CHAPTER 3

XEN VIRTUAL MACHINE MONITOR

3.1 Virtual Machine

The term Virtual Machine (VM) spans a range of software and hardware mechanisms, from bytecode interpreters to virtual computer architectures. The term signifies what it is—a machine that is virtual in the sense that it imitates real machines.

The benefits of VMs are many. VMs can be run as a software mechanism on top of a conventional OS and allow more flexible use of the computer and OS's resources. The resources can be divided into separate execution environments, called domains or in many cases, simply VMs. There are three essential concepts that are valuable in virtualization.

- If the VM maintains isolation between domains, execution in one domain should not adversely affect execution in another domain. Also, one domain should not be able to inspect the state of another domain.
- Since the VM encapsulates the domains, the VM has access to the domains' states and can control and monitor their execution. If necessary, a program running in a domain should not be aware that the underlying execution environment is virtual.
- Also, VMs that maintain software compatibility allow the programs written for the same VM architecture be executed on VMs running on different hardware architectures.

Different methods for virtualizing a computer architecture exist. One method involves interpreting each instruction, and then emulating the computer architecture by simulating what will happen if this instruction is executing on a real computer architecture. Another method involves running the guest software directly on the hardware, while executing sections of the software in a monitoring mechanism. The advantage of the former method is that every instruction may be verified, logged and monitored. The drawback, however, is that a severe performance penalty incurs as each instruction is interpreted, translated and then executed. In simulating architectures, this method is often referred to as execution-driven simulation. The advantage of the latter method is that the guest software runs much faster, as there is less overhead from interpreting instructions. This method can also be used in simulating architectures, by executing instructions on a processor and logging the results, the execution can be monitored. This latter method is often referred to as trace-driven simulation.

Various techniques are used by the VMs in order to gain performance. The methods Microsoft's Virtual PC [11] and QEMU [12] use, involve translating blocks of instructions and running them natively, allowing blocks to be rerun without retranslating. This method gives a performance increase compared to pure interpreting, and may also still allow software to be run across real architectures. The method used by VMware [13] to virtualize an x86 processor involves running some code directly on hardware and some interpreted, giving better performance than the former methods. One of the drawbacks with VMware's solution is that it only allows x86 software to run on x86 machines.

With the modularity of virtualization technology, VM (*Virtual Machine*) has the following properties, such as flexible, secure, reliable, and easy to manage and configure, which can greatly minimized the hardware cost. However, besides these benefits, it also increases the complexity of virtual computing systems, and maximizes the performance of virtual computing systems. Furthermore, end user desktop will also receive the benefits of virtualization. For desktop users, the output of high-performance virtual machine display is an important metrics[11] for application. Nevertheless, there is a difference between the virtual environment of the display mechanism and the traditional computer display mechanism, which impact into the fact that we cannot copy the evaluation methods and techniques of the traditional computer. An important problem as how to effectively and evaluate virtual machine display performance pops up. Virtualization is great and it can do a lot of good for data center but in order to take advantage of its benefits, one needs to know how to deal with some of the problems that come with it.

Virtualization is a relatively young technology and even though it is adopted in many enterprises, there are still lots of technical challenges that need to be overcome. Discovering the reason for performance degradation is not always easy and this alone can cause a lot of problem in a production environment. Both server and storage virtualization come with issues but their nature differs[14].

Performance must be tuned for different types of application workloads[12]. Tuning for large files and continuous performance isn't optimal for small files and vice versa, and tuning for high performance of small files isn't best for large files and continuous performance. If one or two VMs are running especially I/O-intensive applications and hammering a disk, other VMs that

utilize the same data store may suffer the effects of disk I/O contention. This problem is more difficult to identify than overloaded CPU or memory. Increase the memory size and processor cores (if supports) to guest operating system. One has to consider host usage while allocating memory, but processor cores can be set to maximum without considering host usage. Nowadays virtualization is ubiquitous and virtualization technologies play an important and significant role in many IT fields. The main advantages of virtualization is it can rapidly reduce cost and uncertain of the experiments, portability of a virtual machine to another is simple, it has improved security, it enables parallelization, it decreases time expenses needed for administration of a large amount of desktops and workstations, etc. The virtual machine is a technology that creates one or multiple virtual environments on a single physical machine [13]. The virtual machines are separated from each other and the underlying physical machine, and they give users the illusion of accessing a real machine directly.

