The RISC Takers:
Milestone Report I

Sean Keever
swkeever@uw.edu

Yokesh Jayakumar
mcmahjoh@uw.edu

John McMahon
karthj@uw.edu

Abstract

The goal of our project is to create a means of letting a user not only use an OS, but to allow a user to see what is going on inside the OS and CPU. To this end, our project aims to maximize availability to users by providing a browser-based interface that lets the user visit a webpage and visualizing an OS from the moment he or she visits the page.

1 Introduction

There are multiple implementations of operating systems made available via a web browser. Instead of re-implementing that functionality, we chose to find a project that has the low-level emulation from C to JavaScript already handled. This way, we can focus on solving a new problem: visualizing the OS and CPU.

We looked at multiple different OS implementations and decided on risv-angel, which seemed to offer the functionality we needed without too much underlying complexity. This way, we could focus more attention to creating the visualization layers without having to worry too much about the emulation implementation details.

2 Initial Design

To start, we stripped the riscv-angel project to only show the Terminal. We got rid of all the elements and libraries that were not going to be used in our project. Before we started on our implementation, we prototyped the UI in Figma, shown in 1.

Figure 1: Initial prototype of design

Figma allowed us to come up with a design before we implement anything in code. We aimed for a dashboard-like interface, showing the user all the most important information about the CPU and OS. Right now, we know we want to show:

- Contents of registers (show values of registers in real time)
- Contents of memory (let users zoom in on a particular region of memory)

Question for the reader: What else would be useful data to show to the user? We are having some trouble figuring out what exactly would be useful data to show.

3 Beginning Development

Setting up a development environment

One of the challenges for the team was facing ambiguity in this project. It was difficult to interpret exactly what needed to be done.

To facilitate this problem and make development easier, we started approaching the project with an Agile workflow. We

are using GitHub's Kanban board functionality to track issues and milestones. We hold weekly stand-ups to find out

- what we did over the past week, and
- what we will do in the next week.

We use the GitHub issues that we created to assign tasks for each member of a group. In doing this, we are hoping we can be more productive and have a more concrete understanding of what needs to be done to complete the project.

To ensure robustness and quality of our project, we have also developed some tooling for continuous integration. To ensure consistency in the codebase, we use ESLint, a tool that we have configured to apply the rules defined in AirBnB's style guide.

We also set up GitHub Actions, which are essentially hooks that execute when code is pushed to the repository. These hooks will run the style checker and our unit tests before code is pushed to remote. The hook will reject any commits if any of these actions fail.

Choosing technologies

We want our interface to be interactive, so we chose React.js as our library of choice, which offers a means of rapidly developing interactive client-side applications.

This doesn't come without its challenges, though. The existing project uses vanilla JavaScript in order to run the emulator. The legacy code accomplishes this by spawning a worker thread that handles all the emulation tasks. The main thread interacts with this worker thread via message passing.

This existing framework is most likely not going to work in our React app. To use the existing code, we would need to fetch the OS/CPU state via message passing. But we want our application to update in real time as the user uses the OS. Thus, the only solution we see is polling the worker thread and asking it for the most up-to-date view of CPU state. This strategy bogs down the application and will not be a strategy that we can move forward with long term.

We will need to do work to port some of the legacy code into the React scope. This way, when the OS is being used, React can know about the changes in state in real-time. We think this refactor will increase performance substantially and will be necessary if we want this application to be usable.

Future Plans

By Milestone II, we aim to

- finalize our UI prototype
- have all the base functionality of the application working
- start implementing the styling of the application

Then, in the time between Milestone II and the demo, we aim to

- finish styling the application
- add bells and whistles to the application
- attempt to optimize performance, SEO, and possibly deploy the app.

Question for the reader: What do you think about these goals? Are they too ambitious or not ambitious enough? Do you have any suggestions or recommendations in general?

Acknowledgments

Thanks to all of the contributors of riscv-angel, in which this project is based on.

Availability

This project is open-source and is available at https://github.com/swkeever/riscv-angel-extended