The RISC Takers:
Milestone Report II

Sean Keever
swkeever@uw.edu

Yokesh Jayakumar
karthj@uw.edu

John McMahon
mcmahjoh@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

In Milestone Report I, we made a plan to have the following items completed by Milestone II.

- Finalize the UI prototype
- Implement a working prototype of the application
- Begin implementing the application stylesheets

We are happy to announce that we have met these goals, and we're currently on schedule in our application development.

In this report, we discuss our experience with creating a design system and developing a modular dashboard interface. We will end the report by once again going over our plans for the remainder of the quarter, leading up to the demo.

2 Establishing a Design System

None of the members on our team are trained designers. We had to be resourceful to learn some of the patterns of good UI design. To make this easier, we took a lot of inspiration from professionals by using the Material Design system used and maintained at Google. We imported a Material Design kit into Figma, which includes many Material Design assets such as buttons, cards, etc. Doing so allowed us to wireframe a UI prototype that we feel was closer to our vision for the app's design. Figure 1 shows the first steps of prototyping our UI using Material Design and Figma.

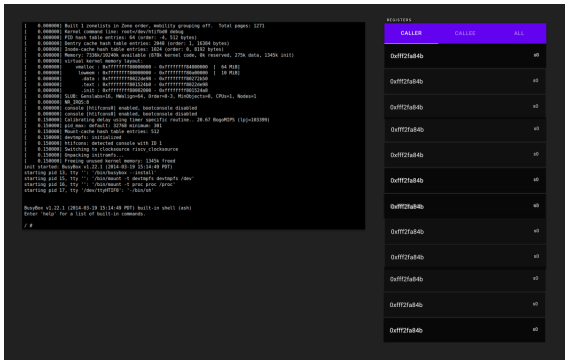

Figure 1: Initial prototype using Material Design

In addition to our team not being expert designers, we are also not experts in CSS, which is vital to making the application look good. We wanted our application to be as lightweight as possible, so we wanted to have our own custom CSS, as opposed to using a CSS framework like Bootstrap. To learn CSS and UI design patterns further, we read the book Refactoring UI, which provided many design tips from a developer's point of view.

We established a design system that is used throughout the app. We defined a system for colors, fonts, spacing/font size, and more. The system restricts properties of the UI to follow a discrete set of values. Although at first glance, it may seem that this restriction would limit our ability to create a design. In fact, the restriction provided by the design system helped to make the application feel polished and consistent.

3 Implementing a Working Prototype

A Modular Design

We describe our app as a modular, dashboard user interface that lets users use a basic OS and see properties about the OS and CPU in real time.

Our modular design means we have separate components for each visualization in the dashboard. For example, there is

a separate component for the window showing the registers, a separate component for the view showing the ratio of instructions being executed, and so forth. We split our app into modular components for several reasons:

- easier to maintain and test
- allows features to rapidly developed
- offers more flexibility in UI structure

We will elaborate on each of these points.

First, because our dashboard interface uses modular UI components, whenever we work on a single component, we can be confident that breaking a single component won't break other parts of the app.

The modular design also allows us to split the work and develop features independently. This is great for us during these times of quarantine because of COVID-19. So, for example, while one team member is working on some feature *A*, another team member can be styling feature *B*. And since the components are modular, we can do so without worrying about conflicting with another person's work.

Finally, having the components act as modular entities allows us to change the layout of the UI very easily should we choose too. The components are simply independent building blocks that can be moved around as we please. This flexibility will become very important when it comes time to finalizing our app.

Finishing our Prorotype

For our minimum viable product, we have chosen the following features for our dashboard:

- show the contents of the 32 user registers
- show the ratio of CPU instructions executed
- a time series graph showing memory utilization

We feel that this offers a useful feature set to get a glance at what is going on under the hood of an operating system utilizing the RISC-V instruction set. Figure 2 shows our working prototype which includes these features.

Future Plans

Before we demo, we plan to

- Make our application responsive; right now it only works on desktop screens.

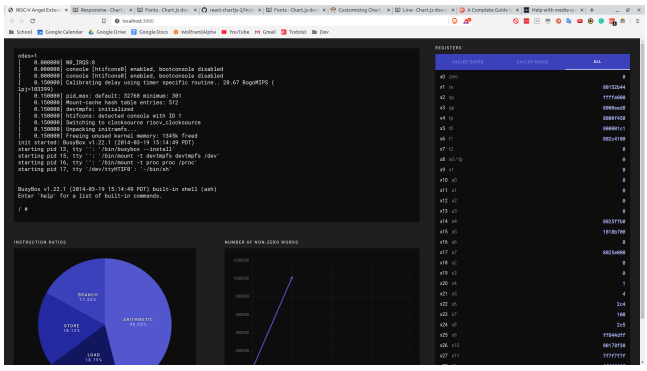

Figure 2: MVP

- Improve the application performance; try to remove message passing between React code and the VM.
- Deploy the application to the web.

We have some ideas for potential bells and whistles for our dashboard which include

- Allow user to change endianness
- Show information about the file system like which files are open
- Show information about the processes that running on the OS

To wrap up the software development lifecycle for this app, we may open a pull request on the base riscv-angel branch. Hopefully, this could invoke some inspiration to keep riscv-angel up to date and maintained in the future, making riscv-angel a tool that can be used by educators and enthusiasts alike to visualize what goes on under the hood of an operating system.

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