At a glance:

- Memory management and SMB 2.0
- NTFS self-healing, Windows Hardware Error Architecture and the Driver Verifier
- Scalability with I/O completion ports, thread pools and NUMA
- Hyper-V virtualisation

Inside Windows Server 2008 kernel changes

Mark Russinovich

Windows Server 2008 is the latest release of the Microsoft server platform and it includes system-level changes that span every functional area of the operating system, from memory management to thread scheduling and from networking to security, just to name a few.

Because Windows Server® 2008 shares the same kernel as Windows Vista® SP1, it includes many of the enhancements that I covered in previous *TechNet Magazine* articles: 'Inside the Windows Vista Kernel' (April and July 2007) and 'Inside Windows Vista User Account Control' (September and November 2007). Only a handful of the features I described in those articles are exclusively client-focused and not included in Windows Server 2008, such as SuperFetch, ReadyBoost, ReadyDrive, ReadyBoot and the Multimedia Class Scheduler Service (MMCSS).

Rather than repeat coverage of the important kernel changes introduced in Windows Vista that are also in Windows Server 2008, such as I/O prioritisation, the new boot architecture, BitLocker™, code integrity, and mandatory integrity levels, I'm going to focus on the key changes that I didn't cover in those articles, including ones related to reliability, performance, scalability, as well as the new Microsoft hypervisor machine virtualisation technology, Hyper-V™.

Also, like the previous articles, the scope of this one is restricted to the operating system kernel, Ntoskrnl.exe, as well as closely associated system components. It does not cover changes to installation (WIM, or Windows® Imaging Format, and Component-Based Servicing), management (Group Policy and Active Directory® improvements), general diagnostics and monitoring (Windows Diagnostic Infrastructure), core networking (the new firewall and TCP/IP implementation), Server Core or Server Roles, for example.

This article is based on a prerelease version of Windows Server 2008. All information herein is subject to change.

Working on multiprocessor systems

One of the low-level changes to the system is that Windows Server 2008 only includes a version of the kernel designed to work on multiprocessor systems. In the past, Windows used a version specific to uniprocessors on machines with a single CPU because that

version could achieve slightly better performance by omitting the synchronisation code required only in multiprocessor environments. As hardware has become faster, the performance benefit of the optimisations has become negligible, and most server systems today include more than one processor, making a uniprocessor version unnecessary.

Figure 1 shows the variants of the Windows Server 2008 kernel, where the version used on a system depends on whether it's the debug (Checked) or retail version of the operating system, whether the installation is 32-bit or 64-bit (Itanium, Intel 64 or AMD64) and, if it's a 32-bit installation, whether the system has more than 4GB of physical memory or supports Data Execution Prevention (DEP). Windows Server 2008 is also the last Windows Server operating system that is expected to offer a 32-bit version.

Every release of Windows Server focuses on improving the performance of key server scenarios such as file serving, network I/O and memory management. In addition, Windows Server 2008 has several changes and new features that allow Windows to take advantage of new hardware architectures, adapt to high-latency networks and remove bottlenecks that constrained performance in previous versions of Windows. This section reviews enhancements in the memory manager, I/O system and the introduction of a new network file system, SMB 2.0.

Memory management

The memory manager includes several performance enhancements in Windows Server 2008. For example, it issues fewer and larger disk I/Os than it does on Windows Server 2003 when fetching data from the paging file or performing read-ahead I/Os on mapped files. The larger file I/Os are facilitated by changes in the I/O system that remove a 64KB I/O-size limit that's been present since the first release of Windows NT®.