3.2 Virtual Machine Monitors

In virtualizing computer architectures, the mechanism that provides the user with VMs is often referred to as a Virtual Machine Monitor (VMM) or a hypervisor. Figure 3.1 illustrates a traditional VMM running in a host OS. The host OS runs directly on physical hardware, while the VMM runs as a process inside the host OS's execution environment, along with other user processes. Two guest OS are hosted by the VMM and run on the VMM's virtualized hardware. OSs are usually designed to run directly on a particular computer architecture, and thus they expect the computer to behave in a certain way. As the design of an OS has many degrees of freedom, if the VMM fails to virtualize the processor completely, in every detail, the OS might fail in execution. Most traditional VMMs aim to let any OS run unmodified on its hypervisor, that is, to completely virtualize the underlying architecture. While aiming to give its hosted OSs a virtual architecture logically equal to the OSs native architecture, VMMs also try to give good performance characteristics. As an x86 OS kernel protects itself from user processes through the use of ring protection, as do an x86 hypervisor require the notion of protection in order to be able protect sensitive system state from alteration and inspection from domains. Thus, whenever a domain's execution is performed directly on the processor—that is, uninterpreted—the guest OS must run in a less privileged ring than zero. When an OS is run in a ring other than zero, the

processor behaves differently from what the OS expects, and thus the execution may fail. This is the reason why VMWare interprets and emulates code which belongs to the guest OS kernel. Xen's approach to this problem is discussed in the next section.

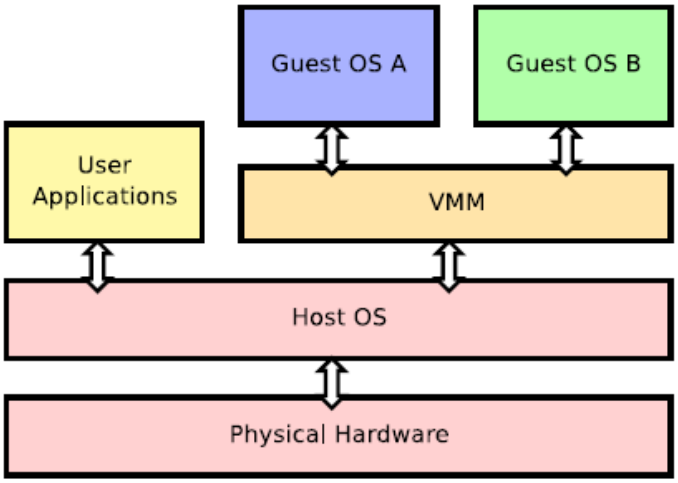

Figure 3.1: Illustration of a VMM hosted on a host OS, running two guest OSs.

3.3 Xen Virtual Machine Monitor

The Xen VMM addresses the main problem with traditional x86 VMMs, namely efficient utilization of processor resources. Most OSs require to run in ring zero and the processor to behave in a certain way. They thus incur the performance penalties of translation of OS code. Xen attacks this problem at the roots by modifying the OS so that it behaves differently. Instead of doing this during execution, the changes are made in the source code before compilation and execution. This has been inspired by some of the work in Denali by Andrew Whitaker et al. [15].

The term guest OS refers to an OS which is hosted in a VMM and the term domain to refer to the virtual machine in which the guest OS is running, i.e. on a VMM there are one or more domains, and in each domain a guest OS is running. Xen does not provide a fully virtualized platform on which guest OSs may execute. Instead, the guest OS needs to be modified to run on this platform—the hardware is effectively para-virtualized [15]. The modifications do not, however, affect the guest OS's interface to the user applications, thus they need not be modified. A few

OSs have been modified in order to run on the Xen/x86 architecture, such as NetBSD and Linux, with Linux being the most mature. Since Microsoft Windows' source code is closed for the public, a port of Windows relies on the willingness and efforts of Microsoft. Para-virtualization is further discussed in section 3.3.1.

The components of a complete Xen virtualization infrastructure include a Xen hypervisor, a privileged domain, one or more unprivileged domains and some user-space control tools, as shown in Figure 3.2. This contrasts to traditional VMMs (see Figure 3.1) in that the VMM runs as hosting mechanism, in full control of the hardware. As the names suggests, the privileged domain has higher privileges than the unprivileged domain. Control software running in the privileged domain has the privileges to control the operation of the hypervisor, including starting, suspending and shutting down unprivileged domains. The privileged domain is commonly referred to as domain 0.

