Project Overview
This project is a C++ implementation of an HTTP server called Webserv. The server supports multiple ports, handles various HTTP request methods, serves static files, functional CGI, supports file uploads, and is non-blocking using poll(). It is designed to be compliant with HTTP/1.1 standards and can be tested in an actual browser.

The project was developed by M. Haan, J. van der Laan, and C. ter Maat.

Features
- Multi-Port Support: The server can listen on multiple ports, as specified in the configuration file.
- Static Website Hosting: The server can serve static websites and handles GET, POST, DELETE requests.
- File Uploads: Users can upload files to the server.
- Non-Blocking I/O: The server uses poll() to monitor multiple client connections simultaneously without blocking the main server loop, allowing for efficient handling of concurrent client requests.
- CGI Support: The server supports CGI execution of various different files.
- HTTP 1.1 Compliance: The server supports HTTP 1.1 features like persistent connections and multipart requests.
- Default Error Pages: Customizable error pages for common HTTP status codes.

More information about the project and its requirements can be found as webserver_subject.pdf in the root of the git repository.

Components Developed By Chavert
- Dockerfile and Makefile: To build the project in a controlled environment.
- Main Program (main.cpp): Handles errors, parsing, server initialization, and the main loop.
- Signal Handling (Signals.cpp & Signals.hpp): Manages graceful shutdowns and signal catching.
- Client Connection (ClientConnection.cpp & ClientConnection.hpp): Manages individual client connections.
- Server Connection (ServerConnection.cpp & ServerConnection.hpp): Manages the server-side socket connection.
- Makefile: Provides build commands for compiling the project, including all, clean, fclean, re.
- Logging (log.cpp & log.hpp): Logs server activities and errors.

Docker

Docker is a platform for building, deploying, and running applications in containers. To download Docker, visit the official Docker website (https://www.docker.com/) and follow the instructions for your operating system.

Docker Makefile Targets

- Build the Docker Image: Run “make build” in the terminal to build the Docker image.
- Run the Docker Container: Run “make run” in the terminal to build the Docker image (if necessary) and start the container, exposing the specified ports.
- Stop the Docker Container: Run “make stop” in the terminal to stop and remove the Docker container.
- Remove the Docker Image: Run “make clean” in the terminal to remove the Docker image.
- Rebuild the Docker Image and Run the Container: Run “make re” in the terminal to stop the container, remove the image, rebuild the image.
- Clean everything: Run “make fclean” in the terminal to stop containers and remove images.
- Check the Docker Container Status: Run “make status” in the terminal to check if the Docker container is running.

Building the Project

1. In the root git repository, enter: make build
2. To enter the interactive shell, enter: make run

Running the Webserver

1. In the interactive shell, to compile the project, enter: make
2. To rebuild the the project enter: make re
3. To start the executable, enter: ./webserv basic_config.txt
4. Open a browser and enter: http://localhost:8080/index.html . Or use any other port then 8080 mentioned in the basic_config.txt.
5. Go nuts on the webpage!
6. Close the webserver by pressing "control + c".

Testing the Webserver for Leaks

Valgrind is a tool for detecting memory issues in programs, such as leaks and invalid accesses, by running them in a virtual environment that tracks memory use.

1. If not compiled, in the interactive shell, enter: make
2. To rebuild the the project enter: make re
3. To start the executable with Valgrind, enter: valgrind --leak-check=full ./webserv basic_config.txt
4. Open a browser and enter: http://localhost:8080/index.html . Or use any other port then 8080 mentioned in the basic_config.txt.
5. Go nuts on the webpage!
6. Close the webserver by pressing "control + c".

Stress Test Connection with Siege

Siege is a powerful HTTP load testing and benchmarking tool. It allows you to stress test the web server by simulating multiple users accessing it simultaneously. This is useful for evaluating the performance and stability of your server under heavy load.

1. If not compiled, in the interactive shell, enter: make
2. To rebuild the the project enter: make re
3. To start the executable with Valgrind, enter: valgrind --leak-check=full ./webserv basic_config.txt
4. To start Siege, open a new terminal and enter: siege -b -c250 -t2M http://localhost:8080/index.html
    - -b: Runs Siege in benchmark mode, where it does not delay between requests.
    - -c255: Simulates 255 concurrent users. 255 users is the max Siege can offer.
    - -t2M: Siege will run its test for 2 minutes.
    - http://localhost:8080/index.html: The URL of the resource on your server to be tested.
5. Inspect the Siege report after finishing.
6. Close the webserver by pressing "control + c".

Cleaning the Project

1. To exit the shell enter: exit
2. To clean the Webserv Docker files enter: make clean