Introduction to Virtualization and KVM

Important Notice : Courseware - Legal

kaiwanTECH Linux OS Corporate Training Programs
Please do check out our current offering of world-class, seriously-valuable, high on returns, technical Linux OS corporate training programs here: http://bit.ly/ktcorp

First see:

- Very good (introductory technical) series on Virtualization :
  "Hardware Virtualization: the Nuts and Bolts"
- "KVM for Server Virtualization: An Open Source Solution Comes of Age". RH whitepaper.
- "Best Practices for KVM", IBM whitepaper
- *Additional Resources – see Appendix A.*

"I once heard that hypervisors are the living proof of operating system's incompetence" :-D
- *see the full article here (LWN).*

Introduction

"... The implicit assumption is that the hardware resources of a system are managed by a single operating system. This binds all hardware resources into a single entity under a single management regime. And this, in turn, limits the flexibility of the system, not only in terms of available application software *<< ever required MS Office on Linux? :-) >>* , but also in terms of security and failure isolation, especially when the system is shared by multiple users or groups of users.

Virtualization provides a way of relaxing the foregoing constraints and increasing flexibility. When a system (or subsystem), is *virtualized*, its interface and all resources visible through the interface are mapped onto the interface and resources of a real system actually implementing it. Consequently, the real system is transformed so that it appears to be a different, virtual system or even a set of multiple virtual systems.

Formally, virtualization involves the construction of an isomorphism[1] that maps a virtual *guest* system to a real *host* (Popek and Goldberg 1974). This isomorphism, illustrated in Fig 1.2, maps the guest state to the host state (function V in Fig 1.2), and for a sequence of operations, *e*, that modifies the state in the guest (the function *e* modifies state *Si* to state *Sj)* there is a corresponding sequence of operations *e'* in the host that performs an equivalent modification to the host's state (changes *S'i* to *S'j*).

1 iso=same, morph=shape or form. Isomorphism: "Mathematics. a one-to-one relation onto the map between two sets, which preserves the relations existing between elements in its domain."

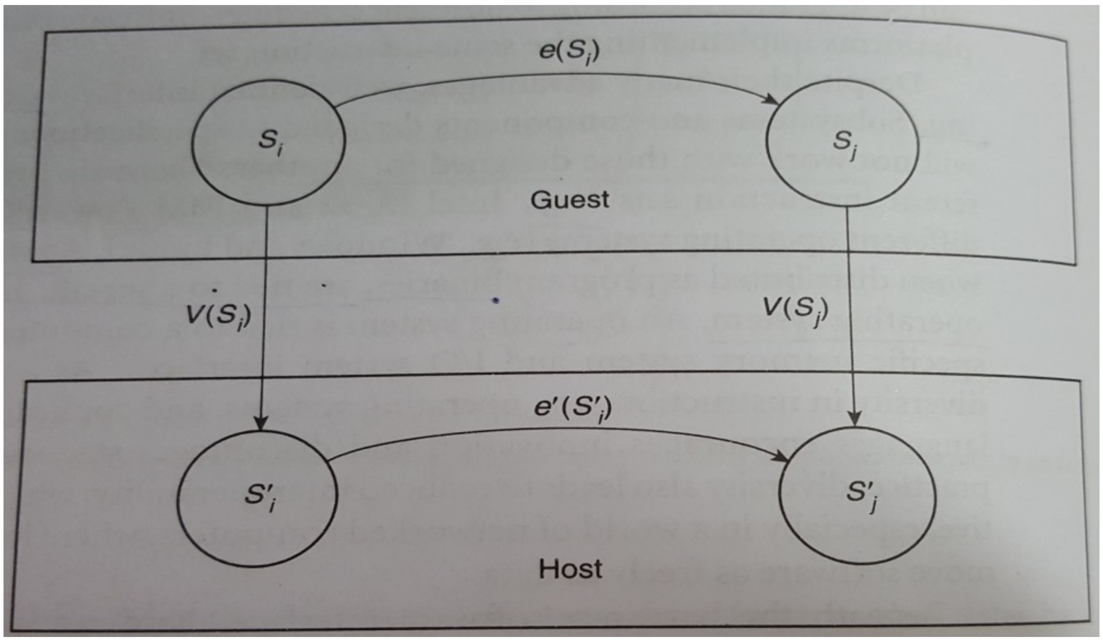


Fig 1.2 Virtualization. Formally, virtualization is the construction of an isomorphism between a guest system and a host; e' o V(Si) = V o e(Si).

Although such an isomorphism can be used to characterize abstraction as well as virtualization, we distinguish between the two: Virtualization differs from abstraction in that virtualization does not necessarily hide details: the level of detail in a virtual system is often the same as that in the underlying system.

...

Virtual machines can also employ emulation techniques to support cross-platform software compatibility. For example, a platform implementing the PowerPC instruction set can be converted into a virtual platform running the IA-32 instruction set. Consequently, software written for one platform will run on the other.

This compatibility can be provided either at the system level (for eg. to run a Windows OS on a Macintosh) or at the program or process level (eg to run Excel on a Sun Solaris/SPARC platform) << *system virtualization vs process virtualization* >>.

In addition to emulation, virtual machines can provide dynamic, on-the-fly optimization of program binaries.

Finally, through emulation, virtual machines can enable new, proprietary instruction sets, eg, incorporating very long instruction words (VLIWs), while supporting programs in an existing, standard instruction set.

...

The Versitality of Virtual Machines

... A computer user might have a Java application running on a laptop PC. This is nothing special; it is done via the Java virtual machine (JVM) developed for IA-32/Linux.

However, <<*consider a scenario where*>> the user happens to have a Linux installed as an OS VM via VMware executing on a Windows PC. And, as it happens, the IA-32 hardware is in fact a Transmeta Crusoe, a codesigned VM implementing a VLIW ISA with binary translation (what Transmeta calls *code morphing*) to support the IA-32 ISA. By using the many VM technologies, a Java bytecode program is actually executing as native VLIW.

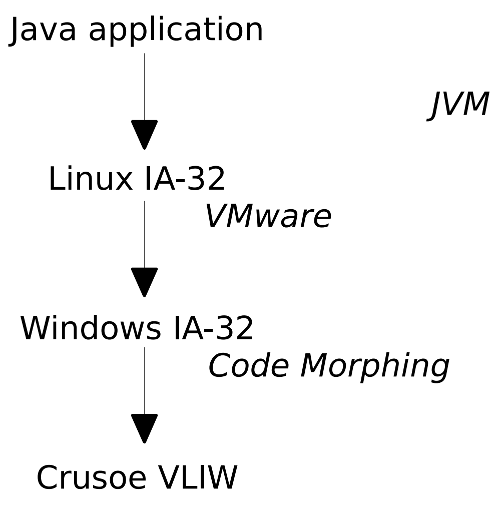


Fig 1.14 Three Levels of VMs. *A Java application running on a Java VM, running on a system VM, running on a codesigned VM.*

<<
Also, do not forget about the Linux kernel's *Containers* technology! A separate discussion is reserved for this very important and still emerging technology.
>>

Instruction Set Architecture (ISA)

“A complete ISA consists of many parts, including the register set and memory architecture, the instructions, and the trap and interrupt architecture. A virtual machine is usually concerned with all aspects of ISA emulation.”
Virtual Machines, Smith & Nair.

Instruction Set Emulation

Host, Guest ISA	*Basic Emulation Technique*	*Optimization*
Same and OS's same [eg. x86 host running Windows OS, x86 guest running Windows OS]	Pass-through	Runtime optimization ; [1]
Same but OS's differ [eg. x86 host running Linux OS, x86 guest running Windows OS]	[Dynamic] Binary Translation (system calls require detection & translation) [1]	
Different [eg. x86 host running Linux OS, ARM guest running Linux OS]	[Threaded] Interpretation [2]	Binary Translation [3], Dynamic Binary Translation [4], Predecoding

[1] : Several application areas of host:guest :: same ISA & OS:
- simulation, managing privileged instructions, program sheperding (security), runtime software optimization (!), etc.

[2] : Interpretation: “a cycle of fetching a source[2] instruction, analyzing it, performing the required operation, and then fetching the next source instruction – all in software”.

