Virtual Machines

CS 256/456

Dept. of Computer Science, University of Rochester

(slides developed by Brandon Shroyer, Sandhya Dwarkadas)

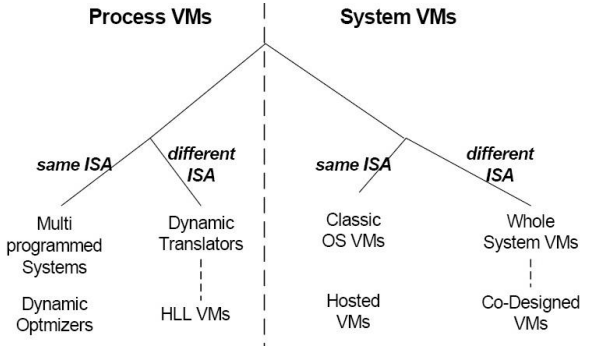
Virtualization

- *Virtualization*: Providing an interface to software that maps to some underlying system.
  - A one-to-one mapping between a guest and the host on which it runs [9, 10].
- Virtualized system should be an "efficient, isolated duplicate" [8] of the real one.
- *Process virtual machine* just supports a process; *system virtual machine* supports an entire system.

Why Virtualize?

- Reasons for Virtualization
  - Hardware Economy
  - Versatility
  - Environment Specialization
  - Security
  - Safe Kernel Development
  - OS Research [12]

A Taxonomy of Virtual Machine Architectures



History

- VM/370
  - Developed by IBM for OS/360 in the 1970s.
  - Introduced timesharing.
  - Provided multiprogramming and an extended machine with a more convenient interface than bare hardware.

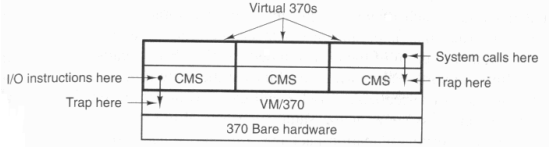
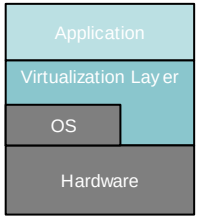


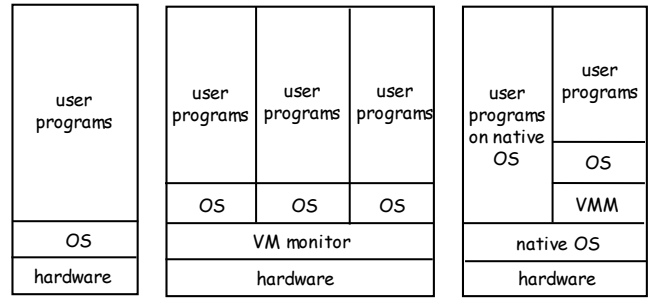
Figure 1-28. The structure of VM/370 with CMS.

Process Virtualization

- VM interfaces with single process
- Application sees "virtual machine" as address space, registers, and instruction set [10].
- Examples:
  - Multiprogramming
  - Emulation for binaries
  - High-level language VMMs (e.g., JVM)



System Virtual Machines



Non-VM Support | Native VM Support | Hosted VM Support

OS sees VM as an actual machine – raw hardware – CPU, memory, I/O

Emulation

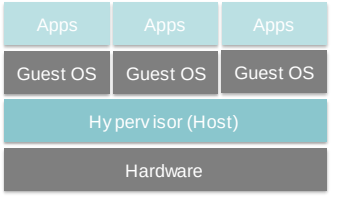
- Providing an interface to a system so that it can run on a system with a different interface [10].
  - Lets compiled binaries, OSes run on architectures with different ISA (binary translation)
  - Performance usually worse than classic virtualization.
- Example: QEMU [11]
  - Breaks CPU instructions into small ops, coded in C.
  - C code is compiled into small objects on native ISA.
  - dyngen utility runs code by dynamically stitching objects together (dynamic code generation).

