KVM – KERNEL BASED VIRTUAL MACHINE

BACKGROUND

Virtualization has begun to transform the way that enterprises are deploying and managing their infrastructure, providing the foundation for a truly agile enterprise, so that IT can deliver an infrastructure that is flexible, scalable, and most importantly economical by efficiently utilizing resources.

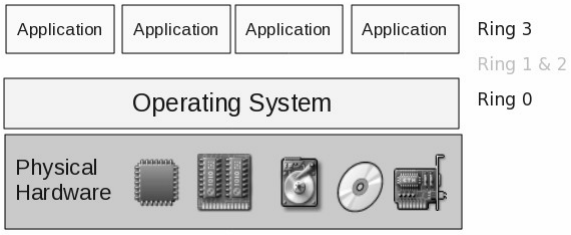
10 years ago virtualization was unheard of in the x86 market it was reserved for mainframe and high end UNIX systems. Over the last 3 to 4 years there has been exponential growth in the virtualization market both in terms of customer adoption and in terms of the rise of the number vendors in the virtualization space; from new hypervisor vendors to virtualization management vendors too numerous to mention.

VIRTUALIZING THE X86 ARCHITECTURE

The x86 architecture has proven to be the dominate platform in enterprise computing, moving from its humble beginnings in desktop systems to now, powering the large enterprise applications that run businesses across the globe. The current generation of x86 CPUs include features such as large scale multi-threading with 8 or more processing cores, support for large memory systems with NUMA and integrated memory controllers, high speed CPU interconnects and chipset for support for advanced reliability, availability and serviceability (RAS) features. These features were once reserved for mainframe and high end UNIX systems, today x86 servers with 2 or 4 sockets are replacing expensive UNIX/RISC systems while delivering better performance and 4 and 8 socket servers are challenging mainframe class systems.

While the x86 platform has evolved significantly over it's lifetime it has maintained it's core architecture to provide backward compatibility. A number of elements from the original architecture threatened to limit the growth of the x86 platform, the most significant of which was the physical address space which was limited to 32 bits. In 2003 Intel and AMD added 64bit extensions to address that limitation and today the x86_64 family of processors from Intel and AMD are ubiquitous from laptops and desktops through to high end servers supporting large scale symmetric multiprocessing systems and terabytes of ram.

In order to provide a secure operating environment the x86 architecture provide a mechanism for isolating user applications from the operating system using the notion of privilege levels.



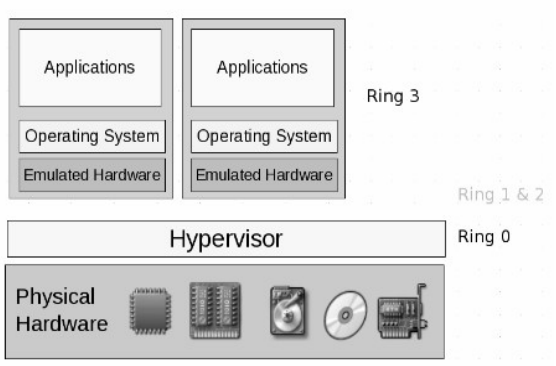
In this model the processor provides 4 privilege levels, also known as *rings* which are arranged in a hierarchical fashion from ring 0 to ring 3. Ring 0 is the most privileged with full access the hardware and is able to call privileged instructions. The operating system runs in ring 0 with the operating system kernel controlling access to the underlying hardware. Rings 1, 2 and 3 operate at a lower privilege level and are prevented from executing instructions reserved for the operating system. In commonly deployed operating systems such as Linux and Microsoft Windows the operating system runs in ring 0 and the user applications run in ring 3. Rings 1 and 2 historically have not used by modern commercial operating systems. This architecture ensures that an application running in ring 3 that is compromised cannot make

privileged system calls, however a compromise in the operating system running in ring 0 hardware exposes applications running in the lower privileged levels.

While this model provides benefit for traditional, “bare metal” deployments it presents challenges in a virtualized environment.

In a virtualized environment the hypervisor must run at the most privileged level, controlling all hardware and system functions. In this model the virtual machines run in a lower privileged ring, typically in ring 3.