[3] Binary Translation: “attempts to amortize the fetch and analysis costs by translating a *block* of source instructions and saving (caching) the translated code for repeated use. Bigger initial translation cost but a smaller execution cost”.

[4] Dynamic Binary Translation: “The system translates one block of source code [guest] at a time. In a simple translation scheme, the natural unit for translation is the *dynamic basic block.* A dynamic basic block is slightly different from a conventional basic block, determined by the static structure of a program. A *static basic block* of instructions contains a sequence with a single entry point and a single exit point. In essence, static basic blocks begin and end at all branch/jump instructions and branch/jump targets.

Above from the excellent book Virtual Machines, Smith & Nair.

2 “Source” implies guest, “target” implies host

From the Wikipedia article on Hypervisors

A **hypervisor** or **virtual machine monitor** (**VMM**) is a piece of computer software, firmware or hardware that creates and runs virtual machines.

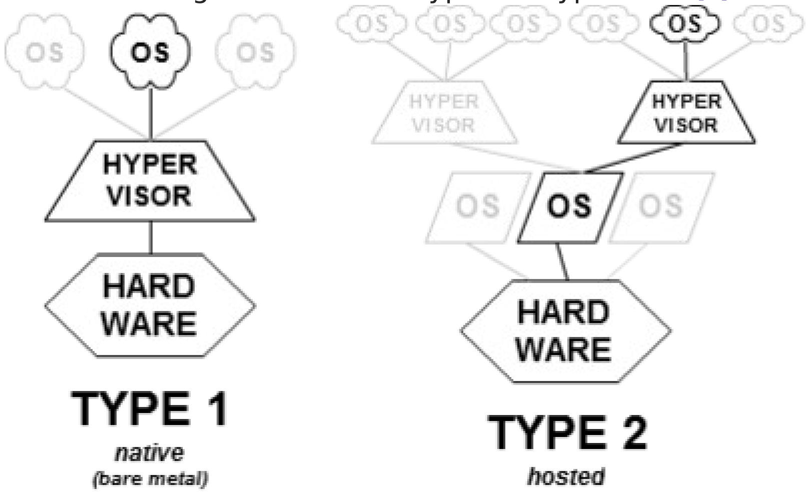
A computer on which a hypervisor is running one or more virtual machines is defined as a *host machine*. Each virtual machine is called a *guest machine*. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources.

…

Classification [edit]

Type-1 and type-2 hypervisors

In their 1974 article "Formal Requirements for Virtualizable Third Generation Architectures" Gerald J. Popek and Robert P. Goldberg classified two types of hypervisor:[1]



Type-1: native or bare-metal hypervisors

These hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems. For this reason, they are sometimes called bare metal hypervisors.

A guest operating system runs as a process on the host. The first hypervisors, which IBM developed in the 1960s, were native hypervisors. [*citation needed*] These included the test software SIMMON and the CP/CMS operating system (the predecessor of IBM's z/VM). Modern equivalents include Oracle VM Server for SPARC, Oracle VM Server for x86, the Citrix XenServer, VMware ESX/ESXi and Microsoft Hyper-V 2008/2012.

Type-2: hosted hypervisors

These hypervisors run on a conventional operating system just as other computer programs do. Type-2 hypervisors abstract guest operating systems from the host operating system. VMware Workstation, VMware Player, VirtualBox, Parallels Desktop for Mac and QEMU are examples of type-2 hypervisors.

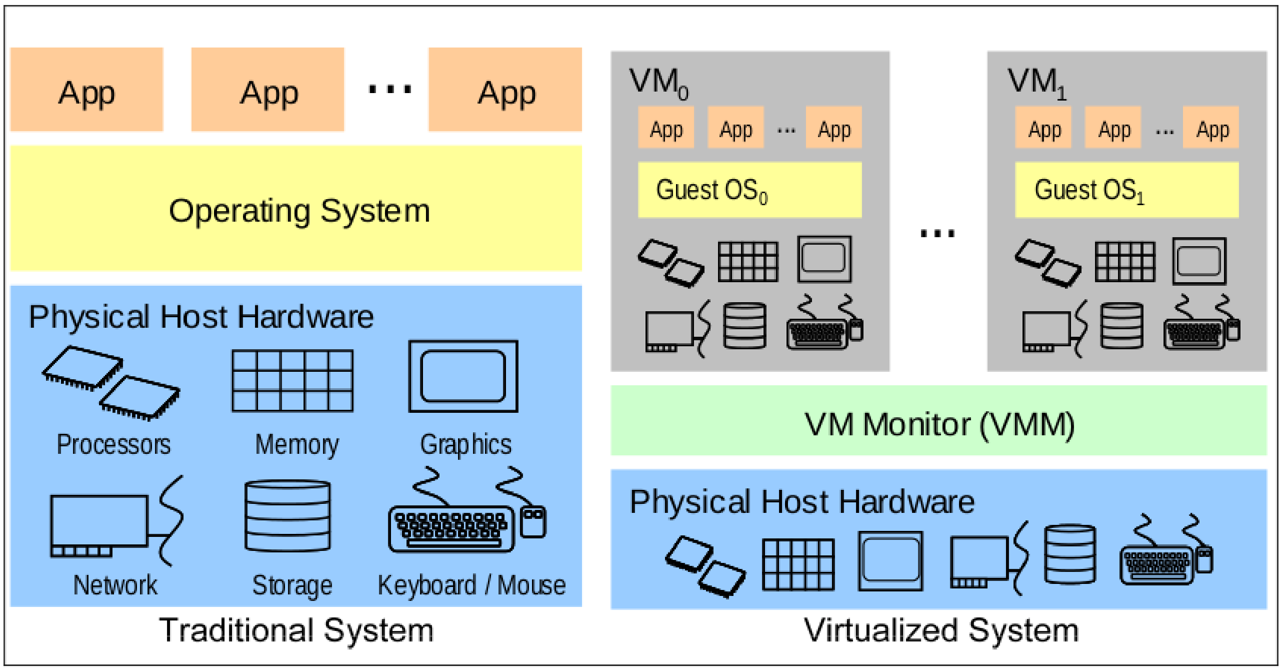
The distinction between these two types is not necessarily clear. Linux's Kernel-based Virtual Machine (KVM) and FreeBSD's bhyve are kernel modules[5] that effectively convert the host operating system to a type-1 hypervisor.[6] At the same time, since Linux distributions and FreeBSD are still general-purpose operating systems, with other applications competing for VM resources, KVM and bhyve can also be categorized as type-2 hypervisors.[7]

In 2012, a US software development company called LynuxWorks proposed a type-0 (zero) hypervisor—one with no kernel or operating system whatsoever[5][6]—which might not be entirely possible.[7]

<< Redhat has a similar (?) product: ***RedHat Enterprise Virtualization Hypervisor*** *(RHEV-H)*. RHEV-H is a subset of RHEL, including only the necessary components required to run KVM. It strips out unnecessary items such as some kernel drivers, tools, and applications that aren't relevant to KVM to reduce the overall distribution size and also the attack surface for greater security. It runs off of a read-only live CD, ensuring that the system always boots secure, unmodified code. >>
...

Intro article: https://www.ibm.com/developerworks/library/l-hypervisor/

"Hypervisors do for operating systems what operating systems roughly do for processes. ... Operating systems virtualize access to the underlying resources of the machine to processes. Hypervisors do the same thing, but instead of processes, they accomplish this task for entire guest operating systems."



Pic Source

VM Monitor (VMM) is another term for the hypervisor.

See: Comparison of platform virtualization software on Wikipedia.

Source

Kernel-based Virtual Machine (**KVM**) is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. It was merged into the Linux kernel mainline in kernel version 2.6.20, which was released on February 5, 2007.[1] KVM requires a processor with hardware virtualization extensions.[2] KVM has also been ported to FreeBSD[3] and illumos[4] in the form of loadable kernel modules.

KVM originally supported x86 processors and has been ported to S/390,[5] PowerPC,[6] and IA-64. An ARM port was merged during the 3.9 kernel merge window.[7]

A wide variety of guest operating systems work with KVM, including many flavours and versions of Linux, BSD, Solaris, Windows, Haiku, ReactOS, Plan 9, AROS Research Operating System[8] and OS X.[9] In addition, Android 2.2, GNU/Hurd[10] (Debian K16), Minix 3.1.2a, Solaris 10 U3 and Darwin 8.0.1, together with other operating systems and some newer versions of these listed, are known to work with certain limitations.[11]

Source

KVM (Kernel Virtual Machine) is a Linux kernel module that allows a user space program << *like QEMU* >> to utilize the hardware virtualization features of various processors. Today, it supports recent Intel and AMD processors (x86 and x86_64), PPC 440, PPC 970, S/390, ARM (Cortex A15, AArch64), and MIPS32 processors.

