Virtualization

Shivanshu Agrawal
Abhimanyu Arora

Virtual Machine Monitor (VMM)

VMM is the virtualization layer responsible for hosting and managing all the virtual machines. The following characteristics of VMM known as Popek and Goldberg virtualization requirements were introduced in 1974[1]

- Equivalence - A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly
- Efficiency - A statistically dominant fraction of machine instructions must be executed without VMM intervention.
- Resource Control - The VMM must be in complete control of the virtualized resources.

The VMM is responsible for managing CPU, Memory and Device and I/O resources for each guest VM

CPU Virtualization

- Three techniques -
  - Full Virtualization
  - Paravirtualization
  - Hardware Assisted Virtualization

Privileges

- Modern operating systems have two modes -
  - User mode - allows only instructions that are necessary to calculate and process data. All applications run in this mode.
  - Kernel mode - allowed to run almost any CPU instructions, including "privileged" instructions that deal with interrupts, memory management, and so on. Operating System runs in this mode.

- The privilege separation is done at the hardware level using CPU rings. In x86, Kernel code runs in ring 0 while user code runs in ring 3. An attempt to run a protected instruction outside of ring zero causes a general-protection exception

- In which ring should the code of a guest VM’s kernel run?

Full Virtualization

In the earliest virtual machines by IBM in 1970’s

- VMM runs in ring 0
- Since all the resources, I/O requests of the guest machines are to be handled by the VMM, guest OS runs in ring 3
- All the unprivileged instructions run normally on the CPU
- Every privileged instruction by a virtual machine caused a "trap"/interrupt. VMM intercepts all those traps and emulates the instruction, without jeopardizing the integrity of the other guests

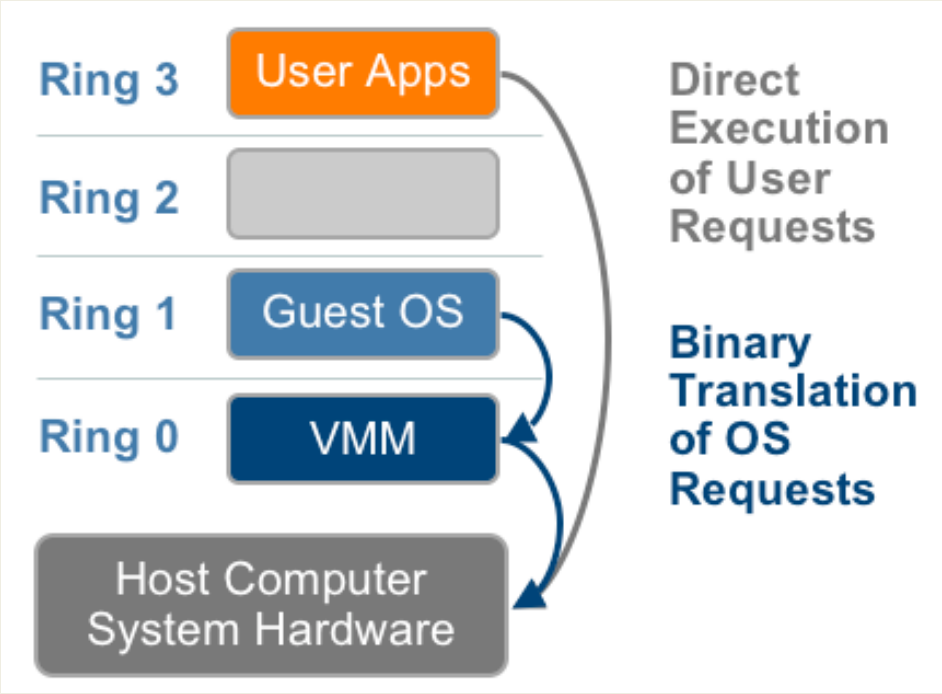
This worked for PowerPC but this kind of virtualization is not possible in x86 Instruction set architecture because it does not trap every incident that should lead to VMM intervention.

Trap and Emulate

- Kernel calls in guest OS:
  - User program running under guest OS issues kernel call instruction.
  - Traps always go to VMM (not guest OS).
  - VMM analyzes trapping instruction, simulates system call to guest OS:
    - Move trap info from VMM stack to stack of guest OS
    - Find interrupt vector in memory of guest OS
    - Switch simulated mode to "privileged"
    - Return out of VMM to interrupt handler in guest OS.
  - When guest OS returns from system call, this traps to VMM also (illegal instruction in user mode); VMM simulates return to guest user level.

Binary Translation

- VMWare solved the problem for x86 virtualization by using binary translation and released their product in 1999
- There are four privilege rings in x86 - 0 to 3. User mode applications run in ring 3, host kernel and VMM in ring 0 and guest kernel in ring 1.

