

Virtual Linux

An overview of virtualization methods, architectures, and implementations

Level: Intermediate

M. Tim Jones (mtj@mtjones.com), Consultant Engineer, Emulex

29 Dec 2006

Virtualization means many things to many people. A big focus of virtualization currently is *server virtualization*, or the hosting of multiple independent operating systems on a single host computer. This article explores the ideas behind virtualization and then discusses some of the many ways to implement virtualization. We also look at some of the other virtualization technologies out there, such as operating system virtualization on Linux.

To *virtualize* means to take something of one form and make it appear to be another form. Virtualizing a computer means to make it appear to be multiple computers or a different computer entirely.

Virtualization also can mean making many computers appear to be a single computer. This is more commonly called server aggregation or *grid computing*.

Let's begin with the origins of virtualization.

A historical view of virtualization

Virtualization is not a new topic; in fact, it's over four decades old. The earliest uses of virtualization include the IBM® 7044, the Compatible Time Sharing System (CTSS) developed by the Massachusetts Institute of Technology (MIT) on the IBM 704, and Manchester University's Atlas project (one of the world's first supercomputers), which pioneered demand paging and supervisor calls.

Hardware virtualization

IBM recognized the importance of virtualization in the 1960s with the development of the System/360™ Model 67 mainframe. The Model 67 virtualized all of the hardware interfaces through the Virtual Machine Monitor, or VMM. In the early days of computing, the operating system was called the *supervisor*. With the ability to run operating systems on other operating systems, the term *hypervisor* resulted (a term coined in the 1970s).

The VMM ran directly on the underlying hardware, permitting multiple virtual machines (VMs). Each VM could then run an instance of its own private operating system -- in the early days this was the CMS, or Conversational Monitor System. The VM continued to be advanced, and today you can find it running on the System z9™ mainframe. This provides backwards compatibility, even to the System/360 line.

Processor virtualization

Another early use of virtualization, in this case of a simulated processor, is the P-code (or pseudo-code) machine. P-code is a machine language that is executed in a virtual machine rather than in actual hardware. P-code was made famous in the early 1970s by the University of California, San Diego (UCSD) Pascal system, which compiled Pascal programs into P-code, and then executed them on a P-code virtual machine. This allowed P-code programs to be highly portable and run anywhere a P-code virtual machine was available.

The same concept was used in the 1960s for Basic Combined Programming Language (BCPL), an ancestor of the C language. In this usage, a compiler would compile BCPL code into an intermediate machine code called O-code. As a secondary step, the O-code was

Java Virtual Machine (JVM)

The Java™ language followed the P-code model for its virtual machine. This permitted the wide distribution of Java

compiled into the native language of the target machine. This model is used by modern compilers to provide flexibility in porting compilers to new target architectures (separating the front-end and the back-end by an intermediate language).

programs over countless architectures by simply porting the JVM.

Instruction set virtualization

A more recent aspect of virtualization is called instruction set virtualization, or binary translation. In this model, a virtual instruction set is translated to the physical instruction set of the underlying hardware, most commonly dynamically. As code is to be executed, a translation occurs for a segment of code. If a branch occurs, that new set of code is brought in and translated. This makes it very similar to caching operations, where blocks of instructions are moved from memory into a local fast cache memory for execution.

A recent example of this model was used in the Crusoe central processing unit (CPU) designed by Transmeta. This architecture implemented binary translation under the trademarked name Code Morphing. A similar example is runtime code scanning used by full virtualization solutions to find and redirect privileged instructions (to work around issues in certain processor instruction sets).

Types of virtualization

When it comes to virtualization, there's not just one way to do it. In fact, there are several ways that achieve the same result through different levels of abstraction. This section introduces you to three of the most common methods of virtualization in Linux and identifies their relative strengths and weaknesses. The industry sometimes uses different terms to describe the same virtualization method. The most common term is used here, with references to the other terms for consistency.

