



## 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: 22.1*

# Contents

<b>I</b>	<b>Introduction</b>	<b>2</b>
<b>II</b>	<b>General rules</b>	<b>3</b>
<b>III</b>	<b>Mandatory part</b>	<b>4</b>
III.1	Requirements . . . . .	6
III.2	For MacOS only . . . . .	8
III.3	Configuration file . . . . .	8
<b>IV</b>	<b>Bonus part</b>	<b>10</b>
<b>V</b>	<b>Submission and peer-evaluation</b>	<b>11</b>

# 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

## Mandatory part

<b>Program name</b>	webserv
<b>Turn in files</b>	Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, configuration files
<b>Makefile</b>	NAME, all, clean, fclean, re
<b>Arguments</b>	[A configuration file]
<b>External functs.</b>	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.
<b>Libft authorized</b>	n/a
<b>Description</b>	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.

## III.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 equivalent) before `read()` to retrieve your configuration file.



Because you have to use non-blocking file descriptors, it is possible to use `read/recv` or `write/send` functions with no `poll()` (or equivalent), and your server wouldn't be blocking. But it would consume more system resources. Thus, if you attempt to `read/recv` or `write/send` on any file descriptor without using `poll()` (or equivalent), your grade will be 0.

- 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.



## III.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.

## III.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 IV

## 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 V

## 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.



16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA77916734D1  
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C281404901890F  
619D682524F5