Project 1: Project 1: Web Server Implementation using BSD Sockets

Christopher Thai 804595489

Ning Hu 904751747

COM SCI 118 Spring 2018

Professor: Songwu Lu

Introduction

This project explored HTTP requests and helped us understand how they worked behind the scenes. We analyzed a browser's HTTP requests, constructed properly formatted responses, and provided the requested files for a single client.

Server Design

We used two functions: one to parse the HTTP request and one to set up the server.

The code to set up the server was provided. In order to make the server persistent, a perpetual while loop was implemented that repeatedly searched for connections with a client over localhost. The server would then serve the client web browser's HTTP request and then close the socket. However, a new socket is then created that waits to service another HTTP request from the same client. Inside the server function, the parseHeader function is called.

For the parseHeader function, the HTTP request string is passed as input. From the HTTP request, the only data we needed to retrieve was the request header which provides the name of the file being requested and the HTTP version. The extension of the file was also extracted for the Content-Type header line. An HTTP response header is constructed. After retrieving data from the request header, the requested file is checked for in the current directory with the dirent library. A case-insensitive string comparator was used to check if any file in the current directory matches the file being requested.

If the file is discovered, then a status code of 200 is attached to the response header, and the file's bytes are extracted into allocated memory. From the file, the Content-Length header line is also filled. Using the extension retrieved earlier, several if-else statements are used to create the proper Content-Type header line, giving proper content types to *.txt, *.jpeg, *.jpg, *.gif, *.htm, and *.html. If the file has a different extension, it is treated as a binary file. Finally, the response header, Content-Length line, Content-Type line, and file content are concatenated to each other and sent back over the socket to the client.

However, if the file is not discovered in the current directory, a 404 status code is attached to the response header, a Content-Length line is added, a Content-Type for HTMl is added, and HTML to display the header is attached and sent over the socket to display in the localhost client.

Difficulties

We were unfamiliar with string parsing and behavior in C, so some time was needed to understand how character pointers and character arrays could be concatenated and stored in a large enough chunk of memory for the entire HTTP response.

We were unaware that null bytes could be present in image files, so images originally did not work properly, and the HTTP response did not contain the number of bytes specified in the Content-Length header line. Once we discovered that this was due to null bytes in the contents of the image file, we decided to attach the file one byte at a time to the final output which solved the issue.

There was a persistent error where *.txt and *.htm files were being downloaded instead of displayed despite a specified Content-Type: text/plain header line. The HTTP response appeared to be accurate, and the server was functioning properly for all other files. We eventually determined that the error was due to a miscalculation where the chunk of allocated memory which was used to store the requested file was one byte greater than the actual file size. Similarly, *.txt files were being downloaded if they had binary content. To combat this, we added the X-Content-Type-Options: nosniff header line.

Project Manual

In the folder with server.c and Makefile, use the command “make” to create the executable. Once the executable has been created, the server can be run with the command: ./server *portno* where *portno* is a specified port number greater than 1024.

Sample Outputs

From the terminal, here are some HTTP requests and responses:

GET /test.jpg HTTP/1.1
Host: localhost:9000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie:
csrftoken=WLO18XeP51bYEpsU29Tk0c1mjQXGzkl7JoDVpmlt7cyYx0Ydz5BpIqAL97UOurTl;
sessionid=bukcgj1eudx9605b7je8c1avu73yo41u
DNT: 1

An image is displayed at localhost:9000/test.jpg in the browser.

GET /test.txt HTTP/1.1

Host: localhost:9000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie:
csrftoken=WLO18XeP51bYEpsU29Tk0c1mjQXGzkl7JoDVpmlt7cyYx0Ydz5BpIqAL97UOurTl;
sessionid=bukcgj1eudx9605b7je8c1avu73yo41u
DNT: 1

Sample HTTP response for the above HTTP request. In the browser, only the text “fdsjlfdjsakl” is displayed.
HTTP/1.1 200 OK
Content-Length: 13
Content-Type: text/plain

fdsjlfdjsakl

GET / HTTP/1.1
Host: localhost:9000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie:
csrftoken=WLO18XeP51bYEpsU29Tk0c1mjQXGzkl7JoDVpmlt7cyYx0Ydz5BpIqAL97UOurTl;
sessionid=bukcgj1eudx9605b7je8c1avu73yo41u
DNT: 1

Sample HTTP response for the above HTTP request. In the browser, HTML is displayed to let the user know that the file could not be found.
HTTP/1.1 404 Not Found
Content-Length: 309
Content-Type: text/html

<!DOCTYPE HTML><html><head><meta http-equiv='Content-Type' content='text/html;charset=utf-8'><title>Error response</title></head><body><h1>Error response</h1> Error code: 404 Message: File not found. Error code explanation: HTTPStatus.NOT_FOUND - Nothing matches the given URI. </body></html>

Conclusion

Through this project, we gained more insight as to how the internet functions behind the scenes and as to how a browser communicates with a server to receive the data it needs. It was also a useful application of material learned in lecture to solidify our understanding of client-server relationships over the internet.