Within ring 3 we can see the virtual machine running with an operating system running on virtual (emulated) hardware. Since the operating system was originally designed to run directly on hardware it expects to be running in ring 0 and will make privileged calls that are not permitted in ring 3. When the operating system makes these privileged calls the hardware will trap the instructions and issue a fault, which will typically destroy the virtual machine.



Much of the work performed in an x86 virtualization solution centers around handling the deprivileging of the operating system running in the virtual machine, moving the operating system kernel from ring 0 to ring 1 (or higher) this is sometimes referred to as “ring compression”.

Early x86 hypervisors such as Bochs created a fully emulated system with the x86 CPU completed emulated in software. This technique resulted in very poor performance so a more advanced technique was developed for use in the first generation of commercial x86 hypervisors.

Binary Translation

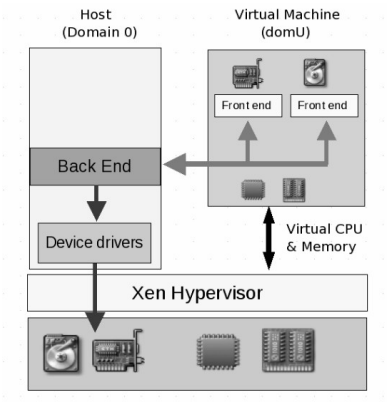
In this model, pioneered by VMware, instead of emulating the processor, the virtual machine runs directly on the CPU. When privilege instructions are encountered the CPU will issue a trap that could be handled by the hypervisor and emulated. However there are a number of x86 instructions that do not trap for example *pushf/popf* and there are some cases where the virtual machine could identify that it was running in ring 3. To handle these cases a technique called *Binary Translation* was developed. In this model the hypervisor scans the virtual machine memory and intercepts these calls before they are executed and dynamically rewrites the code in memory. The operating system kernel is unaware of the change and operates normally. This combination of trap-and-execute and binary translation allows any x86 operating system to run unmodified upon the hypervisor. While this approach is complex to implement it yielded significant performance gains compared to full emulating the CPU.

Paravirtualization

While the emulation and binary-translation approached focused on how to handle a privileged instruction executed in a virtual machine a different approach was taken by the open source Xen project.
Instead of handling a privileged instruction the approach with paravirtualization is to modify the guest operating system running in the virtual machine and replace all the privileged instructions with direct calls into the hypervisor. In this model, the modified guest operating system is aware that it is running on a hypervisor and can cooperate with the hypervisor for improved scheduling and I/O, removing the need to emulate hardware devices such as network cards and disk controllers.

Since paravirtualization requires changes to the operating system it needs to be implemented by the operating system vendor. These changes were made to the Linux operating system initially in the form of custom patches to the Linux kernel and later were incorporated into the mainline Linux kernel, starting with Kernel 2.6.23. Linux distributions that use kernels earlier than 2.6.23, for example Red Hat Enterprise Linux 5 use kernels with a customized set of patches.

The Xen Hypervisor Platform is comprised of two components – the Xen hypervisor which is responsible for the core hypervisor activities such as CPU, memory virtualization, power management and scheduling of virtual machines.
The Xen hypervisor loads a special, privileged virtual machine called Domain0 or dom0. This virtual machine has direct access to hardware and provides device drivers and I/O management for virtual machines.



Each virtual machine, known as an unprivileged domain or domU, contains a modified Linux kernel that instead of communicating directly with hardware interfaces with Xen hypervisor.
CPU and memory access are handled directly by the Xen hypervisor but I/O is directed to domain 0. The Linux kernel includes “front end” devices for network and block I/O. Requests for I/O are passed to the “back end” process in domain 0 which manages the I/O.

In this model the guest kernel in domU runs in ring 1 while user space runs in ring 3.

Domain 0 can be implemented using Linux, BSD or Solaris but is most commonly implemented using a modified Linux distribution. Red Hat, Citrix and Oracle all use a domain 0 based on Red Hat Enterprise Linux 5 sources with the 2.6.18 kernel.
In order to operate as domain 0 the Linux kernel has to be modified. The Xen modifications to the Linux domain 0 have not been incorporated into the *upstream* Linux kernel so all vendors shipping a Xen solution based on Linux maintain a forked copy of the Linux kernel.

