Virtual Machines
Part 1: 54 years ago
It’s 1964 ...

- IBM wants a multiuser time-sharing system.

- TSS project
  - large, monolithic system
  - lots of people working on it for years
  - total, complete flop

- CMS
  - single-user time-sharing system for IBM 360

- CP67
  - virtual machine monitor (VMM)
  - supports multiple virtual IBM 360s

- Put the two together ...
  - a (working) multiuser time-sharing system
Virtual machines are an interesting extension of the virtual-memory concept: not only do we give processes the illusion that they have all of memory to themselves, but also we give them the illusion that they have an entire machine to themselves. The kernel is a relatively simple piece of software that provides environments, called virtual machines, that look and behave exactly like real machines. Within a virtual machine, since it behaves just like a real machine, one can run a standard operating system, or perhaps a specialized operating system.
Why?

- Structuring technique for a multi-user system
- OS debugging and testing
- Multiple OSes on one machine
- Adapt to hardware changes in software
- Server consolidation and service isolation
User vs. Privileged Mode

- Privileged mode
  - may run all instructions, access all registers
  - for example:
    - modify address translation for virtual memory
    - access and control I/O devices
    - mask and unmask interrupts
    - start and stop system clock
- User mode
  - may run only “innocuous” instructions
  - may access only normal registers
How?

- Approach 1
  - system has “normal” scheduler and virtual memory
  - its processes run in privileged mode
How?

• Approach 2
  – system has “normal” scheduler and virtual memory
  – its processes run an emulator of the real machine
How?

• Approach 3
  – system has “normal” scheduler and virtual memory
  – its processes execute user-mode code directly, but run the emulator when going into privileged mode
How?

• Approach 4
  – system has “normal” scheduler and virtual memory
  – its processes execute user-mode code directly, but emulate only privileged instructions
The key to the design of a system supporting virtual machines is the distinction between privileged and user modes. When the processor is running in privileged mode, there is no protection of the various system components. Thus if we are trying to isolate the various virtual machines from one another, we cannot allow the virtual machines to have the processor in privileged mode. But, on the other hand, since we want the virtual machine to behave as a real machine does, we must support the execution of privileged instructions, as well as the dichotomy between privileged and user modes, on each virtual machine.

The solution is to allow only the virtual machine monitor to run in privileged mode; all other software runs in user mode. However, for each virtual machine, we maintain the concepts of "virtual privileged mode" and "virtual user mode." Whenever a virtual machine is in privileged mode, the virtual machine monitor represents the virtual machine’s state as being virtual privileged mode, though the (real) processor is in user mode. When a privileged instruction is executed on a virtual machine, the real processor traps to the virtual machine monitor. The virtual machine monitor will then check the state of the virtual machine: if the virtual machine is in virtual user mode, then this attempt to execute a privileged instruction should result in a trap on the virtual machine. However, if the virtual machine is in virtual privileged mode, then the privileged instruction should be allowed to execute. The instruction is not executed directly—the virtual machine monitor must make certain that the effect of executing the instruction applies only to the virtual machine. For example, one virtual machine must not be allowed to start an operation on another virtual machine’s virtual disk drive.
Requirements

- A virtual machine is an efficient, isolated duplicate of real machine

Sensitive Instructions

- Control-sensitive instructions
  - affect the allocation of resources available to the virtual machine
  - change processor mode without causing a trap
- Behavior-sensitive instructions
  - effect of execution depends upon location in real memory or on processor mode
Privileged Instructions

- Cause a fault in user mode
- Work fine in privileged mode
Theorem (!)

- For any conventional third-generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.
The photo, from http://old.cs.ncl.ac.uk/events/anniversaries/40th/images/ibm360_672/, is of an IBM 360 Model 67, the first (and only) 360 to support virtual memory. IBM’s VM/360 virtual machine monitor ran on it.
The (Real) 360 Architecture

- Two execution modes
  - supervisor and problem (user)
  - all sensitive instructions are privileged instructions
- Memory is protectable: 2k-byte granularity
- All interrupt vectors and the clock are in first 512 bytes of memory
- I/O done via channel programs in memory, initiated with privileged instructions
- Dynamic address translation (virtual memory) added for Model 67
When an interrupt or trap occurs, the current state is pushed on the kernel stack. Control is transferred to the location given in the interrupt vector; the particular element is selected by the type of interrupt. When the handler returns, the saved state is popped off the stack and restored, thus returning control to whatever was interrupted.

Note: the IBM 360 architecture did not directly support stacks. They have to be implemented explicitly in software. What’s in each element of the interrupt vector is a program status word (PSW) that specifies the address of the handler and other information.
A real interrupt or trap occurs, of course, on the real machine. The VMM might determine that it should be handled by a virtual machine, as if the interrupt or trap had happened on that machine. The VMM simulates the interrupt or trap on the virtual machine by causing to happen what happens on the real machine: it saves the current state of the virtual machine on the kernel stack, looks up the interrupt or trap in virtual machine’s interrupt vector, then passes control to the virtual machine at that location.
## Actions on Real 360

<table>
<thead>
<tr>
<th></th>
<th>User mode</th>
<th>Privileged mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-sensitive</td>
<td>executes fine</td>
<td>executes fine</td>
</tr>
<tr>
<td>instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>errant instruction</td>
<td>traps to kernel</td>
<td>traps to kernel</td>
</tr>
<tr>
<td>sensitive instruction</td>
<td>traps to kernel</td>
<td>executes fine</td>
</tr>
<tr>
<td>access low memory</td>
<td>traps to kernel</td>
<td>executes fine</td>
</tr>
</tbody>
</table>
# Actions on Virtual 360

<table>
<thead>
<tr>
<th></th>
<th>User mode</th>
<th>Privileged mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-sensitive</td>
<td>executes fine</td>
<td>executes fine</td>
</tr>
<tr>
<td>instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>errant</td>
<td>traps to VMM; VMM causes trap to occur on guest OS</td>
<td>traps to VMM; VMM causes trap to occur on guest OS</td>
</tr>
<tr>
<td>instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sensitive</td>
<td>traps to VMM; VMM causes trap to occur on guest OS</td>
<td>traps to VMM; VMM verifies and emulates instruction</td>
</tr>
<tr>
<td>instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>access low</td>
<td>traps to VMM; VMM causes trap to occur on guest OS</td>
<td>traps to VMM; VMM verifies and emulates/translator access</td>
</tr>
<tr>
<td>memory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quiz

Can a VMM (supporting other virtual machines) run on a virtual machine?
   a) yes, no problem
   b) it requires some changes to a VMM for it to run on a virtual machine
   c) no, can't be done
Virtual Devices?

- Terminals
  - connecting (real) people
- Networks
  - didn’t exist in the 60s
  - (how did virtual machines communicate?)
- Disk drives
  - CP67 supported “mini disks”
  - extended at Brown into “segment system”
- Interval timer
  - virtual or real?
Coping

• Invent new devices
  – recognized by VMM as not real, but referring to additional functionality
    - e.g., mini disks
• Provide new VM facilities not present on real machine
  – e.g., Brown segment system
  – special instructions on VM to request service from VMM
    - sort of like system calls (supervisor calls on 360), but ...
      • hypervisor calls
        – 360 had an extra, unused privileged instruction
          – the diagnose instruction