Figure 1 **Windows Server 2008 kernel variants**

Kernel	32-bit	64-bit
Multiprocessor	Yes	Yes
Multiprocessor Checked	Yes	Yes
Multiprocessor Physical Address Extension (PAE)	Yes	No
Multiprocessor PAE Checked	Yes	No

Also, it's important to note that data reads for read-ahead (speculative reads) from mapped files by the Cache Manager are typically twice as large on Windows Server 2008 than they are on Windows Server 2003 and go directly into the standby list (the system's code and data cache). This behaviour occurs instead of requiring the Cache Manager to map virtual memory and read the data into the System's working set (memory assigned to the System by the memory manager), which might cause other in-use code or data to be needlessly evicted from the working set.

The memory manager also performs larger I/Os when writing data to the paging file. Whereas Windows Server 2003 often performed writes of even less than 64KB, on Windows Server 2008 the memory manager commonly issues 1MB writes.

Besides improving performance by reducing the number of writes to the paging file, the larger writes also reduce fragmentation within the paging file. This in turn causes a reduction in the number of reads and disk seeks that are required to read back multiple pages, since they will more often than not be adjacent.

The memory manager tries to write out other modified pages that are close to the one being written out in the owning process's address space, and it targets the area of the paging file where other neighbouring pages already reside. This minimises fragmentation and can improve performance because pages that might eventually be written out to the paging file have already been written. It also reduces the number of paging reads required to pull in a range of adjacent process pages. Look at the sidebar 'Experiment: seeing large disk I/Os' for more information on the memory manager's use of large I/Os.

SMB 2.0

The Server Message Block (SMB) remote file system protocol, also known as Common Internet File System (CIFS), has been the basis of Windows file serving since file serving functionality was introduced into Windows. Over the last several years, SMB's design limitations have restricted Windows file serving performance and the ability to take advantage of new local file system features. For example, the maximum buffer size that can be transmitted in a single message is about

60KB, and SMB 1.0 was not aware of NTFS client-side symbolic links that were added in Windows Vista and Windows Server 2008.

Windows Vista and Windows Server 2008 introduce SMB 2.0, which is a new remote file serving protocol that Windows uses when both the client and server support it. Besides correctly processing client-side symbolic links and other NTFS enhancements, SMB 2.0 uses batching to minimise the number of messages exchanged between a client and a server. Batching can improve throughput on high-latency networks like wide-area networks (WANs) because it allows more data to be in flight at the same time.

Whereas SMB 1.0 issued I/Os for a single file sequentially, SMB 2.0 implements I/O pipelining, allowing it to issue multiple concurrent I/Os for the same file. It measures the amount of server memory used by a client for outstanding I/Os to determine how deeply to pipeline.

Because of the changes in the Windows I/O memory manager and I/O system, TCP/IP receive window auto-tuning, and improvements to the file copy engine, SMB 2.0 enables significant throughput improvements and reduction in file copy times for large transfers. Since both operating systems implement SMB 2.0, deploying Windows Server 2008 file servers with Windows Vista clients enables the use of SMB 2.0 and the realisation of these performance benefits.

Reliability with NTFS self-healing

Reliability is a key server attribute, and Windows Server 2008 delivers various improvements that help administrators keep their server running smoothly, including online NTFS consistency repair, a new hardware error-reporting infrastructure and extensions to the Driver Verifier.

With today's multi-terabyte storage devices, taking a volume offline for a consistency check can result in a multi-hour service outage. Recognising that many disk corruptions are localised to a single file or portion of metadata, Windows Server 2008 implements a new NTFS self-healing feature to repair damage while a volume remains online.

When NTFS detects corruption, it prevents access to the damaged file or files and creates a system worker thread that executes Chkdsk-like corrections to the corrupted data structures, allowing access to the repaired files when finished. Access to other files continues as normal during this operation, minimising service disruption.

WHEA infrastructure

The Windows Hardware Error Architecture (WHEA) infrastructure included in Windows Server 2008 promises to simplify the management of hardware failures and

Experiment: seeing large disk I/Os

You can use a file system monitoring tool like TechNet Sysinternals Process Monitor (technet.microsoft.com/sysinternals/bb896645.aspx) to look for large file I/O operations on a Windows Server 2008 system.

