Lecture 8: February 22

Lecturer: Prashant Shenoy *Scribe:* ***Basundhara Chakrabarty***

8.1 Virtualization

Virtualization is the process of providing an interface to mimic the behavior of another system or component. Typically, virtualization mimics one of the following:

- Assembly instructions
- System calls
- APIs

Depending on what is replaced or mimicked, we obtain different forms of virtualization.

8.1.1 Types of Virtualization

8.1.1.1 Emulation

In emulation, a software simulation of a particular type of hardware or architecture is done using another. Once you've the emulated hardware, you can run an OS on it. An example would be to emulate an Intel process on ARM hardware, and thereafter any OS that was compiled for ARM will run unmodified on the emulated Intel processor. The OS will execute machine instructions on the emulated hardware, which won't run as-is on the native hardware, and therefore binary translation will need to be performed to convert them to native instructions. This caused significant overhead/slowdown, which causes a performance penalty.

Examples: Box, QEMU (Used by much Linux software), VirtualPC for MAC (Before, MAC computers were PowerPC based, and VirtualPC emulated a x86 software layer on PowerPC machine, allowing one to run Windows)

8.1.1.2 Full/Native Virtualization

In native virtualization, one or more unmodified OSs and the applications they contain are run on top of virtual hardware, given that the underlying native hardware and virtual hardware are of the same type. Here the VM simulates "enough" hardware to allow the OSs to be run in isolation. One limitation is that one can run OSs designed for the same hardware only. One use case is that the virtual OS can be used as a sandbox for testing components in different environments.

Examples: IBM VM family, VMWare Workstation, Parallels, VirtualBox.

8.1.1.3 Para-virtualization

Similar to full/native virtualization except that the kernel of the guest OS is modified so that it uses special APIs to call the hypervisor instead of directly accessing the hardware. The applications run are unmodified.
Examples: Xen, VMWare ESX Server.

8.1.1.4 OS-level Virtualization

Here the OS allows multiple secure VMs to be run on top of a native OS kernel. Each application run on a VM instances sees an isolated OS. This is lightweight as different OSs are not run.

Examples: Solaris Containers, BSD Jails, Linux Vserver, Linux containers, Docker.

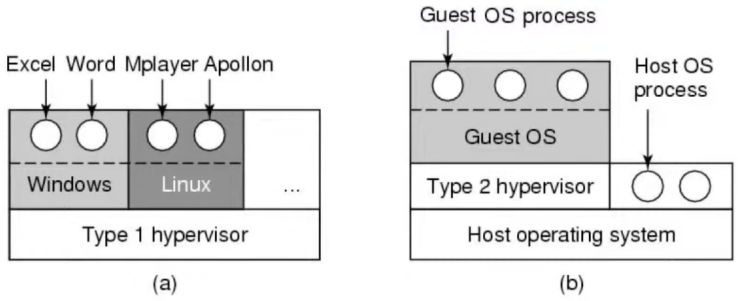
8.1.1.5 Application level virtualization

This type of virtualization uses an application interface to emulate another application. Here, the application is given its own copy of components that are not shared like global objects, registry files etc.

Examples: JVM written using C libraries exports the JAVA interface. Rosetta on Mac (also emulation) allows us to run binaries compiled on one architecture on another architecture WINE.

8.2 Hypervisors

The hardware virtualization layer is called a hypervisor or a virtual machine monitor (VMM). A hypervisor is the virtualization layer, which takes care of resource management, isolation and scheduling. There are 2 types of hypervisors that act like real hardware and implement full/native virtualization.



8.2.1 Type 1 Hypervisor

Type 1 hypervisor runs on "bare metal". It can be thought of as a special OS that can run only VMs as applications. On boot, it's the hypervisor that boots.

Question: Do the multiple OSs running expect the same hardware?
Answer: Yes, because the hypervisor gives the illusion that the OS-s is running on the underlying native hardware. The hypervisor allocates resources between VMs, schedules VMs, etc., the hypervisor behaves like a pseudo-OS for the VMs.

