that the file existed in the “content” folder and was less than our designated chunk size of 5 MB. Additional headers like content length, content type, connection keep-alive, and last-modified were included here. For last-modified, I hardcoded it to be a date in the past. I appended the file content in bytes onto the encoded headers and sent this to the client.

Not Found

This is the case where the filename cannot be found in the “content” folder. This is very similar to the Forbidden case. I followed the same format just encoding HTML code that displayed the Not Found status code instead.

Partial Content
This case is very similar to OK. However, we must send the file in multiple chunks because it exceeds our chunk size of 5 MB. It just takes into account that we need to use the "Range" header if it was provided in the HTTP request. If no "Range" header is provided, I send the first 5 MB of the file. If the header is provided, the start index is given and extracted when the request is parsed. Using this range index, I called f.seek() to navigate to that part of the correct file and send 5 MB (or however much of the file is left) to the client. Because we are only sending parts of a file in this case, it was very important to create a correct content-range header that contained the start and end indices in bytes of the file chunk being sent. Each of the indices was inclusive.

Handling Multiple Clients
I varied the socket.listen() function and used threads to process requests concurrently rather than sequentially.

By changing the socket.listen() parameter, I enabled my server to listen up to 1000 clients. Therefore, I think this is how many clients my server could handle correctly.

By using threads, each client has its own process running continuously processing requests all at the same time. This concurrency greatly sped up my implementation and enabled me to pass the concurrency stress tests.

Libraries Used
- socket: implementing client-server connections
- sys: system calls like getting the command line arguments and the size of a file
- threading: creating threads for each client-server connection
- os: getting size of files, checking if files exist under a directory
- time: getting time according to the RFC-1123 format