Advanced Operating Systems Project Proposal

Team RustyRiceOS: Patrick Tser Jern Kon, Zhiwei Liang, Jason Ludmir

October 14, 2021

1 Introduction

If there is anything that the papers we have read in class thus far have proven, it is that kernels can be very complex, especially when grinding down to the details of how certain mechanisms are implemented. Many of the concepts presented in said papers are fairly abstract, and it can be unclear, without real experience, how some of them would be designed in practice.

Tangential to this, we were exploring the Firecracker [2] virtual machine monitor (VMM), and observed that aside from the popular AWS Lambda using AWS Linux 2 (a custom kernel which is a derivative of Red Hat Enterprise Linux (RHEL)) within the Firecracker VMM, there is currently no widely known alternative kernel being used in the context of Firecracker VMM. Further, there has not been a detailed comparison of AWS Linux 2 against other kernels being used in Firecracker. The problem here is that, even if AWS Linux 2 is the most optimal kernel for AWS Lambda, it is unclear whether this assumption still holds if we were to deploy AWS Linux 2 inside Firecracker, but for other application domains.

Therefore, our goals of our project are first, to create a simple kernel to be used as both a learning device and programmatic playground, second, to deploy our kernel inside Firecracker, and third, to test it against the traditional setup of: AWS Linux 2 inside Firecracker. The proposed performance metrics of said test will be detailed in a separate section below.

Therefore, the first part of our project deals with gaining a detailed understanding of how to build a kernel from scratch, implemented in the Rust programming language. Unlike C/C++, Rust is a memory safe programming language and has many compile time restrictions, thus we can avoid many potential security issues and bugs when building the kernel. Once we build the kernel, and have comprehensive knowledge of each mechanism we implement, we then will move on to attempting to run our kernel on the Firecracker VMM. This will involve not only ensuring that Firecracker can run our kernel, but that our kernel might have some optimizations specifically designed for use with Firecracker.

Since a large part of our project involves building a kernel, we will first evaluate our kernel based

on whether or not we can perform efficient virtual memory management. Once implemented, for the operations our kernel can perform, we will benchmark it comparing with either AWS Linux 2 or another popular Linux kernel. Possible things to compare are including time and memory space usage of some specific algorithm or application.

Our anticipated contribution for this semester project is a lightweight and secure kernel (RustyRiceOS) that can be run on a Firecracker VMM; to provide a solid base for others to extend upon this kernel, and to give a detailed comparison between RustyRiceOS and AWS Linux 2, in the context of running within a Firecracker VMM.

2 Background

As newcomers to the world of operating systems, creating a kernel seems like a monumental task. To help guide us through the initial process of building one, we intend to use a very helpful resource created by Philip Oppermann [6] in which he explains in detail the process of getting started building a Kernel in the Rust language. If there is one thing that his writing proves, it is that there are many nuances in kernel design, each of which would take potentially days or even weeks for a novice programmer to discover. For instance, understanding what the red zone is on a stack and how to disable it (and why we need to in the first place), or how to prevent the compiler from optimizing away specific segments of code are not trivial to someone who does not know of their existence in the first place. In fact, he explicitly mentions how difficult it was even for him to discover some of the odder bugs when building his kernel, some taking days to resolve. Without such knowledge beforehand, it would not be possible to complete anything meaningful within the timeline of this course. Therefore, the first thing to learn is how to construct such a kernel via his methodology.

As the kernel we will develop will need to run on QEMU (another popular VMM) [4] initially for development purposes, before we attempt to migrate over to Firecracker, we will need at least some understanding of both platforms. For this task, both the Firecracker Github repository [3] and QEMU documentation [7] provide a great starting point for learning. To obtain the necessary background to understand these two VMMs however, we will be perusing this textbook on Virtual Machines: [8].

Further, to extend RustyRiceOS with new features, most notably multi-threading capabilities, we will require a thorough understanding of the concepts and implementation details. To acquire this, we will read the following books in order of increasing technical completeness: [1] and [5].

3 Methods and Plan

To complete our first task, we will build our kernel based on Opperman's blog. His documentation is divided into four major sections:

1. An intro section that creates a kernel that runs on bare metal and prints a message.
2. A section on how to deal with exceptions, faults and hardware interrupts.
3. A section on memory management, detailing the creation of a paging system, virtual memory, and heap creation/management.
4. A currently incomplete section on multiplexing.

Once we have completed working through what he has so far introduced, we would like to see the differences between having our kernel on QEMU and Firecracker, which would lead us to being able to deploy our kernel on Firecracker instead.
Ideally, we would also like to implement some multi-threading support for our kernel without any guidance as an optimization that we implement from scratch.
Finally, we would like to then benchmark the performance of our kernel, and make the comparison against AWS Linux 2. Regarding measuring our kernel, we are hoping to accomplish this by implementing a simple timing mechanism and running a small program (such as a sort or random number generator) repeatedly to obtain a mean runtime.

4 Milestones

- 10/26: Create a basic Rust Kernel (RustyRiceOS) that accomplishes all of what Oppermann's kernel can.
- 10/30: Feasibility study: Determine the feasibility of deploying RustyRiceOS on Firecracker; outline the limitations.
- 11/9: Experimental test: Run RustyRiceOS on a single Firecracker process.
- 11/11: Experimental test: Run AWS Linux 2 on a single Firecracker process.
- 11/13: Experimental test: Run a cluster of RustyRiceOS and AWS Linux 2 on Firecracker processes.
- 11/20: Extend RustyRiceOS with a more complex feature: multi-threading capabilities.
- 11/30: Evaluation: Performance comparison (using metrics defined earlier) of RustyRiceOS and AWS Linux 2 on Firecracker.

References

[1] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. 1.00. Arpaci-Dusseau Books, Aug. 2018.

[2] AWS. *Firecracker*. Oct. 2021. URL: https://firecracker-microvm.github.io/.

[3] AWS. *Firecracker Github Repository*. Oct. 2021. URL: https://github.com/firecracker-microvm/firecracker.

[4] Fabrice Bellard. "QEMU, a fast and portable dynamic translator." In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. California, USA. 2005, p. 46.

[5] Intel. *Intel Hyper-Threading Technology - User Guide*. Jan. 2003. URL: http://www.cslab.ece.ntua.gr/courses/advcomparch/2007/material/readings/Intel%20Hyper-Threading%20Technology.pdf.

[6] Philipp Oppermann. *Testing and Writing an OS in Rust*. Oct. 2021. URL: https://os.phil-opp.com/testing/.

[7] QEMU. *QEMU Documentation*. Oct. 2021. URL: https://www.qemu.org/documentation/.

[8] Jim Smith and Ravi Nair. *Virtual machines: versatile platforms for systems and processes*. Elsevier, 2005.