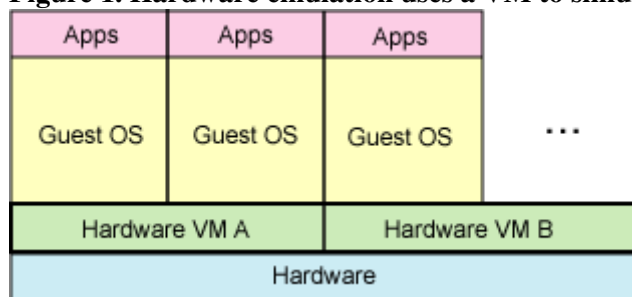
Hardware emulation

Arguably the most complex of the virtualizations is provided by hardware emulation. In this method, a hardware VM is created on a host system to emulate the hardware of interest, as shown in Figure 1.

Virtualization and games

An article on virtualization wouldn't be complete without at least a mention of the Multiple-Arcade Machine Emulator (MAME). MAME, as its name implies, is a full machine emulator of many arcade games of the past. In addition to virtualizing the processors used in those games, the entire machine is virtualized, including sound and graphics hardware and controls. MAME is a great application, but it's also interesting to peruse the source to understand the scope of what they've accomplished.

Figure 1. Hardware emulation uses a VM to simulate the required hardware



As you can probably guess, the main problem with hardware emulation is that it can be excruciatingly slow. Because every instruction must be simulated on the underlying hardware, a 100 times slowdown is not uncommon. For high-fidelity emulations that include cycle accuracy, simulated CPU pipelines, and caching behaviors, the actual speed difference can be on the order of 1000 times slower.

Hardware emulation does have its advantages. For example, using hardware emulation, you can run an unmodified operating system intended for a PowerPC® on an ARM processor host. You can even run multiple virtual machines, each simulating a different processor.

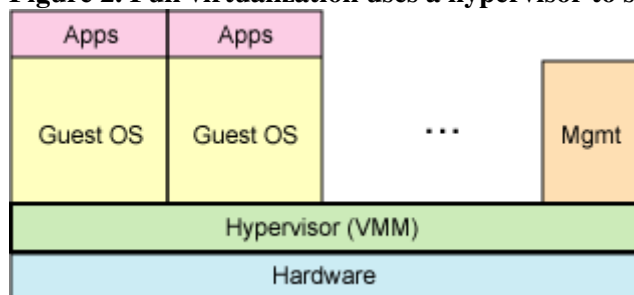
Emulation and development

One of the most interesting uses of hardware emulation is in co-development of firmware and hardware. Rather than wait until the real hardware is available, firmware developers can use a target hardware VM to validate many aspects of their actual code in simulation.

Full virtualization

Full virtualization, otherwise known as native virtualization, is another interesting method of virtualization. This model uses a virtual machine that mediates between the guest operating systems and the native hardware (see Figure 2). "Mediate" is the key word here because the VMM mediates between the guest operating systems and the bare hardware. Certain protected instructions must be trapped and handled within the hypervisor because the underlying hardware isn't owned by an operating system but is instead shared by it through the hypervisor.

Figure 2. Full virtualization uses a hypervisor to share the underlying hardware



Full virtualization is faster than hardware emulation, but performance is less than bare hardware because of the hypervisor mediation. The biggest advantage of full virtualization is that an operating system can run unmodified. The only constraint is that the operating system must support the underlying hardware (for example, PowerPC).

