

Test Cases:

To get a file from my own server:

I ran the following command to start my http server.

thanlim@Thans-MBP server % ./myserver 5678



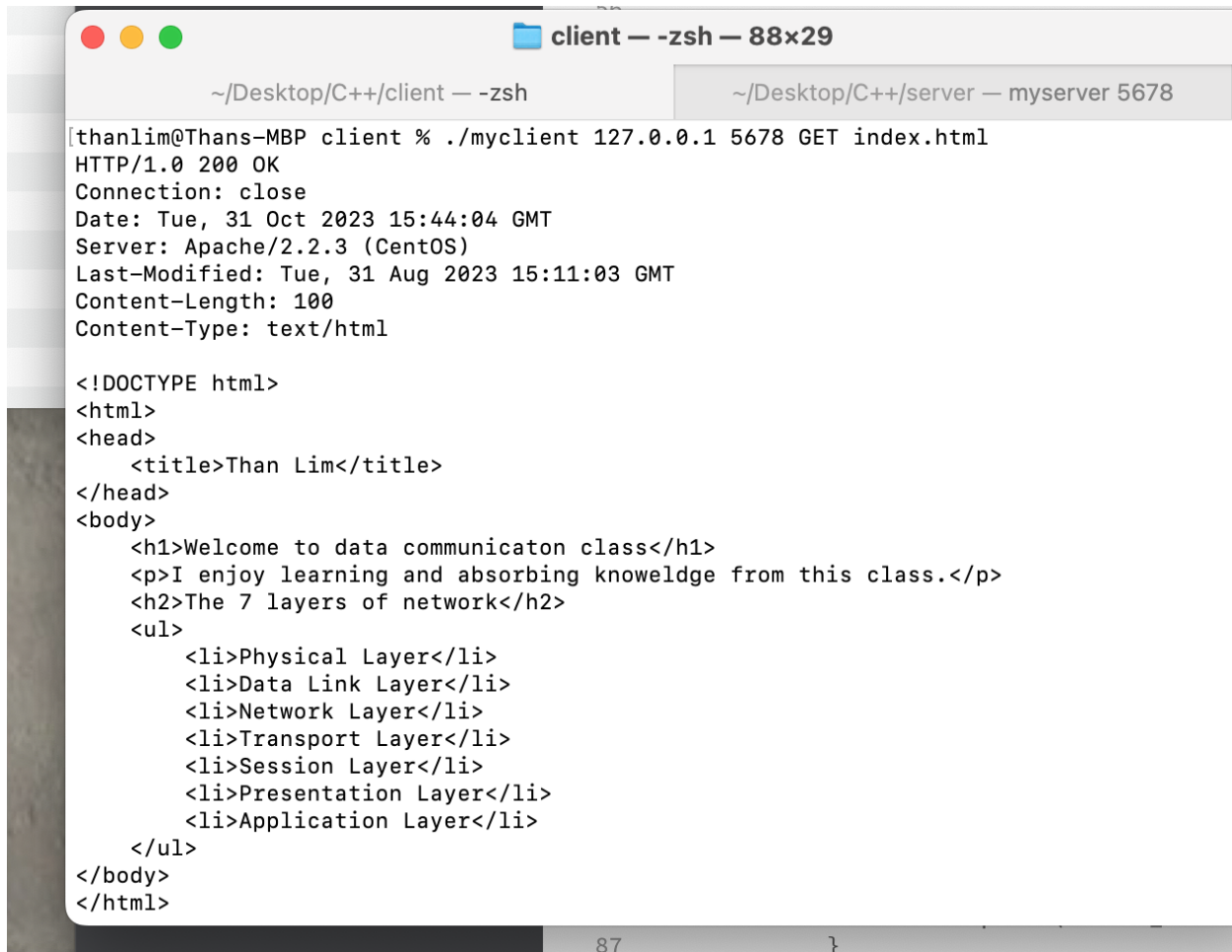
After implementing the command, the server was listening for incoming connection requests from a client.

