Assignment 4: Simple HTTP Server

Computer Networks (CS-UH 3012) - Spring 2022

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

1. Any document and program code that you submit must be fully written by yourself.
2. You can discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. In other words, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit.
3. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution.
4. You are not allowed to possess solutions by someone from a different year or section, by someone from another university, or code from the Internet, etc.
5. There is never a valid reason to share your code with fellow students.
6. There is no valid reason to publish your code online in any form.
7. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g. if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi. More details can be found at: https://students.nyuad.nyu.edu/academics/registration/academic-policies/academic-integrity/

2 Assignment Goal

The goal of this assignment is to implement an HTTP/1.0 server using TCP socket programming and threading in C. This will give you practice in working with sockets, processes, and getting familiar with the HTTP protocol.

3 Task Description

The task of this assignment is to write a C program that replies to HTTP GET requests on TCP port 80. In contrast to the simple HTTP server discussed during the lab, the program should also serve an image, a javascript and a css file as well as responding with a 404 (Not Found)

response for objects that are not available. The server must implement concurrent HTTP/1.0 and hence the additional files (image, javascript and css file) will be served in parallel, i.e. the browser will open separate TCP connections for every request/object. That means you will need to use threading to independently handle the individual requests.
We will provide you with the required objects that show a web page as depicted in Figure 1.

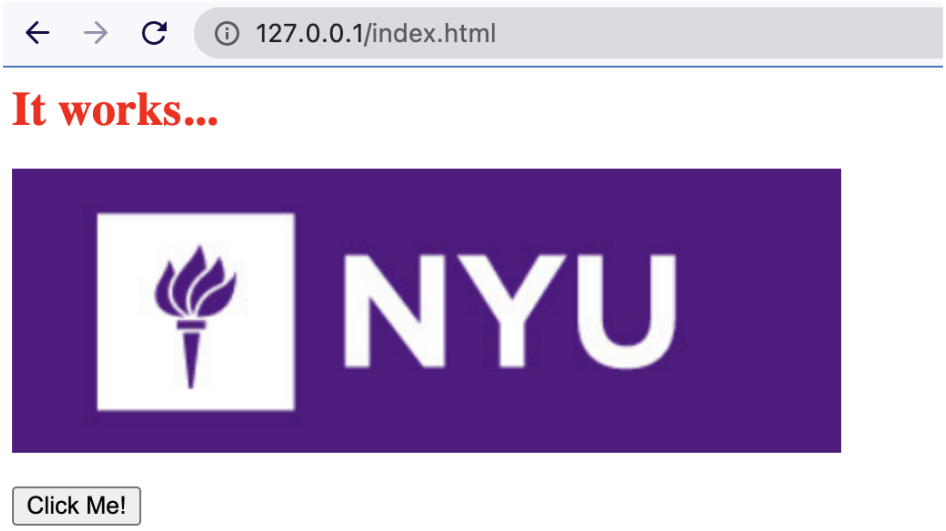

Figure 1: Example web page

When entering http://127.0.0.1/index.html into your browser, the browser will make an HTTP request to the index.html file. Upon receiving the index.html, the browser will make further concurrent requests to the additional files, such as style.css, script.js and logo.png. After every request and a successful response, the server must close the corresponding TCP connection. Requests to non-existing files, such as index.htm, main.php, logo.jpg, favicon.ico, etc. should return a 404 response (including a textual response in HTML indicating the file was not found) from the server. You can test this behavior by either modifying the URL to request non-existing files or simply renaming the file in your server folder.
Please note that common browsers will make an HTTP/1.1 instead of an HTTP/1.0 request. In this assignment, you can ignore this and treat HTTP/1.1 as HTTP/1.0.
Your server implementation should show some debug output, similar to the one shown below:

```
127.0.0.1 [13/Mar/2022 12:23:06] "GET /index.html HTTP/1.1" 200
127.0.0.1 [13/Mar/2022 12:23:08] "GET /script.js HTTP/1.1" 200
127.0.0.1 [13/Mar/2022 12:23:08] "GET /style.css HTTP/1.1" 200
127.0.0.1 [13/Mar/2022 12:23:10] "GET /logo.png HTTP/1.1" 200
```

4 HTTP Response Message Format

The HTTP server should reply to every GET request with an HTTP response, according to the HTTP/1.0 specification [RFC 1945]. For example, the response message format for the index.html file looks a shown below:

```
HTTP/1.0 200 OK
Server: SimpleHTTPServer
Date: Wed Mar 23 18:14:21 2022
Content-type: text/html
Content-Length: 352

<html>
   <head>
      <script
           src="script.js"
           type="text/JavaScript">
      </script>
      <link rel="stylesheet" href="style.css">
      <title>My Web page</title>
   </head>
   <body>
      <h1> This is a heading.</h1>
      <input type="button"
        onclick="Hello();"
        value="Click Me" />
      <img src="logo.png">
    </body>
</html>
```

