System VMs

Resource Control

- Issue: How to retain control of resources in the VMM?
- Timer interval control performed by VMM
- Also, guest OS not allowed to read the timer value
  - Guest OS sees a *virtual interval timer*
- VMM also gains control whenever guest OS executes privileged instructions.

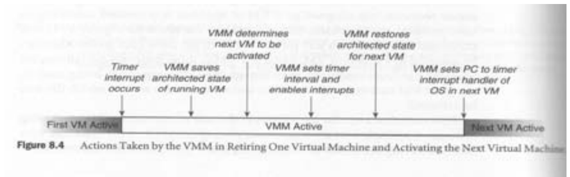


Figure 8.4 Actions Taken by the VMM in Retiring One Virtual Machine and Activating the Next Virtual Machine

Instruction Types

- Non-privileged: Do not cause traps
- Privileged : Cause Traps
- Sensitive: Change/depend upon system state
- Innocuous: Not "sensitive"

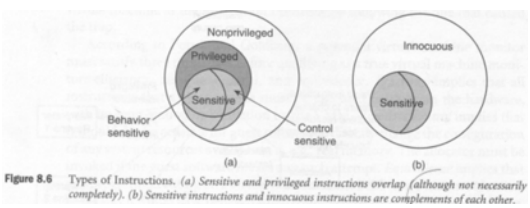


Figure 8.6 Types of Instructions. *(a) Sensitive and privileged instructions overlap (although not necessarily completely). (b) Sensitive instructions and innocuous instructions are complements of each other.*

Conditions of ISA Virtualizability

- Theorem: A computer architecture is ***fully*** virtualizable if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

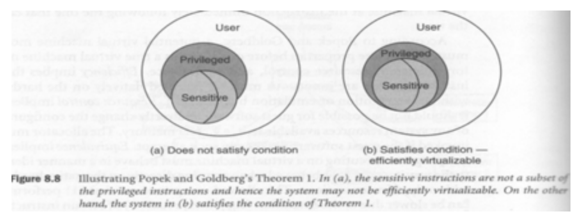
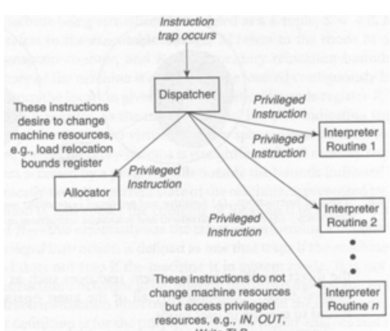
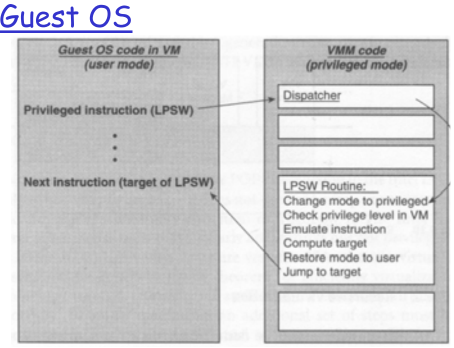


Figure 8.8 Illustrating Popek and Goldberg's Theorem 1. *In (a), the sensitive instructions are not a subset of the privileged instructions and hence the system may not be efficiently virtualizable. On the other hand, the system in (b) satisfies the condition of Theorem 1.*

Execution of Privileged Instruction



Handling Privileged Instructions in a Guest OS



Handling Problem Instructions - Sensitive But Not Privileged

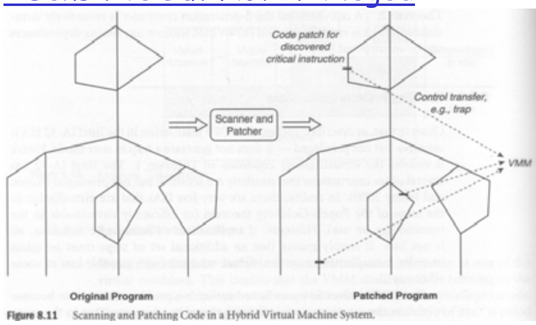
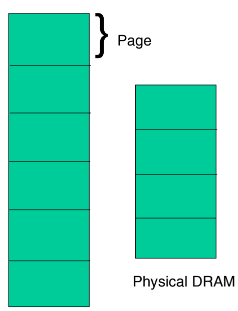


Figure 8.11 Scanning and Patching Code in a Hybrid Virtual Machine System.

