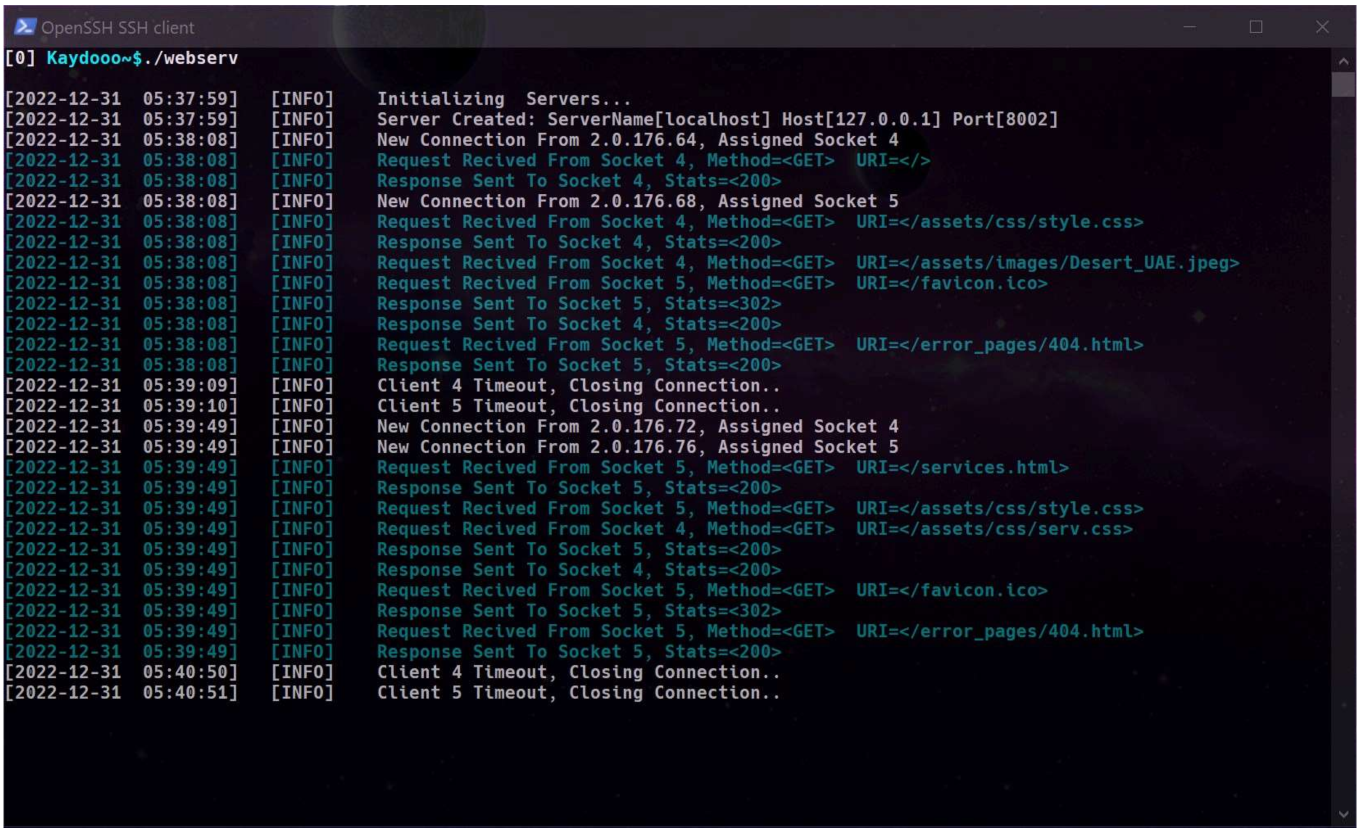
Kaydooo Update Response.hpp 5abfd8c · 3 years ago

cgi-bin	Delete user_database	3 years ago
configs	readme	3 years ago
docs/fusion_web	readme	3 years ago
inc	Update Response.hpp	3 years ago
src	logger update	3 years ago
subject_files	Logger	3 years ago
.gitignore	readme	3 years ago
Makefile	readme	3 years ago
README.md	logger update	3 years ago