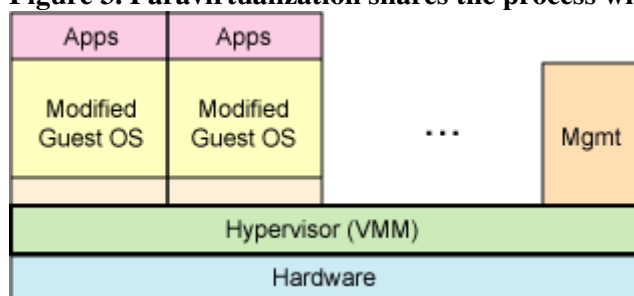
Paravirtualization

Paravirtualization is another popular technique that has some similarities to full virtualization. This method uses a hypervisor for shared access to the underlying hardware but integrates virtualization-aware code into the operating system itself (see Figure 3). This approach obviates the need for any recompilation or trapping because the operating systems themselves cooperate in the virtualization process.

Hypervisors on older hardware

Some older hardware, such as x86, create problems for the full method of virtualization. For example, certain sensitive instructions that need to be handled by the VMM do not trap. Therefore, hypervisors must dynamically scan and trap privileged-mode code to handle this problem.

Figure 3. Paravirtualization shares the process with the guest operating system

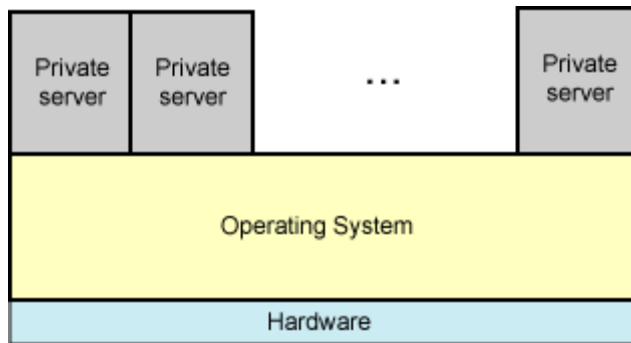


As I mentioned, paravirtualization requires the guest operating systems to be modified for the hypervisor, which is a disadvantage. But paravirtualization offers performance near that of an unvirtualized system. Like full virtualization, multiple different operating systems can be supported concurrently.

Operating system-level virtualization

The final technique we'll explore, operating system-level virtualization, uses a different technique than those covered so far. This technique virtualizes servers on top of the operating system itself. This method supports a single operating system and simply isolates the independent servers from one another (see Figure 4).

Figure 4. Operating system-level virtualization isolates servers



Operating system-level virtualization requires changes to the operating system kernel, but the advantage is native performance.

Why is virtualization important?

Before looking at some of the virtualization options available for Linux today, let's look at the advantages of virtualization.

From a business perspective, there are many reasons for using virtualization. Most come down to what's called *server consolidation*. Simply put, if you can virtualize a number of under-utilized systems on a single server, there are distinct savings in power, space, cooling, and administration due to having fewer servers. Because it can be difficult to determine server utilization, virtualization technologies support what's called live migration. *Live migration* allows an operating system and its applications to be migrated to a new server to balance the load over the available hardware.

Virtualization is also important to developers. The Linux kernel occupies a single address space, which means that a failure of the kernel or any driver results in the entire operating system crashing. Virtualization means that you can run multiple operating systems, and, if one crashes due to a bug, the hypervisor and other operating systems continue to run. This can make debugging the kernel similar to debugging user-space applications.

Linux-related virtualization projects

Table 1 shows several virtualization possibilities for Linux and focuses primarily on those solutions that are open source.

Table 1. Linux-related virtualization projects

Project	Type	License
Bochs	Emulation	LGPL
QEMU	Emulation	LGPL/GPL
VMware	Full virtualization	Proprietary
z/VM	Full virtualization	Proprietary
Xen	Paravirtualization	GPL
UML	Paravirtualization	GPL
Linux-VServer	Operating system-level virtualization	GPL
OpenVZ	Operating system-level virtualization	GPL

For information on other solutions, see the [Resources](#) section.

Bochs (emulation)

Bochs is an x86 computer simulator that is portable and runs on a variety of platforms, including x86, PowerPC, Alpha, SPARC, and MIPS. What makes Bochs interesting is that it doesn't just simulate the processor but the entire computer, including the peripherals, such as the keyboard, mouse, video graphics hardware, network interface card (NIC) devices, and so on.

