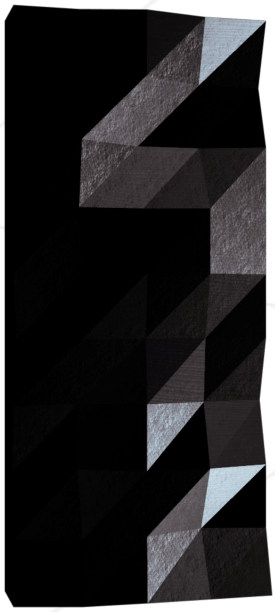

Webserv

This is when you finally understand why URLs start with HTTP

Summary:
This project is about writing your own HTTP server.
You will be able to test it with an actual browser.
HTTP is one of the most widely used protocols on the internet.
Understanding its intricacies will be useful, even if you won't be working on a website.

Version: 23.1

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 clicking a mouse button or tapping the screen on a web browser.

HTTP was developed to support hypertext functionality and the growth of the World Wide Web.

The primary function of a web server is to store, process, and deliver web pages to clients. Client-server communication occurs through 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, splitting traffic between multiple physical machines.

A user agent, commonly a web browser or web crawler, initiates communication by requesting 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 storage, or the result of a program. But this is not always the case and can actually be many other things.

Although its primary function is to serve content, HTTP also enables clients to send data. This feature is used for submitting web forms, including the uploading of files.

Chapter II

General rules

- Your program must not crash under any circumstances (even if it runs out of memory) or terminate unexpectedly.
  If this occurs, your project will be considered non-functional and your grade will be `0`.
- You must submit a `Makefile` that compiles your source files. It must not perform unnecessary relinking.
- Your `Makefile` must at least contain the rules:
  `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code must comply with the **C++ 98 standard** and should still compile when adding the flag `-std=c++98`.
- Make sure to leverage as many `C++` features as possible (e.g., choose `<cstring>` over `<string.h>`). You are allowed to use `C` functions, but always prefer their `C++` versions if possible.
- Any external library and `Boost` libraries are forbidden.

Chapter III

AI Instructions

Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

Main message

- Use AI to reduce repetitive or tedious tasks.
- Develop prompting skills — both coding and non-coding — that will benefit your future career.
- Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- Continue building both technical and power skills by working with your peers.
- Only use AI-generated content that you fully understand and can take responsibility for.

Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

> I ask AI: "How do I test a sorting function?" It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

> I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

> I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

> I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter IV

Mandatory part

Program Name	webserv
Files to Submit	Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, configuration files
Makefile	NAME, all, clean, fclean, re
Arguments	[A configuration file]
External Function	All functionality must be implemented in C++ 98. execve, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair, htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, chdir, bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, close, read, write, waitpid, kill, signal, access, stat, open, opendir, readdir and closedir.
Libft authorized	n/a
Description	An HTTP server in C++ 98

You must write an HTTP server in C++ 98.

Your executable should be executed as follows:

```
./webserv [configuration file]
```

Even though poll() is mentioned in the subject and evaluation sheet, you can use any equivalent function such as select(), kqueue(), or epoll().

Please read the RFCs defining the HTTP protocol, and perform tests with telnet and NGINX before starting this project.
Although you are not required to implement the entire RFCs, reading it will help you develop the required features.
The HTTP 1.0 is suggested as a reference point, but not enforced.

IV.1 Requirements

- Your program must use a configuration file, provided as an argument on the command line, or available in a default path.
- You cannot `execve` another web server.
- Your server must remain non-blocking at all times and properly handle client disconnections when necessary.
- It must be non-blocking and use only **1** `poll()` (or equivalent) for all the I/O operations between the clients and the server (listen included).
- `poll()` (or equivalent) must monitor both reading and writing simultaneously.
- You must never do a read or a write operation without going through `poll()` (or equivalent).
- Checking the value of `errno` to adjust the server behaviour is strictly forbidden after performing a read or write operation.
- You are not required to use `poll()` (or an equivalent function) for regular disk files; `read()` and `write()` on them do not require readiness notifications.

