Cuteserver: A web server written in C

Rahel Kempf Ephraim Siegfried

June 5, 2024

Introduction

”Web servers are everywhere”, Shakespeare once said. This is especially true today. But ”Hark! Dost the common folk comprehend what these web servers be? And doth they grasp the manner of their workings?”.
In this project we aspire to understand the inner workings of web servers. It is of interest for us, as we both recently set up our own home servers and came in contact with different tools such as nginx. Building our own web server helped us be more competent in using those tools. It also helped us deepen our knowledge of OS topics such as socket programming, thread/process creation & management and inter-process communication.
This project resulted in Cuteserver, a simple web server written in C. It can handle various HTTP/1.1 requests from multiple clients concurrently. It is configurable and supports hosting content for multiple domains and is thus similar to a reverse proxy. It can be hosted easily with a prepared docker image. In the following chapters, we will discuss the steps taken and the challenges we faced in this project.

Background

In this section we explain some concepts we had to understand implementing our project.

HTTP Protocol

CGI

Implementation

Development Environment

The development of Cuteserver was primarily accomplished through pair programming. We held regular meetings where we shared our terminals using a shell sharing tool called sshx. This tool was particularly effective for collaboration since we both used Neovim as our code editor.

The implementation of cuteserver was mainly done by pair programming. We regulary had meetings where we shared our terminals with a shell sharing tool called sshx. This tool was particularly useful for working together as we both use Neovim as a code editor which is terminal based.

Initially, we used Makefiles to build the source code, but this approach was inefficient due to frequent adjustments. As the project grew more complex, we decided to use CMake for more efficient project building and file generation.

Our implementation strategy for the web server was iterative, starting with small, manageable tasks and progressively tackling more complex ones. We regularly tested the code by running it with different inputs to ensure functionality.

Project Structure

Static File Requests

CGI

Server Configuration

Example Application

Containerization

We decided to containerize our application for two main reasons. Initially, we used chroot jails to sandbox our application, i.e., a mechanism that isolates a process and its children from the rest of the system by changing their apparent root directory to a specified path. This approach worked until we implemented CGI support, which required the web server to start new processes with the necessary libraries for the CGI scripts. Manually copying those dependencies into the chroot jail would have been too laborious. Docker containers offered an easier solution by securely sandboxing execution and including dependencies. Additionally, Dockerfiles enabled us to automate the building of the web server and the example application, making it easy to use and run. We utilized multistage builds, first creating containers for building and then copying only the necessary files to the final containers. Multistage build reduced the container size from 1.2 GB to approximately 200 MB. This process enhanced our understanding of containerization.

Results

Conclusion

We were successful in our project and comparing to our original plan we implemented everything but websockets. What we miscalculated a bit was the POST-Request handling (as we didn't know about CGI before). This took more time than we anticipated.
Project Planning in General: we never looked at the JIRA Board during the project. so maybe next time find a different way to plan.

Lessons learned

Future Outlook

References

Libraries

Declaration of Independent Authorship

We attest with our individual signatures that we have written this report independently and without outside help. We also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.
Additionally, we affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used.
This report may be checked for plagiarism and use of AI-supported technology using the appropriate software. We understand that unethical conduct may lead to a grade of 1 or "fail" or expulsion from the study program.