Bochs can be configured as an older Intel® 386, or successor processors such as the 486, Pentium, Pentium Pro, or a 64-bit variant. It even emulates optional graphics instructions like the MMX and 3DNow.

Using the Bochs emulator, you can run any Linux distribution on Linux, Microsoft® Windows® 95/98/NT/2000 (and a variety of applications) on Linux, and even the Berkeley Software Distribution (BSD) operating systems (FreeBSD, OpenBSD, and so on) on Linux.

QEMU (emulation)

QEMU is another emulator, like Bochs, but it has some differences that are worth noting. QEMU supports two modes of operation. The first is the Full System Emulation mode. This mode is similar to Bochs in that it emulates a full personal computer (PC) system with processor and peripherals. This mode emulates a number of processor architectures, such as x86, x86_64, ARM, SPARC, PowerPC, and MIPS, with reasonable speed using dynamic translation. Using this mode, you can emulate the Windows operating systems (including XP) and Linux on Linux, Solaris, and FreeBSD. Many other operating system combinations are also supported (see the [Resources](#) section for more information).

QEMU also supports a second mode called User Mode Emulation. In this mode, which can only be hosted on Linux, a binary for a different architecture can be launched. This allows, for example, a binary compiled for the MIPS architecture to be executed on Linux running on x86. Other architectures supported in this mode include ARM, SPARC, and PowerPC, though more are under development.

VMware (full virtualization)

VMware is a commercial solution for full virtualization. A hypervisor sits between the guest operating systems and the bare hardware as an abstraction layer. This abstraction layer allows any operating system to run on the hardware without knowledge of any other guest operating system.

VMware also virtualizes the available I/O hardware and places drivers for high-performance devices into the hypervisor.

The entire virtualized environment is kept as a file, meaning that a full system (including guest operating system, VM, and virtual hardware) can be easily and quickly migrated to a new host for load balancing.

z/VM (full virtualization)

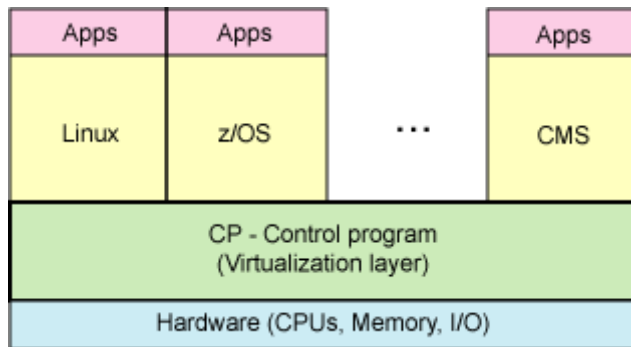
While the IBM System z™ is a new brand name, it actually has a long heritage originating back in the 1960s. The System/360 supported virtualization using virtual machines in 1965. Interestingly, the System z retains backward compatibility with the older System/360 line.

The z/VM® is the operating system hypervisor for the System z. At its core is the Control Program (CP), which provides the virtualization of physical resources to the guest operating systems, including Linux (see Figure 5). This permits multiple processors and other resources to be virtualized for a number of guest operating systems.

Figure 5. Operating system-level virtualization using z/VM

Library-level virtualization

While not discussed here, another method of virtualization that emulates portions of an operating system through a library is library-level virtualization. Examples of this include Wine (a partial Win32 API for Linux) and LxRun (a partial Linux API for Solaris).



The z/VM can also emulate a guest local area network (LAN) virtually for those guest operating systems that want to communicate with each other. This is emulated entirely in the hypervisor, making it highly secure.

Xen (paravirtualization)

Xen is a free open source solution for operating system-level paravirtualization from XenSource. Recall that in paravirtualization the hypervisor and the operating system collaborate on the virtualization, requiring operating system changes but resulting in near native performance.