While Xen is often categorized as being a thin “type-1” hypervisor the entire platform requires a domain 0 operating system in order to access hardware.

Hardware Assisted Virtuallization

Both Intel and AMD developed extensions to the x86 architecture to provide features that could be used by hypervisor vendors to simplify CPU virtualization. The first CPUs including these features were released late in 2005. Today most Intel and AMD CPUs include hardware virtualization support including desktop, laptop and server product lines.

The implementations of these features by Intel (VT-X) and AMD (AMD-V) are different but use a similar approach. A new operating mode is added to the CPU which can now operate in host mode or guest mode. A hypervisor can request that a process operates in guest mode, in which it will see still see the four traditional ring/privilege levels, but the CPU is instructed to trap privileged instructions and then return control to the hypervisor.
Using these new hardware features, a hypervisor does not need to implement the binary translation that was previously required to virtualize privileged instructions.

While VT-X and AMD-V reduced the overhead for virtualizing the CPU a significant amount of resources are expended by the hypervisor in handling memory virtualization.
Because the guest operating system cannot directly access memory the hypervisor must provide a virtualized memory implementation in which the hypervisor provides mapping between the physical host memory and the virtual memory used by the virtual machine. This is often implemented using shadow page tables within the hypervisor.

AMD developed the Rapid Virtualization Indexing (RVI) feature, previously know as nested page tables, and Intel developed the Extended Page Table (EPT) feature. These are incorporated into the recent generation of Intel and AMD CPUs. These features provide a virtualized memory management unit (MMU) in hardware that delivers significant performance improvements compared to the software only implementation.

Both Intel and AMD continue to add new features to hardware to improve performance for virtualization, offloading more features from the hypervisor into “the silicon” to provide improved performance and a more robust platform. The current generation of Intel and AMD CPUs, and supporting chipsets, are adding support for I/O offload with features such as secure PCI pass-through, using Intel VT-D or AMD IOMMU, allowing PCI devices on the host to be passed directly into the virtual machine. Single Root I/O virtualization (SR/IOV) extends those features to allow special PCI devices to be split into multiple virtual PCI devices that can be passed through to individual virtual machines. These features allow virtual machines to achieve the same I/O performance as bare metal systems.

KVM

Kernel-based Virtual Machine (KVM) project represents the latest generation of open source virtualization. The goal of the project was to create a modern hypervisor that builds on the experience of previous generations of technologies and leverages the modern hardware available today.

KVM is implemented as a loadable kernel module that converts the Linux kernel into a bare metal hypervisor. There are two key design principals that the KVM project adopted that have helped it mature rapidly into a stable and high performance hypervisor and overtake other open source hypervisors.

Firstly, because KVM was designed after the advent of hardware assisted virtualiaztion, it did not have to implement features that were provided by hardware. The KVM hypervisor *requires* Intel VT-X or AMD-V enabled CPUs and leverages those features to virtualize the CPU.
By requiring hardware support rather than optimizing with it if available, KVM was able to design an optimized hypervisor solution without requiring the “baggage” of supporting legacy hardware or requiring modifications to the guest operating system.

Secondly the KVM team applied a tried and true adage – “don't reinvent the wheel”.
There are many components that a hypervisor requires in addition to the ability to virtualize the CPU and memory, for example: a memory manager, a process scheduler, an I/O stack, device drivers, a security manager, a network stack, etc. In fact a hypervisor is really a specialized operating system, differing only from it's general purpose peers in that it runs virtual machines rather than applications.
Since the Linux kernel already includes the core features required by a hypervisor and has been hardened into an mature and stable enterprise platform by over 15 years of support and development it is more efficient to build on that base rather than writing all the required components such as a memory manager, scheduler, etc from the ground up.

