Virtual Machines

Today
- VM over time
- Implementation methods
- Hardware features supporting VM

Next time
- Midterm!

*Partially based on notes from C. Waldspurger, VMware, 2010*
Types of virtualization

- **Process virtualization**
  - Language-level: Java, .NET, Smalltalk
  - OS-level: processes, Solaris Zones, BSD Jails
  - Cross-ISA emulation: Apple 68K-PPC-x86

- **Device virtualization**
  - Logical vs. physical: VLAN, VPN, LUN, RAID

- **System virtualization**
  - Type 1 – Run on the raw hardware, VMware ESX, Xen, Microsoft Hyper-V, **Northwestern’s Palacios**!
  - Type 2 – Run on hosted on another OS; VMware Workstation, Microsoft VPC, Parallels
System virtualization starting point

- Physical hardware
  - Processors, memory, chipset, I/O devices, etc.
  - Resources often grossly underutilized
- Software
  - Tightly coupled to physical hardware
  - Single active OS instance
  - OS controls hardware
Adding a virtualization layer

- **Software abstraction**
  - Behaves like hardware
  - Encapsulates all OS and application state

- **Virtualization layer**
  - Extra level of indirection
  - Decouples hardware, OS
  - Enforces isolation
  - Multiplexes physical hardware across VMs
Virtualization properties

- **Isolation**
  - Fault isolation
  - Performance isolation

- **Encapsulation**
  - Cleanly capture all VM state
  - Enables VM snapshots, clones

- **Portability**
  - Independent of physical hardware
  - Enables migration of live, running VMs

- **Interposition**
  - Transformations on instructions, memory, I/O
  - Enables transparent resource overcommitment, encryption, compression, replication …

- *Why not doing these things in the OS?*
Virtual Machine Monitor

- **Classic definition***

> A virtual machine is … an efficient, isolated duplicate of the real machine. … the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

- **VMM properties**
  - Fidelity – SW on the VMM executes identically to its execution on HW, other than timing effects
  - Performance – HW runs most of the instructions w/o the VMM involvement
  - Safety and isolation – VMM manages all HW resources

Classic virtualization

- Classical VMM
  - IBM mainframes:
    IBM S/360, IBM VM/370
  - Co-designed proprietary hardware, OS, VMM
  - “Trap and emulate” model

- Applications
  - Timeshare several single-user OS instances on expensive hardware
  - Compatibility

From IBM VM/370 product announcement, ca. 1972
Modern virtualization renaissance

- Recent proliferation of VMs
  - Considered exotic mainframe technology in 90s
  - Now pervasive in datacenters and clouds
  - Huge commercial success (partly lead by Rosenblum’s VMware)
    - There was support for running DOS and early Windows on 32b machines before this

- Why?
  - Introduction on commodity x86 hardware
  - Ability to “do more with less” saves $$$
  - Innovative new capabilities
  - Extremely versatile technology
Modern virtualization applications

- **Server consolidation**
  - Convert underutilized servers to VMs, saving cost
  - Increasingly used for virtual desktops

- **Simplified management**
  - Datacenter provisioning and monitoring
  - Dynamic load balancing

- **Improved availability**
  - Automatic restart
  - Fault tolerance
  - Disaster recovery

- **Test and development**
Virtualization’s building blocks

- Processor virtualization
  - Trap and Emulate
  - Binary Translation
- Memory virtualization
- I/O virtualization
Virtualizing the CPU

- Running a virtual machine
  - Limited direct execution – remember processes?
  - Wishing to boot a new VM, jump to the first address and go …

- To switch between two VMs, a machine switch
  - Like a processes switch, but using a Virtual CPU
  - VCPU – the state of the CPU as the guest machine believes it to be; a VCPU per guest (~PCB)
  - The VM loaded can be either within the OS or within a process running on that OS

- *Easy right?*
Virtualizing the CPU

- What if the OS or running app tries to perform a privileged instruction?
  - E.g., update the TLB in a software managed TLB
  - The OS cannot be allowed to do it, the VM must intercept it
  - Consider `open(path, flags, mode)` in FreeBSD

```
push dword mode
push dword flags
push dword path
mov eax, 5
push eax
int 80h
```

<table>
<thead>
<tr>
<th><strong>Process</strong></th>
<th><strong>Hardware</strong></th>
<th><strong>OS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.Execute instruction</td>
<td>3.Switch to kernel mode; jump to trap handler</td>
<td></td>
</tr>
<tr>
<td>(add, load, etc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.System call: trap to OS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.In kernel mode; handle syscall; return from trap</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.Switch to user mode; return to user code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.Resume execution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Trap and Emulate

- Trap and emulate
  - Kernel in guest attempts a privileged instruction, traps to VMM
  - VMM emulates the requested action, updates VCPU, and returns control to VM

<table>
<thead>
<tr>
<th>Process</th>
<th>OS</th>
<th>VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System call: trap to OS</td>
<td>3. OS trap handler; decode, trap and execute syscall; return from trap</td>
<td>2. Process trapped, call OS trap handler (at reduce privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. OS tried to return from trap; do real return</td>
</tr>
<tr>
<td>5. Resume execution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Trap and emulate

