Cloud Computing
Project 1
Webserver

32185225
타미라
Dankook University
Free days left: 5

2021 Spring

Cloud Computing: Webserver
타미라 32185225

1. Project introduction

In this project, using socket programming, I will program a webserver that can handle HTTP 'GET' request. The requested file should be an HTML file. If no HTML file is available, a sample request "pink" will be provided by the program. The project supports multi-threading using *pthread* library.

2. Project goals

The goal of this project is to have a clear understanding of how a webserver is created with sockets and how webservers are able to return requests.

3. Concepts used

a) Basic socket functions are used such as socket(), bind(), listen(), accept(), and write(). These functions are the backbone of socket programming. Along with these, other socket operations are also used.

b) Parsing the requested file uses a variety of string manipulation functions such as strcpy(), strncpy(), and strcmp().

c) To open files when requested by client, file operations such as fopen(), fseek(), fread(), fclose() are used.

d) So that the server can serve more than one client, the program implements the *pthread* library.

e) Because the program implements multithreading, the *semaphore* library was also used to coordinate shared resources.

4. Program structure

1. *Set up sockets*
   Creating sockets and setting them up for the server with socket(), setsockopt(), bind(), and listen(). All set up to catch exceptions.

2. *Connecting with client*
   While the server accepts connection, operations will be done in the loop to write to client.

3. *Creating thread*
   Inside the 'connection' loop, thread is created, and connection handler function is invoked.

4. *Connection handler*

Inside the connection handler, get_webpage() function is called to get GET request.

5. *Parsing for request*
   Inside the get_webpage function, request will be parsed and written back to client.

5. Problems and solutions

At the start of the project, I encountered “Segmentation Fault” error a lot, and although I’ve used *malloc* before, after learning more about the error, I now understand the importance of it and hence got rid of the Segmentation Fault error.

But even though I don’t get segmentation fault anymore, another problem related to it I encountered is the “Incorrect checksum for freed object” which is because I used a pointer after it was being freed. I tried to free at different checkpoints, but I kept getting that error and the program ended up aborting. To solve this issue, I commented out all the free() functions. I know that this will produce a bug if the program is to be scaled up, since none of the memory allocated will be freed, but for the purpose of this project, since the data is lightweight, getting rid of most of the free() functions works.

The characteristics of HTTP to request for favicon.ico causes some unpredictable behaviors. From my observation, when requesting a page in the browser at the same time as the previous favicon request, the page will return the ‘error’ page. I suspect the error is because of the favicon request. All my attempts to catch this error and close the thread was unsuccessful, so the only solution is in the error page to let user know that they should try the request again after some time if they reach an error page unexpectedly.

This characteristic also leads to “Connected…” being printed multiple times in the console.

Cloud Computing: Webserver
타미라 32185225

6. Build environment

Compilation: Mac terminal with gcc

Instructions:

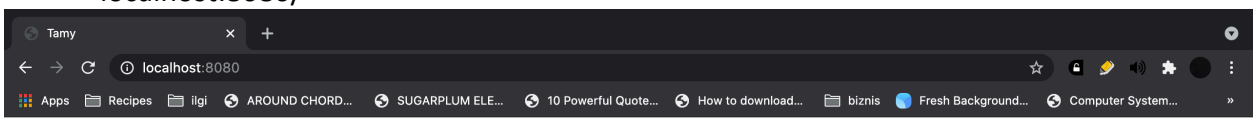
1. Open the project directory on local terminal.
2. Type to compile: `gcc main.c -lpthread -o webserver`
    a. You may encounter a warning during compilation, a function is deprecated, but the code will still accomplish the goal of this project.
3. Type to run: `./webserver` [make sure to run on local terminal]
4. Go to preferred browser, go to: localhost:8080
    a. You may request for 'localhost:8080/pink' which is provided by the program
    b. You may request for .html file in your local directory
    (NOTE: update PATH constant to local directory and make sure to have '/' at the very end [example: User/Tamyra/src/])
5. Lastly, to stop the program, [control+C] in terminal.

7. Screen capture

```
(base) Tamyui-MacBook-Pro:32185225_cloudcomp_webserver tamysiby$ gcc main.c -o ws -lpthread
main.c:49:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
    sem_init(&mutex, 0, 1);   //semaphore is for resource handling when multi-threading
    ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/semaphore.h:55:42: note: 'sem_init' has been explicitly marked deprecated here
int sem_init(sem_t *, int, unsigned int) __deprecated;
                                         ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h:187:40: note: expanded from macro '__deprecated'
#define __deprecated    __attribute__((__deprecated__))
                                       ^
1 warning generated.
(base) Tamyui-MacBook-Pro:32185225_cloudcomp_webserver tamysiby$ ./ws
Will listen to localhost:8080
Listening to port...
Connected.
Connected.
Connected.
Connected.
Connected.
Connected.
File Found.
Connected.
Connected.
```