There are several ways to go about generating large I/Os. If you have a second system running either Windows Vista Service Pack 1 or Windows Server 2008, then you can run Process Monitor on the server and monitor file copies to the second system. You can also usually generate large paging file I/Os by running a memory-intensive program that causes the memory manager to write pages out to the paging file.

Figure A shows Process Monitor after running a memory-intensive program on a Windows XP system, with the Enable Advanced Output option checked in the Process Monitor Options menu and a filter set to only show writes to the paging file, pagefile.sys. The Detail column shows that the writes are 64KB in size.

When you run the same steps on Windows Server 2008, you'll most likely see something similar to what is displayed in **Figure B**, which shows most of the writes to be approximately 1MB in size.

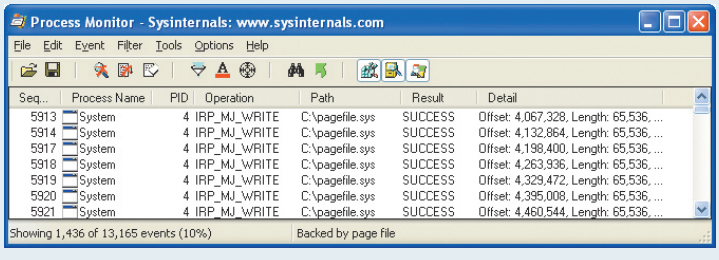

Figure A **Page file writes on Windows XP**

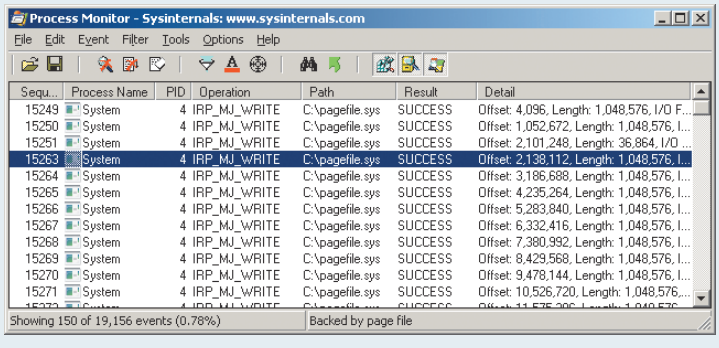

Figure B **Larger page file writes on Windows Server 2008**

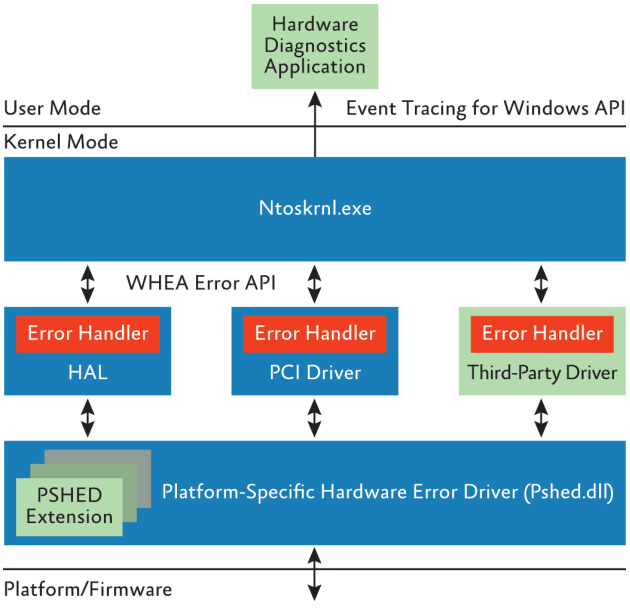

Figure 2 **WHEA error-reporting infrastructure**

enable proactive response to non-fatal errors. Servers often have strict uptime guarantees, so identifying and responding to errors in a timely manner on such systems is critical.

