Virtual Machines (section 5.6)

• Why do we have operating systems
  – Protect system and hardware resources
  – Isolate and protect processes
  – Share resources among processes (better utilization)
  – Emulate services not directly available in hardware (example: timers)

• Sharing resources in an OS:
  – Time sharing (multiplexing) the processor
  – Space sharing (multiplexing) the memory
  – On demand sharing of I/O

• Overhead of having an OS:
  – No overhead when process is executing and does not perform system calls
  – System call overhead when process invokes the OS
  – Overhead of managing and scheduling processes

Virtual Machine Monitors (VMM)

• Multiple Operating systems executing simultaneously on the hardware
• VMM emulates guest operating system and machine resources
  – Each OS gets the illusion that it owns the hardware
  – Improved isolation of multiple guest OSs
  – Avoids security and reliability problems

• Virtualization has some performance impact
  – Feasible with modern high-performance computers

• Examples
  – IBM VM/370 (1970s technology!)
  – VMWare
  – Microsoft Virtual PC
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on instructions that access protected resources
- VMM handles real I/O devices (emulates virtual I/O devices)

Example: Virtualization of interrupts

- On an interrupt in the native machine, the OS suspends the current process, handles the interrupt, then either resumes the process or selects another process to run.
- With a Virtual Machine Monitor, an interrupt suspends the current VM, handles the interrupt on behalf of the appropriate VM, then either resumes the suspended VM or selects another VM to run.

Example: Virtualizing a hardware timer:

- One hardware timer can be virtualized to interrupt multiple OSs
  - If OS₁ requests an interrupt at time t₁
  - and OS₂ requests an interrupt at time t₂
  - the times t₁ and t₂ are kept in a software list and the hardware timer is set to the minimum of t₁ and t₂.
Virtual Memory (section 5.7)

- Main memory can act as a cache for the secondary storage (disk)

**Virtual addresses** ➔ **Address translation** ➔ **physical addresses**

- **Disks storage contains the virtual address space**

- **Advantages:**
  1) illusion of having more physical memory

---

Virtual Memory

**Virtual addresses**

- **for process 1**
- **for process 2**

**Advantages:**

2) more than one process can share the same physical memory (protection ??)
3) program relocation (important for multiprogramming)
4) page sharing among processors

---
Physical memory as a cache for the virtual memory

- Miss penalty (cost of a page fault) = 1M~10M cycles (to access hard drive)
  - (as opposed to 20~150 cycles (L2/memory))
- Block (page) size = 1KB ~ 64KB (large because of huge miss penalty)
  - (as opposed to 16B~64B in L1/L2)
- Hit time = 50 ~ 150 cycles
  - (as opposed to 1~3 in L1 and 6~12 in L2)
- Miss rate = 0.00001~0.001% (called page fault rate)
  - (as opposed to 0.1~10% in L1)
- Always write back (why??)
  - (always write back in L2, and commonly in L1)
- Fully associative (why??)
  - (as opposed to 4-8 way set associative in L1/L2)
- Replacement implemented in OS
  - (as opposed to hardware implementation for L1/L2)
- Locate pages using page tables
  - (efficient way to implement full associativity)

Address translation through Page Tables

- Table lookup replaces tag comparison in L1/L2.
Virtual to physical address translation

In this example, the virtual address space is 4 times larger than the physical address space.

- Page table is part of the addressable space (which? -- virtual or physical?)

Example of address translation

Examples:
reference to 0011000 0101 (page 24), translates to 110 0101
reference to 0011001 0011 (page 25) causes a page fault.