README

Webserv: 42 School Project



```
[0] Kaydooo~$./webserv

[2022-12-31  05:37:59]   [INFO]    Initializing  Servers...
[2022-12-31  05:37:59]   [INFO]    Server Created: ServerName[localhost] Host[127.0.0.1] Port[8002]
[2022-12-31  05:38:08]   [INFO]    New Connection From 2.0.176.64, Assigned Socket 4
[2022-12-31  05:38:08]   [INFO]    Request Recived From Socket 4, Method=<GET>  URI=</>
[2022-12-31  05:38:08]   [INFO]    Response Sent To Socket 4, Stats=<200>
[2022-12-31  05:38:08]   [INFO]    New Connection From 2.0.176.68, Assigned Socket 5
[2022-12-31  05:38:08]   [INFO]    Request Recived From Socket 4, Method=<GET>  URI=</assets/css/style.css>
[2022-12-31  05:38:08]   [INFO]    Response Sent To Socket 4, Stats=<200>
[2022-12-31  05:38:08]   [INFO]    Request Recived From Socket 4, Method=<GET>  URI=</assets/images/Desert_UAE.jpeg>
[2022-12-31  05:38:08]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</favicon.ico>
[2022-12-31  05:38:08]   [INFO]    Response Sent To Socket 5, Stats=<302>
[2022-12-31  05:38:08]   [INFO]    Response Sent To Socket 4, Stats=<200>
[2022-12-31  05:38:08]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</error_pages/404.html>
[2022-12-31  05:38:08]   [INFO]    Response Sent To Socket 5, Stats=<200>
[2022-12-31  05:39:09]   [INFO]    Client 4 Timeout, Closing Connection..
[2022-12-31  05:39:10]   [INFO]    Client 5 Timeout, Closing Connection..
[2022-12-31  05:39:49]   [INFO]    New Connection From 2.0.176.72, Assigned Socket 4
[2022-12-31  05:39:49]   [INFO]    New Connection From 2.0.176.76, Assigned Socket 5
[2022-12-31  05:39:49]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</services.html>
[2022-12-31  05:39:49]   [INFO]    Response Sent To Socket 5, Stats=<200>
[2022-12-31  05:39:49]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</assets/css/style.css>
[2022-12-31  05:39:49]   [INFO]    Request Recived From Socket 4, Method=<GET>  URI=</assets/css/serv.css>
[2022-12-31  05:39:49]   [INFO]    Response Sent To Socket 5, Stats=<200>
[2022-12-31  05:39:49]   [INFO]    Response Sent To Socket 4, Stats=<200>
[2022-12-31  05:39:49]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</favicon.ico>
[2022-12-31  05:39:49]   [INFO]    Response Sent To Socket 5, Stats=<302>
[2022-12-31  05:39:49]   [INFO]    Request Recived From Socket 5, Method=<GET>  URI=</error_pages/404.html>
[2022-12-31  05:39:49]   [INFO]    Response Sent To Socket 5, Stats=<200>
[2022-12-31  05:40:50]   [INFO]    Client 4 Timeout, Closing Connection..
[2022-12-31  05:40:51]   [INFO]    Client 5 Timeout, Closing Connection..
```

About

This project was completed as part of a 42 core curriculum. it was done with Anastasiia-Ni & AhmadMHammoudeh

The goal of the project is to build a C++98 compatible HTTP web server from scratch. The web server can handle HTTP GET, HEAD, POST, PUT, and DELETE Requests, and can serve static files from a specified root directory or dynamic content using CGI. It is also able to handle multiple client connections concurrently with the help of select().

Usage

```
make
./webserv [Config File] ## leave empty to use the default configuration.
```

Introduction

HTTP (Hypertext Transfer Protocol) is a protocol for sending and receiving information over the internet. It is the foundation of the World Wide Web and is used by web browsers and web servers to communicate with each other.