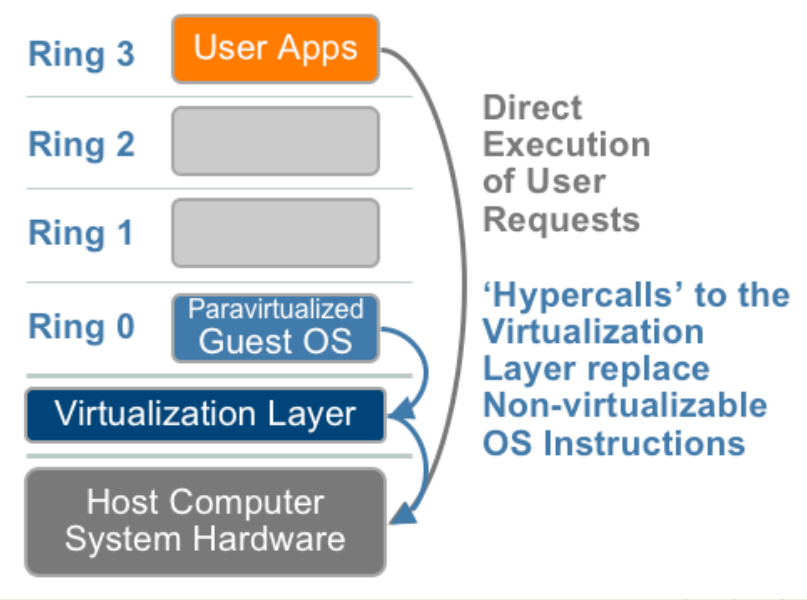


Binary Translation

- User mode code runs directly without any translation
- VMM provides a virtual hardware layer to the guest OS
- Binary code of the guest OS kernel is translated on the fly and replaces privileged code instructions with safer code or with its own system calls which does the desired effect on the virtualized hardware
- Thus, all the syscalls are intercepted by the VMM which then transfers them to the host OS

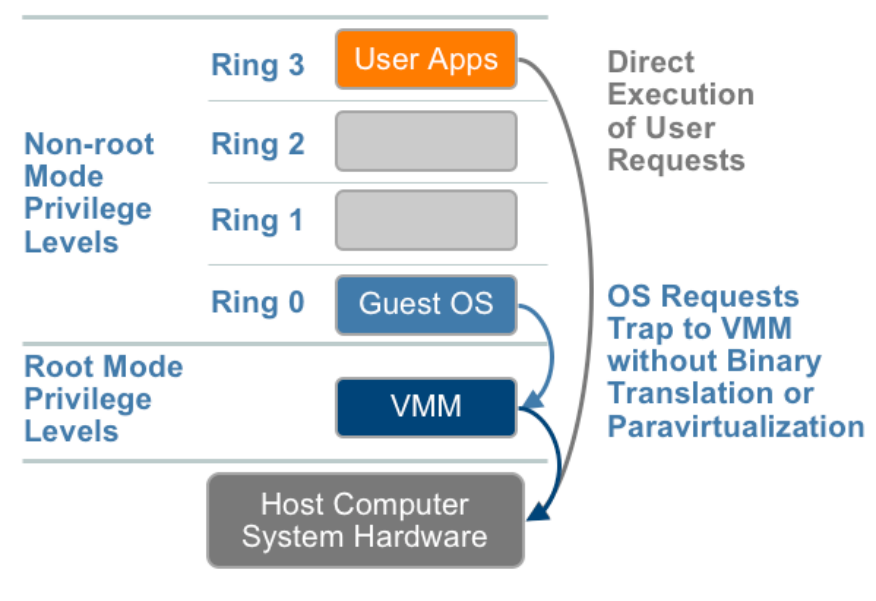
Paravirtualization

- Proposed in 2003 as a better/faster alternative to Full Virtualization along with an implementation which was open sources as Xen.
- Replaces the non-virtualizable/privileged instructions in the source code with hypercalls which directly call the VMM
- Requires modification of the guest kernel
- The hypervisor provides hypercall interfaces for critical kernel operations such as memory management, interrupt handling, and time keeping.



Hardware Assisted Virtualization

- Hardware Assisted virtualization was released in 2005. Both Intel(Intel VT-x) and AMD(AMD-V) have technologies for this.
- Introduces a new root mode below ring 0 where VMM runs. The guest OS now runs at Ring 0.
- Privileged and sensitive calls are set to automatically trap to the VMM, removing the need for either binary translation or paravirtualization



Hardware Assisted Virtualization

- Certain instructions and events cause VM to exit to VMM, which emulates the operation.
- Some new instructions have also been introduced.
- The guest state is stored in Virtual Machine Control Structures (VT-x) or Virtual Machine Control Blocks (AMD-V). Controls what operations trap, records info to handle traps in VMM.
- Does not provide performance gains because each transition from the VM to the VMM (VMexit) and back (VMentry) requires a fixed (and large) number of CPU cycles
- More detailed information given at - http://linux.linti.unlp.edu.ar/images/f/f1/Vtx.pdf