Your HTTP server implementation must compose the HTTP header with the header fields as shown above. All other header fields can be ignored.
The content-type for the four objects are the following:

index.html	text/html
script.js	application/javascript
style.css	text/css
logo.png	image/png

5 Testing and Expected Output

You can use a browser (preferably Chrome) to test your implementation by requesting the index.html of the web page. Make sure you are requesting the web page using the URL described above and **not** the actual file path, e.g. file:///Users/tp53/http_server/index.html.
Helpful tools to investigate and verify the requests and responses are Wireshark and Chrome DevTools (right-click on a page and select Inspect, then select the Network tab). It is highly recommended that you tick the box to disable the cache, otherwise all files (image, javascript, css) will be loaded from the browser cache and not from the server. Also, it might be helpful to set the throttling to Slow 3G (dropdown menu next to Disable cache) to emulate a slow network.
If your program works as expected, the output should look like shown in Figure 2. Please note the concurrent requests for the javascript, css and image files.

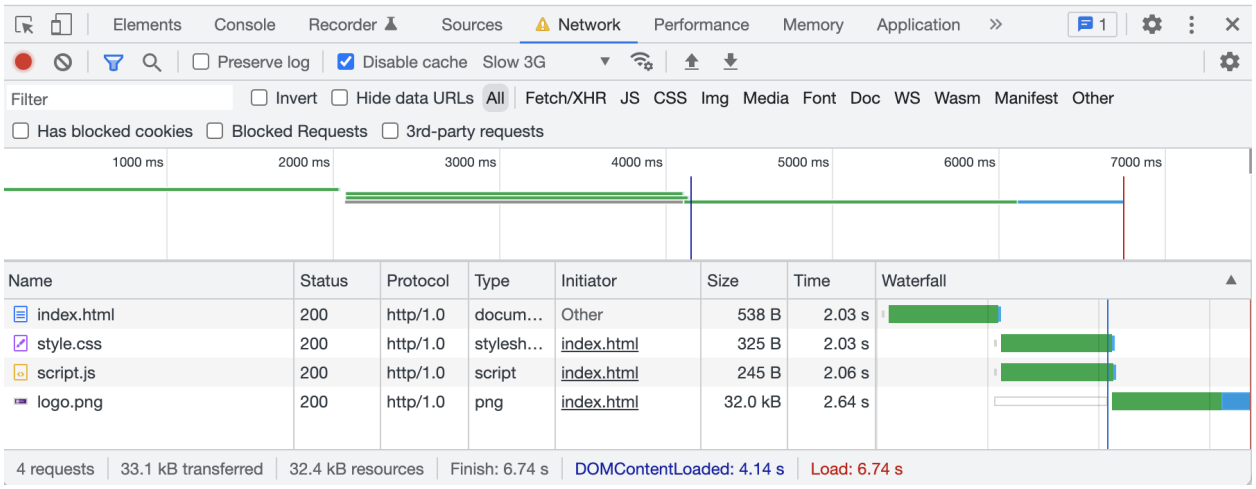

Figure 2: Chrome DevTools

You can also use Wireshark to verify the individual requests. For the example above, you should see four TCP streams in total, one for each file.
Before you start implementing the HTTP server in C, it might be helpful to investigate the requests and responses of an existing HTTP server implementation. You can use a simple Python HTTP/1.0 server and run it in the folder where the four files are stored by using the following command

```
python -m SimpleHTTPServer 80
```

or

```
python3 -m http.server 80
```

6 Grading

Description	Score (/8)
Successful compiling of the program using a Makefile	0.5
Opening and closing TCP connections for each object	0.5
Serving all objects as described in Section 3 and depicted in Figure 2	2.5
Adding correct HTTP headers as described in Section 4	1
Using threading to handle concurrent connections	1.5
Handling non-existing files (404 response)	1
Printing of events, as described in Section 3	0.5
Usage of meaningful comments and coding style	0.5

7 Submission Details and Policy

Submission Deadline: The deadline of this assignment is after 12 days of its release via Brightspace. No extensions will be given.

Submission Format and System: You can directly submit your files (C files and Makefile) as a zip file on Brightspace (https://brightspace.nyu.edu/). Due to technical limitations, submissions via email are not accepted.

Late Submissions: Late submissions will be penalized by 10% per 24 hours, with a maximum of 3 days late.