Analysis of crashes submitted to Microsoft via Online Crash Analysis (OCA) shows that roughly 10 percent of operating system crashes are in response to a hardware failure, but determining the root cause of these crashes has been difficult or impossible because there's insufficient error information provided by the hardware for capture in a crash. In addition, prior to Windows Server 2008, Windows had not provided built-in support for monitoring the health of devices or implemented remediation or notification of imminent failure. The reason behind this is that hardware devices don't use a common error format and provide no support for error management software.

WHEA provides a unified mechanism for error-source discovery and reporting for platform devices, including processors, memory, caches and busses like PCI and PCI Express. It does so by implementing the architecture shown in **Figure 2**, where the core is a kernel API that error sources call to report errors. The API requires all errors to be formatted in a common way, and it logs errors using Event Tracing for Windows (ETW) events (fatal errors are logged after a reboot).

ETW was introduced in Windows 2000, and the WHEA use of ETW makes it easy for hardware manufacturers and software vendors to develop device diagnostics management applications that consume WHEA events. If an event is severe enough to warrant a system crash, WHEA ensures that the fatal error record is stored in the crash dump file so that administrators can determine the crash's root cause.

Another key piece of WHEA is the Platform Specific Hardware Error Driver (PSHED) found in %Systemroot%\System32\Pshed.dll. The kernel links with PSHED, and it interfaces with platform and firmware hardware, essentially serving as a translation layer between their error notifications and the WHEA error-reporting API. There's a Microsoft-supplied PSHED for each platform architecture (x86, x64, Itanium), and PSHED exposes a plug-in model so that hardware vendors and manufacturers can override the default behaviours with ones specific to their platforms.

Finally, components of the system that interface with other error sources – including device drivers, the Hardware Abstraction Layer (HAL) and the kernel – can implement Low-Level Hardware Error Handlers (LLHELs) that initially handle an error condition. An LLHEL's job is to extract error information from the device, notify PSHED to allow it to collect additional platform error information and then call the kernel's WHEA error-reporting API.

Driver Verifier

The Driver Verifier, a powerful tool for tracking down buggy device drivers and faulty hardware, has been included in every copy of Windows since Windows 2000. Administrators commonly configure the Driver Verifier (%Systemroot%\System32\Verifier.exe) to closely monitor the behaviour of device drivers suspected of causing system crashes. The Driver Verifier catches illegal driver operations so that a crash dump file points directly to the guilty party.

A drawback of previous Driver Verifier implementations is that most configuration changes require restarting the system, something that is obviously undesirable on a production server. The Windows Server 2008 implementation of Driver Verifier improves this process by removing the restart requirement for the most useful verifications, making it possible to troubleshoot a problematic server without having to restart it.

Driver Verifier introduces three new verifications, visible in **Figure 3**. 'Security checks' ensures that device drivers set secure permissions on the objects they use to interface with applications. 'Force pending I/O requests' tests a driver's resilience to asynchronous I/O operations that complete immediately rather than after a delay. And 'Miscellaneous checks' looks for drivers improperly freeing in-use resources, incorrectly using Windows Management Instrumentation (WMI) registration APIs, and leaking resource handles.

Scalability

Scalability refers to the ability of an operating system or application to effectively utilise multiple processors and large amounts of memory. Every release of Windows improves scalability by minimising or eliminating the use of locks that reduce parallelism on multiprocessors, and Windows Server 2008 is no exception to this trend.

A relatively small but significant improvement is in the code that executes timer expiration, which no longer acquires the dispatcher lock, a systemwide scheduler lock used by all low-level synchronisation operations. The resulting reduction in CPU synchronisation overhead enables Windows Server 2008 terminal server systems to support about 30 percent more concurrent users than Windows Server 2003.

Other scalability improvements in Windows Server 2008 include completion port enhancements, a new threadpool implementation, more efficient use of Non-Uniform Memory Access (NUMA) hardware, and dynamic system partitioning.