QEMU can make use of KVM when running a target architecture that is the same as the host architecture. For instance, when running *qemu-system-x86* on an x86 compatible processor, you can take advantage of the KVM acceleration - giving you benefit for your host and your guest system.

The reality is that QEMU is a standalone full virtualization software application, capable of emulating a hardware platform in it's entirety *without* the requirement of a host OS-level hypervisor.

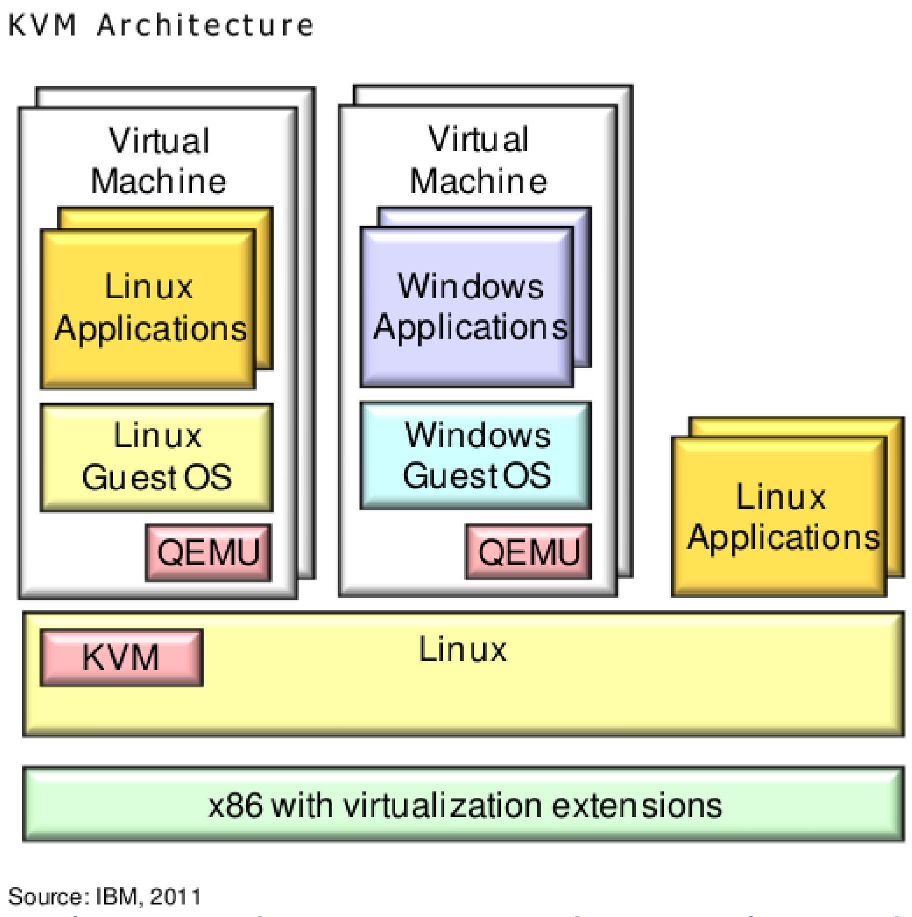
If a host hypervisor (like kvm) is present, and the guest and host are the same architecture (cpu family), then QEMU has the capability to take advantage of the hypervisor to (greatly!) accelerate the virtual machine.

(Put another way) KVM is a virtualization feature in the Linux kernel (a hypervisor), that lets a program like qemu safely execute guest code directly on the host CPU. This is only possible when the target architecture is supported by the host CPU; today that means x86-on-x86 virtualization only.

CPU Privilege Levels

After all, Qemu is a user application- it always runs in Ring 3 (unprivileged) mode on an Intel/AMD cpu.

The hypervisor kvm runs as a kernel module- hence it always executes in Ring 0 (OS privileged) mode.



Above, source: "KVM for Server Virtualization: An Open Source Solution Comes of Age". RH whitepaper

Source
[...]
How KVM Compares to Existing Hypervisors

KVM is a fairly recent project compared with its competitors. In an interview with Avi Kivity, the main developer, he compared KVM with alternative solutions:

> "In many ways, VMware is a ground-breaking technology. VMware manages to fully virtualize the notoriously complex x86 architecture using software techniques only, and to achieve very good performance and stability. As a result, VMware is a very large and complex piece of software. KVM, on the other hand, relies on the new hardware virtualization technologies that have appeared recently. As such, it is very small (about

10,000 lines) and relatively simple. Another big difference is that VMware is proprietary, while KVM is open source.

Xen is a fairly large project, providing both paravirtualization and full virtualization. It is designed as a standalone kernel, which only requires Linux to perform I/O. This makes it rather large, as it has its own scheduler, memory manager, timer handling and machine initialization.

KVM, in contrast, uses the standard Linux scheduler, memory management and other services. This allows the KVM developers to concentrate on virtualization, building on the core kernel instead of replacing it.

QEMU is a user-space emulator. It is a fairly amazing project, emulating a variety of guest processors on several host processors, with fairly decent performance. However, the user-space architecture does not allow it to approach native speeds without a kernel accelerator. KVM recognizes the utility of QEMU by using it for I/O hardware emulation. Although KVM is not tied to any particular user space, the QEMU code was too good not to use—so we used it."

[...]

KVM

Source: "Anatomy of a Linux hypervisor", by M Tim Jones, IBM DeveloperWorks

...

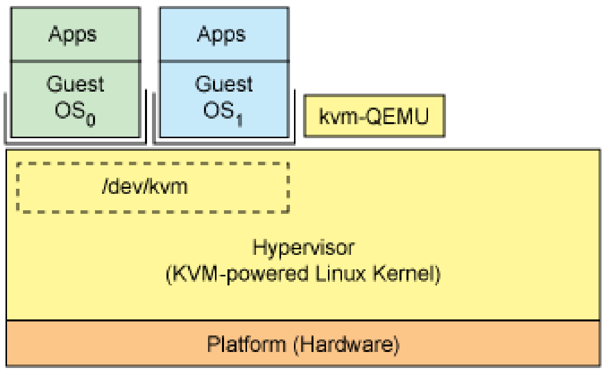
KVM is a kernel-resident virtualization infrastructure for Linux on x86 hardware. KVM was the first hypervisor to become part of the native Linux kernel (2.6.20) and was developed and is maintained by Avi Kivity through the Qumranet startup, now owned by Red Hat.

This hypervisor provides x86 (and x86-64) virtualization, with ports to the PowerPC® and IA-64 in process << *done, including ARM and S/390* >>. Additionally, KVM has recently added support for symmetrical multiprocessing (SMP) hosts (and guests) and supports enterprise-level features such as live migration (to allow guest operating systems to migrate between physical servers).

KVM is implemented as a kernel module, allowing Linux to become a hypervisor simply by loading a module. KVM provides full virtualization on hardware platforms that provide hypervisor instruction support (such as the Intel® Virtualization Technology [Intel VT] or AMD Virtualization [AMD-V] offerings). KVM also supports paravirtualized guests, including Linux and Windows®.

This technology is implemented as two components. The first is the KVM-loadable module that, when installed in the Linux kernel, provides management of the virtualization hardware, exposing its capabilities through the /proc file system (see Figure 4).

The second component provides for PC platform emulation, which is provided by a modified version of QEMU. QEMU executes as a user-space process, coordinating with the kernel for guest operating system requests.