- How does the VMM know where the trap handler is?
  - The OS tried installed them at boot time, a privilege instruction, and the VMM took notes

- Trap costs may be high

- VMM consumes a privilege level
  - In Rosenblum’s Disco work they used a MIPS supervisor mode
    - Access additional memory but not privilege instructions
    - Additional memory is enough for the OS to keep its data
  - Without it, need to virtualize protection levels, use page tables and TLBs to protect OS data structs
Binary translation

- Not all architectures support trap and emulate, i.e. are
  ''Strictly virtualizable''
  - A processor (or mode of a) is strictly virtualizable if, when executed in a lesser privileged mode:
    - all instructions that access privileged state trap
    - all instructions either trap or execute identically
  - x86 was not strictly virtualizable 😞 – example problem instruction – “pop flags” or popf
    - In privilege mode, popf may change system flags such as IF, which controls interrupt delivery
    - For a deprivileged guest, we need the kernel to trap so that the VMM can emulate the virtual IF
    - But all user-mode popf simply suppresses attempts to modify IF
Binary translation

- Simply binary translation – fidelity and safety, but with bad performance
- Dynamically translate potentially dangerous instructions (non-virtualizable) into safe instructions
  - VMM inspects next sequence of instructions (up to next control transfer, like a branch)
  - Translate each instruction and cache translation, jump to start of the translated block and run with VCPU state on hardware
- Issues with binary translation
  - Translation cache management
  - PC synchronization on interrupts
  - Self-modifying code
    - Notified on writes to translated guest code
  - Protecting VMM from guest
Virtualization’s building blocks

- Processor virtualization
  - Trap and Emulate
  - Binary Translation
- Memory virtualization
- I/O virtualization
Virtualizing memory

- Add another level of indirection too for memory too
- Physical memory is now a virtualization on top of “machine memory”
  - Each OS maps virtual-to-physical addresses via its per-process page table
  - VMM maps physical mappings to underlying machines via its per-OS page tables
**Traditional address spaces**

- Consider address translation with a software-managed TLB

<table>
<thead>
<tr>
<th>Process</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load from memory; TLB miss: trap to OS</td>
<td>2. OS TLB miss handler; extract VPN from VA; do page table lookup; if present and valid, get PFN, update TLB; return from trap</td>
</tr>
<tr>
<td>3. Resume execution; instruction is retried, results in TLB hit</td>
<td></td>
</tr>
</tbody>
</table>
### TLB miss flow with virtualization

- With a VMM, upon a miss TLB is not the OS TLB miss handler that runs but the VMM’s

<table>
<thead>
<tr>
<th>Process</th>
<th>OS</th>
<th>VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load from memory; TLB miss: trap to OS</td>
<td>2. VMM TLB miss handler; call into OS TLB handler (reducing privilege)</td>
<td></td>
</tr>
<tr>
<td>3. OS TLB miss handler; extract VPN from VA; do page table lookup; if present and valid, get PFN, update TLB</td>
<td>4. Trap handler; unprivileged code trying to update TLB; OS is trying to install VPN-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)</td>
<td></td>
</tr>
<tr>
<td>5. Return from trap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Resume execution; instruction is retried, results in TLB hit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Issues with virtualized memory

- **Positives**
  - Simplifies monitor design
- **TLB misses significantly more expensive**
  - In Disco they added a software TLB to reduce this cost
- **With a hardware-managed TLB**
  - The hardware walks the TLB and updates it as needed
  - The VMM must monitor changes the OS makes to each page table and keep a shadow page table that instead maps the virtual address of each process to the VMM’s desired machine pages
  - The VMM installs a process’ shadow table when the OS tries to install the process’ OS-level page table
I/O Virtualization implementations

Virtualized I/O

Guest OS

- Device driver

Device Emulation

I/O Stack

Device Driver

Guest device driver

Virtual device

Virtualization layer
- Emulates the virtual device
- Remaps guest and real I/O addresses
- Multiplexes and drives the physical device
- I/O features

Real device
- May be different from virtual

VMware ESX
I/O Virtualization implementations

- Only Domain0, created at boot, has direct access
  - Can also create/terminate domains, control scheduling parameters, physical memory allocation,..

- All others domains access through virtual device

- Information is pass between domains through shared-memory, asynchronous buffer-descriptor rings

Virtualized I/O: Hosted or Split

Guest OS

- Device driver

Host OS/Dom0/Parent Domain

- Device Emulation
- I/O Stack
- Device Driver

VMware Workstation, VMware Server, Xen, Microsoft Hyper-V, Virtual Server
I/O Virtualization implementations

- Fast but inflexible
- How to handle
  - Discontinuous physical memory
  - Read-only physical memory (COW)
  - Paged out physical memory
  - VM migration
  - ...

Passthrough I/O

Guest OS

Device driver

Device Manager

VMware ESX (FPT)
Summary

- Virtualization
  - Not a new concept but technological improvements brought a renaissance and many new usages
  - Interested? Take EECS 441 – much of it focused on Linux kernel development

- Second half of the quarter
  - The power and danger of concurrency
  - I/O and persistent storage