

## Virtual Machines

CS 256/456  
Dept. of Computer Science, University  
of Rochester

12/7/2010

CSC 2/456

1

## Virtual Machine Architectures

- Allow a system or execution platform to appear to be a different (or multiple) platform(s)
  - Implemented by adding a software layer

12/7/2010

CSC 2/456

2

## Why Virtualize?

- Emulation
- Optimization
- Server consolidation
  - Robust level of security
  - Reliability
- Kernel development

12/7/2010

CSC 2/456

3

## Why Virtual Machines?

- Allow flexible management of "machines" at software level
  - experimenting with new architecture
  - debugging an OS
  - checkpointing and migrating all state on a machine
- Enhanced reliability and security
  - VM monitor much smaller than OS, therefore:
  - the full privileged code base (VM monitor) is small
  - the trusted code base (VM monitor) is small
- Strong isolation between VMs
  - fault and resource isolation
  - your Qemu/Linux assignment

12/7/2010

CSC 2/456

4

### 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.

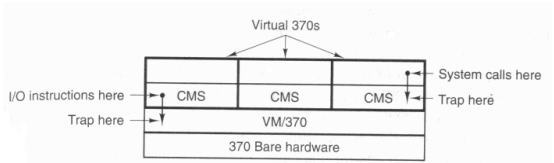
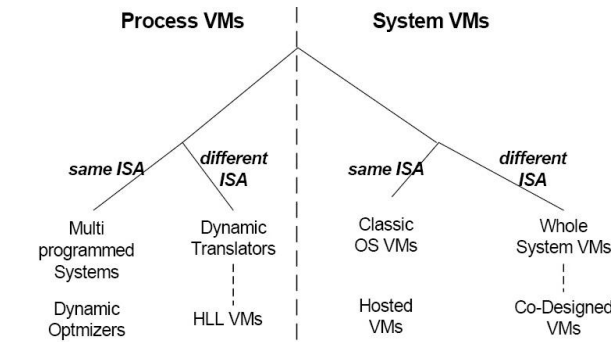


Figure 1-28. The structure of VM/370 with CMS.

### A Taxonomy of Virtual Machine Architectures



12/7/2010

CSC 2/456

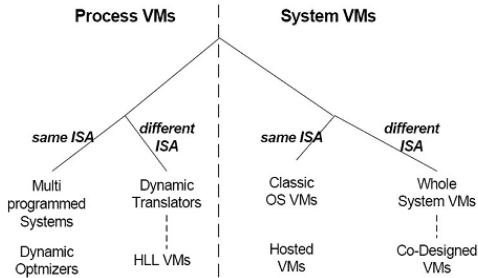
6

### Applications

- Cheaper fault tolerance than independent systems
- Strong isolation
- Environment portability
- Easier load balancing
- Running legacy applications
- Software development

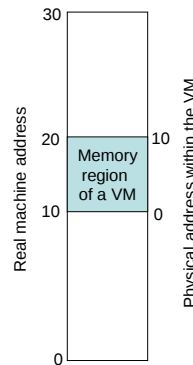
### Types of Virtualization

- Process VM vs. System VM



## Virtualization Challenges

- CPU virtualization
  - how to switch out a VM?
- Memory virtualization
  - VM physical memory address may not be real machine address
  - a VM's memory access must be restricted
- I/O virtualization
  - similar issues with memory virtualization



12/7/2010

CSC 2/456

9

## Virtualization Approach - Interpretation

- Do not directly run VM code  $\Rightarrow$  Interpretation
  - inspect each instruction in software and realize its intended effects using software
  - Nachos VM does this
- CPU virtualization
- Memory virtualization
- I/O virtualization
- Problem: too slow!

12/7/2010

CSC 2/456

10

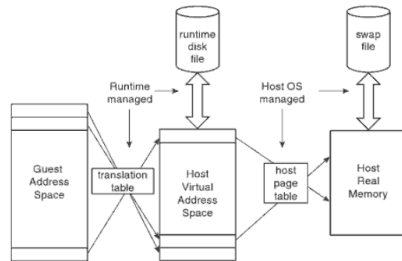
## Process VMs

- Virtualization of single processes
- ABI = application binary interface
  - Combination of ISA and operating system (ex., x86 and Windows)
- Instruction emulation: interpretation vs. binary translation
- I/O is similar to non-virtualized processes.

## Process VMs – Memory

- Memory address space mapping
  - Runtime software-supported translation tables:
    - Use a software table, similar to a page table, to map corresponding memory addresses.
    - Similar to the process of virtualizing hardware memory.
  - Alternative: Direct mapping to host virtual address space.

## Process VMs – Memory



**Figure 3.8** Mapping a Guest Address Space into a Region of the Host Space. The runtime software manages the region of host address space in much the same way as the host OS manages its real memory.

