Virtual PC™ for Windows®

The Technology of Virtual Machines

A Connectix white paper

Connectix Corp.
2955 Campus Dr. #100
San Mateo, CA. 94403
(650) 571-5100
www.connectix.com

Connectix

Table of contents

Introduction

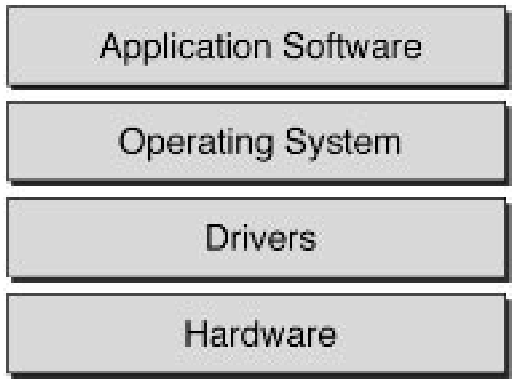
This white paper discusses the technology of *Virtual PC for Windows*, a software program that uses virtual machine technologies developed by Connectix to create in software a complete PC environment. Each virtual machine operates as if it were a stand-alone machine with its own sound card, video board, network adapter and processor. Each virtual computer runs its own operating system; users can install Windows 3.x, Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, MS-DOS, Linux, BSD, OS/2, Novell Netware, and others.

Virtual PC for Windows is the newest addition to the *Virtual PC* line of products from Connectix. It carries forward a record of proven innovation, extending to computer users many years of cross-platform expertise and award-winning technologies that Connectix has originally developed and refined for Macintosh users. The company's flagship Virtual PC products break platform barriers for over one million users worldwide, making Virtual PC "The Ultimate Compatibility Solution".

A PC within a PC

The concept of a virtualized PC is relatively simple. To the user, a virtualized PC has nearly all of the capabilities of a real PC. The difference is that many virtual machines can coexist and run simultaneously on a single “host” PC.

A traditional computer system consists of several layers of hardware and software. The hardware might include a central processor, hard drives, a video card, a network adapter, etc. These hardware resources are managed by a layer of software known as an operating system (or “OS” for short).



Operating systems manage these hardware resources through the use of “drivers” (modular software components that manage a piece of hardware within the system). OSes are designed to be the underlying arbiter for the hardware resources in the machine, controlling the use of memory, CPU cycles, I/O channels, etc. Because of this role, assumptions about “ownership” are built into the code that makes up an OS. For example, a video driver assumes that it “owns” the video adapter. Any software that wants to make use of the video adapter must “talk to” the video driver to make use of this resource. When assumptions about ownership are broken, the system functions incorrectly. For example, if two devices in a PC are both configured to use the same IRQ (interrupt request line) or I/O port range, both of these devices will likely fail.

