Virtual Machines

1DV512 – Operating Systems

Dr. Kostiantyn Kucher

kostiantyn.kucher@lnu.se

December 9, 2020

Based on the Operating System Concepts slides by Silberschatz, Galvin, and Gagne (2018)

Suggested OSC book complement: Chapter 18 *in 10th edition (2018)*

Agenda

- Motivation and Introduction
- Virtualization Concepts
- Virtual Machine Building Blocks
- Virtual Machine Implementation
- Summary
- Course Summary

Motivation

- "All problems in computer science can be solved by another level of indirection" (David Wheeler)
- "... except for the problem of too many layers of indirection" (Kevlin Henney)

- Fundamental idea $\Rightarrow$ abstract hardware of a single computer into several different execution environments, or *virtual machines* (VM), on which operating systems or applications can run
- Single physical machine can run multiple (and typically different!) operating systems concurrently, each in its own virtual machine

Agenda

Virtualization Concepts

- *Host* ⇒ underlying hardware system
- *Virtual machine manager* (VMM) or *hypervisor* ⇒ creates and runs virtual machines by providing interface that is (typically) identical to the host
  - Is hypervisor itself an OS?..
- *Guest* ⇒ environment (usually an OS) provided with a virtual copy of the host
- Virtualization ideas emerged in 1960s and first became available commercially on IBM mainframes in 1972
- In late 1990s ⇒ virtualization on general-purpose x86 computers

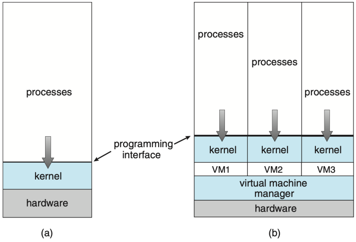

Fig. 18.1 in OSC book

Virtualization Requirements

- *Fidelity* ⇒ a VMM provides an environment for programs that is essentially identical to the original machine
- *Performance* ⇒ programs running within that environment show only minor performance decreases
- *Safety* ⇒ the VMM is in complete control of system resources

Virtualization Implementation

- *Type 0 hypervisors* ⇒ hardware-based solutions that provide support for virtual machine creation and management via firmware (e.g., solutions from IBM and Oracle)
- *Type 1 hypervisors* ⇒ operating-system-like software built to provide virtualization (e.g., VMWare ESX and Citrix XenServer)
- . . . but also general-purpose OS that provide standard functions as well as VMM functions (e.g., Linux with KVM and Windows Server with HyperV)
- *Type 2 hypervisors* ⇒ applications that run on standard OS, but provide VMM features to guest operating systems (e.g., VirtualBox, VMWare Workstation, Parallels Desktop)
- (more details about all these types below. . .)

Virtualization Implementation (cont.)

- *Paravirtualization* ⇒ the guest operating system is modified to work in cooperation with the VMM to optimize performance (e.g., some versions of Linux kernel); involves *hypercalls* ⇒ system calls to the hypervisor
- *Programming-environment virtualization* ⇒ VMMs do not virtualize real hardware, but instead create an optimized virtual system for specific code execution (e.g., Java Virtual Machine and .Net CLR)
- *Emulators* ⇒ allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU (e.g., QEMU emulating MIPS; extra computational overheads. . .)
- *Application containment* ⇒ not virtualization per se, but rather provides virtualization-like features by segregating individual applications from the operating system, making them more secure and manageable (e.g., Wine)
- *OS-level virtualization* ⇒ a complete user space environment (running on host OS kernel) is available in a *container* (e.g., Unix chroot jails, BSD jails, Solaris zones, LXC, OpenVZ virtual private servers, DragonflyBSD virtual kernels, Docker containers. . .)

Virtualization Benefits

- *Protection* through *isolation* $\Rightarrow$ host system protected from VMs, and VMs protected from each other
  - Sharing is provided though via shared file system volumes or network communication
- Ability to *suspend* (freeze) & resume, *clone*, or take a *snapshot* of a VM
- *Consolidation* $\Rightarrow$ run multiple, different OSes on a single machine simultaneously (cf. *physical-to-virtual* conversion)
- *Templating* and *live migration* $\Rightarrow$ eventually, orchestration and cloud computing...

Agenda

CPU Support

- Generally difficult to provide an exact duplicate of underlying machine, especially machines with kernel and user mode ⇒ not relevant for type 0 hypervisors
- *Virtual CPU (VCPU)* represents the state of CPU per guest as guest believes it to be
- Dual mode CPU typically means the guest only executes in user mode
- VM needs two modes⇒ *virtual user mode* and *virtual kernel mode*
- Actions in guest that required kernel mode must cause switch to *virtual* kernel mode
- *Trap-and-emulate* ⇒ attempting a privileged instruction within the guest leads to VMM emulating the code on the host
- Performance issues for such instructions; but the situation has improved over the years due to hardware support such as additional CPU modes

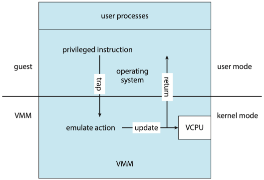

Fig. 18.2 in OSC book

CPU Support (cont.)

