Effective Virtualization

Dr. Amit Praseed

A Simple Solution - Emulation

- A machine could simply be emulated in software
  - File to represent disk
  - Case-statement to implement individual opcodes
  - Registers could be implemented as variables

Conditions for Effective Virtualization

- **Efficiency** : All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program
- **Resource Control** : It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt.
- **Equivalence** : Any program *K* executing with a control program resident performs in a manner indistinguishable from the case when the control program did not exist and *K* had whatever freedom of access to privileged instructions that the programmer had intended

[Popek and Goldberg 1974: "Formal Requirements for Virtualizable Third Generation Architectures" Communications of the ACM]

Issue with Emulation

- A machine could simply be emulated in software
  - File to represent disk
  - Case-statement to implement individual opcodes
  - Registers could be implemented as variables
- This violates the efficiency criteria!!!
  - Software does not allow effective implementations of hardware mechanisms such as interrupts
  - Each guest instruction is executed by several host instructions.
- **Instead, can we directly use the host hardware?**
  - VM executes as a process on host
  - Host processor executes its instructions.

Trap and Emulate

- The guest code runs directly on the CPU
  - with reduced privilege – why?
- When the guest attempts to read or modify privileged state, the processor generates a trap that transfers control to the VMM.
- The VMM then emulates the instruction using an interpreter and resumes direct execution of the guest at the next instruction.

Types of Machine Instructions

- **Privileged instructions**
  - Executed in kernel mode
  - When attempted to be executed in user mode, they cause a trap and so executed in kernel mode.
- **Non-privileged instructions**
  - Can be executed in user mode
- **Sensitive instructions**
  - Can be executed in either kernel or user but they behave differently
  - Require special precautions at execution time.

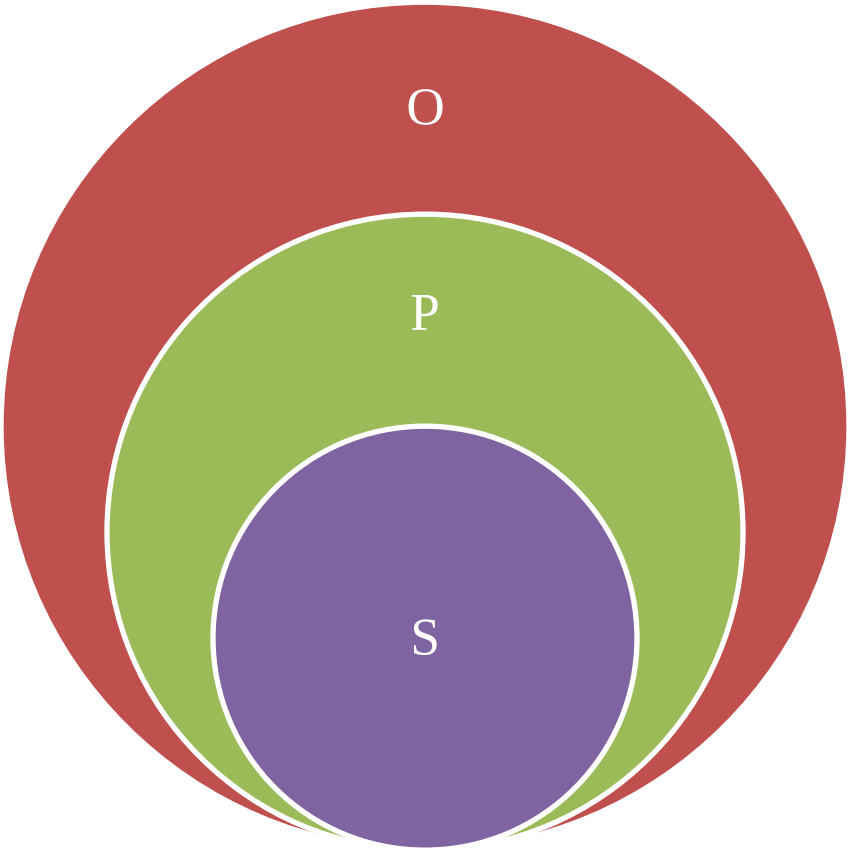
Issues

Let O be the set of instructions, then O = P ∪ S ∪ I, where

- P is the set of **Privileged Instructions**
  - Execution in system mode is possible, execution in user mode traps
- S is a set of **Sensitive instructions**, and S = C ∪ B where
  - C – Control sensitive instructions: Change configuration of system resources
  - B – Behavior sensitive instructions: Behavior depends on configuration of system resources
- I is the set of **Insensitive instructions**

Issues

- Popek and Goldberg theorized that the construction of an efficient VM is possible if $S \subseteq P$
  - All sensitive instructions must also be privileged
- Sadly, this is not the case with the x86 architecture
  - 17 instructions are sensitive but not privileged



Why the x86 Architecture is Not Virtualizable

- 17 instructions are sensitive, but not privileged
- Eg: POPF instruction
  - loads a set of flags from the stack into the %eflags register.
  - When executed in privileged mode, popf loads all flags, which includes a mix of ALU flags (zero flag, carry flag, etc.) and system flags (interrupt flag, I/O privilege level, etc.).
  - When executed in user mode, system flags remain unmodified
  - If a deprivileged guest kernel attempts to clear the interrupt flag using popf, no trap is generated and the VMM has no way of knowing it should not deliver interrupts to the guest.