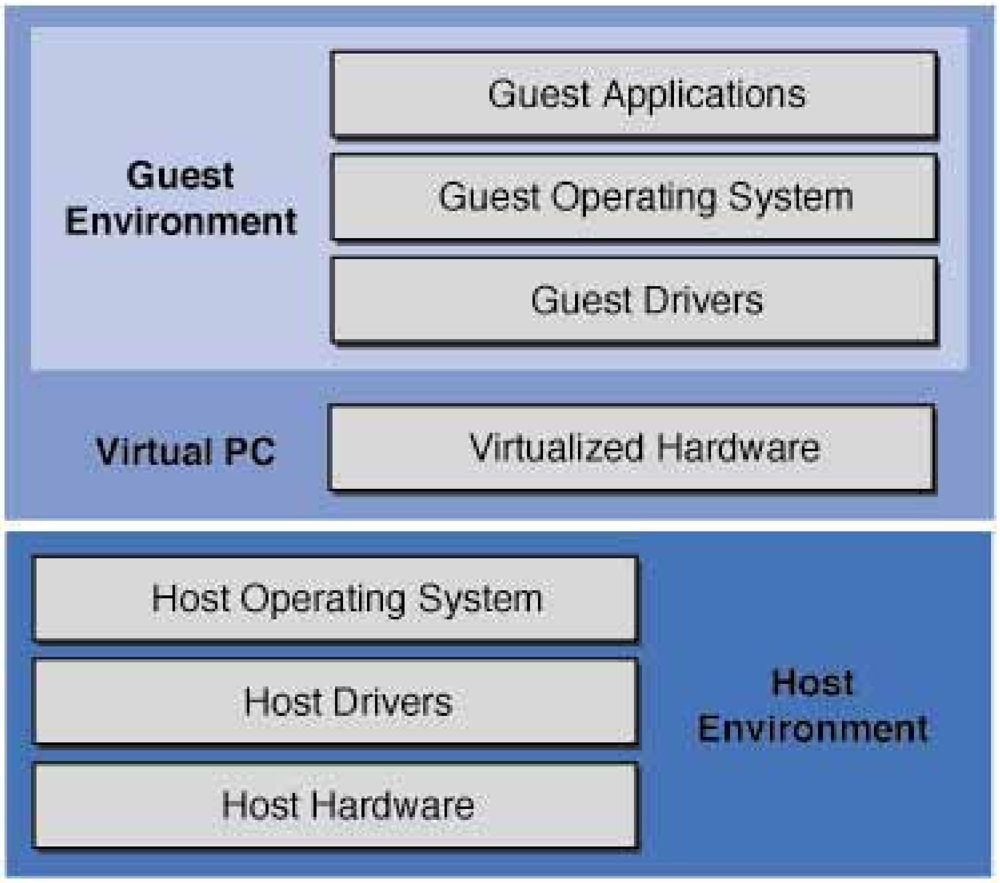
The concept of ownership built into an operating system and its drivers precludes the possibility of running more than one operating system on a particular computer at one time. Or does it?

This is where “virtualization” comes into play. Virtualization involves replacing one of the layers within a computer system in such a way that the higher-level layers are unaffected. In the case of Virtual PC, the lowest layer – the hardware layer – is replaced by a flexible “virtual hardware” implementation. This allows the operating system and its drivers to operate in an environment that matches their built-in assumptions. Each driver is allowed to “own” its corresponding virtual device. For example, the video driver is allowed to manage the “virtual video adapter”.

Because of the flexibility of this virtualization layer, it is easy to construct multiple copies of the virtual machine, allowing for the simultaneous execution of multiple operating systems – all on the same underlying “host” hardware.

Through the magic of virtualization, software running within the virtual machine is unaware that the “hardware” layer has been virtualized. It believes it is running on its own PC – the environment for which it was originally designed. Each virtualized PC is its own independent environment. Software running within one environment is unaware of other virtualized environments running on the same host. It is also unaware of the underlying host operating system. Each OS is “sandboxed” and isolated from the other operating systems.

Once the hardware has been replaced with a virtualized software implementation, it is possible to add new features and capabilities that are technically infeasible or cost-prohibitive to implement in hardware. Virtual PC's hard drive images offer a good example. Virtual PC allows users to "containerize" a virtual machine's hard drive into a single file on the host system. This approach allows for easy portability, backup and recovery and provides an elegant mechanism for deploying a pre-configured system throughout an entire classroom or business.



Virtual PC incorporates other features that are unique to virtual machines such as the ability to suspend the machine and save its state to disk. Users can quickly restore the virtual machine to its previous state without rebooting the guest OS.

Virtual PC makes it possible to combine several individual PCs running different OSes into a single system. This results in an obvious cost savings. However, there are other benefits to running multiple operating systems on a single host system. Virtual PC offers special integration features like copy & paste, which allows the user to transfer text and graphics between virtual machines and the host environment. Another integration feature – "drag and drop" – allows the user to drag files between the virtualized guest environment and the host environment or vice versa, making data sharing fast and simple.

For more information on how virtual machines can be used to solve a variety of problems experienced by computer users, refer to the companion white paper titled "Understanding Virtual Machine Technology." The rest of this white paper discusses some of the techniques used by Virtual PC in implementing the virtual hardware layer.

Virtualizing the x86 processor

At the core of a computer – and a virtual machine – is the central processor. For some virtual machines, it is necessary to create a completely synthetic CPU because the virtual processor is of a completely different type than the host processor. For example, *Virtual PC for Macintosh* emulates an x86 processor using a host PowerPC processor.