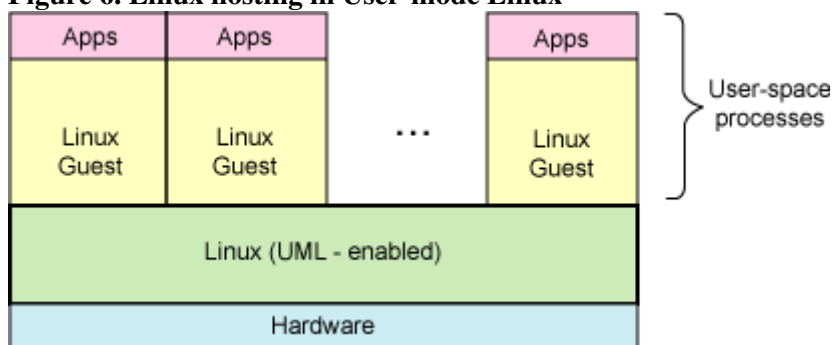
As Xen requires collaboration (modifications to the guest operating system), only those operating systems that are patched can be virtualized over Xen. From the perspective of Linux, which is itself open source, this is a reasonable compromise because the result is better performance than full virtualization. But from the perspective of wide support (such as supporting other non-open source operating systems), it's a clear disadvantage.

It is possible to run Windows as a guest on Xen, but only on systems running the Intel Vanderpool or AMD Pacifica. Other operating systems that support Xen include Minix, Plan 9, NetBSD, FreeBSD, and OpenSolaris.

User-mode Linux (paravirtualization)

User-mode Linux (UML) allows a Linux operating system to run other Linux operating systems in user-space. Each guest Linux operating system exists within a process of the host Linux operating system (see Figure 6). This permits multiple Linux kernels (with their own associated user-spaces) to run within the context of a single Linux kernel.

Figure 6. Linux hosting in User-mode Linux



As of the 2.6 Linux kernel, UML resides in the main kernel tree, but it must be enabled and then recompiled for use. These changes provide, among other things, device virtualization. This allows the guest operating systems to share the available physical devices, such as the block devices (floppy, CD-ROM, and file systems, for example), consoles, NIC devices, sound hardware, and others.

Note that since the guest kernels run in application space, they must be specially compiled for this use (though they can be different kernel versions). This results in what's called the host kernel (which resides on the hardware) and the guest kernel (which runs in the user space of the host kernel). These kernels can even be nested, allowing a guest kernel to run on another guest kernel that is running on the host kernel.

Linux-VServer (operating system-level virtualization)

Linux-VServer is a solution for operating system-level virtualization. Linux-VServer virtualizes the Linux kernel so that multiple user-space environments, otherwise known as *Virtual Private Servers* (VPS), run independently with no knowledge of one another. Linux-VServer achieves user-space isolation through a set of modifications to the Linux kernel.

To isolate the individual user-spaces from one another, you begin with the concept of a context. A *context* is a container for processes of a given VPS, so that tools like `ps` know only about the processes of the VPS. For initial boot, the kernel defines a default context. A spectator context also exists for administration (to view all executing processes). As you can guess, the kernel and internal data structures are modified to support this approach to virtualization.

Linux-VServer also uses a form of `chroot` to isolate the root directory for each VPS. Recall that `chroot` allows a new root directory to be specified, but additional functionality is required (called a *Chroot-Barrier*) so that a VPS can't escape its isolated root directory to the parent. Given an isolated root directory, each VPS has its own user list and root password.

The Linux-VServer is supported by both the 2.4 and 2.6 Linux kernels and operates on a number of platforms, including x86, x86-64, SPARC, MIPS, ARM and PowerPC.

