VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization.
In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM.
To save processor states, mode switching is completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

Hardware Support for Virtualization

- Modern operating systems and processors permit multiple processes to run simultaneously.
- If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash.
- Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware.
- Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions.
- In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack.

CPU Virtualization

- A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode.
- Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability.
- The critical instructions are divided into three categories: privileged instructions, control- sensitive instructions, and behavior-sensitive instructions.
- Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.
- Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

- A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode.
- When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM. In this case,

the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system.

- However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.
- On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions.
- When these instructions execute in virtualization, they cannot be trapped in the VMM.
- On a native UNIX-like system, a system call triggers the 80h interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call.
- On a para-virtualization system such as Xen, a system call in the guest OS first triggers the 80h interrupt normally. Almost at the same time, the 82h interrupt in the hypervisor is triggered.
- Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel.
- Certainly, the guest OS kernel may also invoke the hypercall while it's running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

Hardware-Assisted CPU Virtualization

- This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors.
- Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically.
- This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

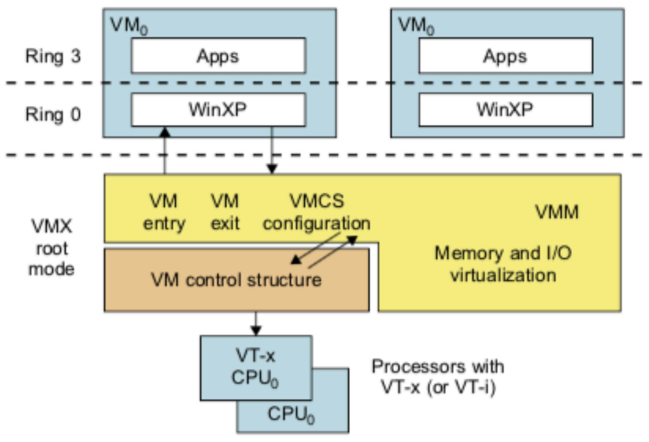

FIGURE 3.11

Intel hardware-assisted CPU virtualization.

Memory Virtualization

- Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory.
- All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.
- However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.
- That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory.
- Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory.

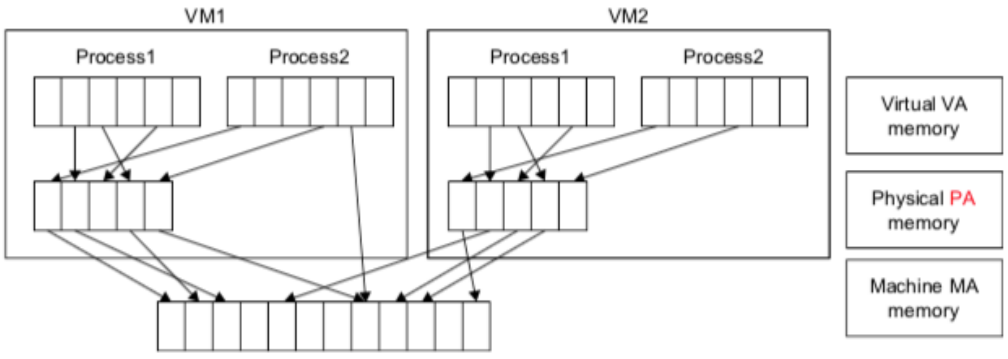

FIGURE 3.12

Two-level memory mapping procedure.

- The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.
- Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table.
- Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor.
- VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access.

I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.

At the time of this writing, there are three ways to implement I/O virtualization:

- full device emulation,

- para-virtualization, and
- direct I/O.
- **Full device emulation** is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices.
- All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are **replicated in software.**
- This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices. The full device emulation approach is shown in Figure 3.14.

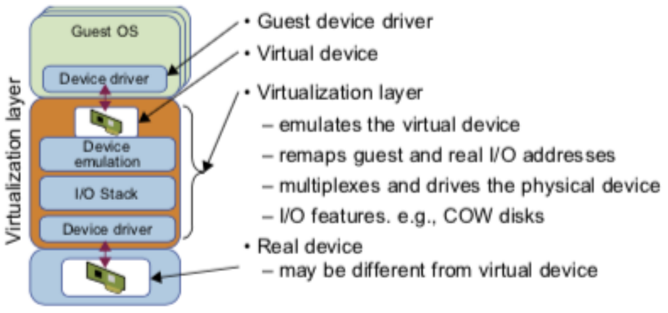
Guest OS
Device driver
Device emulation
I/O Stack
Device driver
Virtualization layer
• Guest device driver
• Virtual device
• Virtualization layer
– emulates the virtual device
– remaps guest and real I/O addresses
– multiplexes and drives the physical device
– I/O features. e.g., COW disks
• Real device
– may be different from virtual device

FIGURE 3.14

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

- **The para-virtualization** method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a **frontend driver and a backend driver**. The frontend driver is running in Domain U and the backend driver is running in Domain 0.
- They interact with each other via a block of shared memory.
- The **frontend driver** manages the I/O requests of the guest OSes and the **backend driver** is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.
- Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.
- **Direct I/O virtualization** lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs.
- However, current direct I/O virtualization implementations focus on networking for mainframes.
- There are a lot of challenges for commodity hardware devices. For example, when a physical device is reclaimed (required by workload migration) for later reassign- ment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system.
- Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical.
- Another way to help I/O virtualization is via **self-virtualized I/O (SV-IO).** The key idea of SV-IO is to harness the rich resources of a multicore processor.
- All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. It provides virtual devices and an associated access API to VMs and a management API to the VMM.