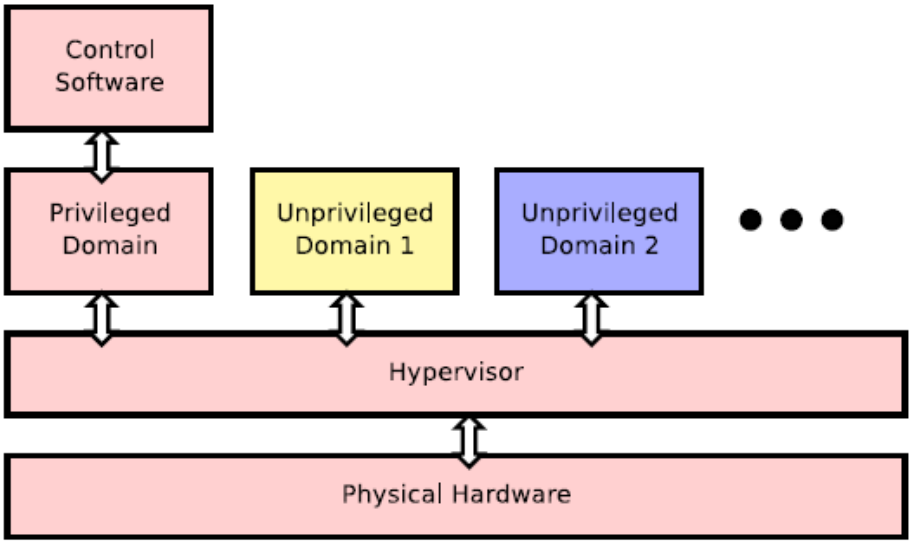

Figure 3.2: The Xen hypervisor, hosting several domains.

The usual mechanisms for interrupts from devices are replaced with an asynchronous event mechanism provided by the Xen hypervisor. CPU resources are distributed dynamically and fully utilized, while main memory resources are partitioned among domains. Block and network interface drivers must also be virtualized in order to divide these resources among domains. Also, in addition to managing its own memory, the hypervisor needs to manage the domains'

memory usage to make sure that they do not access each other's memory spaces. This is necessary in order to maintain isolation. The execution of domains is scheduled, similarly to the scheduling of processes in Linux. Xen provides three different scheduling algorithms, Borrowed Virtual Time (BVT) [16], Atropos and Round Robin, behind a unified API, making it easy to add different schedulers.

3.3.1 Paravirtualization

In Xen guest OSs, instructions that in some way may reveal that the OS is running on a VMM are replaced into functional procedures which emulate the original operation, statically in the source code. The guest OS is modified such that when it originally would execute privileged instructions, instead it makes a hypercall into the hypervisor, in which the operations are executed. This is similar to how a user process makes a system call into the kernel to execute privileged operations. System state which is protected is virtualized through the hypervisor. For instance, the hypervisor keeps a virtual CPU for each domain. When a guest OS would normally write to a protected register, instead, the hypervisor writes the value to the corresponding virtual register in the virtual CPU.

3.3.2 Event Handling

When sharing a single device resource such as, for instance, a harddrive, two guest OSs using the device simultaneously can lead to problems. One malicious or faulty domain can adversely affect the execution of other domains. Also, if several domains rely on a single device, if the device fails due to hardware failure, all the domains will be affected. If several domains are to share a device, it is preferable, in order to enforce isolation and maintain stability, to virtualize devices' resources. In order to maintain isolation, one domain does not know what the other is doing, and if a resource is to be shared, the sharing must be fair. Similarly, if a device is to be managed only by one domain, the hypervisor must ensure that only that domain accesses that device. Thus, the hypervisor manages access to devices. Keir Fraser et al. [17,18] discuss the Xen I/O model in detail.

The hypervisor uses an event mechanism to manage access. Instead of letting the domains handle interrupts directly, they are handled within the hypervisor. This way, the hypervisor can schedule the execution of the ISR of the domain which manages the device. After recieving an interrupt, the hypervisor schedules the execution of the corresponding domain and issues to it an asynchronous event notification. Event notifications are delivered to domains through event channels. The hypervisor communicates these events with a domain through a shared memory page. This approach lets guest OSs use their own drivers unmodified to control devices.

