일디나 자스완 2021320033

COSE342- Computer Networs
Term Project 2023's Report

Process on implementing this web server.

1. Creating a socket
2. Bind
3. Listen
4. Accept
5. Read/Write

Before creating socket, it is necessary to extract files extensions from filename such as .html , .png , .js, .jpg and etc.

So, I created this function

```
const char* get_file_extension(const char* filename) {
    const char* dot = strrchr(filename, '.');
    if (!dot || dot == filename) {
        return "";
    }
    return dot + 1;
}

const char* get_content_type(const char* extension) {
    // Content type mapping
    if (strcmp(extension, "html") == 0 || strcmp(extension, "htm") == 0) {
        return "text/html; charset=UTF-8";
    } else if (strcmp(extension, "js") == 0) {
        return "application/javascript";
    } else if (strcmp(extension, "png") == 0) {
        return "image/png";
    } else if (strcmp(extension, "jpg") == 0 || strcmp(extension, "jpeg") == 0) {
        return "image/jpeg";
    } else {
        return "application/octet-stream";
    }
}
```

The code includes several standard C header files (stdio.h, stdlib.h, string.h, unistd.h, sys/types.h, sys/socket.h, netinet/in.h, arpa/inet.h, sys/select.h) to provide the necessary functions and data types for socket programming and file handling.
Also, The code defines two constants: BUFFER_SIZE (set to 1024) and MAX_CLIENTS (set to 10). These constants are used to define the size of the buffer for reading data from the sockets and the maximum number of concurrent client connections, respectively.

Function to serve file to clients
Since web server is the medium between clients and web contents, this is where everything begins.

```
FILE* file = fopen(filePath, "rb");
    if (file == NULL) {
        // File not found
        const char* notFoundResponse = "HTTP/1.1 404 Not Found\r\n"
                                       "Content-Length: 16\r\n"
                                       "Content-Type: text/plain\r\n"
                                       "\r\n"
                                       "File Not Found\r\n";
        write(clientSocket, notFoundResponse, strlen(notFoundResponse));
        return;
    }
```

This block is responsible when there is a problem where the file requested in not found.
I attempt of opening the file in binary hence `file = fopen(filePath, "rb");`

If the pointer returned is 'NULL' by 'fopen' , it means the file cant be find, doesn't exist, and to be assumed that the file is not found.

In a way, this block of codes works to tell when the requested file is not found.

Next ,

```
fseek(file, 0, SEEK_END);
    long fileSize = ftell(file);
    fseek(file, 0, SEEK_SET);

    char* fileContent = malloc(fileSize);
    if (fileContent == NULL) {
        fclose(file);
        perror("Error allocating memory for file content");
        exit(1);
    }
```

This block is responsible for determining the size of file and allocating memory to store the contents.

Next,

```
fread(fileContent, 1, fileSize, file);
    fclose(file);

    const char* extension = get_file_extension(filePath);
    const char* contentType = get_content_type(extension);
```

this code retrieves file content, store in the memory, get file extension and content type.

Next,

```
char response[BUFFER_SIZE];
    int responseLength = snprintf(response, BUFFER_SIZE,
        "HTTP/1.1 200 OK\r\n"
        "Content-Length: %ld\r\n"
        "Content-Type: %s\r\n"
        "\r\n",
        fileSize, contentType);

    write(clientSocket, response, responseLength);
    write(clientSocket, fileContent, fileSize);

    free(fileContent);
```

this block of code construct and sends HTTP response header to clients, write file content to clients' socket, free the memory used for file content.

These few steps are very important to serve file's content to the client in a proper HTTP response format.

Now, lets follow this process from building socket, binding, listen, accept, read/write.

1. Building socket

```
int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket < 0) {
        perror("Error in socket creation");
        exit(1);
    }

    struct sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port);
```

socket() function is used with IPv4 and socket type TCP to create a server socket. When the server socket creation fails, error message will be printed out and program will exit with non-zero status code.

It define a server address structure type 'struct sockaddr_in' and initialize its member.

This block sets up server socket and initialize the server address structure for accepting incoming connections.

2. Binding

```
// Bind the server socket to the specified port
    if (bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress))
< 0) {
        perror("Error in binding");
        exit(1);
    }
```

This code bind the server socket to the specified port and ip address. If binding process fails, error message will be printed by perror().

