CSE 511: Operating Systems Design

Lectures 12-13

Virtualization

Virtualization

- Allows multiple instances of an operating system to run on a single computer
  - Originally introduced for IBM VM/370
  - Has later been revitalized for modern platforms
- A ***hypervisor*** is a software layer that
  - Separates the ***virtual*** hardware an OS sees from the ***actual*** hardware
  - Arbitrates access to physical resources such as CPU or memory

Virtualization

- Widely known hypervisors
  - Xen, KVM, VMware ESX, Hyper V, VirtualBox, qemu etc
- Virtualization
  - Improves ***isolation*** and ***reliability*** because each OS runs independently from the others on its own ***virtual*** processors so that OS failures are ***contained***
  - Increases ***resource utilization*** as the same hardware can be used for multiple purpose
  - Leads to better ***productivity*** as large pieces of software can be preconfigured and installed very easily

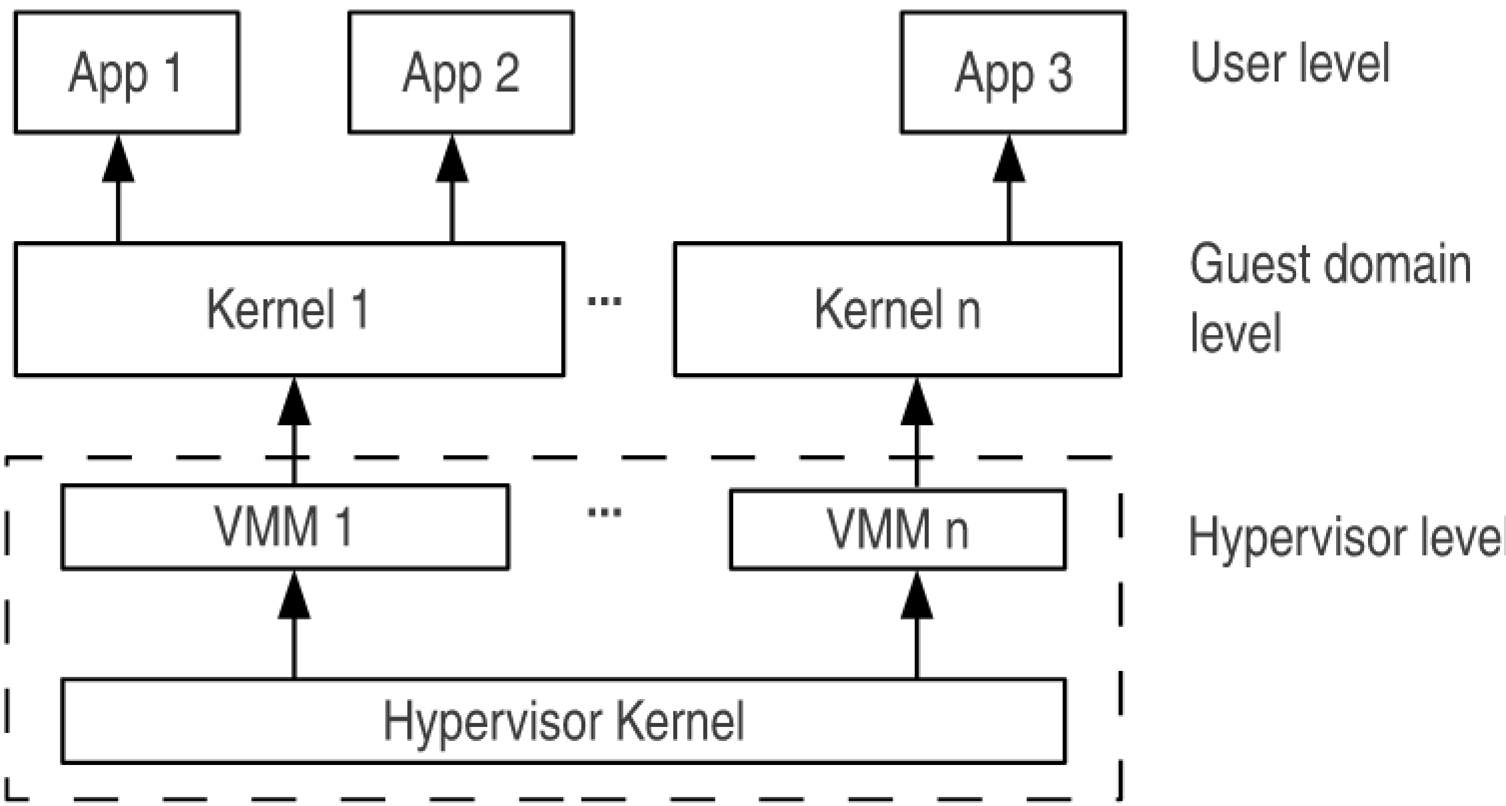
Virtualization

- Virtual appliances
  - Software bundles containing preconfigured packages with their own OS aimed at simplified distribution and deployment
  - They run along with other virtual appliances and general purpose OS on a single machine
- Infrastructure as a service (IaaS) solutions
  - Rely on virtualization to provide business solutions for server consolidation
- Hypervisors can be utilized to provide better security and robustness for operating systems

Hypervisor Types

- Type I
  - Allows execution of virtual machines in ***guest*** domains
- Type II
  - Runs on top of some OS, known as a ***host*** OS
- Type I hypervisors avoid unnecessary overheads, latencies; they are also not subject to host OS vulnerabilities
  - The hypervisor is at the lowest layer: the hypervisor kernel and Virtual Machine Monitors (VMMs)
  - The hypervisor kernel has direct access to hardware, resource allocation
  - The VMM layer is responsible for virtualizing resources