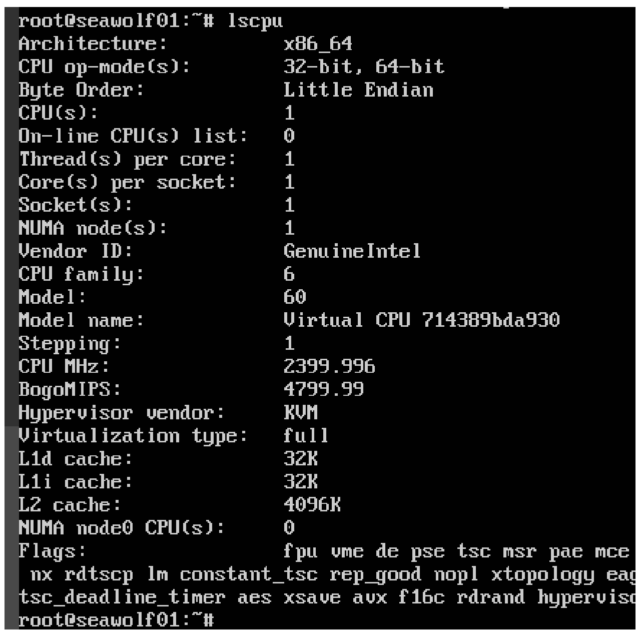
When a new operating system is booted on KVM (through a utility called `kvm`), it becomes a process of the host operating system and therefore scheduleable like any other process.

<<

Note 1:

An example of cloud providers using Linux/KVM: here's a screenshot of Ubuntu 16.04 LTS (storage instance) running off the Vultr.com cloud provider:

(Notice how the 'lscpu' command shows us the *Hypervisor vendor* and *Virtualization type*.)



```
root@seawolf01:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Model name:            Virtual CPU 714389bda930
Stepping:              1
CPU MHz:               2399.996
BogoMIPS:              4799.99
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
NUMA node0 CPU(s):     0
Flags:                 fpu vme de pse tsc msr pae mce
 nx rdtscp lm constant_tsc rep_good nopl xtopology eag
tsc_deadline_timer aes xsave avx f16c rdrand hypervis
root@seawolf01:~#
```

<<
FYI:
- *Amazon EC2* is powered by (custom) Xen virtualization technology

UPDATE! 07 Nov 2017
AWS adopts home-brewed KVM as new hypervisor

"…
AWS has revealed it has created a new hypervisor based on KVM, not the Xen hypervisor on which it has relied for years.

The new hypervisor was unveiled as a virtual footnote in news of new EC2 instance type called the "C5" powered by Intel's Skylake Xeons. AWS's FAQ about the new instances notes "C5 instances use a new EC2 hypervisor that is based on core KVM technology."
…
More interesting still is AWS's news that "going forward, we'll use this hypervisor to power other instance types."
…
It is built on core Linux Kernel-based Virtual Machine (KVM) technology, but does not include general purpose operating system components. ..."

- *Digital Ocean* (the third largest cloud provider) is powered by KVM virtualization tech
>>

Note 2: How does one know if we're running in a VM?

See this SO Q&A: "Find out if the OS is running in a virtual environment".

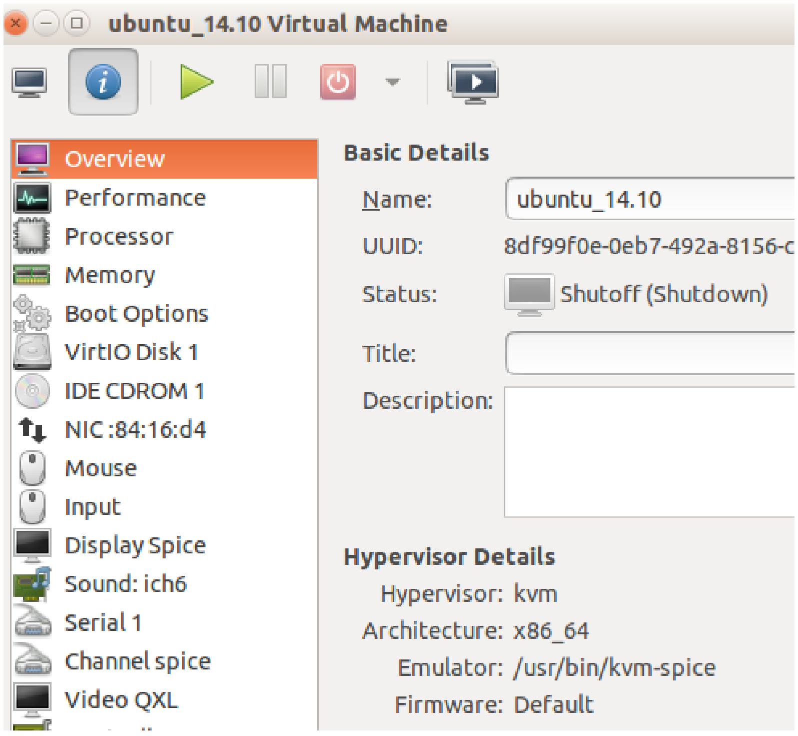
In particular, install and read the code of this bash script which seems to work pretty well:
virt-what

```
sudo apt install virt-what
```

Note 3: Interestingly, the 'kvm' utility on Ubuntu Linux (currently running on Ubuntu 14.10), is a wrapper over the KVM-modified Qemu, enabling KVM kernel-level support:

```
$ file `which kvm`
/usr/bin/kvm: POSIX shell script, ASCII text executable
$
$ cat `which kvm`
#! /bin/sh
exec qemu-system-x86_64 -enable-kvm "$@"
$ qemu-system-x86_64 --help |grep enable-kvm
-enable-kvm      enable KVM full virtualization support
$
```

Update:
Using *virt-manager*, when one clicks on the Info / “Overview” button, the following shows up:



Basic Details
Name: ubuntu_14.10
UUID: 8df99f0e-0eb7-492a-8156-c
Status: Shutoff (Shutdown)
Title:
Description:

Hypervisor Details
Hypervisor: kvm
Architecture: x86_64
Emulator: /usr/bin/kvm-spice
Firmware: Default

Look at the bottom-right of the above screenshot: the hypervisor is kvm. But:
“Emulator: /usr/bin/kvm-spice” ??
Lets find out:

```
$ which kvm-spice
/usr/bin/kvm-spice
$ file `which kvm-spice `
/usr/bin/kvm-spice: symbolic link to `kvm'
$ ls -l `which kvm-spice `
lrwxrwxrwx 1 root root 3 Jul 28 00:57 /usr/bin/kvm-spice -> kvm*
$
```

Okay, so it's just (a soft link to) “kvm”, which as we just saw above, is nothing but KVM's Qemu with the “-enable-kvm” switch.
>>

<<
Experiment:

The instructor will demonstrate the *enormous difference in performance* when running a guest with KVM (hypervisor as a loadable kernel module) support and without.

(x86_kvm_deb project:run_simple.sh)

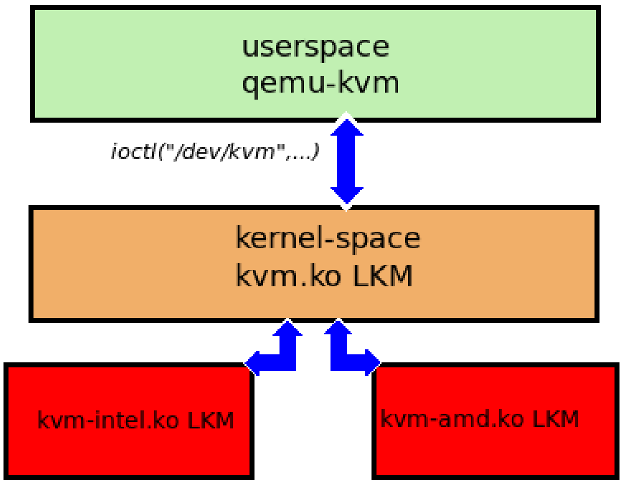
The tremendous difference in performance when running Qemu (x86_64 guest (sample) Debian 7

Linux) on an x86_64 Linux host with kvm running is primarily due to:

- the hypervisor kvm being able to take advantage of:
  - host and guest using the same CPU arch (typically x86_64), and thus,
  - hardware virtualization support
    - Intel (or AMD) processor hardware VirTualization eXtensions (VT-x/EPT/NPT)
    - kvm's ability to execute Qemu guest code directly on the host cpu (in Guest/Non-Root or "-1" privilege level / mode).

  These details are covered later...