Virtual PC for Windows is able to make direct use of the host's x86 processor. However, virtualization is still required to "fool" the guest operating system into believing that it is the exclusive owner of the x86 processor without actually handing over control. Connectix engineers developed a variety of techniques to accomplish this.

Privileged versus user mode

Most processor architectures contain two or more "privilege levels" – a feature that can often be exploited when virtualizing the processor. Typically, the most privileged level is used by the operating system and driver software. Application software, on the other hand, uses the least-privileged level (sometimes called "user mode"). These levels limit access to certain functionality within the processor, allowing the operating system to maintain control over the system at all times.

A standard technique for virtualizing a processor involves executing privileged code in user mode. Any functionality that is prohibited within user mode causes the processor to generate an exception. Exception handlers can then "emulate" the privileged functionality while still maintaining control over the system. This technique relies on the fact that most processors faithfully generate exceptions for all privileged-level operations executed in user mode. Unfortunately, the x86 architecture doesn't follow this rule.

For example, the x86 makes use of a "global descriptor table" (GDT) to define segments through which all memory accesses are performed. Because the GDT is a global resource, it must be maintained by the operating system. It would logically follow that all processor instructions that modify or explicitly access the GDT should be considered privileged, and therefore inaccessible from user-mode code. The x86 architecture does treat the LGDT (load global descriptor table) instruction as privileged, but the SGDT (store global descriptor table) instruction is not. This lack of symmetry, and other similar behaviors, makes the standard technique for virtualization ineffective. Because the SGDT instruction doesn't cause an exception in user mode, there is no opportunity to virtualize the location of the GDT through the use of exception handlers.

Because of the limitations of the x86 architecture, Connectix was forced to develop other techniques for executing x86 code within the virtual machine environment. One such technique draws on experience gained by Connectix engineers during the development of *Virtual PC for Macintosh*, where the technique known as "binary translation" was employed. This mechanism uses a runtime just in time (JIT) compiler to translate the original code into a different form. In the case of *Virtual PC for Macintosh*, x86 code is translated to PowerPC code. Within *Virtual PC for Windows*, x86 code is translated into modified x86 code. In most cases, little or no translation is necessary. However, if a privileged instruction (e.g. SGDT) is encountered, the code can be translated in such a way as to correctly virtualize the processor.

The binary translation process imposes a certain overhead, negatively impacting the performance of the virtual machine. Luckily, this process is only necessary for privileged-mode code. All user-mode code can be executed in “raw mode” (i.e. executed directly by the host processor without the use of translation). In most modern OS environments, the vast majority of the time (typically greater than 90%) is spent executing user-mode code. Therefore, the overhead of the translation process is typically insignificant.

Virtualizing I/O

Although the processor is at the center of a computer system, there are many other peripheral devices that work in concert with the processor. X86 software uses two mechanisms to communicate with these external devices:

- *Programmed I/O (PIO)*: The processor can communicate with other devices through I/O “ports” using a dedicated 16-bit address space. This mechanism makes use of specialized instructions such as IN and OUT (which read from and write to an I/O port, respectively). When an I/O instruction is executed within the virtual machine environment, the host processor is programmed to generate an exception. Virtual PC’s exception handler can then interpret the I/O access, forwarding it on to the appropriate virtualized device. For example, an IN from port $1F7 instructs the processor to read data from the IDE controller. Instead of allowing this instruction to access the host’s IDE controller, the processor is programmed to generate an exception. Through the use of an exception handler, Virtual PC redirects this operation to the virtual IDE controller, which in turn returns data read from the associated virtual hard disk image.
- *Memory-mapped I/O (MMIO)*: The processor can also communicate with other devices through normal load and store instructions. These instructions, usually used to access the machine’s RAM, can also be used to send data to or from external devices. Virtual PC handles such addresses through the host’s MMU (memory management unit). Address ranges corresponding to MMIO ranges are left unmapped, so any access to these ranges causes a page fault. Virtual PC’s page fault handler then interprets the access, forwarding it on to the appropriate virtual device

Faithful emulation of an entire PC requires the virtualization of about two dozen devices including the following:

- PIC (programmable interrupt controller)
- DMA (direct memory access) controller
- IDE controller
- CMOS (battery-backed RAM)
- RTC (battery-backed real time clock)
- PIT (programmable interval timer)
- Ethernet NIC (network interface card)
- Game port controller
- COM (serial) controller
- LPT (parallel) controller
- Memory controller & I/O controller
- PCI bus and host & PCI bridge
- Speaker
- Video adapter (including text, CGA, VGA and SVGA modes)
- Video accelerator (2D) and hardware cursor

- Keyboard controller
- Mouse and keyboard
- Power management hardware
- Sound card (both DSP and FM generation)

The "ownership" problem discussed above means that most of the virtual devices cannot be implemented directly using the corresponding host hardware. For example, the virtual interrupt controller cannot make use of the host interrupt controller because the latter is already "owned" by the host OS. Rather, the virtual interrupt controller is implemented in software.

This means the hardware "seen" by the OS running within the virtual machine may not match the hardware "seen" by the host OS. For example, the Windows Device Manager on the host may indicate a Matrox Millenium video card and a 3COM ethernet adapter, but the Device Manager within the Virtual PC's virtual machine environment will indicate an S3 Trio 32 video card and a DEC 21041 ethernet adapter. This can be confusing to first-time Virtual PC users, but it can be explained by the presence of the virtual devices emulated within Virtual PC.

The types of virtual devices implemented within Virtual PC were chosen due to the broad availability of drivers among various operating systems. For example, the S3 Trio video card was nearly ubiquitous in its day, so almost every PC-compatible OS supports this card with built-in drivers. Likewise, nearly every PC-based operating system contains a driver for the DEC 21041 ethernet adapter. This allows older operating systems to run within Virtual PC on a new host computer even if the guest OS contains no driver support for the new host hardware.

There are some restrictions imposed by this virtualization technique. Because of the "ownership" problem, Virtual PC doesn't allow virtual machines to directly access PCI cards or PC cards installed in the host computer. Generally, these cards are "owned" by a host driver, so they can't also be owned by a driver within the virtual machine. This limitation precludes the use of arbitrary plug-in hardware within the virtual machine environment.

Sharing the host processor

It is the job of the host OS to arbitrate and control global resources like CPU cycles, address space, etc. How, then, does Virtual PC allow multiple virtual machines to share the CPU with the host OS?

From the perspective of the host OS, Virtual PC is just an application – with its own threads, each of which is preemptively scheduled along with all of the other currently-running threads. If Virtual PC is the only active application running on the host, it will be allocated nearly all of the available CPU cycles, and the virtual machine environment will run at its maximum speed. If Virtual PC is running alongside another application that is actively utilizing the processor, the host OS will probably only allocate half of the CPU cycles to Virtual PC, and the virtual machine environment will run at half of its maximum performance. When multiple concurrent virtual machines are running, available CPU cycles are further subdivided.

Address translation (MMU) virtualization

In an effort to provide maximum performance within the virtual machine environment, Virtual PC makes use of the host processor's MMU (memory management unit) to perform address translation – the conversion of a "logical" address to a "physical" address. This translation must be done on every memory reference, so it's important to use the underlying hardware; a software implementation would be prohibitively slow.

An x86 processor is capable of addressing up to 4GB (32 bits) of virtual address space. Typically, an operating system will carve up this 4GB address space into small bands of address ranges. For example, Windows 2000 usually places its kernel at address 0xC0000000 and user-mode application programs at address 0x00400000. Other parts of the address space may be used for the system BIOS, memory-mapped I/O, etc.

The layout of the address space is "owned" by the host operating system, so, how is it possible for Virtual PC to impose the address space layout of the guest operating system when the virtual machine is executing? Virtual PC uses a technique that involves temporarily commandeering ownership of the MMU from the host OS. This is done for relatively short periods of time (measured in milliseconds) without knowledge of the host OS, so it continues to believe it is in complete control of the host processor the entire time. So, while the virtual machine environment is in control of the host processor, the address space layout is made to reflect the current layout of the guest operating system.

The virtual machine environment must maintain control over the processor rather than completely handing it over to the guest operating system. It does this through a mini-kernel that consists primarily of exception handlers. This kernel is referred to as a "hypervisor". The hypervisor is responsible for performing the address space context switch and for handling any exceptions that occur while executing guest instructions.