An HTTP web server is a software application that listens for and responds to HTTP requests from clients (such as web browsers). The main purpose of a web server is to host web content and make it available to users over the internet.

HTTP consists of requests and responses. When a client (such as a web browser) wants to retrieve a webpage from a server, it sends an HTTP request to the server. The server then processes the request and sends back an HTTP response.

HTTP Message Format

```
start-line CRLF
Headers CRLF
CRLF(end of headers)
[message-body]

CRLF are Carriage Return and Line Feed (\r\n), which is just a new line.
```

HTTP Message can be either a request or response.

HTTP Request

An HTTP request consists of a request line, headers, and an optional message body. Here is an example of an HTTP request:

```
GET /index.html HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

The request line consists of three parts: the method, the path, and the HTTP version. The method specifies the action that the client wants to perform, such as GET (to retrieve a resource) or POST (to submit data to the server). The path or URI specifies the location of the resource on the server. The HTTP version indicates the version of the HTTP protocol being used.

Headers contain additional information about the request, such as the hostname of the server, and the type of browser being used.

In the example above there was no message body because GET method usually doesn't include any body.

HTTP Response

An HTTP response also consists of a status line, headers, and an optional message body. Here is an example of an HTTP response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<Message Body>
```

The status line consists of three parts: the HTTP version, the status code, and the reason phrase. The status code indicates the result of the request, such as 200 OK (successful) or 404 Not Found (resource not found). The reason phrase is a short description of the status code. Following is a very brief summary of what a status code denotes:

`1xx` indicates an informational message only

`2xx` indicates success of some kind

`3xx` redirects the client to another URL

`4xx` indicates an error on the client's part

`5xx` indicates an error on the server's part

Headers contain additional information about the response, such as the type and size of the content being returned. The message body contains the actual content of the response, such as the HTML code for a webpage.

HTTP Methods

Method	Description	Possible Body
`GET`	Retrieve a specific resource or a collection of resources, should not affect the data/resource	No
`POST`	Perform resource-specific processing on the request content	Yes
`DELETE`	Removes target resource given by a URI	Yes
`PUT`	Creates a new resource with data from message body, if resource already exist, update it with data in body	Yes
`HEAD`	Same as GET, but do not transfer the response content	No

GET

HTTP GET method is used to read (or retrieve) a representation of a resource. In case of success (or non-error), GET returns a representation of the resource in response body and HTTP response status code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

POST

HTTP POST method is most often utilized to create new resources. On successful creation, HTTP response code 201 (Created) is returned.

DELETE

HTTP DELETE is stright forward. It deletes a resource specified in URI. On successful deletion, it returns HTTP response status code 204 (No Content).

Read more about HTTP methods RFC9110#9.1

Parts of a web server

A basic HTTP web server consists of several components that work together to receive and process HTTP requests from clients and send back responses. Below are the main parts of our webserver.

Server Core

The networking part of a web server that handles TCP connections and performs tasks such as listening for incoming requests and sending back responses. It is responsible for the low-level networking tasks of the web server, such as creating and managing sockets, handling input and output streams, and managing the flow of data between the server and clients.

Before writing your webserver, I would recommend reading this awesome guide on building simple TCP client/server in C as it will help you get a good understanding of how TCP works in C/C++. also you would need to understand I/O multiplexing, this video will help you grasp the main idea of select().