I ran the following command to begin my http client.

NOTE: my http client takes IP address only (127.0.0.1 is my loopback address of my mackbook labtop).

thanlim@Thans-MBP client % ./myclient 127.0.0.1 5678 GET index.html

Than Lim
http programming assignment



A terminal window titled "client — -zsh — 88x29" is shown. The window has two tabs: "~/Desktop/C++/client — -zsh" (active) and "~/Desktop/C++/server — myserver 5678". The active tab shows the following output:

```
thanlim@Thans-MBP client % ./myclient 127.0.0.1 5678 GET index.html
HTTP/1.0 200 OK
Connection: close
Date: Tue, 31 Oct 2023 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 31 Aug 2023 15:11:03 GMT
Content-Length: 100
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
  <title>Than Lim</title>
</head>
<body>
  <h1>Welcome to data communicaton class</h1>
  <p>I enjoy learning and absorbing knoweldge from this class.</p>
  <h2>The 7 layers of network</h2>
  <ul>
    <li>Physical Layer</li>
    <li>Data Link Layer</li>
    <li>Network Layer</li>
    <li>Transport Layer</li>
    <li>Session Layer</li>
    <li>Presentation Layer</li>
    <li>Application Layer</li>
  </ul>
</body>
</html>
```

The terminal window has a status bar at the bottom showing "87" and a closing brace "}".

My http client was able to connect to the server and got a response for its request.

To get a file from a real server. For example, www.google.com (142.250.65.228)

Than Lim

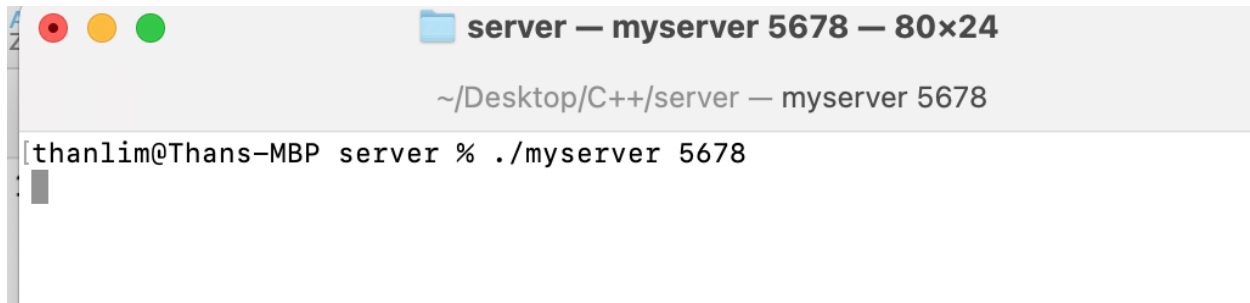
http programming assignment

```
[thanlim@Thans-MBP client % ./myclient 142.250.65.228 80 GET index.html
HTTP/1.0 200 OK
Date: Wed, 25 Oct 2023 18:09:30 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce
-SQ5_k7N7PM_N51CKsjCoEw' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline'
https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-10-25-18; expires=Fri, 24-Nov-2023 18:09:30 GMT; path=/; domain=
.google.com; Secure
Set-Cookie: NID=511=ShlBJbYhNSr7N-ddlXaWpI7QPjuXSouzP_QfsaHG1Nb_oD0k8UqAT-b_EwkCgqtB01g4
sSLp-MaLNThtlLlQORVhF--B9n60oPgFKcF3MzazuDs4rojDIEhiSaMjnRjGVGR7axVna168A_v0vTyY8IE7mf2vV
TehHK0zvUeGG8Zg; expires=Thu, 25-Apr-2024 18:09:30 GMT; path=/; domain=.google.com; Http
Only
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><
meta content="Search the world's information, including webpages, images, videos and mor
e. Google has many special features to help you find exactly what you're looking for." n
ame="description"><meta content="noodp" name="robots"><meta content="text/html; charset=
UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_stan
```

