System Virtual Machines

• Introduction
• Key concepts
• Resource virtualization
  – processors
  – memory
  – I/O devices
• Performance issues
• Applications
Introduction

• System virtual machine
  – capable of supporting multiple system images
  – each guest OS manages a set of virtualized hardware resources
  – VMM owns the real system resources
  – VMM allocates resources to guest OS by partitioning or time-sharing
  – guest OS allocates resources to its user programs
  – each virtual resource may not have physical resource
  – see Figure 8.1
Introduction (2)
Key Concepts

• Outward appearance
• State management
• Resource control
• Native and hosted virtual machines

• Several concepts are simplified
  – same guest and host ISA
  – uniprocessor systems
Outward Appearance

• Create an illusion of multiple machines.
• Illusion in software
  – no replication of hardware resources
  – switch to move devices from one VM to the other
• Replication of hardware resources
  – devices used directly by each user are replicated
  – rest of the hardware is shared
• Hosted VM
  – one OS more important than the other
  – UI of host OS provides way to display UI of the second
State Management

• Refers to how each individual guest state is managed by the SVM
  – each guest OS has an architected state
  – host resources may not be adequate
• Fixed location for all guest state in host
  – VMM manages pointer and switching
  – indirection can be very inefficient (Figure 8.3a)
• Copy approach
  – more efficient (Figure 8.3b)
  – essential to allow direct execution of native code
State Management (2)

State Management (2)
Resource Control

• Refers to how the VMM maintains overall control of all hardware resources.
• Scheme similar to time-sharing OS
  – VMM intercept accesses to privileged resources
  – get control on all privileged instructions
  – VMM handles the timer interrupt (Figure 8.4)
  – similar tradeoffs between fair scheduling and large switching overhead
VMM Timesharing

- VMM timeshares resources among guests
  - similar to OS timesharing

![Diagram of VMM Timesharing process]

- Timer interrupt occurs
- VMM saves architected state of running VM
- VMM determines next VM to be activated
- VMM sets timer interval and enables interrupts
- VMM sets PC to timer interrupt handler of OS in next VM
- VMM restores architected state for next VM

First VM Active

VMM Active

Next VM Active
Native and Hosted VM

• Refers to the runtime privilege level of the VMM (and OS).
  – for efficiency, some part of VMM should have higher privileges

• Figure 8.5
  – analogous relationship to OS and user programs
  – *native VM system*: only the VMM operates in privilege mode
  – *hosted VM*: VMM installed on top of existing OS
    • some or no part of VMM in privilege mode
Native and Hosted VM (2)

Traditional uniprocessor system
Native VM system
User-mode Hosted VM system
Dual-mode Hosted VM system

Privileged modes
Non-privileged modes
Resource Virtualization – Processors

• Techniques to execute guest instructions
  – emulation
  – direct native execution
• Emulation
  – interpretation or binary translation
  – necessary if different host and guest ISAs
• Direct native execution
  – same host and guest ISAs required
  – may still require emulation of some instructions
Conditions for ISA Virtualizability

• Conditions laid out by Popek and Goldberg.
• Assume native system VMs
  – VMM runs in system mode
  – all other software runs in user mode
• Other assumptions
  – hardware consists of processor and uniformly addressable memory
  – processor can operate in two modes, user and system
  – some subset of instructions only available in system mode
  – relocation register available for memory addressing
Privileged Instructions

- Instructions that trap if the machine is in *user* mode, and does not trap if the machine is in *system* mode.
- LPSW (Load Processor Status Word)
  - PSW can change the state of the CPU
  - can also change the instruction address (PC)
- SPT (Set CPU Timer)
  - can change the CPU interval timer
Privileged Instructions (cont...)  

- **LRW (Load Real Address)**  
  - translate a virtual address, save corresponding real address in a specified GPR

- **POPF (Pop Stack into Flags Register)**  
  - replace flag register with top of memory stack  
  - *interrupt-enable* flag can only be modified in system mode  
  - no *trap* in user mode
Sensitive Instructions

- These specify instructions that interact with hardware.
- Control-sensitive instructions
  - can change the configuration of system resources
  - e.g., physical memory assigned to a program
  - e.g., LPSW – change mode of operation
  - e.g., SPT – change CPU timer value
Sensitive Instructions (cont...)

• Behavior-sensitive instructions
  – behavior or results depend on configuration of resources
  – e.g., LRA – depends on mapping (state) of real memory resource
  – e.g., POPF – depends on mode of operation

• Innocuous instructions
  – neither context-sensitive nor behavior-sensitive

• see Figure 8.6
Instruction Types – Summary

- Privileged
- Non-Privileged
- Sensitive
- Behavior-sensitive
- Control-sensitive
- Innocuous
Components of a VMM

Instruction trap occurs

Dispatcher

These instructions desire to change machine resources, e.g. Load Relocation Bounds Register

Allocator

These instructions do not change machine resources, but access privileged resources, e.g. IN, OUT, Write TLB

Interpreter Routine 1

Interpreter Routine 2

Interpreter Routine n
Components of a VMM (2)

- **Dispatcher**
  - control module, called on hardware traps
  - decides the next module to be invoked

- **Allocator**
  - decides allocation of system resources
  - invoked by dispatcher to change machine resource allocation for VMs

- **Interpreter routines**
  - emulates trap functionality for each user VM
  - all traps except resource re-assignment traps
Properties for a True VMM

• Efficiency
  – All *innocuous* instructions *must* be natively executed

• Resource control
  – impossible for guest s/w to *directly* change system resource configurations