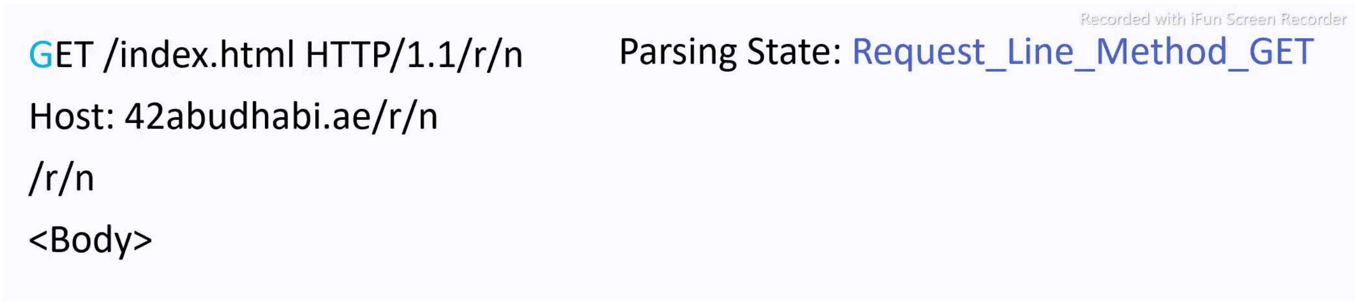
The I/O Multiplexing process in our web server is summarized in the flowchart below. (CGI is not included in the flowchart but may be added in the future)

I/O_Multiplexing

Request Parser

The parsing part of a web server refers to the process that is responsible for interpreting and extracting information from HTTP requests. In this web server, the parsing of requests is performed by the HttpRequest class. An HttpRequest object receives an incoming request, parses it, and extracts the relevant information such as the method, path, headers, and message body(if present). If any syntax error was found in the request during parsing, error flags are set and parsing stops. Request can be fed to the object through the method feed() either fully or partially, this is possible because the parser scans the request byte at a time and update the parsing state whenever needed. The same way of parsing is used by Nginx and Nodejs request parsers.

below is an overview of how the parser works.



GET /index.html HTTP/1.1/r/n
Host: 42abudhabi.ae/r/n
/r/n
<Body>

Parsing State: Request_Line_Method_GET

Response Builder

The response builder is responsible for constructing and formatting the HTTP responses that are sent back to clients in response to their requests. In this web server, the Response class is responsible for building and storing the HTTP response, including the status line, headers, and message body. The response builder may also perform tasks such as setting the appropriate status code and reason phrase based on the result of the request, adding headers to the response to provide additional information about the content or the server, and formatting the message body according to the content type and encoding of the response. For example, if the server receives a request for a webpage from a client, the server will parse the request and pass it to a Response object which will fetch the contents of the webpage and construct the HTTP response with the HTML content in the message body and the appropriate headers, such as the Content-Type and Content-Length headers.

Configuration File

Configuration file is a text file that contains various settings and directives that dictate how the web server should operate. These settings can include things like the port number that the web server should listen on, the location of the web server's root directory, and many other settings.

Here is an example fie that shows config file format and supported directives.

```
server {
  listen 8001;                        # listening port, mandatory parameter
  host 127.0.0.1;                     # host or 127.0.0.1 by default
  server_name test;                   # specify server_name, need to be added into /etc/hosts to work
  error_page 404 /error/404.html;     # default error page
  client_max_body_size 1024;          # max request body size in bytes
  root docs/fusion_web/;              # root folder of site directory, full or relative path, mandatory parameter
  index index.html;                   # default page when requesting a directory, index.html by default

  location /tours {
      root docs/fusion_web;           # root folder of the location, if not specified, taken from the server.
                                      # EX: - URI /tours           --> docs/fusion_web/tours
                                      #     - URI /tours/page.html --> docs/fusion_web/tours/page.html
      autoindex on;                   # turn on/off directory listing
      allow_methods POST GET;         # allowed methods in location, GET only by default
      index index.html;               # default page when requesting a directory, copies root index by default
      return abc/index1.html;         # redirection
      alias  docs/fusion_web;         # replaces location part of URI.
                                      # EX: - URI /tours           --> docs/fusion_web
```