>>



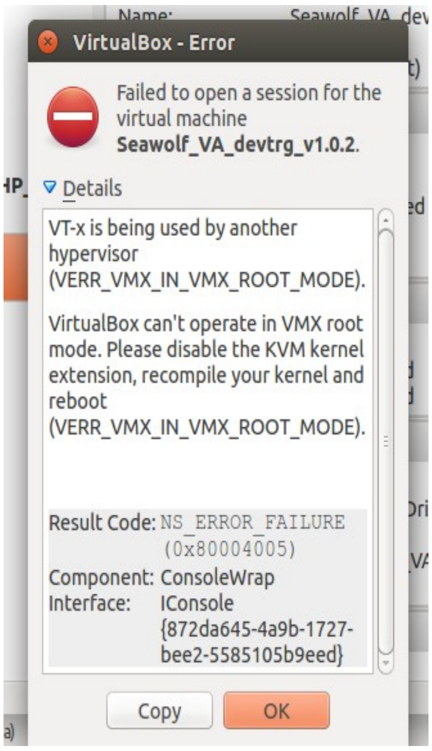
But unlike traditional processes in Linux, the guest operating system is identified by the hypervisor as being in the "guest" (or "-1" privilege level) mode (independent of the kernel and user modes).

<<
Note!

Only ONE hypervisor can make use of the processor hardware virtualization extensions (VT-x on Intel) at a time.

See the error thrown up by VirtualBox (*right*) when attempting to launch a guest when another QEMU guest is already executing via the hypervisor (and thus making use of VT-x on a system that has the feature enabled):
"VT-x is being used by another hypervisor ..."

>>


VirtualBox - Error

Failed to open a session for the virtual machine **Seawolf_VA_devtrg_v1.0.2.**

Details

VT-x is being used by another hypervisor (VERR_VMX_IN_VMX_ROOT_MODE).

VirtualBox can't operate in VMX root mode. Please disable the KVM kernel extension, recompile your kernel and reboot (VERR_VMX_IN_VMX_ROOT_MODE).

Result Code: NS_ERROR_FAILURE (0x80004005)
Component: ConsoleWrap
Interface: IConsole {872da645-4a9b-1727-bee2-5585105b9eed}

Each guest operating system is mapped through the `/dev/kvm` device, having its own virtual address space that is mapped into the host kernel's physical address space. As previously mentioned, KVM uses the underlying hardware's virtualization support to provide full (native) virtualization. I/O requests are mapped through the host kernel to the QEMU process that executes on the host.

KVM operates in the context of Linux as the host but supports a large number of guest operating systems, given underlying hardware virtualization support. You can find a list of the supported guests in Resources. << KVM Guest Support Status >>

<< For high performance virtualization though, running a paravirtualized Linux guest on a (KVM) Linux host is best. >>

... KVM makes use of QEMU for platform virtualization, and with Linux as the hypervisor, it immediately supported the idea of guest operating systems executing in concert with other Linux applications.

KVM is a virtualization feature in the Linux kernel that lets a program like qemu safely execute guest code directly on the host CPU. This is only possible when the target architecture is supported by the host CPU; today that means x86-on-x86 virtualization only.

Always x86 and x86_64 only?
No. "KVM originally supported x86 processors and has been ported to S/390,[4] PowerPC,[5] and IA-64. An ARM port was merged during the 3.9 kernel merge window.[6]".
But x86-on-x86 is highly optimized by having guest code *directly execute* on the host cpu.

Design

By itself, KVM does not perform any emulation. Instead, it simply exposes the */dev/kvm* interface, with which a user-space host can then:

- Set up the guest VM's address space. The host must also supply a firmware image (usually a custom BIOS when emulating PCs) with which the guest can bootstrap into its main OS.
- Feed it simulated I/O.
- Map its video display back onto the host.

On Linux, QEMU versions 0.10.1 and later is one such host. It will use KVM when available to virtualize guests at near-native speeds, but otherwise fall back to software-only emulation.

<<
An interesting blog post that categorises and shows the “palyers” in the VM/Container markerspace as of Aug 2016:
Private Cloud Bakeoff – Top Platforms for Virtualization Playgrounds, Packet

>>

Paravirtualization and Virtio

Source: Virtio: An I/O virtualization framework for Linux : Paravirtualized I/O with KVM and lguest, by M Tim Jones, IBM DeveloperWorks.

...

Full virtualization vs. Paravirtualization

Let's start with a quick discussion of two distinct types of virtualization schemes: full virtualization and paravirtualization. In *full virtualization*, the guest operating system runs on top of a hypervisor that sits on the bare metal (type 1; or more commonly, type 2: on a host OS which sits on bare metal). The guest is unaware that it is being virtualized and requires no changes to work in this configuration.

Conversely, in *paravirtualization*, the guest operating system is not only aware that it is running on a hypervisor but includes code to make guest-to-hypervisor transitions more efficient (see Figure 1).

In the full virtualization scheme, the hypervisor must emulate device hardware, which is emulating at the lowest level of the conversation (for example, to a network driver). Although the emulation is clean at this abstraction, it's also the most inefficient and highly complicated.

In the paravirtualization scheme, the guest and the hypervisor can work cooperatively to make this emulation efficient. The downside to the paravirtualization approach is that the operating system is aware that it's being virtualized and requires modifications to work.

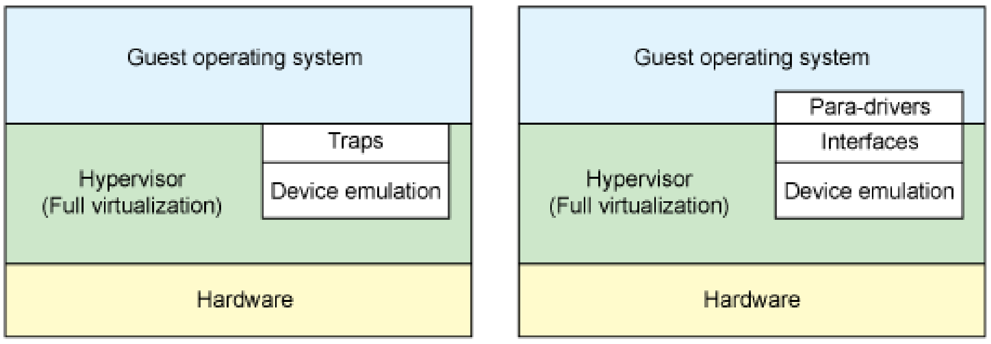


Figure 1. Device emulation in full virtualization and paravirtualization environments

<<

FYI, within the Linux kernel menuconfig (x86_64) : configurables relevant to paravirtualization:

```
config PARAVIRT
      bool "Enable paravirtualization code"
      ---help---
        This changes the kernel so it can modify itself when it is run
        under a hypervisor, potentially improving performance 
significantly
        over full virtualization.  However, when run without a hypervisor
        the kernel is theoretically slower and slightly larger.
```

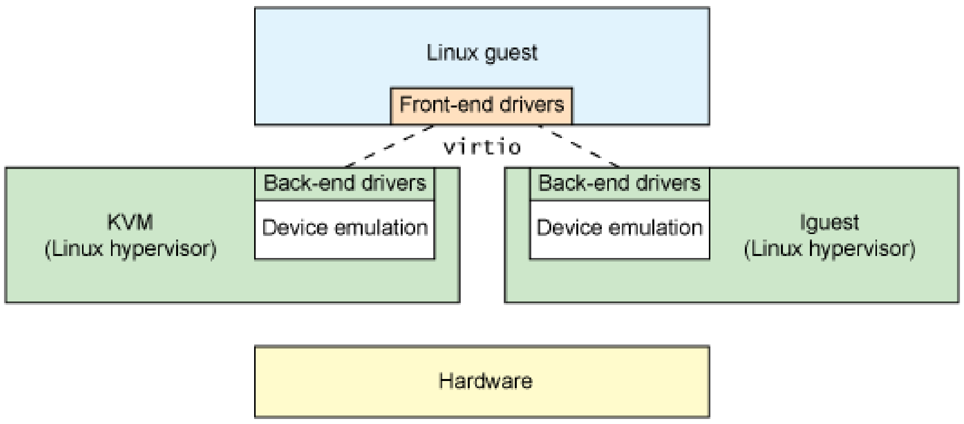
```
config PARAVIRT_DEBUG
      bool "paravirt-ops debugging"
      depends on PARAVIRT && DEBUG_KERNEL
      ---help---
        Enable to debug paravirt_ops internals.  Specifically, BUG if
        a paravirt_op is missing when it is called.

config PARAVIRT_SPINLOCKS
      bool "Paravirtualization layer for spinlocks"
      depends on PARAVIRT && SMP
      select UNINLINE_SPIN_UNLOCK
      ---help---
        Paravirtualized spinlocks allow a pvops backend to replace the
        spinlock implementation with something virtualization-friendly
        (for example, block the virtual CPU rather than spinning).

        It has a minimal impact on native kernels and gives a nice
performance
        benefit on paravirtualized KVM / Xen kernels.

        If you are unsure how to answer this question, answer Y.

source "arch/x86/xen/Kconfig"

config KVM_GUEST
      bool "KVM Guest support (including kvmclock)"
      depends on PARAVIRT
      select PARAVIRT_CLOCK
      default y
      ---help---
        This option enables various optimizations for running under the
KVM
        hypervisor. It includes a paravirtualized clock, so that instead
        of relying on a PIT (or probably other) emulation by the
        underlying device model, the host provides the guest with
        timing infrastructure such as time of day, and system time

>>
```

...

