5.6 Virtual machines

Virtual machines (VM) were first developed in the mid-1960s, and they have remained an important part of mainframe computing over the years. Although largely ignored in the single-user PC era in the 1980s and 1990s, they have recently gained popularity due to

- The increasing importance of isolation and security in modern systems
- The failures in security and reliability of standard operating systems
- The sharing of a single computer among many unrelated users, in particular for Cloud computing
- The dramatic increases in raw speed of processors over the decades, which makes the overhead of VMs more acceptable

The broadest definition of VMs includes basically all emulation methods that provide a standard software interface, such as the Java VM. In this section, we are interested in VMs that provide a complete system-level environment at the binary *instruction set architecture* (ISA) level. Although some VMs run different ISAs in the VM from the native hardware, we assume they always match the hardware. Such VMs are called (Operating) *System Virtual Machines*. IBM VM/370, VirtualBox, VMware ESX Server, and Xen are examples.

System virtual machines present the illusion that the users have an entire computer to themselves, including a copy of the operating system. A single computer runs multiple VMs and can support a number of different *operating systems* (OSes). On a conventional platform, a single OS "owns" all the hardware resources, but with a VM, multiple OSes all share the hardware resources.

The software that supports VMs is called a *virtual machine monitor* (VMM) or *hypervisor*; the VMM is the heart of virtual machine technology. The underlying hardware platform is called the *host*, and its resources are shared among the *guest* VMs. The VMM determines how to map virtual resources to physical resources: a physical resource may be time-shared, partitioned, or even emulated in software. The VMM is much smaller than a traditional OS; the isolation portion of a VMM is perhaps only 10,000 lines of code.

Although our interest here is in VMs for improving protection, VMs provide two other benefits that are commercially significant:

1. *Managing software.* VMs provide an abstraction that can run the complete software stack, even including old operating systems like DOS. A typical deployment might be some VMs running legacy OSes, many running the current stable OS release, and a few testing the next OS release.
2. *Managing hardware.* One reason for multiple servers is to have each application running with the compatible version of the operating system on separate computers, as this separation can improve dependability. VMs allow these separate software stacks to run independently yet share hardware, thereby consolidating the number of servers. Another example is that some VMMs support migration of a running VM to a different computer, either to balance load or to evacuate from failing hardware.

Hardware/Software Interface

Amazon Web Services (AWS) uses the virtual machines in its Cloud computing offering EC2 for five reasons:

1. It allows AWS to protect users from each other while sharing the same server.
2. It simplifies software distribution within a warehouse-scale computer. A customer installs a virtual machine image configured with the appropriate software, and AWS distributes it to all the instances a customer wants to use.
3. Customers (and AWS) can reliably "kill" a VM to control resource usage when customers complete their work.
4. Virtual machines hide the identity of the hardware on which the customer is running, which means AWS can keep using old servers *and* introduce new, more efficient servers. The customer expects performance for instances to match their ratings in "EC2 Compute Units," which AWS defines: to "provide the equivalent CPU capacity of a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor." Thanks to **Moore's Law**, newer servers clearly offer more EC2 Compute Units than older ones, but AWS can keep renting old servers as long as they are economical.
5. Virtual machine monitors can control the rate that a VM uses the processor, the network, and disk space, which allows AWS to offer many price points of instances of different types running on the same underlying servers. For example, in 2012 AWS offered 14 instance types, from small standard instances at $0.08 per hour to high I/O quadruple extra large instances at $3.10 per hour.

In general, the cost of processor virtualization depends on the workload. User-level processor-bound programs have zero virtualization overhead, because the OS is rarely invoked, so everything runs at native speeds. I/O-intensive workloads are generally also OS-intensive, executing many system calls and privileged instructions that can result in high virtualization overhead. On the other hand, if the I/O-intensive workload is also *I/O-bound*, the cost of processor virtualization can be completely hidden, since the processor is often idle waiting for I/O.

The overhead is determined by both the number of instructions that must be emulated by the VMM and by how much time each takes to emulate. Hence, when the guest VMs run the same ISA as the host, as we assume here, the goal of the architecture and the VMM is to run almost all instructions directly on the native hardware.

PARTICIPATION ACTIVITY 5.6.1: Virtual machine basics.