`I/O that can wait for data (sockets, pipes/FIFOs, etc.) must be non-blocking and driven by a single poll() (or equivalent). Calling read/recv or write/send on these descriptors without prior readiness will result in a grade of 0. Regular disk files are exempt.`

- When using poll() or any equivalent call, you can use every associated macro or helper function (e.g., FD_SET for select()).
- A request to your server should never hang indefinitely.
- Your server must be compatible with standard **web browsers** of your choice.
- NGINX may be used to compare headers and answer behaviours (pay attention to differences between HTTP versions).
- Your HTTP response status codes must be accurate.
- Your server must have **default error pages** if none are provided.
- You can't use fork for anything other than CGI (like PHP, or Python, and so forth).
- You must be able to **serve a fully static website**.
- Clients must be able to **upload files**.
- You need at least the `GET`, `POST`, and `DELETE` methods.

- Stress test your server to ensure it remains available at all times.
- Your server must be able to listen to multiple ports to deliver different content (see *Configuration file*).

We deliberately chose to offer only a subset of the HTTP RFC. In this context, the virtual host feature is considered out of scope. But you are allowed to implement it if you want.

IV.2 For MacOS only

Since macOS handles write() differently from other Unix-based OSes, you are allowed to use fcntl().
You must use file descriptors in non-blocking mode to achieve behaviour similar to that of other Unix OSes.

However, you are allowed to use fcntl() only with the following flags:
F_SETFL, O_NONBLOCK and, FD_CLOEXEC.
Any other flag is forbidden.

IV.3 Configuration file

You can take inspiration from the 'server' section of the NGINX configuration file.

In the configuration file, you should be able to:

- Define all the interface:port pairs on which your server will listen to (defining multiple websites served by your program).
- Set up default error pages.
- Set the maximum allowed size for client request bodies.
- Specify rules or configurations on a URL/route (no regex required here), for a website, among the following:
  - List of accepted HTTP methods for the route.
  - HTTP redirection.
  - Directory where the requested file should be located (e.g., if URL `/kapouet` is rooted to `/tmp/www`, URL `/kapouet/pouic/toto/pouet` will search for `/tmp/www/pouic/toto/pouet`).
  - Enabling or disabling directory listing.
  - Default file to serve when the requested resource is a directory.
  - Uploading files from the clients to the server is authorized, and storage location is provided.

- Execution of CGI, based on file extension (for example .php). Here are some specific remarks regarding CGIs:
  - Do you wonder what a CGI is?
  - Have a careful look at the environment variables involved in the web server-CGI communication. The full request and arguments provided by the client must be available to the CGI.
  - Just remember that, for chunked requests, your server needs to un-chunk them, the CGI will expect `EOF` as the end of the body.
  - The same applies to the output of the CGI. If no `content_length` is returned from the CGI, `EOF` will mark the end of the returned data.
  - The CGI should be run in the correct directory for relative path file access.
  - Your server should support at least one CGI (php-CGI, Python, and so forth).

You must provide configuration files and default files to test and demonstrate that every feature works during the evaluation.

You can have other rules or configuration information in your file (e.g., a server name for a website if you plan to implement virtual hosts).

If you have a question about a specific behaviour, you can compare your program's behaviour with NGINX's.
We have provided a small tester. Using it is not mandatory if everything works fine with your browser and tests, but it can help you find and fix bugs.

Resilience is key. Your server must remain operational at all times.

Do not test with only one program. Write your tests in a more suitable language, such as Python or Golang, among others, even in C or C++ if you prefer.

Chapter V

Bonus part

Here are some additional features you can implement:

- Support cookies and session management (provide simple examples).
- Handle multiple CGI types.

The bonus part will only be assessed if the mandatory part is fully completed without issues. If you fail to meet all the mandatory requirements, your bonus part will not be evaluated.

Chapter VI

Submission and peer-evaluation

Submit your assignment in your `Git` repository as usual. Only the content of your repository will be evaluated during the defense. Be sure to double-check the names of your files to ensure they are correct.

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.

16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA77916734D1
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C281404901890F
619D682524F5