One major problem still remains with this technique. If the address space layout is dictated by the guest OS, where in the address space should the hypervisor (including its code and data) be located? One approach to this problem is to pick an arbitrary location within the address space – a location that is seldom used by most operating systems. However, this approach will ultimately fail. If the guest OS decides to use the location currently occupied by the hypervisor, a conflict will arise.

Connectix engineers solved this problem by making the hypervisor (both its code and data) relocatable. The hypervisor monitors the address space usage of the guest OS and is able to relocate itself if and when a conflict arises. This innovation (one of numerous patent-pending technologies incorporated into Virtual PC) allows for greater compatibility with a variety of x86 operating systems.

Virtual hard drives

Many unique applications of Virtual PC come from the use of its flexible virtual hard drive configurations.

A hard drive is an example of a “block storage device” because all data is stored in numbered blocks (sometimes called “sectors”) that can be addressed individually by number. Higher-level constructs such as partitions and files are used to impose order onto the contents of the hard drive. However, at the hardware level, a hard drive is simply a large number of sequentially numbered blocks.

Virtual PC virtualizes hard drives at the hardware level. This allows software within the virtual machine to define the partition and file format on the virtual hard disks. Virtual PC can therefore support old formats such as FAT16 and FAT32 as well as newer formats like NTFS or formats that are yet to be defined.

The default mechanism used by Virtual PC to represent virtual hard drives is a “hard drive image”. From the perspective of the host OS, a hard drive image appears as a single, large file. The simplest type of hard drive image is a “fixed size” image in which all of the blocks within the virtual hard drive are stored sequentially in the file starting at block number zero. For example, a virtual hard drive that is 800Mb in size can be represented by a single 800Mb file.

Computer users are familiar with the concept of a “file” – a container that can grow or to shrink to accommodate more or less data. The size of a hard drive, on the other hand, is predetermined and fixed. However, through the use of virtualization, a hard drive image can take on some of the properties of a file. Virtual PC implements an “expanding drive image” format that grows as new portions of the hard drive are accessed. When an expanding drive image is first created, it is represented as a small file on the host. When a previously unaccessed block is written, the file is enlarged, and the block is appended to the end. Blocks that have never been written contain zeroed data by default. The expanding image format includes internal data structures that track which blocks have been allocated and where they exist within the image file. For example, if a hard drive image is created with a maximum capacity of 16GB, the initial file may only take up several megabytes of space. As data is written to the drive image, the file expands – eventually consuming the full 16GB of space. To the guest OS, the virtual hard disk always appears to have a 16GB capacity.

Virtual PC also implements an extension to the basic hard drive image concept called a “differencing drive”. These drive images work in conjunction with an existing “base drive image”. Any changes made to the hard drive are written only to the differencing drive, leaving the base drive image in its original state. This feature allows several virtual machines to share a single base drive image. Changes to the virtual hard disk are stored in the differencing drive, thus saving storage space.

Finally, Virtual PC implements an “undoable” feature for any hard drive image. When a hard drive image is marked as “undoable”, all changes to the drive are written to a temporary differencing drive rather than the actual drive image. At the end of the session, the user is given a choice between “committing” the changes back to the hard drive image or “undoing” all of the changes made during the session.

