ECE 344: Operating Systems
Lecture 32

Virtual Machines

1.0.1

Jon Eyolfson
December 5, 2022

Virtual Machines Abstract an Entire Machine

Goal: run multiple operating systems on a single machine

Each OS believes they’re the only one running

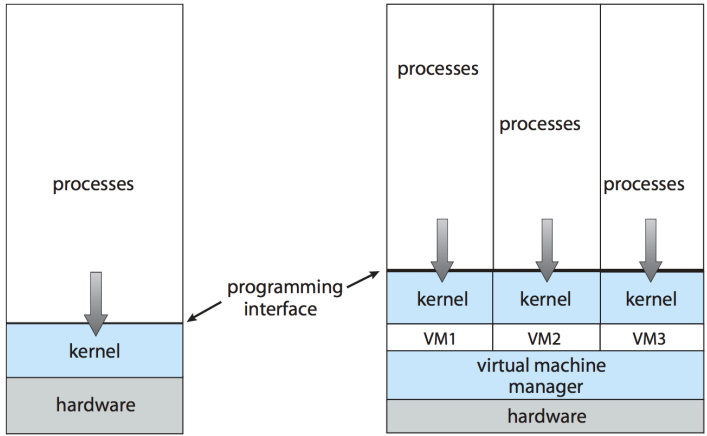
The Host Has Direct Control Over the Hardware

The *hypervisor* or *virtual machine manager* (VMM) controls virtual machines
- Creation, management, isolation (which hardware is it able to access)

There are two kinds of hypervisors: type 1 and type 2
- Type 1: bare metal hypervisor, it runs directly on the host's hardware
  - Requires special hardware support
- Type 2: hosted hypervisor, it simulates a hypervisor and runs as an application
  - Slower, but does not require any special hardware

A *guest* sees it's own virtual copy of the host

(a) Is a Typical Machine (b) Shows One Machine Running 3 Kernels



Virtual Machines are Not Emulation

Emulation is typically used to translate one ISA to another
- e.g. x86_64 to ARM/RISC-V

Our guest operating system executes instructions directly using the same ISA
- Otherwise translating instructions is typically slow

It's OK for some uses, such as a Nintendo Entertainment System (NES) emulator

A virtual machine could use emulation to run a virtual machine for a different ISA
- Performance would suffer greatly

Virtual Machines Enable Pause and Play

Much like our kernel can pause a process, a hypervisor can pause an OS

The hypervisor needs to context switch between virtual machines

- It'll save the current state and restores it later

We could also move it around, exactly like a process

Virtual Machines Provide Protection Through Isolation

The guests are isolated from each other, and the host

The hypervisor and set limits on: CPU time, memory, network bandwidth, etc.

A compromised guest only has access to it's own virtualized hardware

You can easily roll back the infected virtual machine, or remove it

Virtual Machines Also Help Consolidation

In data centers there's many servers running, often not making use of all resources

- Servers with different purposes could be sharing the same hardware

Instead of having lightly used physical systems, make them virtual machines

- Run as many on a single machine as possible

A Virtual CPU (VCPU) Is the Key Abstraction

For processes, part of the process control block (PCB) acted as a virtual CPU

- It doesn't virtualize all parts of the CPU, just enough for user-mode processes

The VCPU is a data structure that stores the state of the CPU

- The hypervisor saves this when the guest isn't running

When the virtual machine resumes, like the PCB, it loads the data and resumes

The Guest Still Uses User and Kernel Modes

There are no changes to the guest operating systems

- A Linux kernel still uses privileged instructions

Recall on x86_64 user mode is ring 3, kernel mode is ring 0

- A hardware hypervisor (type 1) is ring -1, letting it control the guest

For type 2 hypervisors, the host has to create a virtual kernel and user mode

One Strategy is to Trap-and-Emulate

For type 2 hypervisors the guest runs on the host in user mode

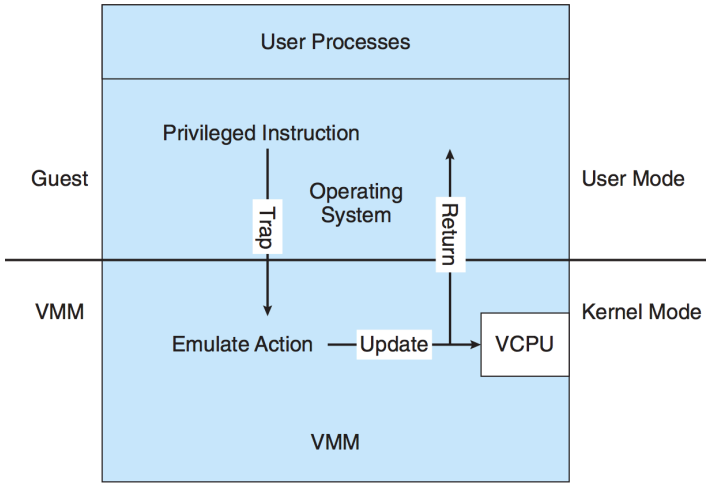
- Any privileged instructions generate a trap (wrong mode)

The hypervisor should explicitly handle this error

- Emulate (or simulate) the operation for the guest and resume it

This will slow down the otherwise native execution

Trap-and-Emulate Visually



Trap-and-Emulate Does Not Always Work

Some CPUs are not clear between privileged and non-privileged instructions
- This includes x86_64, virtual machines didn't exist in the 1970s

One example is the `popf` instruction, it loads the flags register from the stack
- It behaves differently for both kernel and user mode

It does not generate a trap, so you can't trap-and-emulate
- These *special* instructions need another approach