1) A virtual machine (VM) is an emulation that provides a hardware interface.
   - ○ True
   - ○ False

2) A system VM allows a computer to share hardware resources amongst multiple operating systems.
   - ○ True

- False

3) When a computer runs multiple VMs, the first VM launched is called the host, and the other VMs are called the guests.
   - True
   - False

4) Another name for a VMM is a hypervisor.
   - True
   - False

5) A VMM is the same size as the corresponding OS.
   - True
   - False

Requirements of a virtual machine monitor

What must a VM monitor do? It presents a software interface to guest software, it must isolate the state of guests from each other, and it must protect itself from guest software (including guest OSes). The qualitative requirements are:

- Guest software should behave on a VM exactly as if it were running on the native hardware, except for performance-related behavior or limitations of fixed resources shared by multiple VMs.
- Guest software should not be able to change allocation of real system resources directly.

To "virtualize" the processor, the VMM must control just about everything—access to privileged state, I/O, exceptions, and interrupts—even though the guest VM and OS currently running are temporarily using them.

For example, in the case of a timer interrupt, the VMM would suspend the currently running guest VM, save its state, handle the interrupt, determine which guest VM to run next, and then load its state. Guest VMs that rely on a timer interrupt are provided with a virtual timer and an emulated timer interrupt by the VMM.

To be in charge, the VMM must be at a higher privilege level than the guest VM, which generally runs in user mode; this also ensures that the execution of any privileged instruction will be handled by the VMM. The basic system requirements to support VMMs are:

- At least two processor modes, system and user.
- A privileged subset of instructions that is available only in system mode, resulting in a trap if executed in user mode; all system resources must be controllable just via these instructions.

PARTICIPATION ACTIVITY 5.6.2: VMM requirements.

1) A VMM should not allow a guest VM to change how resources are allocated.
   - True
   - False

2) A VMM runs in system mode, while a guest VM runs in user mode.
   - True
   - False

(Lack of) Instruction set architecture support for virtual machines

If VMs are planned for during the design of the ISA, it's relatively easy to reduce both the number of instructions that must be executed by a VMM and improve their emulation speed. An architecture that allows the VM to execute directly on the hardware earns the title *virtualizable*, and the IBM 370 and the ARMv8 architectures proudly bear that label.

Alas, since VMs have been considered for PC and server applications only fairly recently, most instruction sets were created without virtualization in mind. These culprits include x86 and most RISC architectures, including ARMv7 and MIPS.

Because the VMM must ensure that the guest system only interacts with virtual resources, a conventional guest OS runs as a user mode program on top of the VMM. Then, if a guest OS attempts to access or modify information related to hardware resources via a privileged instruction—for example, reading or writing a status bit that enables interrupts—it will trap to the VMM. The VMM can then affect the appropriate changes to corresponding real resources.

Hence, if any instruction that tries to read or write such sensitive information traps when executed in user mode, the VMM can intercept it and support a virtual version of the sensitive information, as the guest OS expects.

In the absence of such support, other measures must be taken. A VMM must take special precautions to locate all problematic instructions and ensure that they behave correctly when executed by a guest OS, thereby increasing the complexity of the VMM and reducing the performance of running the VM.

Protection and instruction set architecture

Protection is a joint effort of architecture and operating systems, but architects had to modify some awkward details of existing instruction set architectures when virtual memory became popular.

For example, the x86 instruction POPF loads the flag registers from the top of the stack in memory. One of the flags is the *Interrupt Enable* (IE) flag. If you run the POPF instruction in user mode, rather than trap it, it simply changes all the flags except IE. In system mode, it does change the IE. Since a guest OS runs in user mode inside a VM, this is a problem, as it expects to see a changed IE.

Historically, IBM mainframe hardware and VMM took three steps to improve the performance of virtual machines:

1. Reduce the cost of processor virtualization.
2. Reduce interrupt overhead cost due to the virtualization.
3. Reduce interrupt cost by steering interrupts to the proper VM without invoking VMM.

AMD and Intel tried to address the first point in 2006 by reducing the cost of processor virtualization. It will be interesting to see how many generations of architecture and VMM modifications it will take to address all three points, and how long before virtual machines of the 21st century for x86 will be as efficient as the IBM mainframes and VMMs of the 1970s.

Elaboration

ARMv8 provides a third state (EL2) specifically to allow the VMM to run at a higher privilege level than the guest operating system.

PARTICIPATION ACTIVITY 5.6.3: VMs and ISAs.

1) Most ISAs were created to be virtualizable.
   - ○ True
   - ○ False

2) If a guest VM tries to execute an instruction that reads or writes a status bit and enables an interrupt, the instruction will result in a trap.
   - ○ True
   - ○ False

Provide feedback on this section