Binding a socket to a specific port and ip address allow the server to listen for incoming connections on that address.

3. Listen

```
if (listen(serverSocket, 5) < 0) {
        perror("Error in listen");
        exit(1);
    }

    fd_set readfds;
    int maxfd;
    int clientSockets[MAX_CLIENTS];
    memset(clientSockets, 0, sizeof(clientSockets));

    printf("Server listening on port %d...\n", port);
```

Here the server socket start listening for incoming connections. The value returned is less than 0, it'll be error in listening.

If the serving is listening to a specified port successfully, for an example port 8080, it'll look like this.

```
irdinajaswan@Dinas-MacBook-Pro tp % gcc webserver.c -o webserver
irdinajaswan@Dinas-MacBook-Pro tp % ./webserver 8080 resources
Server listening on port 8080...
```

4 & 5. Accept and Read/Write

```
for (int i = 0; i < MAX_CLIENTS; i++) {
            int clientSocket = clientSockets[i];
            if (FD_ISSET(clientSocket, &readfds)) {
                char buffer[BUFFER_SIZE];
                ssize_t bytesRead = read(clientSocket, buffer, BUFFER_SIZE - 1);
                if (bytesRead < 0) {
                    perror("Error in reading from client");
                    exit(1);
                } else if (bytesRead == 0) {
                    // Client disconnected
                    printf("Client disconnected, socket fd is %d\n", clientSocket);
```

```
                    close(clientSocket);
                    clientSockets[i] = 0;
                } else {
                    buffer[bytesRead] = '\0';

                    char* requestLine = strtok(buffer, "\r\n");
                    char* requestMethod = strtok(requestLine, " \t");
                    char* requestPath = strtok(NULL, " \t");

                    if (requestPath != NULL) {
                        char filePath[BUFFER_SIZE];
                        snprintf(filePath, BUFFER_SIZE, "%s%s", directory,
requestPath);

                        printf("File path: %s\n", filePath);

                        if (strcmp(requestMethod, "GET") == 0) {
                            // Serve the requested file to the client
                            serve_file(clientSocket, filePath);
                        }
                    }
                }
            }
        }
```

Lastly in this block, if there is no error in reading, code will not exit with error message.
If the client got disconnected, itll close the socket and label as inactive.

However, if it is not the case, it will parse the clients' requests to extract request method and path.
If "GET" request, and path is not empty, the code will find the file through the directory and request path.

Finally, it will serve to the client what the clients have requested.

API Description

Here, the API is described as a very simple HTTP server that serves files from a specific directory.
GET method can be used for clients to connect to the server and request files.

Whevener client request something, the endpoints : 'GET <file_path>'

For an example, with the provided 'resource' for the term project, when client request index.html,
This will come out.

```
New client connection, socket fd is 4, IP is : 127.0.0.1, port : 57301
File path: resources/index.html
File path: resources/script.js
New client connection, socket fd is 5, IP is : 127.0.0.1, port : 57302
File path: resources/gr-small.png
File path: resources/gr-large.jpg
Client disconnected, socket fd is 5
```

For the response Headers :

- ‘Content-Length’: the length of the response body in bytes
- ‘Content-type’: the MIME type of the file being served

Response code:

- ‘200 OK’: the file was found and successfully served
- ‘404 Not Found’: the requested file was not found on the server

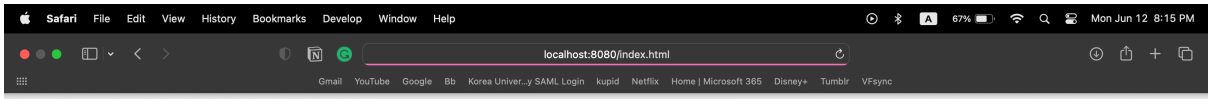
Result of webserver

```
irdinajaswan@Dinas-MacBook-Pro tp % gcc webserver.c -o webserver
irdinajaswan@Dinas-MacBook-Pro tp % ./webserver 8080 resources
Server listening on port 8080...
```

Before

After

I also managed to run my server in another environment (MacOS) through Safari



Before

localhost:8080/index.html
Gmail YouTube Google Bb Korea Univer...y SAML Login kupid Netflix Home | Microsoft 365 Disney+ Tumblr VFsync

After