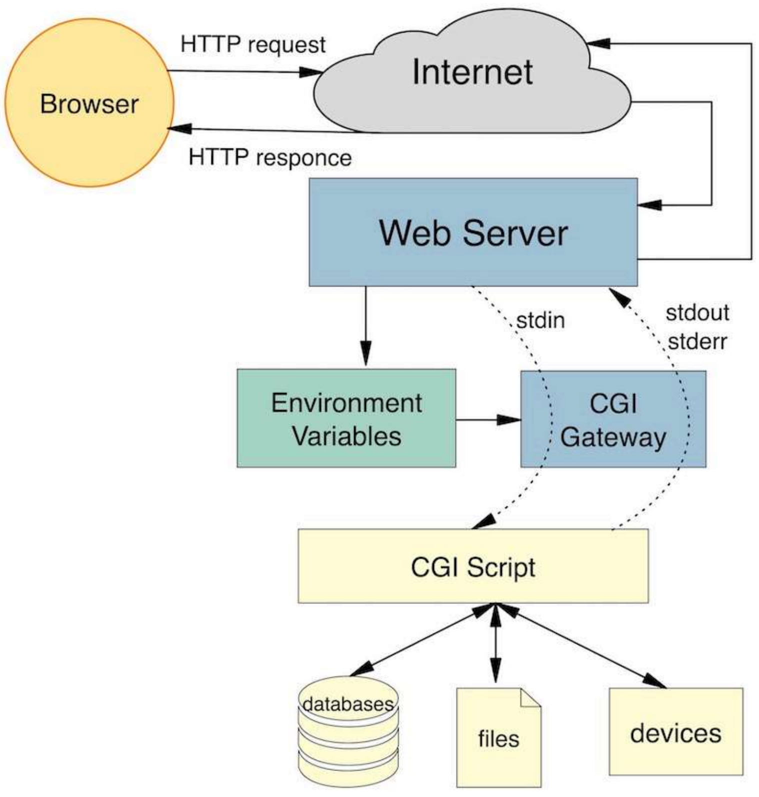
```
                                            #     - URI /tours/page.html --> docs/fusion_web/page.html
    }

    location cgi-bin {
        root ./;                                            # cgi-bin location, mandatory parameter
        cgi_path /usr/bin/python3 /bin/bash;                # location of interpreters installed on the current system, mand
        cgi_ext .py .sh;                                    # extensions for executable files, mandatory parameter
    }
  }
```

CGI

CGI is a standard for running external programs from a web server. When a user requests a web page that should be handled by a CGI program, the web server executes the program and returns the output to the user's web browser.

CGI programs are simply scripts that can be written in any programming language, such as Perl, Python, or bash, and are typically used to process data submitted by a user through a web browser, or to generate dynamic content on a web page.



Resources

Networking

- Create a simple HTTP server in c
- (Video) Create a simple web server in c
- (Video) explaining select()
- IBM - Nonblocking I/O and select()
- All about sockets blocking
- TCP Socket Programming: HTTP
- Beej's Guide to Network Programming

HTTP

- MDN - HTTP
- An Overview of the HTTP as Coverd in RFCs
- How the web works: HTTP and CGI explained
- MIME
- HTTP Status Codes

RFC

- How to Read an RFC
- RFC 9110 - HTTP Semantics
- RFC 9112 - HTTP/1.1
- RFC 2068 - ABNF
- RFC 3986 - (URI) Generic Syntax
- RFC 6265 - HTTP State Management Mechanism (Cookies)
- RFC 3875 - CGI

CGI

- Python web Programming
- CPP web Programming
- (Video) Creating a file upload page

StackOverFlow

- What HTTP response headers are required
- Why do we cast sockaddr_in to sockaddr when calling bind()
- Is an entity body allowed for an HTTP DELETE request?
- Sending images over http to browser in C
- Handling whitespaces in http headers

Tools

- Postman : Send custom requests to the server
- PuTTY : Send raw data to the server (Windows Only)
  - Video: How to use
- Wireshark : Capture request/response traffic
- Sige : Load testing

Other

Releases

No releases published

Packages

No packages published

Languages

● C++ 92.0% ● Python 6.9% ● Other 1.1%