A Type 2 hypervisor runs on a host OS and the guest OS runs inside the hypervisor.

Question: Does type 1 belong to native or paravirtualization?
Answer: Both.

8.2.2 Type 2 Hypervisor

The boot process boots the native OS, the hypervisor runs as a application on the native OS. The native OS is called host OS and the OSs that run on the hypervisor are called guest OSs.

Question: Can you do further virtualization?
Answer: Yes, but it is complicated. It is called nested virtualization

Question: Can multiple guest OSs be run?
Answer: Yes, multiple VMs can be run, each with its own guest OS

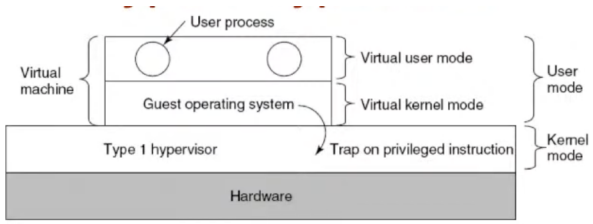
Question: Is the bootloader GRUB a hypervisor?
Answer: No, GRUB used in dual boot machines decides which one OS it has to boot, whereas hypervisors can run multiple guest VMs.

Question: Can a single hypervisor simulate multiple hardwares?
Answer: No, for that emulation needs to be used.

8.3 How Virtualization Works?

For architectures like Intel, different rings signify different levels of privilege. The CPU can run in either user mode (ring 3) or kernel mode (ring 0). In kernel mode, any instructions can be run. But in user mode, only a subset of instructions is allowed to run. The OS kernel is trusted more than user applications, so the OS kernel is run in kernel mode and user applications are run in user mode to limit instruction sets. A simple example is the halt instruction which user applications are not allowed to run. Also, memory management instructions are only allowed to run in kernel mode. Intel CPUs have intermediate rings (ring 1, ring 2) in which guest OSes can be run. Sensitive instructions (I/O, MMU, halt, etc.) are only allowed to run in kernel mode. Privileged instructions cause a trap or interrupt. I/O is one example. These are not related to user mode or kernel mode.

8.3.1 Type 1 Hypervisor

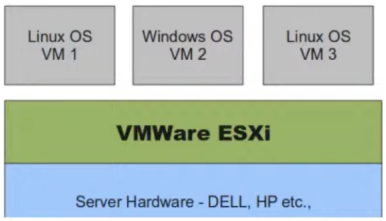


Theory: Type 1 virtualization is feasible if sensitive instruction is subset of privileged instructions or all

sensitive instructions always cause a trap.
Reasoning: On booting a Type 1 hypervisor, it runs in kernel mode. A Windows VM run on the hypervisor should not be trusted as much as the hypervisor and is therefore run in user Mode. Windows assumes it is the kernel and can run sensitive instructions, but these sensitive instructions won't run because it will be running in user mode. The solution is that the hypervisor intervenes and runs each sensitive instruction attempted by the Windows VM. How will the hypervisor be alerted when Windows attempts so? If the sensitive instruction causes a trap, the hypervisor intervenes and executes it for the VM.

Early Intel processors did not have Type 1 support. Recent Intel/AMD CPUs have hardware support, named Intel VT and AMD SVM. The idea is to create containers where a VM and guest can run and that hypervisor uses hardware bitmap to specify which instruction should trap, so that sensitive instruction in guest traps to hypervisor. This bitmap property can also be turned off.

Examples:



a) VMWare ESXI running, a specialized OS kernel that can run any arbritary VMs on it

b) Windows Hyper-V creates partitions, runs Windows in the parent partition (one copy of windows is mandatory), and the child partitions can run Linux or any other OS. Less flexible than ESXI.

c) Linux KVM or kernel virtual machine: Implemented as a device driver that gives barebone support for Type 1. Along with some other components (like QEMU) gives the functionality for Type 1 hypervisor.