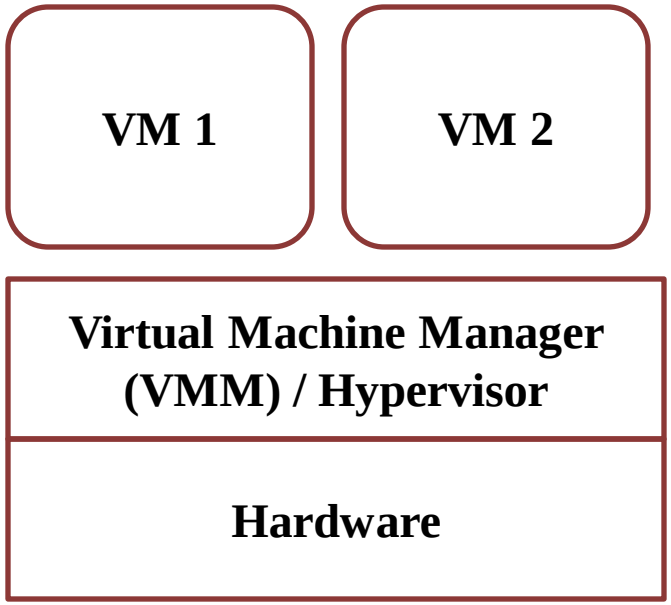
Solutions

- Make sure that no sensitive but unprivileged instructions are executed
  - Instructions are replaced by VMM at runtime
  - Instructions are replaced before runtime
  - Adapting the guest OS
- Hardware becomes virtualization-aware
  - Add new instructions for virtualization
  - Add new mode for virtualization
    - Orthogonal to traditional modes
    - New mode “above” system mode (even more privileged)
  - Example: AMD-V, Intel VT-x, ARM Virtualization Extensions as an extension of ARMv7

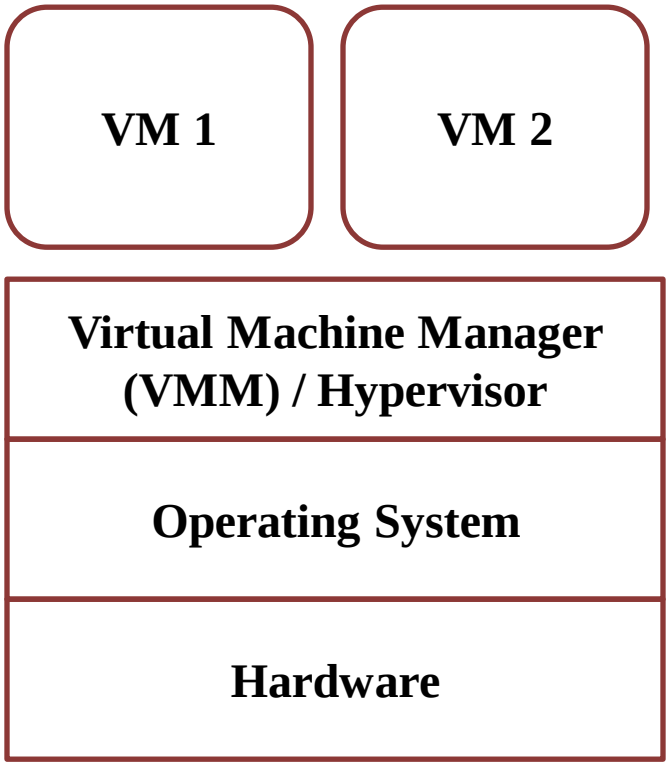
Role of Hypervisor

- Provide an environment for programs which is essentially identical to the original machine
- Ensure that programs run in this environment should show, at worst, only minor decreases in speed
- Ensure complete control of the system resources.
  - Allocation
  - Separation
  - Preemption

Types of Hypervisors



Type 1 (Bare-Metal) Hypervisor



Type 2 (Hosted) Hypervisor

Comparison of Hypervisors

Type - 1 Hypervisor

- Resides directly on the hardware (“bare metal”)
- Communicates directly with the hardware resources
- More efficient
- More secure
- Eg: Citrix/Xen Server, VMware ESXi and Microsoft Hyper-V

Type – 2 Hypervisor

- Resides on top of the operating system (“hosted”)
- Communicates with hardware through the OS
- Less efficient
- Less secure
- Eg: Oracle Virtual Box, VMware Workstation etc.