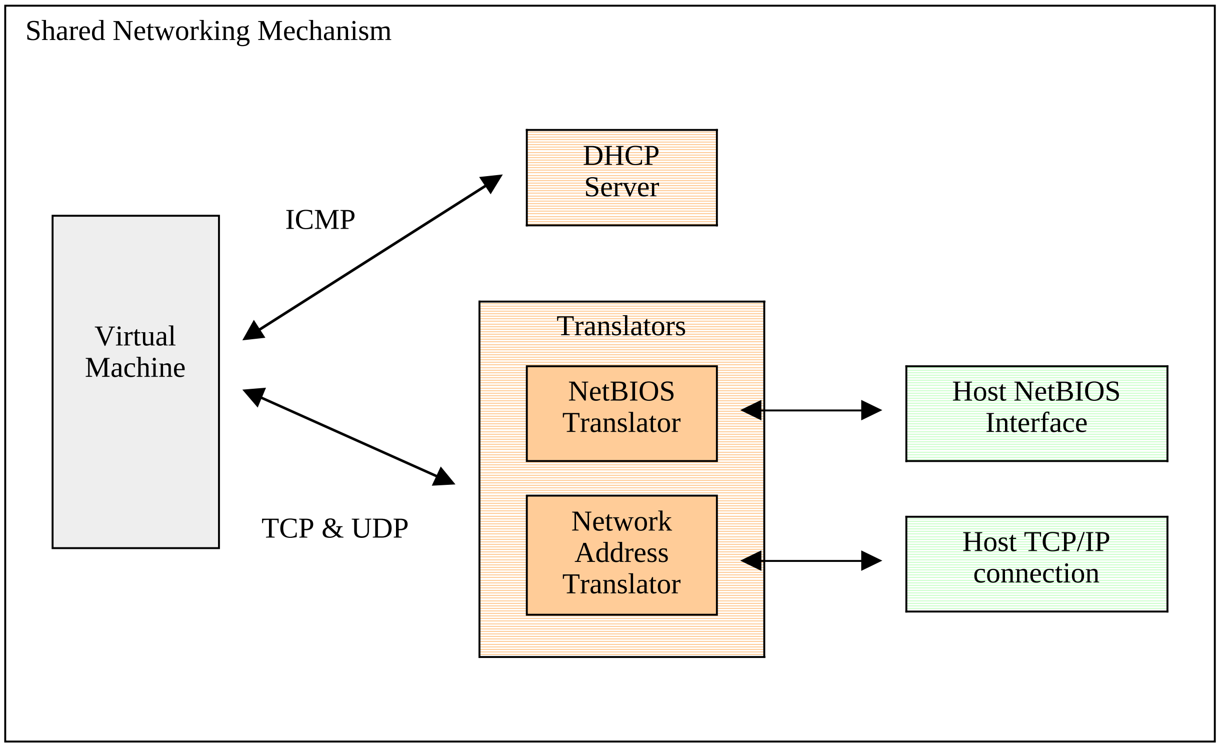
Virtualized networking

Virtual PC offers two powerful virtualized networking modes. Both of these provide network connectivity from within the virtual machine, and both are implemented as a virtual DEC 21041 ethernet card. The similarities end there.

Shared networking

Shared networking is useful for TCP/IP client connectivity. It makes use of an integrated NAT (network address translation) mechanism and a DHCP server. Here’s how it works:

- Packets sent from the virtual DEC card are intercepted by the shared networking subsystem. The ethernet (ARP) portion of the packet is ignored, and the packet data (the payload) is handled by packet type.
- Non-IP packets are ignored.
- ICMP packets that contain DHCP requests are handled by an internal DHCP server which assigns a non-routable IP address to the guest OS.
- UDP and TCP packets that originate from the assigned IP address are translated and sent to the current TCP connection on the host.
- Response UDP and TCP packets are translated and sent as incoming packets to the DEC card.
- The translation is done using IP address/port number combinations. For example, if a packet is sent with a source address that matches the guest and a source port of 756, this packet may be sent out of the host’s port 534. If a response packet is received on the host’s port 534, it is translated back to the original guest port of 756. Thus, multiple virtual machine clients can be running simultaneously, all sharing a single host TCP/IP connection (thus the term “shared networking”).
- Packets that contain NetBIOS commands are specially handled and directed to the host NetBIOS interface. This allows clients running within virtual machines to access files and printers on other machines.



