Rohan Swaby

Operating Systems

Instructor: Professor P. Barnett

Kernel-based Virtual Machine

Introduction

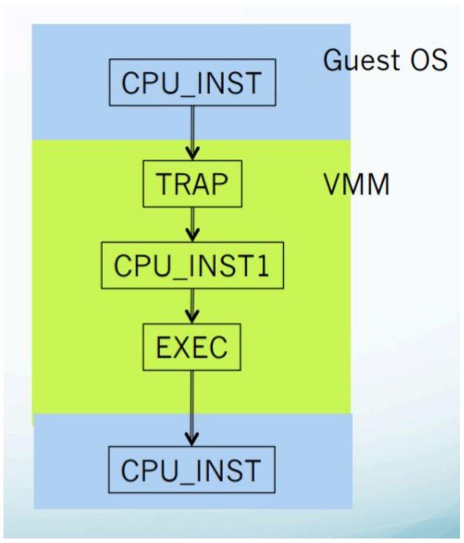
Virtualization have been around since the 1960s let by companies like Bell labs, IBM. Virtualization allow for running task in parallel by sharing computing resources. Virtualization also made it possible to run more than one virtual machine on top of a host. This paper will focus more on kernel based virtual machines (**KVM**). KVMs are designed to take advantage of hardware assisted virtualization. we will also look at how interrupts, traps and signals are handled by the KVM, how the hypervisor communicate with the operating system and the hardware and how does KVM differ from XEN VMWare or PowerVM in the way it handles interrupts, traps and signals.

Interrupts for KVM

Interrupts are asynchronous events generated by external components such as I/O devices.The way interrupts works is the currently executed code is interrupted and control moves to a predefined handler that is specified in an in-memory look up table. Interrupts can be software or hardware generated. The intel architecture has interrupt controllers that are referred to as **Advanced Programmable Interrupt Controllers (APIC)** that are designs intended to solve interrupt routing efficiency issues in multiprocessor computer systems. The APIC design has a **local component(LAPIC)** integrated into the processor itself and an optional I/O APIC component. In hardware assisted virtualization registers are emulated in VMCs (Virtual machine Control structure shadowing), Inside that is a special register called the IDT register. There are also VM execution control fields in the VMCS for controlling and handling interrupts. Some of the CPU hardware capabilities such as LAPIC which are emulated in hardware, hardware capabilities emulated in hardware. Because LAPIC, IDT and other registers are emulated in hardware interrupts can be sent to the LAPIC of a VM which can the be delivered to CPU executing the VM in non root mode. The IDT is in guest mode in the VMCS, the virtual IRQ can vector into it directly and the interrupt handler invoked directly. Once the EOI is written into it is virtualized again. All these without exiting the VM. KVM also provides interrupt injection facilities to userspace. This means to determine when the quest is ready to accept and interrupt, The interrupt flag must be set. To actually inject the interrupt when the quest is ready.

Traps

In general, traps occur when the guest operating system does not the correct privileges to run some instructions. Some operating systems that run on a hypervisor such as KVM do not know they are running on a hypervisor. When the virtual machine tries to execute privileged instructions, those instructions create a trap that goes into the hypervisor. The hypervisor then emulated this instruction and perform the same function the operating system would. Another way to say this is when the quest OS executes a privileged instruction, control is passed to the VMM (VMM traps on instruction) which decides how to handle instruction VMM generates instruction. VMM generates instructions to handle trapped instructions which you could say is emulation. Non-privileged instruction do not trap, they stay in the quest context. The book refers to this as the Trap-and-Emulation, This implementation does have problems. Trap-and-emulate is expensive, it requires context-switch from the quest OS mode to VMM



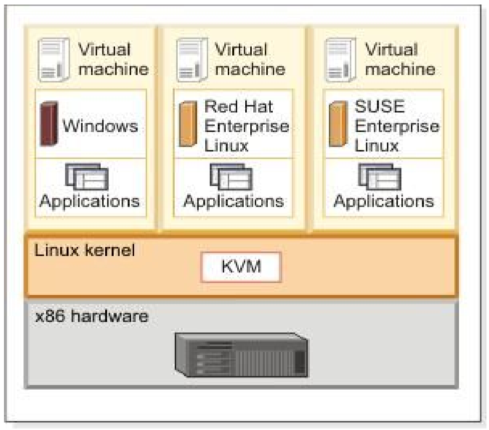
Traps in kvm

Signals kvm

In kernel-based virtual machines the userspace calls the kernel to the execute guest code until it encounters an I/O instructions, or until an external event such as arrival of a network packet or timeout occurs. These external events are represented by signals.