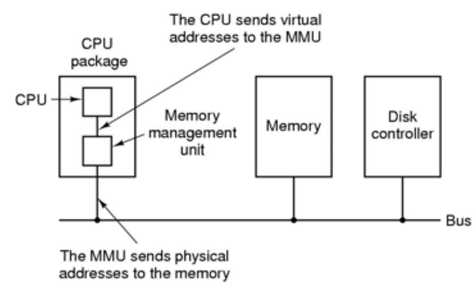
Memory Virtualization

Virtual Memory Basics

- Sometimes, a single process might require more memory than the available DRAM in the system.
- In such cases swapping is not enough. Rather, we need to break up the memory space of a process into smaller equal-sized pieces (called pages).
- Operating system then decides which pages stay in memory and which get moved to disk.
- Virtual memory: means that each process gets an illusion that it has more memory than the physical DRAM in the system.



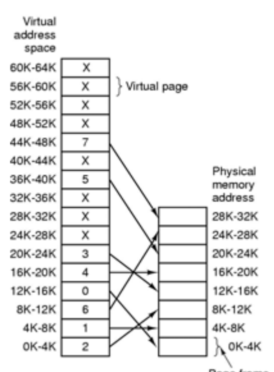
MMU and Virtual Memory



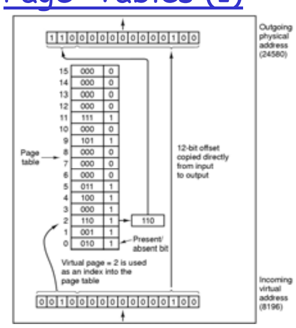
- MMU = Memory Management Unit
- Part of Hardware that accompanies the CPU
- Converts Virtual Addresses to Physical Addresses

Paging

The mapping between virtual page numbers and physical page numbers is stored in the page table

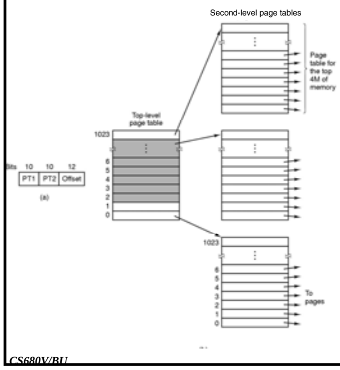


Page Tables (1)



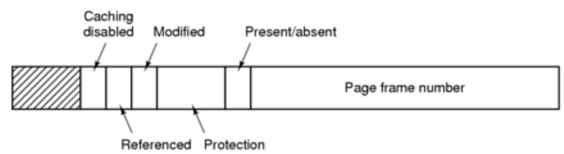
Internal operation of MMU with 16 4 KB pages

Page Tables (2)



- 32 bit address with 2 page table fields
- Two-level page tables
- PT too Big for MMU
  - Place it in main memory
- But how does MMU know where to find PT?
  - Registers (CR2 on Intel)

Page Tables (3)



Typical page table entry

TLBs – Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- TLB is a small cache that speeds up the translation of virtual addresses to physical addresses.
- TLB is part of the MMU hardware (comes with CPU)
- It is not a Data Cache or Instruction Cache. Those are separate.
- TLB simply caches translations from virtual page number to physical page number so that the MMU don't have to access page-table in memory too often.
- On x86 architecture, TLB has to be "flushed" upon every context switch because there is no field in TLB to identify the process context.

Impact of Page Size on Page tables

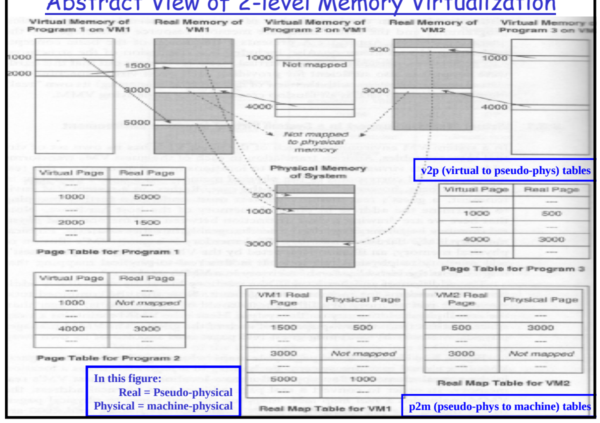
Small page size

- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
- Disadvantages
  - If a process needs more pages, then it has larger page table

