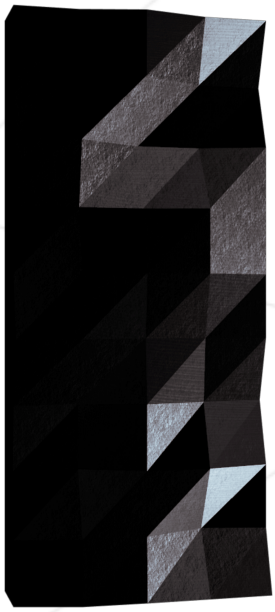

Webserv

This is when you finally understand why a url starts with HTTP

Summary: This project is here to make you write your own HTTP server. You will follow the real HTTP RFC and you will be able to test it with a real browser. HTTP is one of the most used protocol on internet. Knowing its arcane will be useful, even if you won't be working on a website.

Contents

Chapter I

Introduction

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.
HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.
HTTP was developed to facilitate hypertext and the World Wide Web.
The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP).

Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to the text content.

Multiple web servers may be used for a high traffic website.

A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Chapter II

Mandatory part

Program name	`webserv`
Turn in files	
Makefile	`Yes`
Arguments	
External functs.	`malloc, free, write, open, read, close, fork, wait, waitpid, wait3, wait4, signal, kill, exit, getcwd, chdir, stat, lstat, fstat, lseek, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, errno, gettimeofday, strptime, strftime select, socket, accept, listen, send, recv, bind, inet_addr, setsockopt, getsockname`
Libft authorized	`Yes`
Description	`Write a HTTP server in C++`

- You must write a HTTP server in C++
- It must be fully HTTP 1.1 compliant (rfc 2616)
- We will consider that nginx is HTTP 1.1 compliant and may be use to compare headers and answer behaviors
- It must be non blocking and use only 1 select for all the IO between the client and the server (listens includes)
- You should never read or write without going through select
- a request to your server should never hang for ever
- You server should have default error pages if none are provided
- Your program should not leak and should never crash
- You can't use fork for something else than cgi (like php or perl or ruby etc...)
- You can include and use everything in "iostream" "string" "vector" "list" "queue" "stack" "map" "algorithm"

- You can use the openssl library
- You can use the zlib library
- Your program should have a config file in argument or use a default path
- In this config file we should be able to:

You should inspire yourself from the "server" part of nginx configuration file

  - choose the port and host of each "server"
  - setup TLS
  - setup default error pages
  - setup GZIP compression or not
  - setup the server_name or not
  - limit client body size
  - setup routes with one or multiple of the following rules/configuration (routes wont be using regexp):
    - define a list of accepted HTTP Methods for the route
    - define a directory from where the file should be search
    - turn on or off directory listing
    - default file to answer if the request is a directory
    - execute php cli to resolve .php files
    - make the route able to accept uploaded files and configure where it should be saved

You should provide some configuration files for evaluation

If you've got a question about one behavior, you should compare your program behavior with nginx. For example check how server_name works...

Please read the RFC and do some test with telnet and nginx before starting this project.

Chapter III

Bonus part

- If the Mandatory part is not perfect don't even think about bonuses
- You don't need to do all the bonuses
- Make pluggins loadable/unloadable through terminal, like other compression system, charset convertor and so on... (repeatable bonus)
- Your program can have workers define as:
  - a worker can be either processes or threads (and use fork for them)
  - a worker should not be spawn for each client and must able to take care of an infinite number of requests
  - workers are not mandatory
- Add any number of the following in the configuration file:
  - choose a number of worker (if your program implements workers)
  - Configure plugins (works with pluggins see above)
  - Make routes work with regexp
  - Configure a proxy to an other http/https server
  - Use an internal module for php or any other language. it means you aren't calling any external executable to generate a page with this language. (repeatable bonus)