Hypervisors

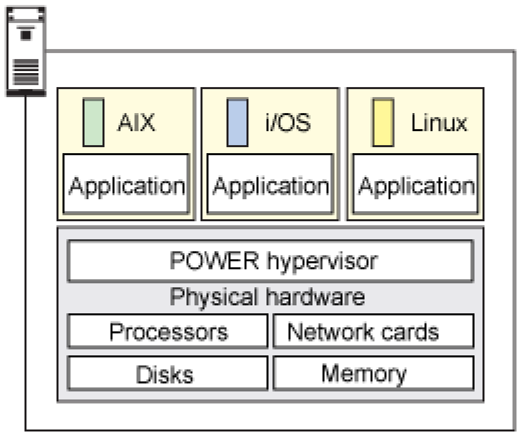
Hypervisors serve a very important role in kernel-base visualization. A hypervisor allows multiple OS to be able to share one single hardware. Hypervisors create virtual machine environments And manages calls to the processor, memory, hard disk and network. For hypervisors to work the kvm requires hardware with virtualization extensions to connect to the guest OS. The image below shows how the kvm architectures looks including the hypervisors on x86 architectures.



XEN

Xen hypervisors are similar to KVM hypervisors in that they both make it possible to run many instances of an operating system or different operating systems in parallel on a single host machine. In xen architecture the hypervisor is referred to a Dom0 and VMs as guest. In xen fully virtualized mode provides guest kernel with emulated interrupt controllers (APICs and IOAPICs). Each instruction that interacts with the APIC requires a trip up into Xen Project and a software instruction decode; and each interrupt delivered requires several of these emulations.

PowerVM



PowerVM Architecture

PowerVM provides flexibility in that it combines dedicated and shared resources in a partition supporting dynamic resource allocation. PowerVM does have some advantages over VMware. One advantage is PowerVM leverages SMT4 even when the number of virtual processors (8) is lower than the number of logical processors (32) in the system. VMware on the other hand maps a logical processor to a virtual processor so at any point in time a VMware virtual machine configured with 8 virtual processors can only consume eight logical processors. PowerVMs hypervisors provide the ability to divide physical system resources into isolated logical partitions. Each logical partition operates like an independent system running its own operating system. The power hypervisor controls hardware I/O interrupt management facilities for logical partitions. Furthermore, because powervms are capable of hosting many system images, the importance of isolating and handling service interrupts(planned and unplanned) becomes significant.

VMWare

According to the textbook operating system concepts, VMware abstracts the Intel X86 and compatible hardware into isolated virtual machines. Vmware can be described as a type 2 hypervisor, meaning there is very little operating system involvement in these application level virtual machine managers. The VMware architecture is another process run and managed by the host, which makes the host unaware virtualization is happening.

References

IO and Interrupt Virtualization - Masum Z. Hasan

https://sites.google.com/site/masumzh/articles/hypervisor-based-virtualization/io-and-interrupt-virtualization

A Comparison of PowerVM and VMware Virtualization Performance - IBM Systems and Technology Group

https://www.mercurymagazines.com/pdf/NCIBMP71.pdf

kvm: the Linux Virtual Machine Monitor

https://www.ece.cmu.edu/~ece845/docs/kvm-paper.pdf

OS Virtualization - Brandon D. Shroyer

http://www.cs.rochester.edu/~sandhya/csc256/seminars/bshroyer_virtualmachines.pdf

ARM® Interrupt Virtualization - Andre Przywara

http://events17.linuxfoundation.org/sites/events/files/slides/ARM_Interrupt_Virtualization_Przywara.pdf

Enabling Optimized Interrupt/APIC Virtualization in KVM - Jun Nakajima

https://www.linux-kvm.org/images/7/70/2012-forum-nakajima_apicv.pdf

Xen user manual

https://wiki.xenproject.org/images/4/47/Xen_3_user_manual.pdf