OS Virtualization: Some Important Terms

- *Virtual Machine (VM)*: An instance of of an operating system running on a virtualized system. Also known as a *virtual* or *guest OS.*
- *hypervisor:* The underlying virtualization system sitting between the guest OSes and the hardware. Also known as a *Virtual Machine Monitor (VMM).*
  - The analogous construct in process virtualization is the runtime (the JVM, for instance).

Guest OS Model

- Hypervisor exists as a layer between the operating systems and the hardware.
- Performs memory management and scheduling required to coordinate multiple operating systems.
- May also have a separate controlling interface.



Virtualization Approach – Direct Execution

- Directly executing VM code to attain high speed
- CPU virtualization
  - VM monitor catches timer interrupts and switches VM if necessary
- I/O access virtualization
  - cause a trap to VM monitor, which processes appropriately
  - extra overhead is not too bad
- Memory virtualization
  - a trap at each memory access is not a very good idea
  - How?

System VMs – Processors

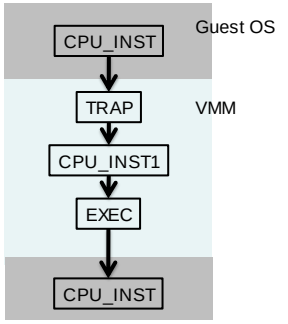
- Instruction execution can be through interpretation or binary translation. Can also use direct native execution (only if ISAs are identical)
- Must address issue of "sensitive" and "privileged" instruction references

Trap and Emulate Basics [8]

- User and superuser modes in [8] roughly correspond to guest OS & VMM modes
  - Analogous to user and kernel modes
- *Trap:* To switch to VMM mode
- *Sensitive Instruction:* Instruction that must be executed in superuser (VMM) mode
  - Instructions that manage system resources or switch modes are sensitive
  - Instructions affected by location in memory are also sensitive
- *Privileged Instruction:* Any instruction which traps when executed in user (OS) mode.
  - Instructions that update memory are privileged, also sensitive
- All sensitive instructions *must* be privileged instructions [8]

Performance

- Modern VMMs based around trap-and-emulate [8].
- When a guest OS executes a privileged instruction, control is passed to VMM (VMM "traps" on instruction), which decides how to handle instruction [8].
- VMM generates instructions to handle trapped instruction (emulation).
- Non-privileged instructions do not trap (system stays in guest context).



Challenges – Instruction Architecture

- "Sensitive" – May only be executed in kernel mode
- "Privileged" – Cause a trap if executed in user mode
  - "Trap" – Switch to kernel mode for execution.
- For a system to be virtualizable, the sensitive instructions must be a subset of the privileged instructions (Popek and Goldberg, 1974)
- Historically, not all ISAs provided this guarantee

Trap-and-Emulate Problems

- Trap-and-emulate is expensive
  - Requires context-switch from guest OS mode to VMM.
- x86 is not trap-friendly
  - Guest's CPL privilege level is visible in hardware registers; cannot change it in a way that the guest OS cannot detect [5].
  - Some instructions are not privileged, but access privileged systems
    - Example: instructions that access page tables do not require 0-level CPL privilege, but anything that affects the MMU does [5].
      - A read is fair game for the user, but if it causes a page fault, a new page has to be fetched by the MMU

Virtualizing Privileged Instructions

- x86 architecture has four privilege levels (rings).
- The OS assumes it will be executing in Ring 0.
- Many system calls require 0-level privileges to execute.
- Any virtualization strategy must find a way to circumvent this.

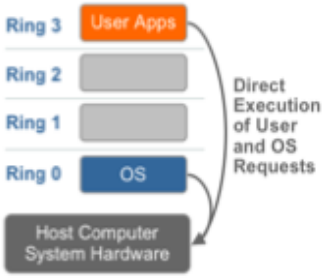


Image Source: VMWare White Paper, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist", 2007.

ISA Challenge: Potential Solution

- Paravirtualization
  - Remove (some or all) sensitive instructions from the guest OS and replace them with hypervisor calls
  - The VMM basically acts as a microkernel by emulating guest OS system calls

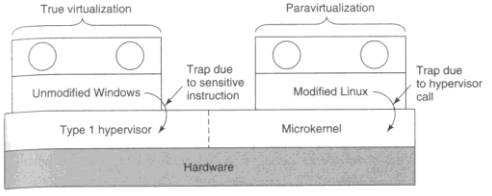


Figure 8-27. A hypervisor supporting both true virtualization and paravirtualization.