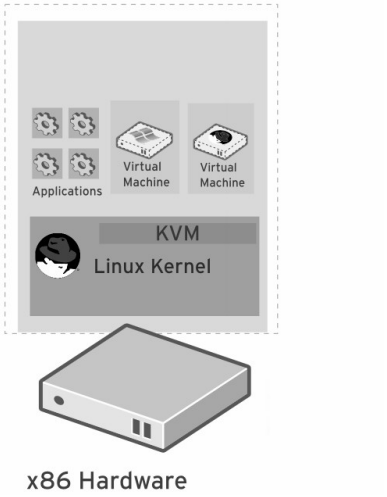
In this regard the KVM projected benefited from the experience of the Xen. One of the key challenges of the Xen architecture is the split architecture of domain0 and the Xen hypervisor. Since the Xen hypervisor provides the core platform features within the stack, it has needed to implement these features, such as scheduler and memory manager from the ground up.
For example while the Linux kernel has a mature and proven memory manager including support for NUMA and large scale systems, the Xen hypervisor has needed to build this support from scratch. Likewise features like power management which are already mature and field proven in Linux had to be re-implemented in the Xen hypervisor.

Another key decision made by the KVM team was to incorporate the KVM into the upstream Linux kernel. The KVM code was submitted to the Linux kernel community in December of 2006 and was accepted into the 2.6.20 kernel in January of 2007. At this point KVM became a core part of Linux and is able to inherit key features from the Linux kernel. By contrast the patches required to build the Linux Domain0 for Xen are still not part of the Linux kernel and require vendors to create and maintain a fork of the Linux kernel. This has lead to an increased burden on distributors of Xen who cannot easily leverage the features of the upstream kernel. Any new feature, bug fix or patch added to the upstream kernel must be back-ported to work with the Xen patch sets.

In addition to the broad Linux community KVM is supported by some of the leading vendors in the software industry including Red Hat, AMD, HP, IBM, Intel, Novell, Siemens, SGI and others

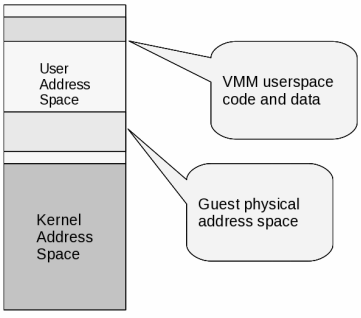
KVM ARCHITECTURE

In the KVM architecture the virtual machine is implemented as regular Linux process, schedule by the standard Linux scheduler. In fact each virtual CPU appears as a regular Linux process. This allows KVM to benefit from all the features of the Linux kernel.
Device emulation is handle by a modified version of QEMU that provides an emulated BIOS, PCI bus, USB bus and a standard set of devices such as IDE and SCSI disk controllers, network cards, etc.



Security

Since a virtual machine is implemented as a Linux process it leverages the standard Linux security model to provide isolation and resource controls. The Linux kernel includes SELinux (Security-Enhanced Linux) a project developed by the US National Security Agency to add mandatory access controls, multi-level and multi-category security as well as policy enforcement. SELinux provides strict resource isolation and confinement for processes running in the Linux kernel. The sVirt project builds on SELinux to providean infrastructure to allow an administrator to define policies for virtual machine isolation. Out of the box sVirt ensures that a virtual machines resources can not be accessed by any other process (or virtual machine) and this can be extended by the administrator to define fine grained permissions, for example to group virtual machines together to share resources.
Any virtual environment is only as secure as the hypervisor itself, as organizations look to deploy virtualization more pervasively throughout their infrastructure security becomes a key concern, even more so in cloud computing environments. The hypervisor is undoubtedly a tempting target for hackers as an exploited hypervisor could lead to the compromise of all virtual machines it is hosting, in fact we have already seen hypervisor exploits for example the *“Invisible Things Lab”* exploit in 2008 where a Xen domU was able to compromise the domain0 host. SELinux and sVirt provide an infrastructure that provides a level of security and isolation unmatched in industry.



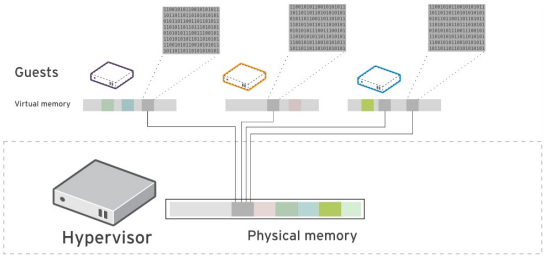
Memory Management

KVM inherits the powerful memory management features of Linux
The memory of a virtual machine is stored as memory is for any other Linux process and can be swapped, backed by large pages for better performance, shared or backed by a disk file.
NUMA support allows virtual machines to efficiently access large amounts of memory.

