Virtual Machine Monitor
Topics

• How to implement VMMs?
• How to reduce overhead?
How to Implement a VMM?

• Emulator
  – Many game consoles are emulated
  – In theory, any h/w can be emulated via s/w

• Language based virtual machine
  – Instead of virtualizing real hardware, it provides a specially designed virtual hardware for the specific language(s)
  – JVM for Java, CLR for MS .Net

• Common issues: performance
Java Virtual Machine

Java program .class files --> class loader --> Java interpreter --> host system (Windows, Linux, etc.) --> Java API .class files

Java byte code

Performance killer
How to Implement a VMM?

• Modern VMMs
  – Normal instructions are executed on the real CPU
    • In case of emulator, each instruction is executed in s/w
    • No performance loss for user-mode instructions
  – Privileged instructions cause *traps* to the VMM
    • Privileged instructions (e.g., addr. space change)
Instructions Types

• Normal instructions
  – add, sub, load/store, branch, ...
  – Execute natively

• Privileged instructions
  – Setup page tables, load/flush TLB and caches
    • LGDT, LLDT, LTR, MOV <Control Reg>, LMSW, ...
  – Mode change, system state monitor
    • HLT, RDMSR, WRMSR
Trap and Emulation in VMM

- Virtualize privileged instructions
  - Guests run in user-mode, generating exceptions
Some instructions are not virtualizable

- Execute in both user and kernel modes, but behave differently (e.g., `popf`)
Types of VMM

• Native (or Type 1) VMM
  – VMM runs directly on top of bare hardware
  –Vmware ESX, Microsoft Hyper-V
  – VMM is a kind of a OS on its own right

• Hosted (or Type 2) VMM
  – VMM runs within an OS
  – VirtualBox, VMWare Workstation
  – VMM relies on functionalities of the host OS
VMware WorkStation (Player)
How to Virtualize Hardware?

• CPU
• Memory
• Events
  – Exceptions, interrupts
• I/O devices
  – Disk, network
Virtualizing the CPU

• Virtual CPU (vCPU)
  – One or more vCPUs for every VM
  – Seen as physical CPU for the guest OS on the VM

• How?
  – Timeslice the CPU
  – Just like CPU scheduling in OS
  – VMM uses CFS like scheduler(s)
VMM Timesharing
Virtualizing Memory

- **OS view**
  - Virtual address $\rightarrow$ physical address

- **VMM view**
  - Guest virtual $\rightarrow$ guest physical $\rightarrow$ VMM physical
  - Does MMU know about VMM physical???
  - Originally no, but now yes
    - Intel/AMD support nest page tables

---

*Intel EPT (extended page table)*
Virtualizing Interrupts & I/O

• VMM receives h/w interrupts
  – Determines which VM to receive
  – Emulate interrupt controller for the VM

• VMM emulate a specific h/w devices
  – Guest OS $\rightarrow$ VMM $\rightarrow$ devices
    • E.g., AMD Lance PCNet ethernet device

• Lots of I/O $\rightarrow$ performance killers
Para-virtualization

• Idea: provides simple/fast APIs to guests
  – Instead of emulating actual hardware (e.g., PCNet32 ethernet card)

  – Pros
    • can be a lot faster (more efficient I/O)

  – Cons
    • need to modify the guest OS
I/O in Xen via Shared Buffer

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

Request queue - Descriptors queued by the VM but not yet accepted by Xen

Outstanding descriptors - Descriptor slots awaiting a response from Xen

Response queue - Descriptors returned by Xen in response to serviced requests

Unused descriptors
IOMMU

• Problem: How to do DMA in a VM?
  – DMA controller needs host physical address, not guest physical address

• IOMMU
  – MMU for IO devices
  – maps guest physical → host physical for the I/O devices

https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit#/media/File:MMU_and_IOMMU.svg
LXC: OS (Linux) Container

- Same kernel, separate user-space
- Virtualize OS, not machine
- Low overhead, flexible
Docker: Application Container

- A container contains one application (process)
- Built on top of OS containers
- Even more flexible
Summary

• Virtual Machine (hardware virtualization)
  – Trap & emulate
  – Binary translation
  – Para-virtualization
  – Hardware support for virtualization

• Containers
  – OS container: same kernel, different user-space
  – App container: same kernel, per-process space