Memory Virtualization

- The hypervisor will have to provide each of the guest OSs with the illusion of a more or less contiguous block of physical memory starting at address 0. Guest OS assumes full control over memory.
- The simplest way to do that is to introduce one more level of virtualization.
  - A mapping between what the guest OS considers as a physical address and the actual physical address.
  - This map is maintained by the VMM.
- But, hardware managed TLB's makes it difficult.
  - When TLB misses the hardware automatically walks through the page tables.
  - So, VMM needs to control access by OS to the page tables

Shadow Page Tables

- The VMM also maintains a shadow page table
  - It contains a direct map from guest OS process Virtual address to host physical address.
  - The shadow page table is used by the hardware MMU for translation. (CR3 register which points to page table of a process points to this page table in case of virtualization).
- VMM needs to keep its shadow page table consistent
  - VMM maps guest OS page tables as read only
  - When OS writes to page tables, trap to VMM
  - VMM modifies guest OS table and shadow page table
- Very high overhead
- For every switch between VM's, whole TLB has to be flushed
- Second generation hardware assisted virtualization has Extended Page Tables(Intel) and Nested Page Tables(AMD) which provides significant performance gains

EPT and NPT

- Original Page tables mapping guest virtual to guest physical managed by guest OS
- New Page tables mapping guest physical to host physical managed by the VMM
- No shadow page tables, so no need to trap to VMM when OS updates its page tables
- Hardware MMU provides support for lookup from two levels of page tables
- Tagged TLB's
  - No need to flush the whole TLB on VM switch

Page Faults and memory allocation

- When page fault occurs, VMM finds physical page and corresponding guest page table entry. Two possibilities:
  - present bit is 0 in the guest page table entry: this fault must be reflected to the guest OS:
    - Simulate page fault for guest OS (similar to kernel call).
    - Guest OS invokes I/O to load page into guest physical memory.
    - Guest OS sets present bit to 1 in guest page table entry.
    - Guest OS returns from page fault, which traps into VMM again (like returning from kernel call).
    - VMM sees that present is 1 in guest page table entry, finds corresponding physical page, creates entry in shadow page table.
    - VMM returns from the original page fault, causing guest application to retry the reference.

Page faults and memory allocation

- present is 1 in the guest page table entry: guest OS thinks page is present in guest physical memory (but VMM may have swapped it out anyway).
  - VMM locates the corresponding physical page, loading it in memory if needed.
  - VMM creates entry in shadow page table.
  - VMM returns from the original page fault, causing guest application to retry the reference.
  - In this situation the page fault is invisible to the guest OS.

- The VMM allocates host physical memory on its first accesses to the memory.
- Memory deallocation
  - guest OS frees physical pages by adding them to its free list
  - Since VMM cannot access the free list of guest OS, it doesn't know that a physical page has been freed and it remains allocated to guest VM.

Memory in Paravirtualization

- Hardware MMU uses the pages of the guest OS directly
- VMM validates all the updates by the guest OS to the guest page table
- The page tables works as before, but the OS os constrained to use only the physical pages it ‘owns’
- Guest OS source code needs to be modified.

Memory Ballooning

- Memory of the host system can be overcommitted.
- Guest memory can be adjusted according to workload
- Requires some mechanism to reclaim memory from other guests
- Balloon Driver
  - Resides on the guest OS
  - It allocates memory from the guest OS(runs as a process on the guest OS) and then hands it back to the host. This is known as balloon inflation. Reverse is balloon deflation
  - Two parameters for each VM - maxmem and memory.
  - maxmem specifies the maximum memory which can be allocated to the guest
  - memory is the current memory of the guest.

Existing Feature Support

- KVM and Xen support ballooning. Only Xen supports auto ballooning. In KVM, auto ballooning is still in experimental stage.
- KVM does not support memory hotplug. Xen supports memory hot add but not hot remove.
- VMWare vSphere has a Distributed Resource Scheduler which is capable of doing automatic Hotspot Mitigation, Load Balancing and consolidation by automatic VM Migration and memory Ballooning

References

[1] Popek, G. and Kgoldberg, R. “Formal Requirements for Virtualizable Third-Generation Architectures”, Communications of the ACM, 1974
[2] VMWare White Paper, “Understanding Full Virtualization, Paravirtualization, and Hardware Assist”, 2007.
[3] Barham , P. et al. “Xen and the Art of Virtualization”, SOSP 2003
[4] Johan De Gelas “Hardware Virtualization: Nuts & Bolts”
[5] John Ousterhout http://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=vmm
[6] Adams, K. and Agesen, O. “A Comparison of Software and Hardware Techniques for x86 Virtualization”, ASPLOS 2006
[7] VMWare White Paper, “Understanding Memory Resource Management in VMWare vSphere 5.0”,