Console

- localhost:8080/



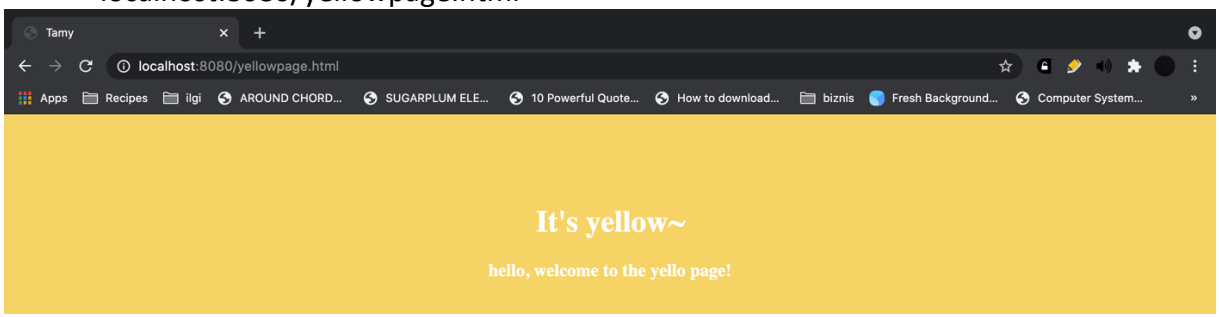
Hello World!

Default page

Cloud Computing: Webserver

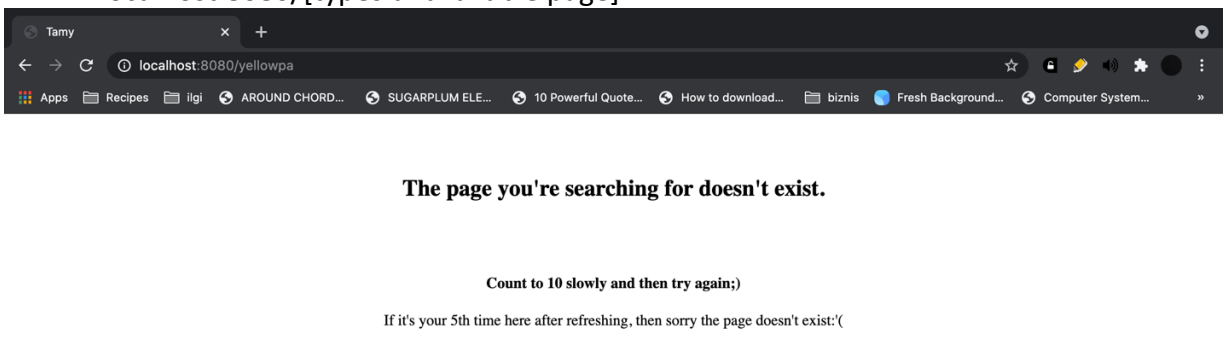
타미라 32185225

- localhost:8080/yellowpage.html



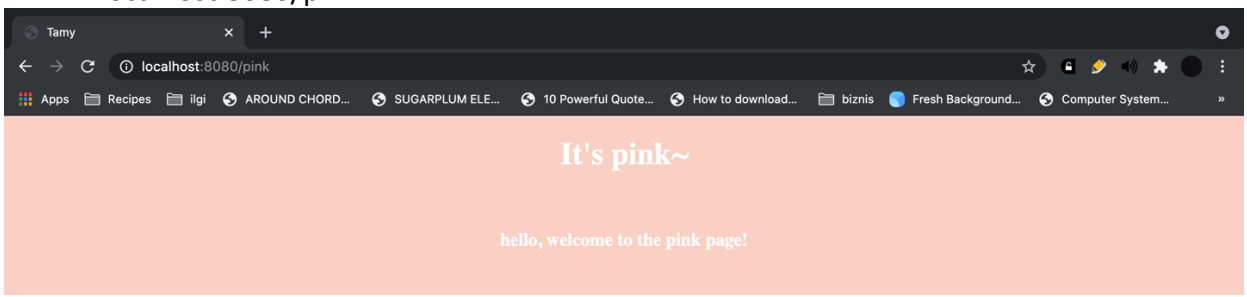
A page from a local .html file.

- localhost:8080/[types unavailable page]



Error page.

- localhost:8080/pink



Pink page is available from the program.

8. Personal comments

This project was another one that was very challenging for me because I have only ever done socket programming using high-level API, and it was not using C. This time, I have to work with everything under the hood, including parsing the GET request. I had to learn and grasp the concepts of webserver in the process of accomplishing the goal of this project. I now understand more of how webserver works, and I think the concepts I have picked up will be stepping stones into learning more about sockets and networks, which I think will be beneficial for me in the future.

I was very curious about making my page dynamic. I think it was a very interesting challenge, but I could not find resources for C that I could digest in the short time span provided to finish this project, so I ended up not implementing it. I wonder if there is another library that would help me make my page dynamic, or if it would just be logical implementations until the page is dynamic. I thought about making a page that would be in a loop and using time interval to refresh and get requests every time, but I think it would take some time for me to understand the time functionalities.

I would also learn more about the HTTP, because there are some behaviors that I only understand partly such as requesting the favicon, and all the other information given to the socket. I wonder how I could utilize the other information returned from the HTTP client.

Overall, through this project I was able to explore more and see all the programming tools available for me to accomplish a given task.