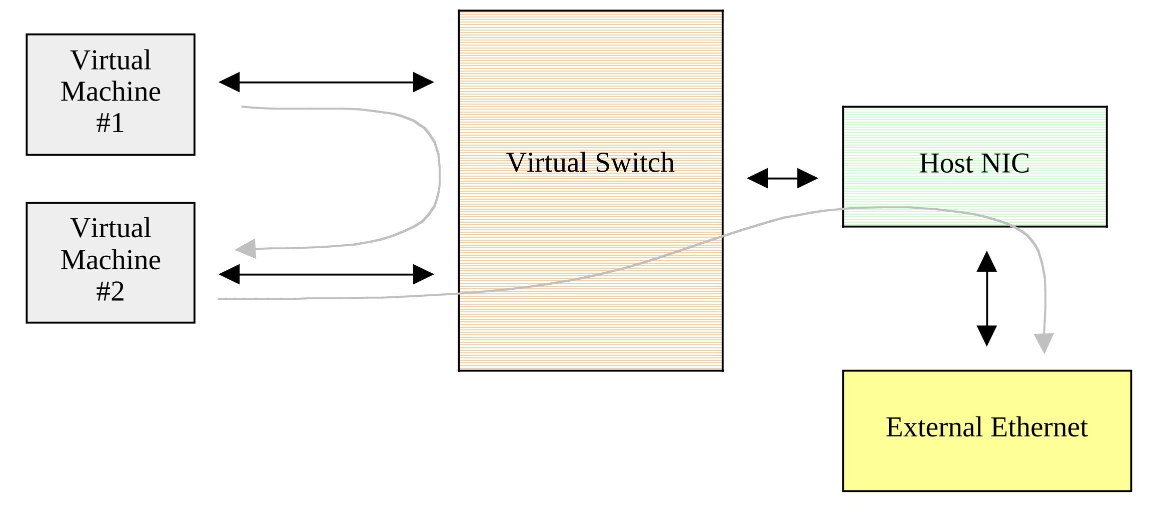
Because shared networking works at the TCP/IP level of the networking stack, it is compatible with any TCP/IP connection including ethernet, token ring, dial-up adapters (PPP), etc.

Virtual switch

Virtual PC's virtual switch networking mode provides lower-level network connectivity at the ethernet (ARP) level. It can be thought of as an Ethernet switch implemented in software. It is useful for a variety of client and server applications. Here's how it works:

- Packets sent from the virtual DEC card are intercepted by the virtual switch subsystem. The packet data (the payload) is not interpreted in any way. This mode is therefore compatible with any packet type that can be encapsulated within an ethernet packet.
- The destination hardware (MAC – "media access control") address of the packet is extracted and examined.
- If the destination MAC address corresponds to the address of a virtual NIC in one of the other currently running virtual machines, the packet is routed directly to that virtual machine.
- If the destination MAC address corresponds to the address of the host's NIC and guest-to-host routing is enabled, the packet is forwarded to the host.

Virtual Switch Mechanism (gray lines show two possible packet routes)



- If the destination MAC address doesn't correspond to any known NIC, it is presumed to be an external address. If external routing is enabled, the packet is sent onto the external network via the specified host NIC.
- Broadcast and multicast packets are routed according to the routing preferences and predefined multicast filtering.
- Incoming packets are routed to the appropriate virtual machine, once again according to the routing preferences.

Because the virtual switch operates at the ethernet (ARP) layer, it is compatible with any higher-level transport protocols including TCP/IP, IPX, AppleTalk, etc.

Virtual PC supports various routing options that can be defined on a per-virtual machine basis. For example, it's possible to limit all network traffic routing between the virtual machines. Likewise, it's possible to allow virtual machines to communicate with the host system and the external ethernet network.

Unlike shared networking, virtual switch mode is tied to one of the host's ethernet NICs. It will not work with other links such as token ring or dialup adapters.

Summary

The concept of a virtual machine is simple, but the technology that makes Virtual PC possible is quite complex. The result is an application that allows for much more than simply running multiple concurrent virtual machines. Connectix has implemented the virtualized devices in such a way as to provide features that are not available on real PCs. The result is a more versatile and flexible PC that can support a much broader range of applications.

Media contact

North America:
Nancy K. Smith
Candelori Communications, Inc.
(408) 774-3414
nancysmith@candelori.com

International:
David Lawson
Connectix Corporation
+1 (650) 638-7393
dlawson@connectix.com

About Connectix

Connectix has been at the forefront of virtual machine (VM) development since it was founded in 1988. Connectix used VM technology to produce the first implementation of virtual memory for personal computers. In 1997, Connectix introduced Virtual PC for Mac. This new product combined the VM technology with a processor emulator to enable users to run Windows applications on their Macs. Today, over one million users make Virtual PC the most widely used cross-platform product of its kind and the benchmark of its category.