CMPSCI 677 Operating Systems **Spring 2015**

Lecture 6: February 17

Lecturer: Prashant Shenoy *Scribe: Dong Chen*

6.1 Virtualization Continued

6.1.1 Type 1 Hypervisor

In this case, the hypervisor runs on bare metal, and the system is booted with the hypervisor. The unmodified OS is running in Ring 3 or Ring 1 mode. On top of the hypervisor a virtual machine that emulates the same hardware environment can be run. So, multiple virtual machines can be running with same hardware environment, but with different OSs. This type of hypervisors are typically only seen in server environments.

6.1.2 Type 2 Hypervisor

Type 2 hypervisor runs just as an application in user level on top of the native OS. It emulates the same underlying hardware and exposes the same hardware interface. Guest OS runs inside hypervisor. VMWare fusion was demonstrated in class as an example of a type 2 hypervisor. Guest OS is unaware that it is running in user mode, so it tries to execute kernel instructions as it would normally do in non-virtualized world. When this happens, it will not be allowed to execute them, as it runs in user mode. To solve this problem type 2 hypervisors scan guest OS instructions as they are about to execute, and whenever there is a kernel instruction it is replaced with a function call to the hypervisor. The hypervisor, in turn, triggers a system call and requests the host OS to execute that instruction. The host OS then executes that instruction for the hypervisor. So type 2 hypervisors require binary translation from code. All sensitive instructions are replaced by procedures that emulate them.

6.1.3 How Virtualization Works

CPU suports kernel mode (Ring 0) and user mode (Ring3). Rings define levels of protection in terms of what privileges a process has when it runs at that level. The OS kernel runs at ring 0, and has full control over things happening in the machine. User processes run at ring 3 i.e. they can only run some restricted set of instructions, not including certain special instructions that only the kernel can run. These rings offer protection from user processes that can corrupt components and cause the system to come down. The instructions that can only be executed in kernel mode are called sensitive instructions, e.g. instructions that can change MMU settings or instructions that get I/O involved. If processes try to execute these sensitive instructions in user mode, their request will be denied, as only the kernel has the privileges to execute them. Any assembly instruction that causes a trap is known as a privileged instruction. Result: It is possible to implement type 1 virtualization only if sensitive instructions are a subset of privileged instructions. An OS running on a guest VM should be not be allowed to change hardware resourses. So, guest OSs are all forced to run in used mode. when sensitive instructions are executed in user space, somehow the hypervisor needs to be notified to execute those instructions on behalf of the guest OS. Therefore, we would like sensitive instructions to cause a trap and use that as a way to let the hypervisor know that the guest OS needs to execute a sensitive instruction. E.g. In Intel 386 sensitive instructions executed in user mode are ignored.

6.1.4 Paravirtualization

Paravirtualization is a technique to run a guest OS on top of a type 1 hypervisor, without requiring hardware support. in paravirtualization, the assumption is that the OS can be modified. In effect, a programmer modifies the OS by taking each kernel mode instruction and replacing it with a function call to the hypervisor. So, in a sense, it is similar to what was being done in type 2 hypervisors, except now it is not done on-the-fly, but instead, it is done beforehand in the source code, by the programmer. The programmer looks at kernel code, and every time the kernel makes a system call, it is replaced with a function call to the hypervisor. These calls to the hypervisor are called 'hypercalls'. So the programmer replaces all sensitive instructions with hypercalls. All kernel instructions are replaced by hypercalls to a type 1 hypervisor, which runs in kernel mode and can execute those instructions on behalf of the OS. Paravirtualization can only be implemented for those OSs, which are open-source e.g Linux. For proprietary OSs like Windows, it can only be implemented internally by the company. Examples of paravirtualized hypervisors include Xen, VMWare ESX server.

6.1.5 Memory Virtualization

Virtualization assume they can full control over memory. If multiple virtual machines can reside in memory, the hypervisor needs to allocate memory to each VM. The OS that runs inside each VM is unaware of the fact that it has access to only a portion of the actual RAM. So it thinks it is running on real hardware, and has full control of memory. So a portion of the RAM needs to be carved out for each VM and it needs to be able to address it like the actual RAM. The hypervisor needs to deal with page table mapping as well. The OS manages page tables for all processes. The page table data structure tracks which pages of memory are allocated to which process. Typically manipulating page tables requires manipulating hardware. So, these are kernel mode operations, and user processes cannot perform them. So the hypervisor needs to do it on behalf of the OS. The hypervisor maintains shadow page tables that mirror actual page tables. So there is a copy of page tables that the OS manages and copy of page tables that hypervisor manages. There is a one-to-one mapping between the two. If the OS tries to modify a page table, a trap is generated, and the hypervisor makes those changes. These changes are also reflected in the real page tables, as it is a mirrored copy. The OS thinks it modified the actual page table, although it does not have the privileges. Any modification to OS level data structures including memory management data structures have to be handled by the hypervisor on behalf of the OS.