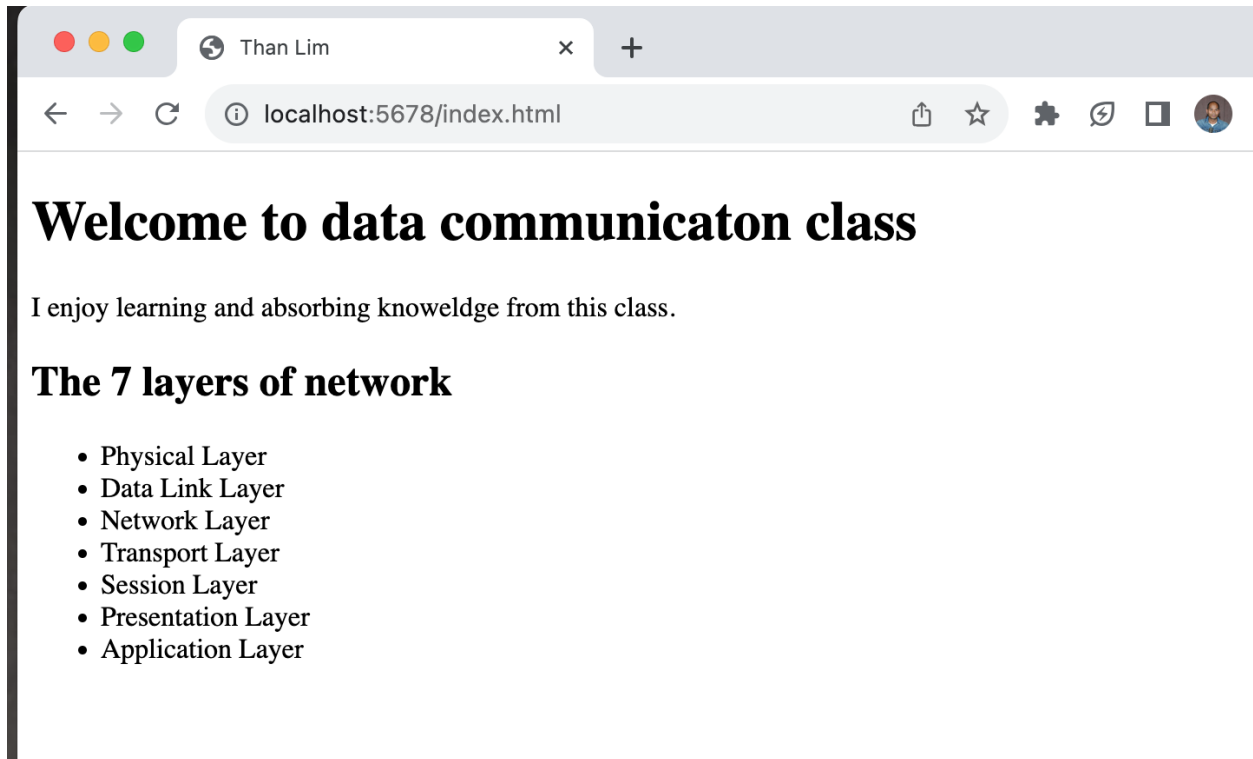
Using Chrome as my browser to get a html file from my server:

First, I ran my server.

A terminal window titled "server — myserver 5678 — 80x24" with a path of "~/Desktop/C++/server — myserver 5678". The prompt shows the user has run the command `./myserver 5678` successfully.