Improved I/O completion port handling

Most scalable Windows server apps, including IIS, SQL Server® and Exchange Server, rely on a Windows synchronisation API called a completion port to minimise switching between multiple threads of execution when executing I/O operations. They do this by first associating notifications of new request arrivals, such as Web server client connections, with a completion port and dedicating a pool of threads to wait for the notifications. When a request arrives, Windows schedules a thread, which then usually executes other I/O operations like reading a Web page from disk and sending it the client to complete the request.

So that the same thread can return to waiting for more client requests as quickly as possible, the thread issues I/Os asynchronously and associates I/O completions with the completion port. The thread then returns to waiting on the completion port, which will schedule the thread when either a new request arrives or one of the I/Os completes. In this way, the same thread remains active on a CPU, alternately processing client requests and waiting on the completion port.

A drawback of the completion port implementation of prior Windows releases is that, when an I/O completed, the I/O system caused the thread that issued the I/O to immediately perform a small bit of completion processing, regardless of whatever else it was doing. If other threads were active, that often caused the scheduler to preempt an active thread and context switch to the issuer.

Windows Server 2008 avoids these context switches by deferring completion processing to the next thread to wait on the completion port with which the I/O is associated. Thus, even if a different thread is the next to wait on the completion port, it will perform the completion processing before executing other code, and the scheduler does not need to switch to the issuing thread. This minimisation of context switches can dramatically improve the scalability of heavily loaded server applications.

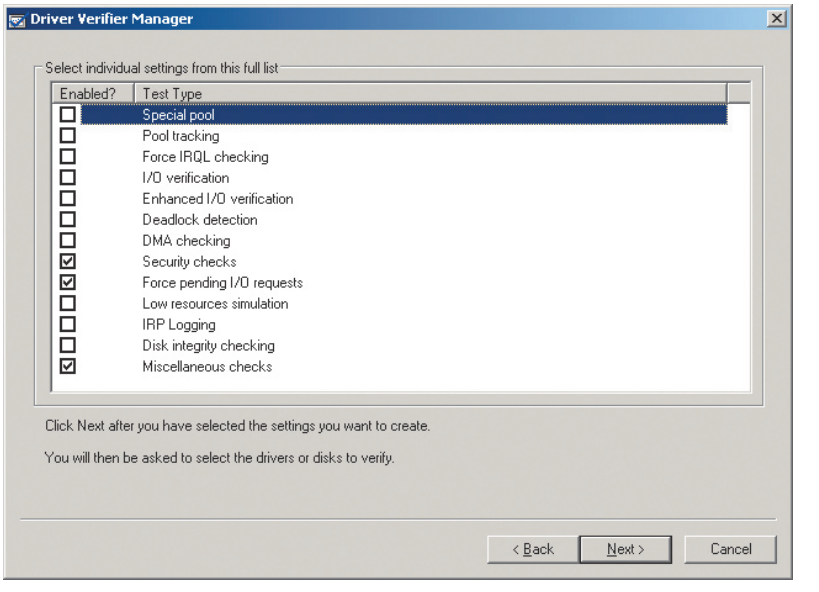

Figure 3 **Driver Verifier with Windows Server 2008 options checked**

The goal of the thread pool is to minimise context switches by using the same threads to execute multiple work items in succession

More efficient thread pools

Writing applications that take advantage of multiple CPUs can be difficult, so Windows XP introduced worker thread pools, an infrastructure and associated API that abstracts the details of executing small units of work across CPUs. An application specifies the work items to the thread pool API, which then executes them on one of a number of threads that it creates and manages for each CPU in the system.

The goal of the thread pool is to minimise context switches by using the same threads to execute multiple work items in succession. When that's not possible because one of its threads is busy already performing other work, it executes the work item using a different thread on a different CPU.