KVM supports the latest memory virtualization features from CPU vendors with support for Intel's Extended Page Table (EPT) and AMD's Rapid Virtualization Indexing (RVI) to deliver reduced CPU utilization and higher throughput.

Memory page sharing is supported through a kernel feature called Kernel Same-page Merging(KSM). KSM scans the memory of each virtual machine and where virtual machines have identical memory pages KSM merges these into a single page that it shared between the virtual machines, storing only a single copy. If a guest attempts to change this shared page it will be given it's own private copy.

When consolidating many virtual machines onto a host there are many situations in which memory pages may be shared – for example unused memory within a Windows virtual machine, common DLLs, libraries, kernels or other objects common between virtual machines.
With KSM more virtual machines can be consolidated on each host, reducing hardware costs and improving server utilization.



Hardware support

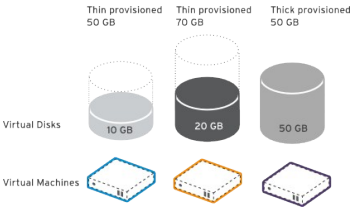
Since KVM is a part of Linux it leverages the entire hardware ecosystem, so any hardware device supported by Linux can be used by KVM. Linux enjoys one of the largest ecosystem of hardware vendors and the nature of the open source community, where hardware vendors are able to participate in the development of the Linux kernel, ensures that the latest hardware features are rapidly adopted in the Linux kernel, allowing KVM to utilize a wide variety of hardware platforms.

As new features are added to the Linux kernel KVM inherits these without additional engineering and the ongoing tuning and optimization of Linux immediately benefits KVM.

Storage

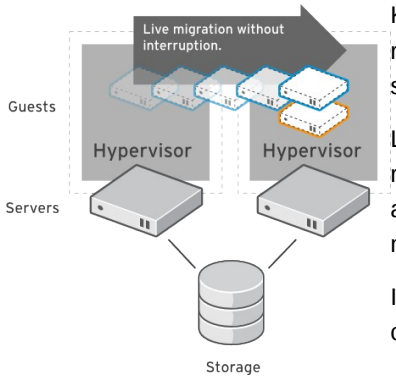
KVM is able to use any storage supported by Linux to store virtual machine images, including local disks with IDE, SCSI and SATA, Network Attached Storage (NAS) including NFS and SAMBA/CIFS or SAN with support for iSCSI and Fiber Channel. Multipath I/O may be used to improve storage throughput and to provide redundancy. Again, because KVM is part of the Linux kernel it can leverage a proven and reliable storage infrastructure with support from all the leading storage vendors with a storage stack that has been proven in production deployments worldwide.

KVM also supports virtual machine images on shared file systems such as the Global File System (GFS2) to allow virtual machine images to be shared between multiple hosts or shared using logical volumes.
Disk images support thin provisioning allowing improved storage utilization by only allocating storage when it is required by the virtual machine rather than allocating the entire storage up front.



The native disk format for KVM is QCOW2 which includes support for snapshots allowing multiple levels of snapshots, compression and encryption.

Live Migration



KVM supports live Migration which provides the ability to move a running virtual machine between physical hosts with no interruption to service.

Live Migration is transparent to the end user, the virtual machine remains powered on, network connections remain active and user applications continues to run while the virtual machine is relocated to a new physical host.

In addition to live migration KVM supports saving a virtual machine's current state to disk to allow it to be stored and resumed at a later time.

Guest Support

KVM supports a wide variety of guest operating systems, from mainstream operating systems such as Linux and Windows to other platforms including OpenBSD, FreeBSD, OpenSolaris, Solaris x86 and MS DOS.

In Red Hat's enterprise offerings, KVM has been certified under Microsoft's Server Virtualization Validation Program (SVVP) to ensure users deploying Microsoft Windows Server on Red Hat Enterprise Linux and Red Hat Enterprise Virtualization Hypervisor (RHEV-H) will receive full commercial support from Microsoft.