Paravirtualization

- Replace certain unvirtualized sections of OS code with virtualization-friendly code.
- Virtual architecture "similar but not identical to the underlying architecture." [3]
- Advantages: easier, lower virtualization overhead
- Disadvantages: requires modifications to guest OS

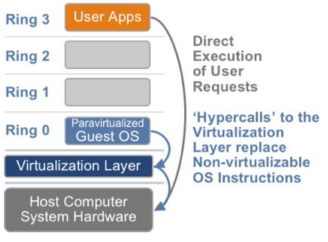


Image Source: VMWare White Paper, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist", 2007.

Full Virtualization

- "Hardware is functionally identical to underlying architecture." [3]
- Typically accomplished through interpretation or binary translation.
- Advantage: Guest OS will run without any changes to source code.
- Disadvantage: Complex, usually slower than paravirtualization.

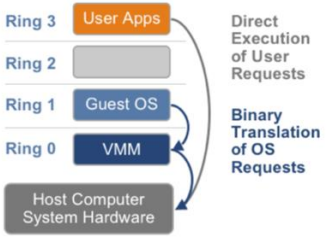


Image Source: VMWare White Paper, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist", 2007.

ISA Challenge: Potential Solution

- Architecture design can limit potential for virtualizability
  - Some ISAs have instructions that can read sensitive information without trapping (Ex: Pentium)
- Solution: Design from the start with virtualization in mind
  - Ex: Intel Core 2 Duo (VT) and AMD Pacific (SVM)

Hardware-Assisted Virtualization

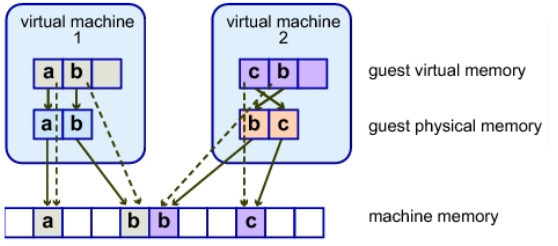
- Hardware virtualization-assist released in 2006 [5].
  - Intel, AMD both have technologies of this type.
- Introduces new VMX runtime mode.
  - Two modes: guest (for OS) and root (for VMM).
  - Each mode has all four CPL privilege levels available [8].
  - Switching from guest to VMM does not require changes in privilege level.
  - Root mode supports special VMX instructions.
  - Virtual machine control block [5] contains control flags and state information for active guest OS.
  - New CPU instructions for entering and exiting VMM mode.
- Does not support I/O virtualization.

Virtualizing Memory

- Virtualization software must find a way to handle paging requests of operating systems, keeping each set of pages separate.
- Memory virtualization must not impose too much overhead, or performance and scalability will be impaired.
- Guest OS must each have an address space, be convinced that it has access to the entire address space.
- SOLUTION: most modern VMMs add an additional layer of abstraction in address space [4].
  - *Machine Address*—bare hardware address.
  - *Physical Address*—VMM abstraction of machine address, used by guest Oses.
  - Guest maintains virtual-to-physical page tables.
  - VMM maintains pmap structure containing physical-to-machine page mappings.

System VMs – Memory

- Example: VMware's ESX Server



Memory Virtualization Under Direct Execution (protected page table)

- From the VM OS's view, the page table contains mapping from virtual to VM physical addresses
- For proper operation, the page table hooked up with MMU must map virtual to real machine addresses
- VM OS cannot directly access the page table
  - each page table read is trapped by VM monitor, the physical address field is translated (from real machine address to VM physical address)
  - each page table write is also trapped, for a reverse translation and for security checking

Memory Virtualization Under Direct Execution (shadow page table)