(In the paravirtualization scheme) the guest operating system is aware that it's running on a hypervisor and includes drivers that act as the front end. The hypervisor << for eg, KVM >> implements the back-end drivers for the particular device emulation. These front-end and back-end drivers are where `virtio` comes in, providing a standardized interface for the development of emulated device access to propagate code reuse and increase efficiency.

From the previous section, you can see that `virtio` is an abstraction for a set of common emulated devices in a paravirtualized hypervisor. This design allows the hypervisor to export a common set of emulated devices and make them available through a common application programming interface

(API). Figure 2 (below) illustrates why this is important. With paravirtualized hypervisors, the guests implement a common set of interfaces, with the particular device emulation behind a set of back-end drivers. The back-end drivers need not be common as long as they implement the required behaviors of the front end.



Note that in reality (though not required), the device emulation occurs in user space using QEMU, so the back-end drivers communicate into the user space of the hypervisor to facilitate I/O through QEMU. QEMU is a system emulator that, in addition to providing a guest operating system virtualization platform, provides emulation of an entire system (PCI host controller, disk, network, video hardware, USB controller, and other hardware elements).

The `virtio` API relies on a simple buffer abstraction to encapsulate the command and data needs of the guest.

... << *Continue to read the article online for more details on virtio internal architecture...* >> ...

<<
SIDEBAR:

Virtio alternatives

`virtio` is not entirely alone in this space; modern hypervisors will provide paravirt drivers though they terminology might differ:

Xen : provides paravirtualized device drivers
VMware : provides what are called Guest Tools
VirtualBox : provides what are called Guest Additions
>>

<<
More on VirtualBox's VirtIO Support

Source

...
As briefly mentioned in Section 3.8, “Network settings”, VirtualBox provides up to eight virtual PCI Ethernet cards for each virtual machine. For each such card, you can individually select

1. the hardware that will be virtualized as well as
2. the virtualization mode that the virtual card will be operating in with respect to your physical networking hardware on the host.

Four of the network cards can be configured in the "Network" section of the settings dialog in the graphical user interface of VirtualBox. You can configure all eight network cards on the command line via VBoxManage modifyvm; see Section 8.8, “VBoxManage modifyvm”.

This chapter explains the various networking settings in more detail.

6.1. Virtual networking hardware

For each card, you can individually select what kind of hardware will be presented to the virtual machine. VirtualBox can virtualize the following six types of networking hardware:

- AMD PCNet PCI II (Am79C970A);
- AMD PCNet FAST III (Am79C973, the default);
- Intel PRO/1000 MT Desktop (82540EM);
- Intel PRO/1000 T Server (82543GC);
- Intel PRO/1000 MT Server (82545EM);
- Paravirtualized network adapter (virtio-net).

The PCNet FAST III is the default because it is supported by nearly all operating systems out of the box, as well as the GNU GRUB boot manager.

[...]

The "Paravirtualized network adapter (virtio-net)" is special. If you select this, then VirtualBox does not virtualize common networking hardware (that is supported by common guest operating systems out of the box). Instead, VirtualBox then expects a special software interface for virtualized environments to be provided by the guest, thus avoiding the complexity of emulating networking hardware and improving network performance. Starting with version 3.1, VirtualBox provides support for the industry-standard "virtio" networking drivers, which are part of the open-source KVM project.

The "virtio" networking drivers are available for the following guest operating systems:

- Linux kernels version 2.6.25 or later can be configured to provide virtio support; some distributions also back-ported virtio to older kernels.
- For Windows 2000, XP and Vista, virtio drivers can be downloaded and installed from the KVM project web page.[30]

...

>>

FYI:

“The Windows guests currently have no VirtIO

Don't lose more time with tweaking anything.
Install the virtIO drivers and come back. The difference is so huge that any enhancement you can find now will have no meaning with virtIO.
Just an example with one of our servers:
- without virtIO a W2k3 can handle about 10 Terminal Server users
- with virtIO, the same machine with the same OS currently handle 120 to 125 users with little slow down. And we added another virtual machine to run SQL Server one the same physical computer ...

Shareeditflag answered Jul 2 '12 at 23:08 Gregory MOUSSAT
”

KVM guest io is much slower than host io: is that normal?
...
You're not done with performance tuning yet.

```
  <driver name='qemu' type='raw' cache='writethrough' io='native'/>
```

First is which I/O mechanism to use.

> QEMU has two asynchronous I/O mechanisms: POSIX AIO emulation using a pool of worker threads and native Linux AIO.

Set either `io='native'` or `io='threads'` in your XML to benchmark each of these.
Second is which caching mechanism to use. You can set `cache='writeback'`, `cache='writethrough'` or you can turn it off with `cache='none'`, which you actually may find works best.

> If you're using raw volumes or partitions, it is best to avoid the cache completely, which reduces data copies and bus traffic.

...
... Virtio rules! ;)
...
io='native' gave almost 20-30% extra WRITE performance in my case – rahul286 Jul 10 '13 at 13:39

Also see:

An API for virtual I/O: virtio [lwn]

Excellent: “Best Practices for KVM”, IBM whitepaper. [PDF]
Relevant section: “Best practice: Para-virtualize devices by using the VirtIO API” pg 1-5.

[2.6.34] **Vhost net: fast KVM networking**

vhost net is a kernel-level backend for virtio networking. The main motivation for vhost is to reduce virtualization overhead for virtio-net by moving the task of converting virtio descriptors to SKBs and back from qemu userspace to the vhost net driver. For virtio-net this means removing up to 4 system calls per packet: vm exit for kick, reentry for kick, iothread wakeup for packet, interrupt injection for packet. This was shown to reduce latency by a factor of 5, and improve bandwidth to almost-native performance. Existing virtio net code is used in guests without modification.
Project web site: http://www.linux-kvm.org/page/VhostNet
Code: (commit)