Device Drivers

KVM supports *hybrid virtualization* where paravirtualized drivers are installed in the guest operating system to allow virtual machines to use an optimized I/O interface rather than emulated devices to deliver high performance I/O for network and block devices.
The KVM hypervisor uses the VirtIO standard developed by IBM and Red Hat in conjunction with the Linux

community for paravirtualized drivers which is a hypervisor independent interface for building device drivers allowing the same set of device drivers to be used for multiple hypervisors, allowing for better guest

interoperability. Today many hypervisors use proprietary interfaces for paravirtualized device drivers which means that guest images are not portable between hypervisor platforms. As more vendors adopt the VirtIO framework guest images will become more easily transferable between platforms and reduce certification testing and overhead.

VirtIO drivers are included in modern Linux kernels (later than 2.6.25), included in Red Hat Enterprise Linux 4.8+, 5.3+ and available for Red Hat Enterprise Linux 3.

Red Hat had developed VirtIO drivers for Microsoft Windows guests for optimized network and disk I/O that have been certified under Microsoft's Windows Hardware Quality Labs certification program (WHQL).

Performance and Scalability

KVM inherits the performance and scalabiltiy of Linux, supporting virtual machines with up to 16 virtual CPUs and 256GB of ram and host systems with 256 cores and over 1TB or RAM.

With up to 95%-135% performance relative to bare metal for real-world enterprise workloads like SAP, Oracle, LAMP and Microsoft Exchange; more than 1 million messages per second and sub 200 micro-second latency in virtual machines running on a standard server; and the highest consolidation ratios with more than 600 virtual machines running enterprise workloads on a single server, KVM allows even the most demanding application workloads to be virtualized.

Improved scheduling and resource control

In the KVM model, a virtual machine (Windows or Linux) is a Linux process. It is scheduled and managed by the standard Linux kernel. Over the past several years, the community has advanced the core Linux kernel to a point where it has industry leading features, performance stability, security and enterprise robustness. The current version of the Red Hat Enterprise Linux kernel supports setting relative priorities for any process including virtual machines. This priority is for an aggregate measure of CPU, memory, network and disk IO for a given virtual machine, and provides the first level of Quality of Service (QoS) infrastructure for virtual machines.

The modern Linux scheduler accrues some further enhancements that will allow a much finer-grain control of the resources allocated to a Linux process and will allow guaranteeing a QoS for a particular process. Since in the KVM model, a virtual machine is a Linux process, these kernel advancements naturally accrue to virtual machines operating under the KVM architectural paradigm. Specifically, enhancements including CFS, control-groups, network name spaces and real-time extensions will form the core kernel level infrastructure for QoS, service levels and accounting for VMs.

The Linux kernel includes a new advanced process scheduler called the completely fair scheduler (CFS) to provide advanced process scheduling facilities based on experience gained from large system deployments. The CFS scheduler has been extended to include the CGroups (control groups) resource manager that allows processes, and in the case of KVM – virtual machines, to be given shares of the system resources such as memory, cpu and I/O. Unlike other virtual machine schedulers that give proportions of resources to a virtual machine based on weights, cgroups allow minimums to be set not just maximums, allowing guaranteed resources to a virtual machine but allowing the virtual machine to use more resources if available. Network name-spaces is a similar kernel-level infrastructure that allows finer grain controls and guarantees a minimum network SLA for a given virtual machine.

These advanced features in the kernel allow resource management and control at all levels – CPU, memory, network and I/O.

Lower latency and higher determinism

In addition to leveraging the new process scheduler and resource management features of the kernel to guarantee the CPU, memory, network and disk IO SLA for each VM, the Linux kernel will also feature real-time extensions. These allow much lower latency for applications in virtual machines, and a higher degree of determinism which is important for mission critical enterprise workloads. Under this operating model, kernel processes that require a long CPU time slice are divided into smaller components and scheduled/processed accordingly by the kernel. In addition, mechanisms are put in place that allow interrupts from virtual machines to be prioritized better than if long kernel processes were to consume more CPU. Hence, requests from virtual machines can be processed faster, thereby significantly reducing application processing latency and improving determinism.

Copyright © 2009 Red Hat, Inc. “Red Hat,” Red Hat Linux, the Red Hat “Shadowman” logo, and the products listed are trademarks of Red Hat, Inc., registered in the U.S. and other countries. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.