• Equivalence
  – guaranty of identical behavior
  – allowed exceptions
    • performance can be slower
    • available resources can be limited
    • differences in performance due to changed timings
Requirement for True VMM

• For any third-generation computer, a VMM may be constructed if the set of sensitive instructions for a computer is a subset of the set of privileged instructions
  – sensitive instructions always trap in user mode
  – non-privileged instructions execute natively
Handling Privileged Instruction

- Privileged instruction traps in the guest OS

**Guest OS code in VM**
*user mode*

- Privileged instruction (LPSW)
- ...
- ...
- ...
- Next instruction (target of LPSW)

**VMM code**
*privileged mode*

- **Dispatcher**

  **LPSW Routine:**
  - Change mode to privileged
  - Check privilege level in VM
  - Emulate instruction
  - Compute target
  - Restore mode to user
  - Jump to target
Problem Instructions

• Sensitive instructions that are not privileged
  – POPF instruction in the Intel IA-32
  – IA-32 not virtualizable by Popek-Goldberg rules

• Handling problem instructions
  – interpret all guest software
  – scan and patch problem instructions before execution
  – see Figure 8.11
Patching Problem Instructions

Original Program

Scanner and Patcher

Code Patch for discovered critical instruction

Control transfer, e.g., trap

Patched Program

VMM
Patching Problem Instructions (2)

- VMM takes control at the head of each basic block.
- Scan to find problem instructions.
- Replace with trap to VMM.
- Place another trap at end of basic block.
- Patched basic blocks can be *chained*.
- Mostly similar to binary translation.
Caching Emulation Code

- High frequency of sensitive instructions requiring interpretation increases VMM overhead
  - cache interpreter actions in code cache
  - cache block containing the problem instruction
- Each instance of the problem instruction in the code associated with a distinct piece of cache code
  - make code instance-specific and optimize
- Simpler management of cached code
  - cached code executed in system mode
Resource Virtualization – Memory

• Generalizes the concept of virtual memory
• Virtual memory basics
  – application provided a logical view of memory
  – OS manages actual real memory
  – page tables provide logical-physical mapping
  – TLBs cache most common mappings
• VMM distinguishes between real memory and physical memory
  – VMM does real-physical memory mapping
VMM Memory Support

- VMM mechanism to virtualize memory
  - maintain a per-VM *real-map table*
  - maps real pages to physical pages
  - VMM maintains a distinct swap space

- Additional indirection layer may be inefficient
  - virtualize architected page tables
  - virtualize architected TLB
Resource Virtualization – I/O

• Virtualization of I/O is difficult
  – large number of I/O devices
  – each I/O device controlled in specific manner
  – OS have to deal with similar issues

• Virtualization Scheme
  – provide VMs with virtual version of a device
  – intercept VM requests made to virtual device
  – convert request, perform equivalent action
Virtualizing Devices

• Dedicated devices
  – display, keyboard, mouse etc., for each user
  – no device virtualization necessary
  – device requests could bypass the VM

• Partitioned devices
  – smaller *virtual* disks for each user
  – VMM use map to translate parameters
  – status information from the device also needs translation and reflection in the map
Virtualizing Devices (cont...)

- **Shared devices**
  - network adapter, virtual network address for a VM
  - maintain state information for each guest VM
  - translate device outgoing/incoming requests

- **Spooled devices**
  - shared at higher granularity, e.g., printer
  - two-level spool tables in OS and VMM
  - VMM schedules requests from multiple guest VMs
Performance Degradation

- Setup
  - initialize the state of the machine
- Emulation
  - sensitive instructions, maybe others
- Interrupt handling
  - handling interrupts generated by the guest programs
- State saving
  - save/restore state of VM during switch to VMM
- Book-keeping
  - special operations to maintain equivalent behavior
- Time elongation
  - some instructions may require more processing time
VM Assists

• Piece of hardware that improves performance of an application when running on a VM.
• May not always improve performance
  – improve user program-VMM system-mode switches
  – improve address translation performance
• VM assists can help improve performance of
  – instruction emulation
  – other aspects of VMM
  – the user-mode system running as the guest
  – specific types of guest systems
Instruction Emulation Assists

• Emulation of privileged instructions in the guest VM is a cause of fundamental overhead
  – causes interrupt that is handled by the VMM
  – VMM emulation depends on whether guest VM in user or system mode when instruction called
• Assist for LPSW on system/370
  – check state of guest VM
  – provides hardware-assisted instructions to modify physical resources
• Depends on VM system implementation
• The overhead of the trap still remains
  – eliminates overhead of emulation and mode switching
VMM Assists

• Context switch
  – hardware to save/restore registers, machine state

• Decoding of privileged instructions
  – helps certain critical parts of the emulation

• Virtual interval timer
  – precise timer for all guest VMs to schedule jobs

• Adding to the instruction set
  – to help commonly executed VMM parts
Improve Guest VM Performance

• Requires guest system to be VMM aware
  – avoid duplication of functionality
  – provide VMM with additional information (handshaking), e.g., DIAGNOSE instruction

• Example; non-paged mode
  – disable dynamic address translation in guest VM
  – no translation from virtual to real address
  – translation from real to physical done by VMM

• Paravirtualization
  – modify guest OS to work around difficult to virtualize ISA features
  – Xen for the IA-32, eliminates code detection, patching, shadow page tables
Applications

- Implementing multiprogramming
- Multiple single-application virtual machines
- Multiple secure environments
- Mixed-OS environments
- Legacy applications
- Multiplatform application development
- Gradual migration to new system
- New system software development
- Operating system training
- Help desk support
- Operating system instrumentation
- Event monitoring
- System encapsulation and checkpointing