OpenVZ (operating system-level virtualization)

OpenVZ is another operating system-level virtualization solution, like Linux-VServer, but it has some interesting differences. OpenVZ is a virtualization-aware (modified) kernel that supports isolated user-spaces, VPS, with a set of user-tools for management. For example, you can easily create a new VPS from the command line:

Listing 1. Creating a VPS from the command line

```
$ vzctl create 42 --ostemplate fedora-core-4
Creating VPS private area
VPS private area was created
$ vzctl start 42
Starting VPS ...
VPS is mounted
```

You can also list the currently created VPSes using the `vzlist` command, which operates in a similar fashion to the standard Linux `ps` command.

To schedule processes, OpenVZ includes a two-level CPU scheduler. First, the scheduler determines which VPS should get the CPU. After this is done, the second-level scheduler picks the process to execute given the standard Linux priorities.

OpenVZ also includes what are called *beancounters*. A beancounter consists of a number of parameters that define resource distribution for a given VPS. This provides a level of control over a VPS, defining how much memory is available, how many interprocess communication (IPC) objects are available, and so on.

A unique feature of OpenVZ is the ability to checkpoint and migrate a VPS from one physical server to another. *Checkpointing* means that the state of a running VPS is frozen and store into a file. This file can then be migrated to a new server and restored to bring the VPS back online.

OpenVZ supports a number of hardware architectures, including x86, x86-64, and PowerPC.

Hardware support for full virtualization and paravirtualization

Recall that the IA-32 (x86) architecture creates some issues when it comes to virtualization. Certain privileged-mode instructions do not trap, and can return different results based upon the mode. For example, the x86 `STR` instruction retrieves the security state, but the value returned is based upon the particular requester's privilege level. This is problematic when attempting to virtualize different operating systems at different levels. For example, the x86 supports four rings of protection, where level 0 (the highest privilege) typically runs the operating system, levels 1 and 2 support operating system services, and level 3 (the lowest level) supports applications. Hardware vendors have recognized this shortcoming (and others), and have produced new designs that support and accelerate virtualization.

Intel is producing new virtualization technology that will support hypervisors for both the x86 (VT-x) and Itanium® (VT-i) architectures. The VT-x supports two new forms of operation, one for the VMM (root) and one for guest operating systems (non-root). The root form is fully privileged, while the non-root form is depriveged (even for ring 0). The architecture also supports flexibility in defining the instructions that cause a VM (guest operating system) to exit to the VMM and store off processor state. Other capabilities have been added; see the [Resources](#) section.

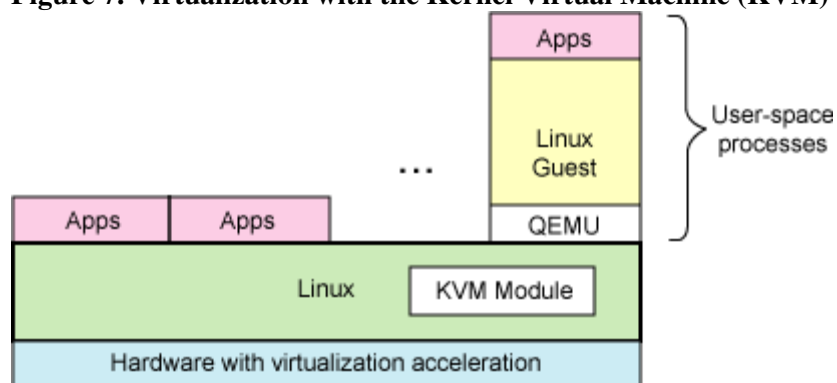
AMD is also producing hardware-assisted virtualization technology, under the name Pacifica. Among other things, Pacifica maintains a control block for guest operating systems that are saved on execution of special instructions. The `VMRUN` instruction allows a virtual machine (and its associated guest operating system) to run until the VMM regains control (which is also configurable). The configurability allows the VMM to customize the privileges for each of the guests. Pacifica also amends address translation with host and guest memory management unit (MMU) tables.