## Process VMs – Memory

- Memory architecture emulation:
  - Three aspects to consider
    - 1) The overall structure of the address space (segmented vs. flat, linear)
    - 2) The supported access privilege types (R,W,X)
    - 3) The protection/allocation granularity (e.g., page sizes)

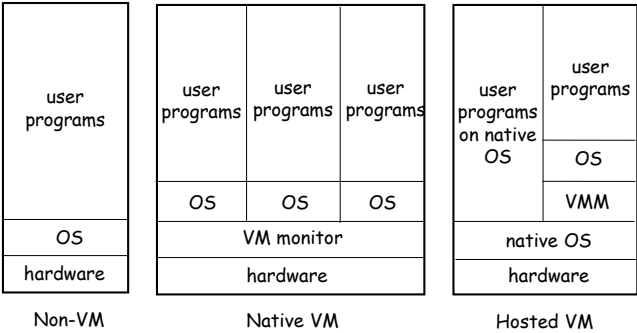
## Process VMs

- OS Emulation
  - Same-OS emulation:
    - May need to translate function parameters and return values for system calls
    - Some system calls may be handled by the runtime implementation
  - Different-OS emulation:
    - Harder than emulating different ISAs
    - Some OS functions required by the guest may not be possible given the host OS
    - Case-by-case process is necessary due to wide range of systems

## System VMs

- Similar to the concept of time-sharing among processes
- Capable of supporting multiple system images simultaneously
- Virtual Machine Monitor (VMM): A layer of software between the host platform and guest VM that manages allocation of and access to the hardware resources of the host

Virtual Machine Architecture



12/7/2010

CSC 2/456

17

Virtual Machine Transparency

- Full transparency (perfect virtualization):
  - stock OS (without change) can run within VM
  - VMware
- Less-than-full transparency (para-virtualization):
  - modified OS runs within VM
  - Xen
  - for performance (memory virtualization)
    - batched page table accesses through explicit monitor calls
  - for simplicity (I/O virtualization)

12/7/2010

CSC 2/456

18

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

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)

## 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

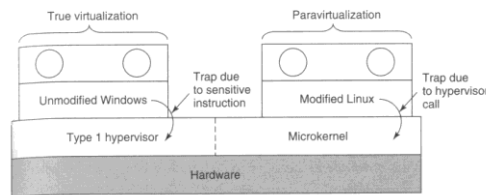


Figure 8-27. A hypervisor supporting both true virtualization and paravirtualization.

## 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)

## 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?

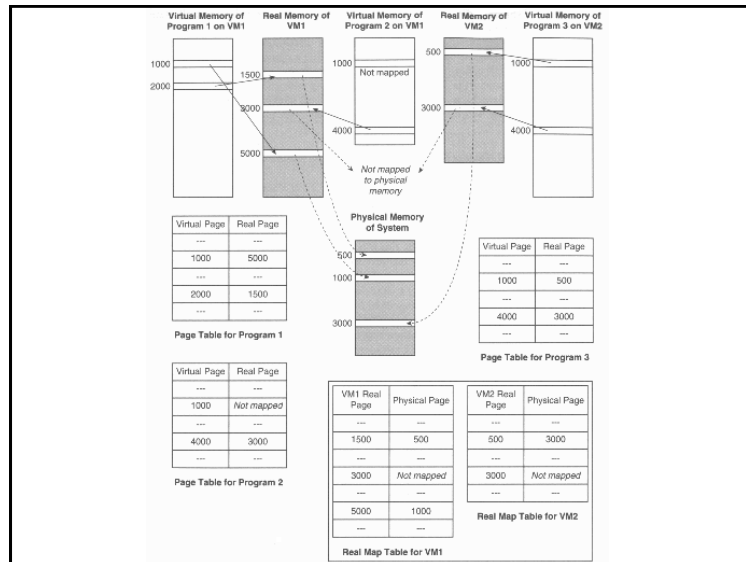
12/7/2010

CSC 2/456

23

## System VMs – Memory

- Each guest VM has its own set of virtual memory tables
- VM virtual addresses are translated to “real” memory (a mapping kept by the VMM), which maps to physical memory
  - Adds an extra level of address translation



## System VMs – Memory

- For ISA-implemented page tables:
  - VMM keeps “shadow page tables,” one for each guest VM
    - Used by the hardware in address translation and TLB caching
    - Page fault handling must take into account differences between shadow page tables and the guest VM page tables

## System VMs – Memory

- Example: VMware's ESX Server
  - VMM intercepts virtual machine instructions that manipulate guest memory so that the actual physical MMU is not directly updated
  - Host ESX Server keeps virtual-to-machine page mappings in shadow page tables, which are updated with the physical-to-machine mappings
  - Shadow page tables then used by the processor's paging hardware

## 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

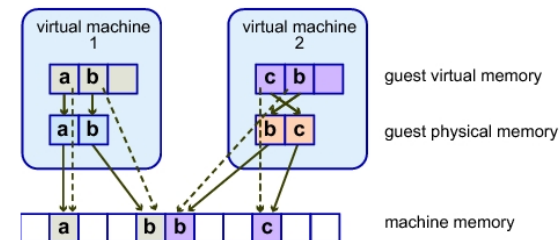
12/7/2010

CSC 2/456

29

## System VMs – Memory

- Example: VMware's ESX Server



## 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

## 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](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.