The Windows Server 2008 thread pool implementation makes better use of CPUs indirectly because it benefits from the completion port improvements and directly by optimising thread management so that worker threads dynamically come and go when needed to handle an application's workload. Further, the core of the infrastructure has moved to kernel mode, minimising the number of system calls made by applications that use the API. Finally, the new API makes it easier for applications to perform certain operations, such as aborting queued work units during application shutdown.

NUMA optimisations

Windows Server 2003 introduced optimisations for NUMA machines in the thread scheduler and memory manager, but Windows Server 2008 adds NUMA optimisations in the I/O manager and extends the memory manager's NUMA optimisations.

NUMA systems are typically multiprocessor systems where memory has different latency depending upon which processor accesses it (see **Figure 4**). Memory is divided into nodes, where the latencies between CPUs and nodes can vary and each CPU is considered part of a node to which it has the fastest access.

NUMA systems, especially ones with more than eight CPUs, are often more cost and performance efficient than uniform-memory access systems. While a uniform-memory access system must make memory equally available to all CPUs, a NUMA system can implement high-speed interconnections for memory directly connected to a CPU and cheaper higher-latency connections for CPUs and memory that are further apart.

To scale effectively on a NUMA system, an operating system or application must be aware of the node topology so that computation executes near the memory containing the computation's data and code. For example, the Windows scheduler assigns each thread a so-called ideal processor, which is the CPU on which the scheduler tries to always execute the thread. Doing this makes it more likely that data the thread places in the CPU's cache is going to be available to the thread every time it runs.

In Windows Server 2003, the scheduler expands this concept by considering the node containing the ideal processor to be the thread's ideal node, and it tries to schedule the thread on another CPU in the ideal node when the ideal processor is busy executing a different thread. The Windows Server 2003 memory manager also became NUMA-aware and, when possible, directs a thread's memory allocations to the memory of the node on which the thread is executing.

In Windows Server 2008, the memory manager divides the kernel's non-paged memory buffers (memory used by the kernel and device drivers to store data that is guaranteed to remain in RAM) across nodes so that allocations come from the memory on the node on which the allocation originates. System page table entries (PTEs) are allocated from the node from where the allocation originates, if the allocation requires a new page table page to satisfy it, instead of from any node, as it does in Windows Server 2003.

In Windows Server 2003, when a thread makes a memory allocation, the memory manager would prefer the node on which the thread is executing at the time of the allocation. If the thread was momentarily scheduled onto a non-ideal node, any allocations performed during that time would come from the non-ideal node. So when the thread eventually executes on its ideal node, it won't be as close as it could be to the data or code stored in the allocated memory.

To address this, the Windows Server 2008 memory manager prefers a thread's ideal node for all of a thread's allocations, even

when the thread is executing close to a different node. The memory manager also automatically understands the latencies between processors and nodes, so if the ideal node doesn't have available memory, it next checks the node closest to the ideal node. In addition, the memory manager migrates pages in its standby list to a thread's ideal node when a thread references the code or data.

Applications that want to control the locality of their allocations can use new NUMA memory APIs that allow them to specify a preferred node for memory allocations, file mapping views and file mapping objects. For an allocation related to a file mapping, the memory manager checks to see whether the mapping operation specifies a node, then checks whether the file mapping object specifies one, and finally it falls back on going to the thread's ideal node if neither do.

Prior to Windows Server 2008, the interrupt and associated deferred procedure call (DPC) for a storage or network I/O can execute on any CPU, including ones from a different node than the one on which the I/O initiated, potentially causing data read or written in the I/O operation to be in a different node's memory than the one where the data is accessed.

To avoid this, the Windows Server 2008 I/O system directs DPC execution to a CPU in the node that initiated the I/O, and systems that have devices that support PCI bus MSI-X (an extension to the Message Signaled Interrupt standard) can further localise I/O completion by using device drivers that take advantage of Windows Server 2008 APIs to direct an I/O's interrupt to the processor that initiated the I/O.