These new technologies can be used by a number of virtualization techniques discussed here, including Xen, VMware, User-mode Linux, and others.

Linux KVM (Kernel Virtual Machine)

The most recent news out of Linux is the incorporation of the KVM into the Linux kernel (2.6.20). KVM is a full virtualization solution that is unique in that it turns a Linux kernel into a hypervisor using a kernel module. This module allows other guest operating systems to then run in user-space of the host Linux kernel (see Figure 7). The KVM module in the kernel exposes the virtualized hardware through the `/dev/kvm` character device. The guest operating system interfaces to the KVM module using a modified QEMU process for PC hardware emulation.

Figure 7. Virtualization with the Kernel Virtual Machine (KVM)






The KVM module introduces a new execution mode into the kernel. Where vanilla kernels support *kernel* mode and *user* mode, the KVM introduces a *guest* mode. The guest mode is used to execute all non-I/O guest code, where normal user mode supports I/O for guests.

The introduction of the KVM is an interesting evolution of Linux, as it represents the first virtualization technology that is part of the mainline Linux kernel. It exists in the 2.6.20 tree, but can be used as a kernel module for the 2.6.19 kernel. When run on hardware that supports virtualization, Linux (32-and 64-bit) and Windows (32-bit) guests are supported. For more information on KVM, see the [Resources](#) section.

Summary

Virtualization is the new big thing, if "new" can include something over four decades old. It has been used historically in a number of contexts, but a primary focus now is in the virtualization of servers and operating systems. Much like Linux, virtualization provides many options for performance, portability, and flexibility. This means that you can choose the approach that works best for you and your application.

Share this...

-  [Digg this story](#)
-  [Post to del.icio.us](#)
-  [Slashdot it!](#)

Resources

Learn

- The [New to IBM Systems](#) page is helpful if IBM Systems are new territory for you. This page details the System i, p, x, z, and more.
- [Grid computing](#) at IBM is based on an open set of standards and protocols that virtualize a distributed computer to create a single powerful system.
- In the [developerWorks Linux zone](#), find more resources for Linux developers.
- Stay current with [developerWorks technical events and webcasts](#).

Get products and technologies

- [Bochs](#) and [QEMU](#) are PC emulators that allow operating systems such as Windows or Linux to be run in the user-space of a Linux operating system.
- [VMware](#) is a popular commercial full-virtualization solution that can virtualize unmodified operating systems.
- [z/VM](#) is the newest VM operating system for the 64-bit z/Architecture. z/VM provides full-virtualization with hardware assist and supports a broad range of operating systems, including Linux.
- [Xen](#) is an open source paravirtualization solution that requires modifications to the guest operating systems but achieves near native performance by collaborating with the hypervisor.
- [User-mode Linux](#) is another paravirtualization solution that is open source. Each guest operating system executes as a process of the host operating system.
- [coLinux](#), or Cooperative Linux, is a virtualization solution that allows two operating systems to cooperatively share the underlying hardware.
- [Linux-Vserver](#) is an operating system-level virtualization solution for GNU/Linux systems with secure isolation of independent guest servers.
- [OpenVZ](#) is an operating system-level virtualization solution that supports checkpointing and migration of live virtual private surfaces.
- The [Linux KVM](#) is the first virtualization technology that has been integrated into the mainline Linux kernel. With a single kernel loadable module, a Linux kernel running on virtualization-capable hardware is able to act as a hypervisor and support unmodified Linux and Windows guest operating systems.

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author



M. Tim Jones is an embedded software architect and the author of *GNU/Linux Application Programming*, *AI Application Programming*, and *BSD Sockets Programming from a Multilanguage Perspective*. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a Consultant Engineer for Emulex Corp. in Longmont, Colorado.

DB2, Lotus, Power PC, Rational, System/360, System z, System z9, Tivoli, WebSphere, and z/VM are trademarks of IBM Corporation in the United States, other countries, or both. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.