Hypervisor Types



CPU Virtualization

- A hypervisor needs to provide guests with ***virtual*** CPUs
  - Entities that allow guests to have access to CPU resources while allowing the hypervisor to schedule different operating systems on the available ***physical*** CPUs or cores
- Complete and transparent CPU virtualization traditionally required ***architectural support***

CPU Virtualization

- Popek and Goldberg formulated necessary and sufficient conditions for virtualization
  - ***Fidelity***: requires that the virtual environment in which programs run should be indistinguishable from the real hardware
  - ***Performance***: a substantial amount of instructions need to be executed directly by hardware and without any additional hypervisor handling
  - ***Safety***: requires that the hypervisor has full control of all system resources

CPU Virtualization

- ***Sensitive instructions*** used by guests need to be trapped and emulated if they could affect the correctness of the hypervisor's behavior
  - ***Trap-and-emulate*** is a technique for intercepting CPU instructions and executing them using special (hypervisor) handlers
- The original x86 architecture does not meet described criteria, as it does not allow to intercept ***all*** necessary instructions when executed in ***unprivileged*** mode

CPU Virtualization

- Techniques to overcome this problem
  - ***Binary translation***: machine code is being processed by a translator program, which allows prefetching, inspecting and special treatment of all relevant instructions (e.g., VMware)
  - ***Paravirtualization***: adaptations of the guest OS kernel code that avoids the use of untrappable instructions, reducing emulation and management costs and yielding better performance (e.g., Xen)

CPU Virtualization

- Xen’s original paravirtualization relied on the 4-ring protection hierarchy present in 32-bit x86 processors
  - Ring 0: privileged hypervisor code
  - Ring 1: guest kernel
  - Ring 3: user processes
- Segmentation protected guest kernel code and data from user processes, but guest kernels had to be adapted to successfully run in this mode
  - x86-64 effectively removed segmentation

CPU Virtualization

- The original paravirtualization approach is not so practical for x86-64
  - Need to run the kernel in Ring 3 and switch page tables every single system call
- But CPU vendors introduced ***hardware extensions*** (Intel VT-x, AMD-V) that allowed the execution of unchanged guest kernels
  - Safe virtualization of the CPU by introducing a VMM mode distinct from the mode in which privileged guest kernel code executes

CPU Virtualization

- Later generations added support for MMU virtualization via ***nested paging*** without resorting to ***shadow page tables***
- Well known hypervisors such as Xen and VMware now provide support for hardware virtualization
- Xen modes
  - ***PV***: paravirtualization
  - ***HVM, PVHVM***: using hardware extensions
  - ***PVH***: in between

CPU Virtualization

Poor Performance
Scope for Improvement
Optimal Performance

PV = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized
HA = Hardware Accelerated

x86 Shortcut	Mode	With	Disk and Network	Interrupts & Timers	Boot Path	Privileged Instructions, Page Tables	QEMU Used
HVM / Fully Virtualized	HVM		VS	VS[1]	VS	VH	Yes
HVM + PV drivers	HVM	PV Drivers Installed	PV	VS[1]	VS	VH	Yes
PVHVM	HVM	PVHVM Capable Guest	PV	PV[2]	VS	VH	Yes
PVH	PVH	PVH Capable Guest	PV	HA[3]	PV[4]	VH	No
PV	PV		PV	PV	PV[5]	PV	No
ARM							
N/A	N/A		PV	VH	PV[6]	VH	No

* The picture is taken from https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

Memory Virtualization

- Conventional OS manage memory assuming that they have unrestricted control over physical memory
- Hypervisors introduce special adaptations to guest OS allowing to run multiple guest domains at the same time
  - ***Virtual***, ***physical***, and ***machine*** memory address space
  - ***Machine*** memory is the actual memory exposed by hardware

Memory Virtualization

- Physical and virtual memory are guest specific and have their conventional meaning
  - Physical addresses correspond to memory exposed by the VMM rather than hardware
- Hypervisors and CPUs provide support for virtual-to-physical, physical-to-machine and virtual-to-machine address translation
  - Page granularity: hypervisors and guests refer to the physical and machine addresses using their physical and machine frame numbers

Memory Virtualization

- Fully-virtualized guests are completely unaware of the underlying hypervisor
  - They know nothing about machine frames and treat physical frames as if they represented actual hardware exposed memory
- Guests create page tables containing virtual-to-physical memory mappings
  - Cannot be used by the hardware since physical frames do not correspond to actual memory location

Memory Virtualization

- The hypervisor traps any attempt to load page tables and replace them with ***shadow page tables***
  - ***Shadow page tables*** are created and maintained by the hypervisor
  - Entries are created on demand
  - Contain virtual-to-machine memory mappings generated by the hypervisor from guest page tables and the physical-to-machine (P2M) hypervisor translation tables
  - Transparent (not exposed to guests)

Memory Virtualization

- CPU vendors also introduced ***nested page tables***
  - Eliminates shadow page tables by providing an additional hardware translation layer
  - The first layer is for regular OS page tables containing virtual-to-physical mappings
  - The second layer is for physical-to-machine mappings and is managed by the hypervisor

Memory Virtualization

- Paravirtualization
  - Guests are adapted to assist the hypervisor with memory management
  - They have direct access to the P2M translation table, allowing them to perform physical-to-machine translation
  - No shadow page tables: guests create page tables with virtual-to-machine mappings
    - The hypervisor will verify the consistency of page tables