As well as letting domains manage devices directly, a Xen managed system as a whole can benefit from devices being virtualized. This allows devices to be shared while ensuring isolation and has other potential benefits. Virtualized devices are managed in isolated driver domains (IDD). Transfers between an IDD and a guest OS are serviced by I/O descriptor rings, in which transfers are asynchronously scheduled. IDDs and guest OSs are notified of queued transfer descriptors through event channels.

Block devices, such as harddrives, are managed using IDDs, as illustrated in Figure 3.3. Domain 0 has direct access to a harddrive through the hypervisor's event mechanism, while providing an abstraction for other domains through an IDD, through which they have access to virtual block devices. A driver that domain 0 uses to provide other domains with a virtual block or network device is called a backend driver, while a driver that a domain uses to control a virtual block or network device is called a frontend driver. Domain 0 uses the same driver as the native driver of the guest OS that instantiates domain 0 uses, to control the device physically.

3.3.3 Memory Management

Since several domains share the same memory, care has to be taken in order to maintain isolation. The hypervisor must ensure that two unprivileged domains do not access the same memory area. Each page or directory table update has to be validated by the hypervisor in order to ensure that domains only manipulate their own tables. The domain may batch these operations to make sequential updates more efficient. Segmentation is virtualized in a similar manner. The hypervisor makes sure that also domains' segments do not map memory areas that overlap or that are invalid in any other way.

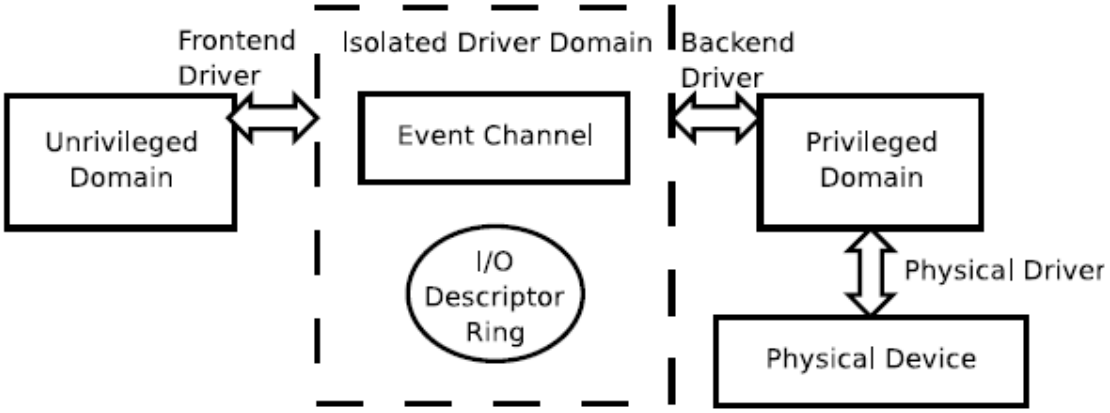

Figure 3.3: Device Virtualization in Xen

Domains are allocated physical memory by the hypervisor, which is not necessarily contiguous. This memory does not map directly to the machine's physical memory and is in that sense pseudo-physical. The translation from physical to pseudo-physical is not transparent, and guest OSs need to translate addresses from pseudo-physical to physical themselves. The hypervisor provides a machine-to-physical and a physical-to-machine, which map pseudo-physical to physical addresses and vice versa.

3.3.4 Time

Guest OSs are provided with different mechanisms to manage time that replace their native counterparts. Wall time is kept by the hypervisor and can be read by a domain. Also, virtual time is kept and only progresses during a domain's execution, which allows correct scheduling of domains' internal tasks. Xen needs to shuffle the execution of domains just as native OSs need to shuffle the execution of tasks in order to maintain the illusion of simultaneous execution. The BVT scheduling algorithm is presented by Kenneth J. Duda [16]. It allows a domain to be dispatched with low latency. This allows the domain to be quickly dispatched to respond to events, such as packets arriving on the network. BVT operates with the concept of Effective Virtual Time (EVT). Whenever the scheduler selects a new domain to run, it picks the domain with the lowest EVT.