- Some CPUs don't have clean separation between privileged and nonprivileged instructions ⇒ earlier Intel processors used *special instructions* that were difficult to handle with trap-and-emulate...
- *Binary translation* ⇒ user mode code in the guest is OK to execute natively on host hardware...
- ...but kernel mode code in the guest must be read ahead by VMM, several instructions ahead of the program counter
- Non-*special* instructions ⇒ can still run natively
- Special instructions ⇒ translated into other instructions (still native binary code...) that are safe to execute natively!
- Difficult to implement and poor performance ⇒ extra steps required (e.g., caching of the translated binary code)

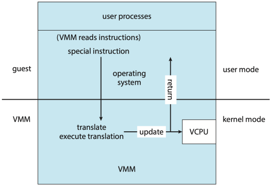

Fig. 18.3 in OSC book

Memory Management

- Memory management is another general challenge ⇒ how can VMM keep page-table state both for (1) guests believing they control the page tables and (2) VMM that does indeed control the tables?
- Common method (for trap-and-emulate and binary translation) is *nested page tables* (NPTs)
- Each guest maintains page tables to translate virtual to physical addresses
- VMM maintains per guest NPTs to represent guest's page-table state, just as VCPU stores guest CPU state
- When guest on CPU ⇒ VMM makes that guest's NPTs the active system page tables
- Guest tries to change page table ⇒ VMM makes equivalent change to NPTs and its own page tables
- Can cause many more TLB misses ⇒ much slower performance

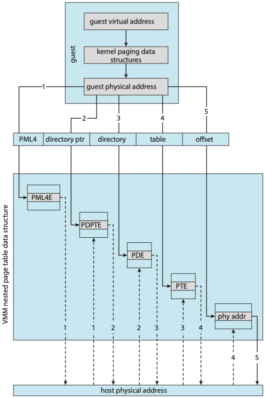

Fig. 18.4 in OSC book

Hardware Assistance

- Full-fledged virtualization needs some hardware support
- More support $\Rightarrow$ more feature rich, stable, better performance of guests
- Intel added new VT-x instructions in 2005 and AMD the AMD-V instructions in 2006
- CPUs with these instructions remove need for binary translation
- Generally define additional CPU modes $\Rightarrow$ *guest* and *host*
- VMM can enable host mode, define characteristics of each guest VM, switch to guest mode and guest(s) on CPU(s)
- In guest mode $\Rightarrow$ guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
- Access to virtualized device, privileged instructions cause trap to VMM
- CPU maintains VCPU, context switches it as needed
- Hardware support for Nested Page Tables, DMA, interrupts as well

Agenda

- Motivation and Introduction
- Virtualization Concepts
- Virtual Machine Building Blocks
- Virtual Machine Implementation
- Summary
- Course Summary

Virtual Machine Lifecycle

- Many variations as well as hardware details
- Assume VMMs take advantage of hardware features
- Hardware features can simplify implementation and improve performance
- Whatever the type, a VM has a lifecycle:
  - Created by VMM
  - Resources assigned to it (number of cores, amount of memory, networking details, storage details)
  - When no longer needed, VM can be deleted, freeing resources
- Steps simpler, faster than with a physical machine install

Type 0 Hypervisors

- Old idea, under many names by hardware manufacturers ⇒ "partitions", "domains", . . .
- A hardware feature implemented by *firmware* ⇒ OS does not do anything special, VMM is in firmware
- Smaller feature set than other types
- Each guest has dedicated hardware
- I/O a challenge as difficult to have enough devices and controllers to dedicate to each guest
- Sometimes VMM implements a *control partition* running daemons that other guests communicate with for shared I/O
- Can provide *virtualization-within-virtualization* (guest itself can be a VMM with guests) ⇒ other types have difficulty doing this!

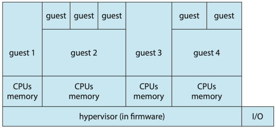

Fig. 18.5 in OSC book

Type 1 Hypervisors

- Special purpose operating systems that run natively on hardware (e.g., VMWare ESX, Citrix XenServer, Xen. . .)
- Rather than providing system call interface ⇒ create, run, and manage guest OSes
- Can run on Type 0 hypervisors but not on other Type 1s
- Run in kernel mode, while guests usually don't know they are in a VM
- Implement device drivers for host hardware because no other component can
- Provide other traditional OS services like CPU and memory management
- Another variation is a general purpose OS that also provides VMM functionality (e.g., RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris)
- Perform normal duties as well as VMM duties
- Typically less feature-rich than dedicated Type 1 hypervisors
- In many ways, treat guests OSes as just another process ⇒ albeit with special handling when guest tries to execute special instructions!

