Virtualization/OS Notes

Katie Baek

Boston College

All of my personal notes complied into one document

November 18, 2025

Virtualization/OS Notes

Katie Baek

November 18, 2025

Abstract

This is motivated by a desire to put things into my own words. In this virtualization research project, the end goal is to add support for microVMs into Openstack. As of today (November 14th), we have mainly been learning and reading various textbooks and papers over operating systems and hypervisors. Starting next week, are going to be going over OS in 1000 lines.

Contents

1 Introduction

The bulk of the project is to add mircoVM support into OpenStack. As mentioned before, Innocent has been giving me lectures on virtualization and the OS so that I can be ready to contribute to his project when the time comes. The following sections are my notes complied from various resources as an exercise to link various ideas in my head and to ensure that I can remember things. I am unsure of how to structure this, as virtualization (of an entire machine) also involves the OS.

There are a few different resources that we're using to learn about virtualization, including [4], [3] (to learn more about the OS), and [2]. The application part of the project is first to write our own OS, then hypervisor, then tackle the Openstack part, where we're trying to add Firecracker support (microVM). This involves having a lot of background knowledge into how an OS works and how a hypervisor works.

2 Breaking through abstractions

The computer is a very complex machine which contains many level of abstractions which people interact with. The "lay-person" has probably only knows of the **GUI**, which allows them to control the computer via icons or graphical interfaces. Applications are accessed by double-clicking on the icon. You can create folders and files and visually see them represented by mini pictures within the file system. At the next "layer", there is the slightly-more technically involved pesron. Maybe they've taken a computer science class and was exposed to the command line interface (**CLI**) where you type in commands rather than use the mouse. Or maybe they're just a nerd and like learning about this stuff. The GUI was developed because there is a pretty steep learning curve to using the CLI, or other text-based interfaces and not everyone wants to dedicate enough time or is nerdy enough to want to learn. Here, the person using the computer has "broken through" one layer of abstraction and probably understands on a deeper level how a computer works. At the top layer, there is a number of protections in case the user accidently does something that they didn't mean to. Accidently click "delete" on a file? No worries, it was stored in the trash. But when you type in the command `rm *`, it deletes everything in the directory, no questions asked and no take-backs. Everything is permanently gone, forever.

This is probably where the average CS person stops. A modern front-end or back-end SWE developer doesn't need to know anything below this. They work on developing the software required to run various applications and they don't need to worry about the environment under which the applications are running. Software like

At the next layer, there is the operating system. The OS is application agonstic and is concerned with presenting a clean abstraction to the applications and managing the system resources. Applications interface with the OS.

The OS in turn, interaces with an abstraction of the hardware. The code and underlyings of the processesors, IO, and memory is very complex so there exists a layer of software inbetween the OS and the physical resources. For IO devices, this is called the driver. The OS interfaces with this driver and makes requests to that driver which represents the actual device. The driver then controls the device, does whatever the OS wanted and gives the output back to the OS.

3 Making connections

Virtualization is a term that's used a lot. At its core, its about *abstraction.* In order for something to be virtualized, there needs to be the concept of some form of hardware, or thing you want to virtualize, and the virtualized version of that thing. There are several different reasons to virtualize something and the definition shifts a little bit with each and every new iteration of virtualization. But formally, virtualization is the application of the layering principle via enforced modularity such that the exposed virtual resource is identical to the underlying physical resource [4]. There is a theorem which lays out what computer architectures are virtualizable [1]

4 The hypervisor

There are two ways to spin up a virtual machine (VM):

1. Have a hypervisor running on the bare machine, which then creates the VMs. The hypervisor multiplexes the compute, memory, and IO for the VMs. For this to work, there there needs to be **trap-and-emulate** functionality where the OS traps into hypervisor for the system calls which require the use of the CPU, memory or IO and then emulates it for the OS. The hypervisor is OS agnostic and the OS believes that it is the only one running on the machine.

2. Have a host OS with the VM running as a regular ol proceses that gets scheduled by that host OS. The host OS handles all of the multiplexing while a hypervisor-like program handles the VM.

5 Knowledge Dump

A compliations of questions that I have had while writing this doc

5.1 What is the difference between containerization and VMs?

From what I understand, containerization is the virtulization of the OS and the VM is a virtualization of the entire machine. With containerization, developers can write applications/serivces which are OS agnostic. "Containarization makes your application portable so that the same code can run on any device" [5]. You package applications so that they can run in any real or virtual environments. This software package acts like the hypervisor and is called the container engine or container runtime. It interfaces with whatever host OS/enviornment and does a translation to make sure that the application works as it should and that it does the same thing every time.

References

[1] Gerald J. Popek and Robert P. Goldberg. “Formal requirements for virtualizable third generation architectures”. In: *Commun. ACM* 17.7 (July 1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073. URL: https://doi.org/10.1145/361011.361073.

[2] James E. Smith and Ravi Nair. *Virtual Machines*. Denise E.M. Penrose, 2005.

[3] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th ed. Pearson, 2015.

[4] Edouard Bugnion, Jason Nieh, and Dan Tsafrir. *Hardware and Software Support for Virtualization*. Morgan & Claypool, 2017.

[5] AWS. *What’s the Difference Between Containers and Virutal Machines*. URL: https://aws.amazon.com/compare/the-difference-between-containers-and-virtual-machines/.