6.1.6 I/O Virtualization

Disks and network interfaces also have to be virtualized. There is one large physical disk, and the storage on that disk is partitioned and allocated to the VMs that are running. Each VM thinks that it is accessing a physical disk. This is achieved by creating a virtual disk abstraction, where all the translations from addresses of the virtual disk to addresses of the physical disk are done by the hypervisor. This can be done in many ways. Actual partitions can be created on the disk and allocated to the VMs, or the entire virtual disk can be a file on the disk. The physical machine has a ethernet card that is used to do network I/O. The VMs share this physical ethenet card. Each VM is assigned a logical ethernet card using which it does network I/O. Network packets that are sent by the VMs are multiplexed by the hypervisor to the actual physical ethernet card. Similarly, when packets arrive at the actual network card, the hypervisor figures out which VM those are for, and delivers them accordingly.

6.1.7 Examples

Java is an example of virtualization at application level. The Java Virtual Machine (JVM) is an abstract computing machine. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time. It has an API that it exposes to Java programs, and you implement the JVM using underlying interfaces, e.g you could write it in C. Java code runs regardless of the underlying hardware, therefore it is portable. Rosetta was a technology that Apple introduced for Mac when they switched from PowerPC processors to Intel processors. So, Rosetta is an application-level abstraction that translates PowerPC instructions to equivalent Intel instructions.

6.1.8 Virtual Appliances and Multi-Core CPUs

One advantage of virtualization is that you can download prepackaged VMs, called appliances, that already have pre-installed and pre-configured software, including OS and applications. These appliances can be downloaded and run as is, and have made VMs a popular method of distributing complex applications that require expert configuration. Multi-core CPUs: Having multiple cores on one system has made it attractive to use VMs. Rather than slicing a single CPU across different machines, each VM can be run on one core. Multiple cores can also be assigned to one VM. This is often used in datacenter environments or other server environments where there are large multiple core servers.

6.1.9 Case Study: Planet Lab

PlanetLab is a global research network that supports the development of new network services. Since the beginning of 2003, more than 1,000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. PlanetLab currently consists of 1343 nodes at 657 sites. Planet Lab uses OS level virtualization. They assign a slice of the machine to a user by giving them a container (like the Solaris container or BSD jails that were described earlier). Therefore, the user does not get a VM running another copy of the OS, but they get a container with the allocated amount of resources (including memory, CPU etc.). This feather enables users to run experiments without impacting other users.

6.1.10 OS Virtualization

OS virtualization is a technology that partitions the operating system to create multiple isolated lighweight virtual machines, as on hypervisor involved. It is a widely used to improve security, manageability and availability of todays complex software environment, with small runtime and resource overhead, and with minimal changes to the existing computing infrastructure, such as Solaris container, BSD jails, LXC.

6.1.11 Linux Containers (LXC)

Linux Containers (LXC) feature is a lightweight virtualization mechanism that does not require you to set up a virtual machine on an emulation of physical hardware. The LXC feature takes the cgroups resource management facilities as its basis and adds POSIX file capabilities to implement process and network isolation. The main use of Linux Containers is to allow you to run a complete copy of the Linux operating system in a container (a system container) without the overhead of running a level-2 hypervisor such as VirtualBox. In fact, the container is sharing the kernel with the host system, so its processes and file system are completely

visible from the host. When you are logged into the container, you only see its file system and process space. Because the kernel is shared, you are limited to the modules and drivers that it has loaded.

6.1.12 Docker

Docker is a container virtualization technology that offers the promise of a more efficient, lightweight approach to application deployment than most organizations are currently implementing. With a traditional virtualization hypervisor like VMware ESX, Microsoft Hyper-V or the open-source Xen and Kernel-based Virtual Machine (KVM) technologies, each virtual machine (VM) needs its own operating system. In contrast, with Docker, applications sit inside a container that resides on a single host operating system, that can serve many containers.