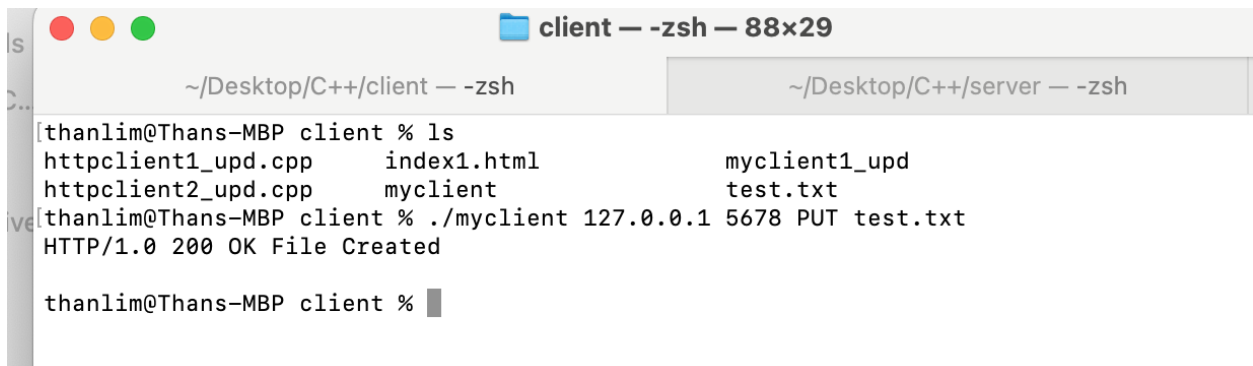
```
[thanlim@Thans-MBP server % ./myserver 5678
```

Second, I hopped on Chrome and entered the URL in the address bar and the html file is being displayed by the browser.



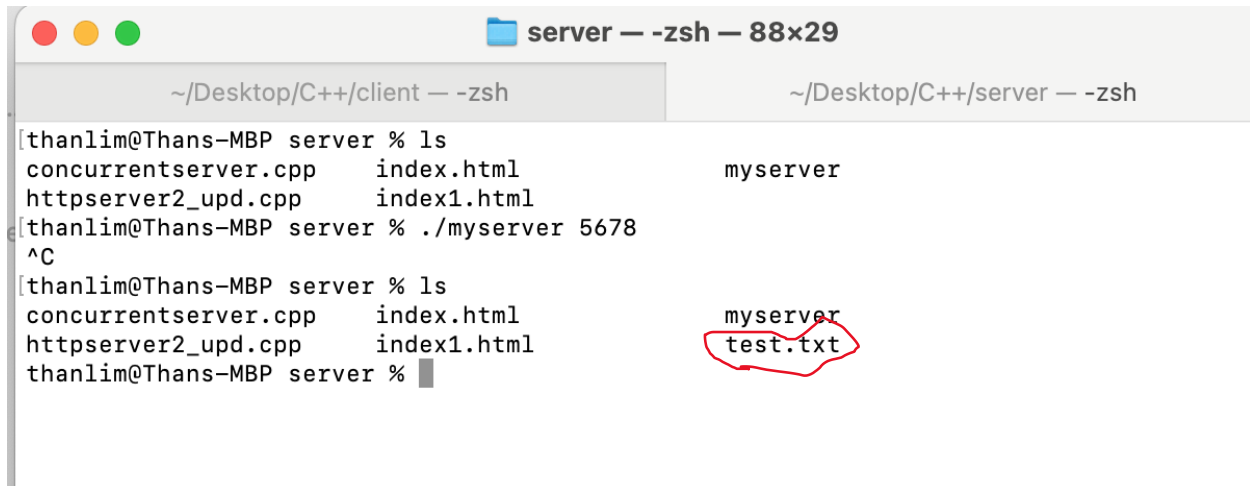
To put a file into my server, I ran the following command:

```
thanlim@Thans-MBP client % ./myclient 127.0.0.1 5678 PUT test.txt
```



The image shows that the file was successfully sent.