Memory Virtualization for VMs

- Not the same as virtual memory for processes
  - But very similar
  - Just another layer of indirection
- Address Translation process
  - Level 1: Guest VM page table
    - Virtual address ➔pseudo-physical address
  - Level 2: VMM Page table
  - Pseudo-physical address ➔ machine-physical address
- Sum of real memory of all guest VMs ***could be*** greater than physical memory of the system
  - Thus VMM may have its own swap space, independent of what guest VMs have.

Abstract View of 2-level Memory Virtualization

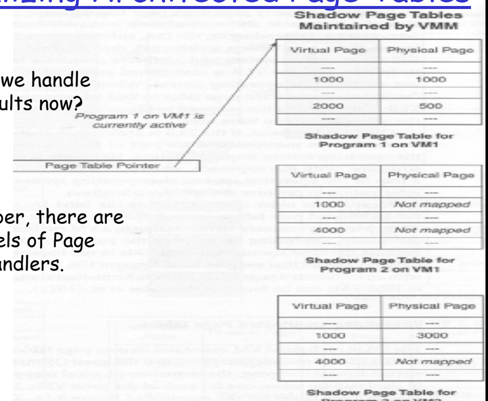


Two Architectures for implementing the abstract view

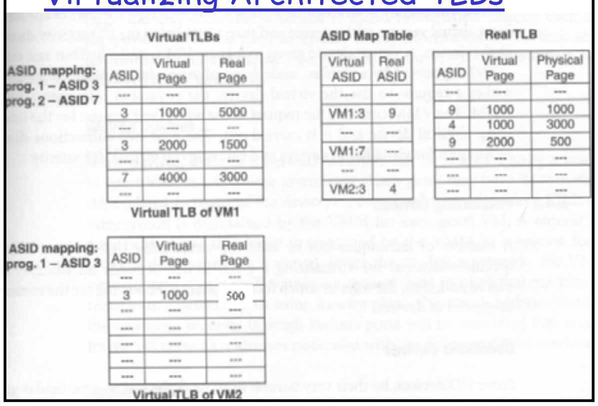
- Architected Page Tables
  - Page table interface defined by ISA and understood by memory translation hardware
  - E.g. x86 architecture
- Architected TLBs
  - TLB interface defined by ISA and understood by memory translation hardware
  - E.g. alpha architecture

Virtualizing Architected Page Tables

- How do we handle page-faults now?
- Remember, there are two-levels of Page Fault handlers.



Virtualizing Architected TLBs



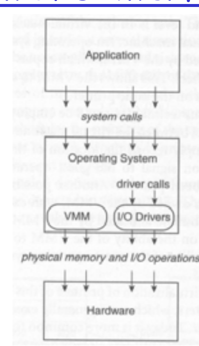
I/O Virtualization

- A number of different types of I/O devices
- Construct a virtual version of the device
- I/O activity directed at the device is intercepted by VMM and converted to equivalent request for underlying physical device.

Device Types

- Dedicated Device
  - Display, keyboard, mouse etc.
  - VMM routes, but does not interpret the I/O instructions
- Partitioned Devices
  - E.g. A hard disk can host several virtual disks
- Shared Devices
  - E.g. network adapter
- Spooled Devices,
  - E.g. printers
  - Shared, but at a higher granularity than shared devices
- Nonexistent Physical Devices
  - E.g. network adapter to communicate only among VMs
  - E.g. Virtual disk hosted on a non-disk physical device

Different I/O interfaces



Virtualizing at the I/O Operations Level

- Directly intercept I/O instructions below the device driver.
- Execute the effect of the instruction on physical device
- Difficult to reverse engineer the combination of individual I/O operations that device drivers intends to do.

Virtualizing at the Device Driver Level

- VMM has knowledge of the guest OS internals
- It inserts virtual drivers for each virtual device in the guest OS.
- Typically used in hosted VMs.

Virtualizing at the System Call Level

- VMM can intercept system calls for more efficient virtualization.
- However, VMM needs to know all the calls in ABI (application binary interface).
- Plus need to account for interaction between ABI and other parts of the OS.