- VM OS maintains virtual to VM physical (V2P) page table
- VM monitor
  - maintains a VM physical to machine (P2M) mapping table
  - combines V2P and P2M table into a virtual to machine mapping table (V2M)
  - supplies the V2M table to the MMU hardware
- Page table updates
  - any VM change on its V2P page table must be trapped by VM monitor
  - VM monitor modifies V2M table appropriately

System VMs – I/O

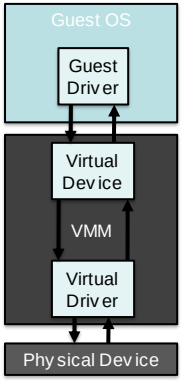
- Challenging for VMM, but can adapt techniques from time-sharing of I/O devices on typical systems
- Create a virtualized version of system devices. VMM intercepts request by guest VM and converts the request to equivalent physical device request

System VMs – I/O

- The VMM can catch and virtualize the I/O action at three levels:
  - I/O operation level
  - device driver level
  - system call level
- Virtualizing at the device driver level is most practical

I/O Virtualization

- Performance is critical for virtualized I/O
  - Many I/O devices are time-sensitive or require low latency [7].
- Most common method: device emulation
  - VMM presents guest OS with a virtual device [7].
  - Preserves security, handles concurrency, but imposes more overhead.



I/O Virtualization Problems

- Multiplexing
  - How to share hardware access among multiple OSes.
- Switching Expense
  - Low-level I/O functionality happens at the VMM level, requiring a context switch.

Packet Queuing

- Both major VMMs use an asynchronous ring buffer to store I/O descriptors.
- Batches I/O operations to minimize cost of world switches [7].
- Sends and receives exist in same buffer.
- If buffer fills up, an exit is triggered [7].

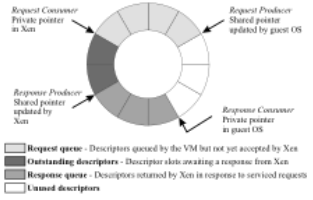


Image Source: Barham, P. et al. "Xen and the Art of Virtualization", SOSP 2003.

I/O Rings, continued

Xen

- Rings contain memory descriptors pointing to I/O buffer regions declared in guest address space.
- Guest and VMM deposit and remove messages using a producer-consumer model [2].
- Xen 3.0 places device drivers on their own virtual domains, minimizing the effect of driver crashes.

VMWare

- Ring buffer is constructed in and managed by VMM.
- If VMM detects a great deal of entries and exits, it starts queuing I/O requests in ring buffer [7].
- Next interrupt triggers transmission of accumulated messages.

References

1. Singh, A. "An Introduction To Virtualization", www.kernelthread.com, 2004.
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. Waldspurger, C. "Memory Resource Management in VMware ESX Server", OSDI 2002.
5. Adams, K. and Agesen, O. "A Comparison of Software and Hardware Techniques for x86 Virtualization", ASPLOS 2006.
6. Pratt, I. et al. "Xen 3.0 and the Art of Virtualization", Linux Symposium 2005.
7. Sugerman, J. et al. "Virtualizing I/O Devices on Vmware Workstation's Hosted Virtual Machine Monitor", Usenix, 2001.
8. Popek, G. and Goldberg, R. "Formal Requirements for Virtualizable Third-Generation Architectures", Communications of the ACM, 1974.
9. Mahalingam, M. "I/O Architectures for Virtualization", VMWorld, 2006.
10. Smith, J. and Nair, R. *Virtual Machines*, Morgan Kaufmann, 2005.
11. Bellard, F. "QEMU, a Fast and Portable Translator", USENIX 2005.
12. Silberschatz, A., Galvin, P., Gagne, G. *Operating System Concepts*, Eighth Edition. Wiley & Sons, 2009.

Sources

- "Modern Operating Systems," Tanenbaum
  - Chapters 1, 8
- "Virtual Machines," Smith and Nair
  - Chapters 1, 2, 3, 8
- VMware Resource Management Guide
  - http://pubs.vmware.com/vi301/resmgmt/wwhelp/wwhimpl/common/html/wwhelp.htm?context=resmgmt&file=vc_advanced_mgmt.11.16.html
- "Survey of Virtual Machine Research," Robert P. Goldberg, 1974.