Type 2 Hypervisors

- Less interesting from an OS perspective ⇒ very little OS involvement in virtualization
- VMM is simply another process, run and managed by host (e.g., VMWare Workstation or Oracle VirtualBox) ⇒ the host doesn't even necessarily know that virtualization is taking place...
- Tend to have poorer overall performance because can't take advantage of some hardware features
- But also a benefit because require no changes to host OS
- PC user could have a Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

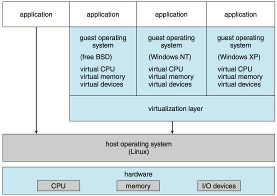

Fig. 18.9 in OSC book

Paravirtualization

- Does not completely fit the definition of virtualization ⇒ VMM not presenting an exact duplication of underlying hardware, and guest system has to be modified ⇒ but still useful!
- VMM provides services that guest must be modified to use ⇒ leads to increased performance
- Xen, for instance, adds several techniques:
  - Efficient I/O via a circular shared buffer
  - Memory management does not include nested page tables ⇒ the guest uses *hypercalls* when page-table changes needed
- Paravirtualization allowed virtualization of older x86 CPUs (and others) without binary translation
- Approach less needed as hardware support for VMs grows
- On modern CPUs Xen no longer requires guest modification ⇒ no longer paravirtualization, but rather Type 1 hypervisor

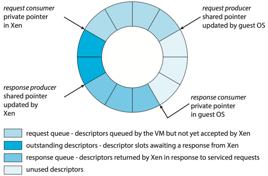

Fig. 18.6 in OSC book

Emulation

- Another (older) way for running one operating system on a different operating system
- *Virtualization* requires the underlying CPU to be same as the guest was compiled for
- *Emulation* allows the guest to run on a different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
- Useful when host system has one architecture, guest compiled for other architecture, e.g.:
  - Company replacing outdated servers with new servers containing different CPU architecture, but still wanting to run old applications
  - Old gaming consoles emulated on new consoles, or PCs or smartphones
- Performance challenge $\Rightarrow$ order of magnitude slower than native code
- New machines faster than older machines so can reduce slowdown

Application- and OS-Level Virtualization

- Application containment (isolation) ⇒ does not required full-fledged virtualization support, if applications are compiled for the host operating system
- OS-level virtualization for complete user space environment ⇒ *containers* used for segregation and control of apps, performance and resource management
- Uses the host system kernel, more lightweight than complete VMs
- Solaris zones:
  - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
  - Each zone has its own applications, networking stack, user accounts, etc.
  - CPU and memory resources divided between zones ⇒ a zone can have its own scheduler to use those resources

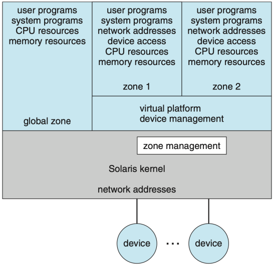

Fig. 18.7 in OSC book

Further OS Aspects for Virtualization

- Usually not enough CPUs ⇒ CPU *overcommitment*
- VMM can use standard scheduling algorithms to put threads on CPUs
- *Cycle stealing* by VMM and oversubscription of CPUs means guests don't get CPU cycles they expect ⇒ issues with clock time and synchronization
- Memory management also suffers from oversubscription⇒ requires extra management efficiency from VMM
- I/O is complicated for VMMs ⇒ many short paths for I/O in standard OSes for improved performance
- Less hypervisor needs to do for I/O for guests, the better
- Possibilities include direct device access, DMA pass-through, etc. ⇒ additional hardware support required
- Secondary storage ⇒ both boot disk and general data access need to be provided by VMM; also various partitions, disks images, network-attached storage. . .
- Networking also complex as VMM and guests all need network access ⇒ use *bridging* (allowing guests direct access to network) or *network address translation* (NAT) (guest gets a local network address)

Agenda

Summary

- Virtualization is a method for providing a guest with a duplicate of a system's underlying hardware $\Rightarrow$ a virtual machine
- Multiple guests can run on a given system, each believing that it is the native operating system and it is in full control
- Hardware support improves the performance of virtualization for compatible architectures $\Rightarrow$ otherwise, software emulation is necessary
- The virtual machine manager, or hypervisor, creates and runs virtual machines $\Rightarrow$ several hypervisor types exist

Agenda

That's It, Folks!

- This was the final lecture!
- Remaining tutoring sessions:
  - December 9
  - December 16
  - January 13
  - Moodle forums and Slack also available, of course...
- Remaining deadlines:
  - **Group Assignment 2** $\Rightarrow$ December 23
  - **Individual Assignment 3** $\Rightarrow$ January 17
- Retakes:
  - March 14
  - April 25
- **Good luck and stay safe!**