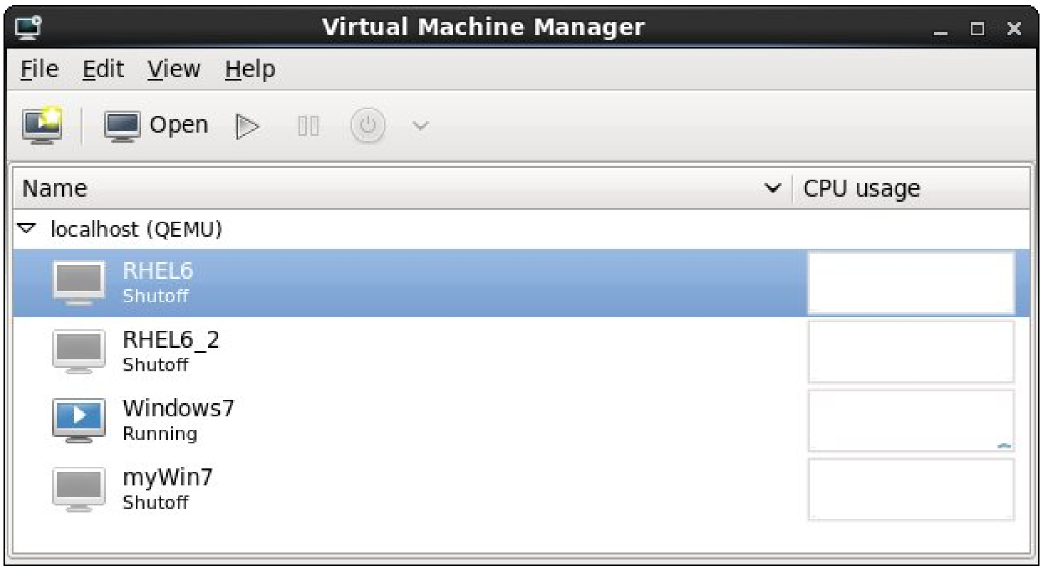
Miscellaneous

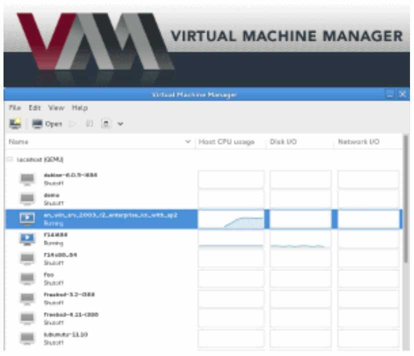
libvirt
libvirt is an open source API, daemon and management tool for managing platform virtualization.[1] It can be used to manage Linux KVM, Xen, VMware ESX, QEMU and other virtualization technologies. These APIs are widely used in the orchestration layer of hypervisors in the development of a cloud-based solution.

[*sudo apt-get install libvirt-bin libvirt-doc*]

VMM - Virtual Machine Manager (or virt-manager)
GUI VM manager app: virt-manager (python-based)
http://virt-manager.org/

The virt-manager application is a desktop user interface for managing virtual machines through libvirt. It primarily targets KVM VMs, but also manages Xen and LXC (linux containers). It presents a summary view of running domains, their live performance & resource utilization statistics. Wizards enable the creation of new domains, and configuration & adjustment of a domain’s resource allocation & virtual hardware. An embedded VNC and SPICE client viewer presents a full graphical console to the guest domain.





[sudo apt-get install virt-manager]

Note- terminology: often the VMM is thought of as equivalent to hypervisor ('Virtual Machine Monitor' and not the 'manager').

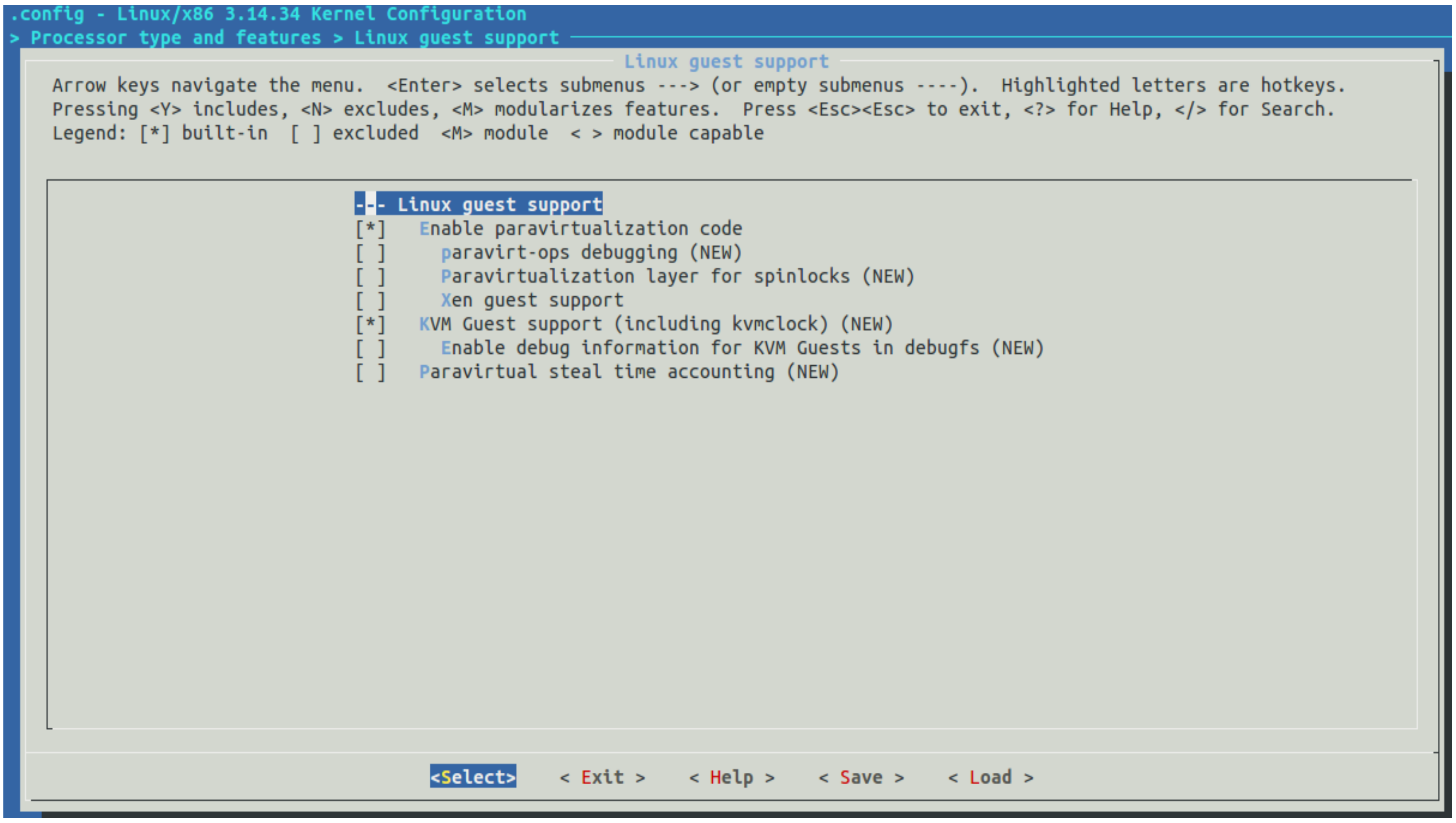
Additional Resource
https://www.ibm.com/developerworks/community/blogs/ibmvirtualization/entry/how_secure_is_your_kvm_environment

---

Kernel Configuration for KVM

On a vanilla Linux kernel ver 3.14.34:

```
$ make menuconfig
...

<< under Processor type and features / Linux guest support >>
```


```
.config - Linux/x86 3.14.34 Kernel Configuration
> Processor type and features > Linux guest support ─────────────────────────
                              Linux guest support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.
  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                                    --- Linux guest support
                                    [*]   Enable paravirtualization code
                                    [ ]     paravirt-ops debugging (NEW)
                                    [ ]     Paravirtualization layer for spinlocks (NEW)
                                    [ ]     Xen guest support
                                    [*]   KVM Guest support (including kvmclock) (NEW)
                                    [ ]     Enable debug information for KVM Guests in debugfs (NEW)
                                    [ ]   Paravirtual steal time accounting (NEW)

                          <Select>    < Exit >    < Help >    < Save >    < Load >
```