Special Instructions Need Binary Translation

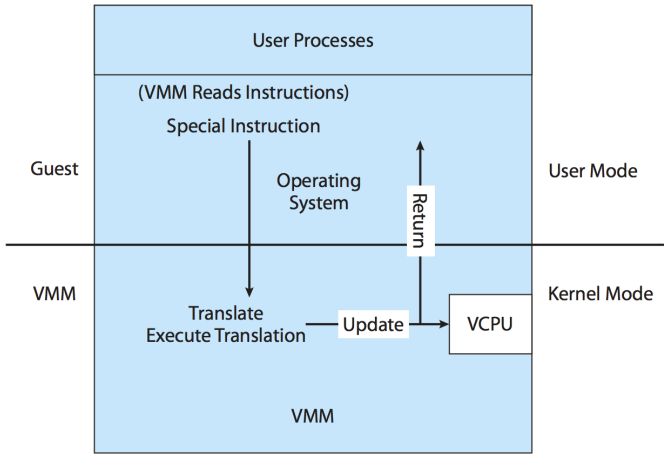
If the guest VCPU is in user mode, we can run instructions natively
    in kernel mode, the hypervisor inspects every instruction before execution

Special instructions need to be translated to instructions with the same effect
    Regular instructions can run natively

The kernel uses a CPU instruction to switch from user to kernel mode
    The hypervisor can handle that using normal trap-and-emulate

Overall performance for type 2 hypervisors suffer, but they’re adequate

Binary Translation Visually



One More Hardware Rescue

In 2005 Intel introduced virtualization as VT-x and in 2006 AMD did as AMD-V

- Intel's codename Vanderpool, published as Virtual Machine Extensions (VMX)
- AMD's codename Pacifica, published as Secure Virtual Machine (SVM)

This added the concept of ring -1, or hypervisor mode

The host kernel claims the hypervisor, and is the only one able to access it

- It can set the isolation for the guests and what hardware to virtualize

Virtualized Scheduling

If there is only one CPU on the physical machine, the guest will not know

- The host could still present multiple virtual CPUs to the guest

We now need to map the VCPUs to physical CPUs, or schedule them like processes

- Like a normal kernel, there will also be hypervisor threads

One Approach is CPU Assignment

If there are more physical cores on the host than all VCPUs, we can map 1:1

- The host can continue using the spare physical cores

If we have to share, things get more complicated (called overcommitting)

- At equal numbers we can still map 1:1, the hypervisor threads don't run often

We have to use a scheduling algorithm, like we used for processes

CPU Overcommitment Causes Additional Problems

The guest operating system runs too unpredictably for soft real-time tasks

- It may be context switched out when the user process says not to

For example, consider a real-time round robin time slice is 10 ms

- The guest will not have a consistent slice of 10 ms, it may be much higher

This may make processes miss deadlines they wouldn't have running on the host

- In this case virtualization has different observable behavior

Virtualized Memory Management Gets a Lot More Complex

Recall: virtual memory allows each process to think it has the entire address space

Now the guest kernel thinks it's managing the entire physical address space

We have to virtualize that too!

The problem gets even worse if memory is overcommitted as well

Nested Page Tables Enable Virtual Memory for Guest Kernels

The guest thinks it controls physical memory, and does page table management

The hypervisor maintains a nested page table the re-translates for the guest
- It translates the guest's page table to the real physical page table

For overcommitted memory the hypervisor can provide double-paging
- The hypervisor does its own page replacement algorithm
  - However, the guest may know it's memory access patterns better

Guests Could Share Pages if They’re Duplicates

Similar to copy-on-write pages, we can get memory saves by sharing pages
- This time instead of sharing between processes, share between guests

The hypervisor can do duplicate detection by hashing the contents of pages
- If two hashes are the same, check they’re the same byte-for-byte

If they’re the same, we can share them until one of the guests try to write
- Then we again do copy-on-write as before

The Hypervisor Provides Virtualized I/O Devices

The hypervisor can multiplex one device to multiple virtual machines

The hypervisor could also emulate devices that don't physically exist

The hypervisor could also map one physical device to a virtual device one VM

The VM has exclusive access to the device, but hypervisor still translates

There is a hardware solution to remove the hypervisor during run-time — IOMMU

The hypervisor maps the devices virtual memory exclusively to the guest

The VM now actually has exclusive control over the device

This allows complex GPUs to work at native speeds in VMs

Virtual Machines Boot from a Virtualized Disk

You create a *disk image* that has all the contents of a physical disk

- It contains partitions, and each partition has a file system

Usually, it's one big file (but some formats allow you to split it up)

- The guest kernel sees it has a normal disk, that it has full control of

The disk image is all you need for the virtual machine, makes it easy to move

- The ova file you downloaded is basically a disk image and guest settings

Virtual Machines Could Be Used to Isolate an Application

Assume your application uses a dynamic library (back to Lecture 13)

- An ABI change would cause your application to no longer work
- Even more subtle, the library's behavior could change

You may want to freeze your dependencies to deploy it in production

- Create a virtual machine for it with all the libraries it needs

Containers, like Docker, Aim to be Faster

The hypervisor sets limits on: CPU time, memory, network bandwidth, etc.

What if the kernel supported this directly, without virtualization?

Linux control groups (`cgroups`) support hypervisor-like limits for processes

Isolate a process to a *namespace*

You can set other resources a namespace can access (mount points, IPC, etc.)

Containers are lighter-weight than full virtual machines, they use a normal kernel

Virtual Machines Virtualize a Physical Machine

They allow multiple operating systems to share the same hardware

- Virtual machines provide isolation, the hypervisor allocates resources
- Type 2 hypervisors are slower due to trap-and-emulate and binary translation
- Type 1 hypervisors are supported by hardware, IOMMU speeds up devices
- Hypervisors may overcommit resources and need to physically move VM
- Containers aim to have the benefits of VMs, without the overhead