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, modes witching are completed by hardware. For the x86architecture, 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 - Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.

Control sensitive instructions - Control-sensitive instructions attempt to change the configuration of resources used.

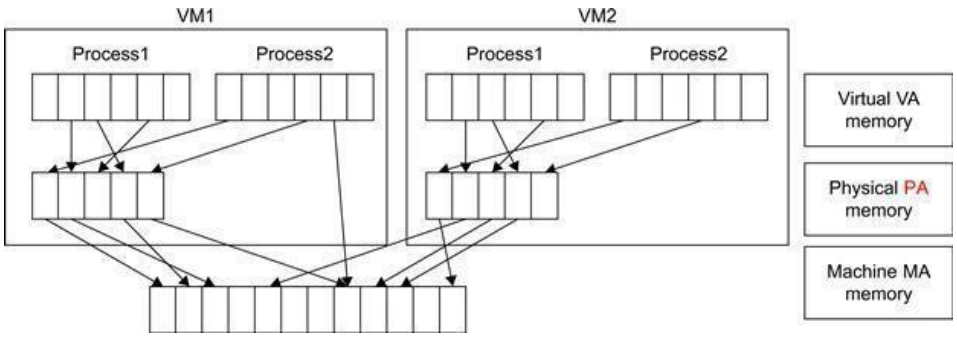
Behavior-sensitive instructions - Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

CPU architecture is virtualizable if it supports the ability to run the VM's privileged and privileged instructions in the CPUs 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. RISC CPU architectures can be naturally virtualized because all control-and behavior sensitive instructions are privileged instructions.

Hardware-Assisted CPU Virtualization: This technique attempts to simplify virtualization because full or para virtualization 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.

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 look aside 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. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure below shows the two-level memory mapping procedure.

Two-level memory mapping procedure

I/O Virtualization: I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. There are three ways to implement I/O virtualization:

• Full device emulation
• Para virtualization
• Direct I/O

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

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.
A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates.
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 front end driver and a back end driver. The front end driver is running in DomainU and the back end driver is running inDomain0.

They interact with each other via a blockof 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.

Virtualization in Multi-Core Processors: Virtualizing a multi-core processor is relatively more complicated than virtualizing a unicore processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualization has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.