Virtual Machines

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

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

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:
      - Full privileged code base (VM monitor) is small
      - Trusted code base (VM monitor) is small
  - Strong isolation between VMs
    - Fault and resource isolation
    - Your Qemu/Linux assignment
### 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.

![Diagram of VM/370 architecture](image)

*Figure 1-2k. The structure of VM/370 with CMS.*

### Applications

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

### A Taxonomy of Virtual Machine Architectures

- **Process VMs**
  - Same ISA
  - Different ISA
  - Multi-programmed Systems
  - Dynamic Translators
  - Dynamic Optimizers
  - HLL VMs

- **System VMs**
  - Same ISA
  - Different ISA
  - Classic OS VMs
  - Hosted VMs
  - Whole System VMs
  - Co-Designed VMs

### 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

Memory region of a VM

<table>
<thead>
<tr>
<th>Real machine address</th>
<th>Physical address within the VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Virtualization Approach - Interpretation

- Do not directly run VM code ⇒ 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!

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.
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)

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

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

- Non-VM
- VM

Native VM

- OS
- VM monitor
- hardware

Hosted VM

- user programs
- user programs
- user programs

OS

Virtual Machine Transperecy

- 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)

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

- 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?

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 pages 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

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