What the configuration variable name (CONFIG_XXX) is and what the options mean:

```
arch/x86/Kconfig
...
menuconfig HYPERVISOR_GUEST          << becomes CONFIG_HYPERVISOR_GUEST
>>
	bool "Linux guest support"
	---help---
	  Say Y here to enable options for running Linux under various
hyper-
	  visors. This option enables basic hypervisor detection and
platform
	  setup.

	  If you say N, all options in this submenu will be skipped and
	  disabled, and Linux guest support won't be built in.

if HYPERVISOR_GUEST

config PARAVIRT
	bool "Enable paravirtualization code"
	---help---
	  This changes the kernel so it can modify itself when it is run
	  under a hypervisor, potentially improving performance
significantly
	  over full virtualization.  However, when run without a hypervisor
	  the kernel is theoretically slower and slightly larger.

config PARAVIRT_DEBUG
	bool "paravirt-ops debugging"
	depends on PARAVIRT && DEBUG_KERNEL
	---help---
	  Enable to debug paravirt_ops internals.  Specifically, BUG if
	  a paravirt_op is missing when it is called.

config PARAVIRT_SPINLOCKS
	bool "Paravirtualization layer for spinlocks"
	depends on PARAVIRT && SMP
	select UNINLINE_SPIN_UNLOCK
	---help---
	  Paravirtualized spinlocks allow a pvops backend to replace the
	  spinlock implementation with something virtualization-friendly
	  (for example, block the virtual CPU rather than spinning).

	  It has a minimal impact on native kernels and gives a nice
performance
	  benefit on paravirtualized KVM / Xen kernels.

	  If you are unsure how to answer this question, answer Y.

source "arch/x86/xen/Kconfig"

config KVM_GUEST
	bool "KVM Guest support (including kvmclock)"
	depends on PARAVIRT
	select PARAVIRT_CLOCK
	default y
	---help---
	  This option enables various optimizations for running under the
KVM
	  hypervisor. It includes a paravirtualized clock, so that instead
	  of relying on a PIT (or probably other) emulation by the
	  underlying device model, the host provides the guest with
	  timing infrastructure such as time of day, and system time

config KVM_DEBUG_FS
	bool "Enable debug information for KVM Guests in debugfs"
	depends on KVM_GUEST && DEBUG_FS
	default n
	---help---
	  This option enables collection of various statistics for KVM
guest.
	  Statistics are displayed in debugfs filesystem. Enabling this
option
	  may incur significant overhead.

source "arch/x86/lguest/Kconfig"

config PARAVIRT_TIME_ACCOUNTING
	bool "Paravirtual steal time accounting"
```

```
     depends on PARAVIRT
     default n
     ---help---
       Select this option to enable fine granularity task steal time
       accounting. Time spent executing other tasks in parallel with
       the current vCPU is discounted from the vCPU power. To account
for
       that, there can be a small performance impact.

       If in doubt, say N here.

config PARAVIRT_CLOCK
     bool

endif #HYPERVISOR_GUEST
...
```

Appendix A :: Resources

Very good (introductory technical) series on Virtualization :
"**Hardware Virtualization: the Nuts and Bolts**"

Intro & terminology to VM hypervisor
http://en.wikipedia.org/wiki/Hypervisor
http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine

Simple guest VM setup
http://www.linux-kvm.org/page/HOWTO1

"Best Practices for KVM", IBM whitepaper

Memory management for virtualization

https://www.kernel.org/doc/Documentation/virtual/kvm/mmu.txt

Internals overview
http://blog.vmsplice.net/2011/03/qemu-internals-big-picture-overview.html

http://www.linux-kvm.org/page/Documents
http://www.linux-kvm.org/page/Tuning_Kernel

Memory management notifiers

http://stackoverflow.com/questions/9832140/what-exactly-do-shadow-page-tables-for-vmms-do

OASIS:
"OASIS is a non-profit, international consortium that drives the development, convergence and adoption of open standards for the global information society. OASIS promotes industry consensus and produces worldwide standards for cloud computing, M2M, IoT, security, privacy, content technologies, energy, emergency management, and other areas. OASIS open standards offer the potential to lower cost, stimulate innovation, grow global markets, and protect the right of free choice of technology. OASIS members broadly represent the marketplace of public and private sector technology leaders, users, and influencers. The consortium has more than 5,000 participants representing over 600 organizations and individual members in 65 countries."

OASIS Members to Develop VIRTIO Interoperability Standard for Virtualization

Nested vmx support coming to kvm, Avi Kivity

Book:
Virtual Machines: Versatile Platforms for Systems and Processes by Smith & Nair.

Misc / Useful:
QEMU how to ping host network?

Appendix B :: Ballooning

Understanding Memory Resource Management in Vmware Vsphere 5.0

Ballooning is the process of being able to dynamically increase or decrease the amount of memory seen as available to a guest OS. It is a technique whereby the guest is aware of the memory situation of it's host.

"Best Practices for KVM", IBM whitepaper. - see section on Ballooning

Source: http://www.quora.com/Virtualization/What-is-memory-ballooning
Answered below by Michael Bogobowicz.

...
How it works

Since the guest virtual machines need to know how much memory they have on boot, the system has to be tricked. An agent resides on the guest OS that communicates directly with the hypervisor. If more memory is needed for other guests, and this guest has additional memory, the agent reserves and locks down part of the memory away from the regular processes, allowing it to be freed up for others.

If the guest VM starts running low on memory, the agent will talk to the hypervisor, and once accepted, it will release some of the space to the guest VM for other processes.

Pros
You can more efficiently use memory and other resources by overcommitting slightly, and having more virtual servers per physical server. This decreases energy, hardware and rack space costs.

Cons
You run the risk of a major performance meltdown. If multiple servers start using more memory at the same time, and more virtual memory is needed than physical, the virtual servers will not be able to get it. As a result, CPU and disk usage will spike, as Windows tries to free up space in other places and page items in memory to disk.

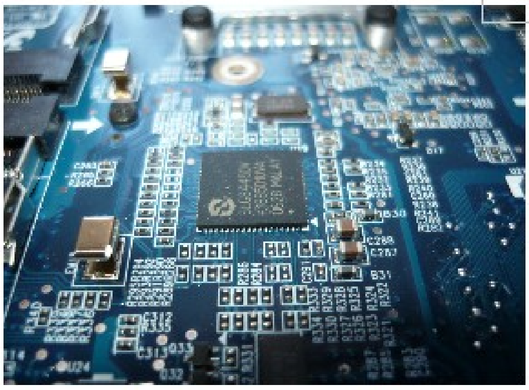
This could then cause all virtual machines on this physical server to start responding poorly, and possibly even affect machines on other virtual servers attached to the same central storage.

Overall, as long as you understand the risk, understand your workloads (and know that most do not regularly use all of their memory), and don't overcommit too greatly, memory ballooning is a great feature that can improve your ROI on virtualization and hardware.

Virtual memory ballooning.

Linux Operating System Specialized

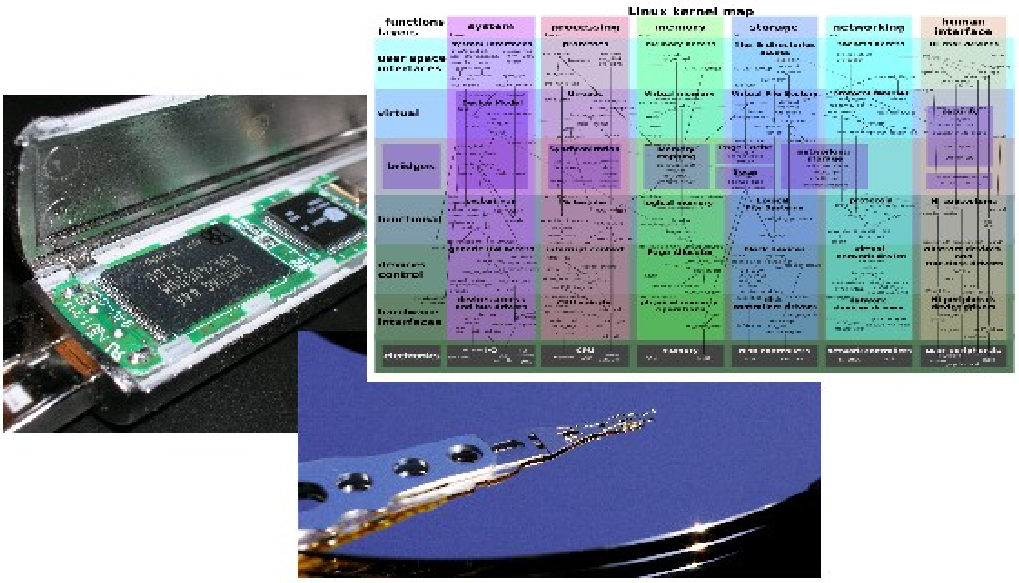
kaiwanTECH



The highest quality Training on:

Linux Fundamentals, CLI and Scripting
Linux Systems Programming
Linux Kernel Internals
Linux Device Drivers
Embedded Linux
Linux Debugging Techniques
New! *Linux OS for Technical Managers*

Please do visit our website for details:
http://kaiwantech.in



http://kaiwantech.in

kaiwanTECH Linux OS Corporate Training Programs
Please do check out our current offering of world-class, seriously-valuable, high on returns, technical Linux OS corporate training programs here: http://bit.ly/ktcorp