Than Lim
http programming assignment

A terminal window titled 'server — -zsh — 88x29' is shown. It has two tabs: '~/Desktop/C++/client — -zsh' and '~/Desktop/C++/server — -zsh'. The active tab is the client one. The terminal shows a user 'thanlim@Thans-MBP' in the 'server' directory. They run 'ls' and see 'concurrentserver.cpp', 'index.html', 'httpserver2_upd.cpp', and 'index1.html'. Then they run './myserver 5678' and press '^C'. They run 'ls' again and see the same files plus 'myserver' and 'test.txt', which is circled in red.

```
[thanlim@Thans-MBP server % ls
concurrentserver.cpp    index.html              myserver
httpserver2_upd.cpp    index1.html
[thanlim@Thans-MBP server % ./myserver 5678
^C
[thanlim@Thans-MBP server % ls
concurrentserver.cpp    index.html              myserver
httpserver2_upd.cpp    index1.html             test.txt
thanlim@Thans-MBP server %
```

and the server successfully created and stored the file.

Overall program design:

HTTP Client

Interface:

It is a command-line interface (CLI).

Request handling:

Requesting methods are GET and PUT.

Network communication:

Establish a network connection to the HTTP server through TCP. Send the HTTP request to the server and receive a response.

Response handling:

Parse the HTTP response, including the status code, headers, and response body.

User output:

Display the response data to the user in html format.

HTTP Server:

Request reception:

Listen for incoming HTTP requests on a specific port (5678).

Routing:

Determine how to route incoming requests to corresponding handlers, which are either GET request handler or PUT request handler.

Request parsing:

Parse the HTTP request, including the request method, headers, query parameters, and request body.

Response generation:

Generate an HTTP response, including the status code, headers, response body. Produce data into appropriate format such as HTML.

Response sending:

Send the HTTP response back to the client over the established connection.

Error handling:

Handle errors and exceptions gracefully, providing appropriate error response when necessary.

Security: (This is an improvement considered)

Than Lim
http programming assignment

Implement security measures to fight against common web attacks such as Cross-Site Scripting (XSS) and SQL injections.

Utilize encryption (HTTPS) to secure data that travels through network.

Concurrency: (This is an extension considered)

Handle multiple client connections concurrently (multi-threading or asynchronous programming) to ensure scalability.

A verbal description of how my programs work:

What are the steps that http client uses to connect to a server?

1. Get all the arguments needed.
2. Store those arguments in variables that will be conveniently used later in the program.
3. Create a socket for the client.
4. Define the server address.
5. Convert server host to IP address.
6. Create connection with server.
7. Prepare the HTTP GET request or HTTP PUT request.
8. Send the request to the server.
9. Receive and display the server's response.
10. Finally, close the connection.

What are the steps for the http server?

1. Get arguments from the command line.
2. Store those arguments in variables that will be conveniently used later in the program.
3. Create a socket for the server.
4. Bind the socket to the specified port (5678)
5. Start listening for incoming connections.
6. Accept client connection when requests are made.
7. Read the HTTP request. (handle differently for GET or PUT request)
8. Send response to client.
9. Close the client socket.
10. Close the server socket.

Tradeoffs, improvement, and extension:

When designing and building the HTTP client and server, I have considered what components the http client and http server programs should have. For http client, I must ask myself if I want to create http client with GUI incorporated or just with command line interface (CLI). To simplify and to be convenient for this assignment, I built http client and http server that ran through command lines. GUI is more friendly and convenient for any users mostly, but CLI could be intimidating for some. This is one of the tradeoffs I made. I went with CLI.

I was thinking of converting my server to be a concurrent server, but it is complex in term of coding and might use lots of resources. For these reasons, I decided to create a regular http server only. If my http

Than Lim
http programming assignment

server were created to simultaneously handle many requests at the same, it could be very beneficial and scalable. However, this is another tradeoff that I was willing to make.

A consideration of improvement is making my http client and server to support different version of HTTP protocols such as HTTP/1.1, HTTP/2, and HTTP/3. HTTP requests can be extended to include other methods such as POST and DELETE. Furthermore, HTTP can be improved to HTTPS.

Another improvement is to implement security measures, including input validation and sanitization (inputs from command line), preventing common web vulnerabilities such as cross-site scripting and SQL injection if http server associate database.