8.3.2 Type 2 Hypervisor

A Type 2 hypervisor runs as an application on the host OS and therefore does not have kernel level privileges. Again, only the host OS can run in kernel mode. The Guest OS can also run only user mode instructions.

Therefore, the Type 2 hypervisor performs *dynamic code translation*. It scans instructions executed by the guest OS, replacing the sensitive instructions with function calls. These function calls in turn make system calls to the OS, thereby involving the host OS. This process is called binary translation. This leads to slowdown as every piece of sensitive code has to be translated. Therefore, VT support is not needed, no support is needed from the hardware to ensure that all sensitive instructions are privileged.

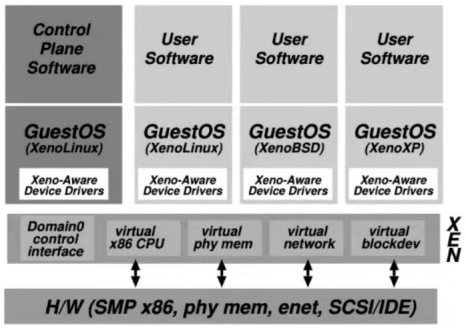
For example, VMWare Fusion, upon loading program, scans code for basic blocks and replaces sensitive instructions by VMWare procedure using binary translation. Only the guess OS's instructions need to be scanned, not the applications.

8.3.3 Para-virtualization

Instead of dynamically translating the sensitive instructions, leading to overhead, we can categorically change the kernel instructions to replace the sensitive instructions with hypercalls (call to the hypervisor) to get a

modified OS with no sensitive instructions. Every time a sensitive function needs to be executed, a hypercall is made instead.

Both Type 1 and 2 hypervisors work on unmodified OS. In contrast, para-virtualization modifies OS kernel to replace all sensitive instructions with hypercalls. Thus, the OS behaves like a user program making system calls and the hypervisor executes the privileged operation invoked by hypercall.



Example: Xen ran as a para-virtualized version of Linux when the hardware did not support Type 1 hypervisors. Xen needed Linux to run in domain 0 (master partition) just like HyperV.

8.3.4 Memory Virtualization

The hypervisor has control over RAM and allocates memory to the guest OS, and the guest OS allocates memory to its processes. If the guest OS tries to change the memory allocation of a process, that is a sensitive instruction because only the host OS is allowed to change page tables. Processes are not allowed to this. But a guest OS in Type 1 does not have those privileges. It still maintains page tables, and it still thinks it owns the machine that it can do whatever it wants, but it is not allowed to do so.

So what we do here is that we change those page tables in guest OSes to be read-only. When OS tries to write to that page table, a trap is created because that is a write instruction to read-only memory page. The hypervisor maintains a second page table which is called a shadow page table. That is a mirror of the original page table. It makes those changes in its actual page table. It utilizes the existing hardware feature that causes a trap when a write instruction happens on read-only region.

8.3.5 I/O Virtualization

Typically, the OS manages the physical disk, the guest OS should not be able to make changes to it. The hypervisor creates a large file on the guest's file system that emulates a disk called virtual disk (eg., vmdk for VMWare) When the guest OS writes to the disk, it effectively writes to this virtual disk. Multiple virtual disk maps to the real disk.

8.3.6 Benefits of Virtualization

One major benefit is that virtualization makes it easier to distribute pre-built applications and software. One can design virtual appliances (pre-built VM with pre-installed OS and pre-configured applications) that are plug-and-play.

For multi-core CPUs, careful allocation of CPU resources per VM can be made.

8.3.7 Use of Virtualization

a) Cloud Computing b) Data Centres c) Software Testing and Development

8.3.8 Brief: OS Virtualization

OS virtualization uses the native OS interface to emulate another OS interface. A use case can be emulating an older version of OS. A popular use case is to allow backward compatibility, allow an older version of an application to be run, or sandboxing.