Dynamic partitioning

One of the ways a system can be more scalable is for it to support the dynamic addition of hardware resources such as CPUs and memory. This same support can also make the system more available when those resources can be replaced without requiring a system reboot. Windows Server 2003 included hot-memory add, allowing servers with dynamic memory support to use RAM as an administrator adds it. Windows Server 2008 extends dynamic memory support by allowing memory to be replaced.

RAM that becomes more reliant on Error Correcting Code (ECC) corrections is at risk of failing altogether, so on a server with hot-replace support, Windows Server 2008 can transparently migrate data off failing memory banks and onto replacements. It does so by migrating data that's under control of the operating system first, then effectively shutting down hardware devices by moving them into a low-power state, migrating the rest of the memory's data, then re-powering devices to continue normal operation.

Windows Server 2008 also supports hot addition and hot replacement of processors. For a hot replacement, the hardware must support the concept of spare CPUs, which can be either brought online or added dynamically when an existing CPU generates failure indications, something currently only available in high-end systems. The Windows Server 2008 scheduler slows activity on the failing CPU and migrates work to the replacement, after which the failing CPU can be removed and replaced with a new spare.

Windows Server 2008 support for hot processor addition allows an administrator to upgrade a server's processing capabilities without downtime. However, the scheduler and I/O systems will only make a new CPU available to device drivers and applications that request notification of CPU arrival via new APIs because some applications build in the assumption that the number of CPUs is fixed for a boot session. For instance, an application might allocate an array of work queues corresponding to each CPU, where a thread uses the queue associated with the CPU on which

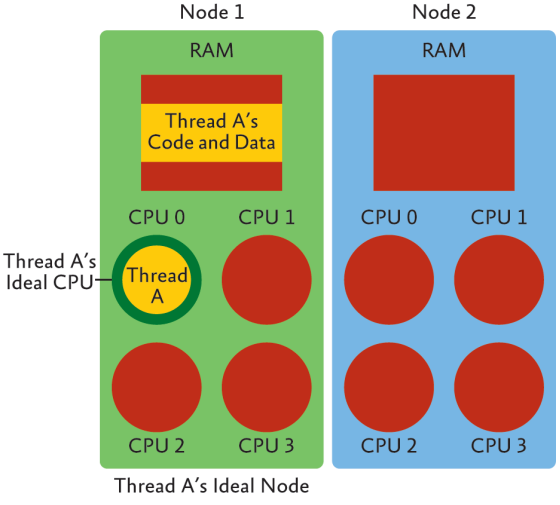

Figure 4 **Example NUMA system**

it's executing. If the scheduler put one of the application's threads on a new CPU, it would try to reference a non-existent queue, potentially corrupting the application's data and most likely crashing the application.

Microsoft server-based applications like SQL Server and Exchange Server are CPU addition capable, as are several core Windows processes, including the System process, Session Manager (%SystemRoot%\System32\Smss.exe,) and Generic Service Hosting processes (%Systemroot%\System32\Svchost.exe). Other processes can also request notification of new CPU arrival using a Windows API. When a new CPU arrives, Windows notifies device drivers of the impending arrival, starts the CPU and then notifies device drivers and applications written to take advantage of new CPUs so that they allocate data structures to track activity on the new CPU, if necessary.

Machine virtualisation

Prior to Windows Server 2008, Microsoft virtualisation products, including Virtual Server 2005, have been implemented using hosted virtualisation, as shown in **Figure 5**. In hosted virtualisation, virtual machines are implemented by a Virtual Machine Monitor (VMM) that runs alongside a host operating system, typically as a device driver. The VMM relies on the host operating system's resource management and device drivers, and when the host operating system schedules it to execute, it time-slices the CPU among active virtual machines (VMs).

Hyper-V, previously code-named 'Viridian', is implemented using hypervisor virtualisation. The hypervisor is in full control of all hardware resources, and even the Windows Server 2008 operating system that boots the system and through which you control VMs is essentially running in a virtual machine, as seen in **Figure 6**.

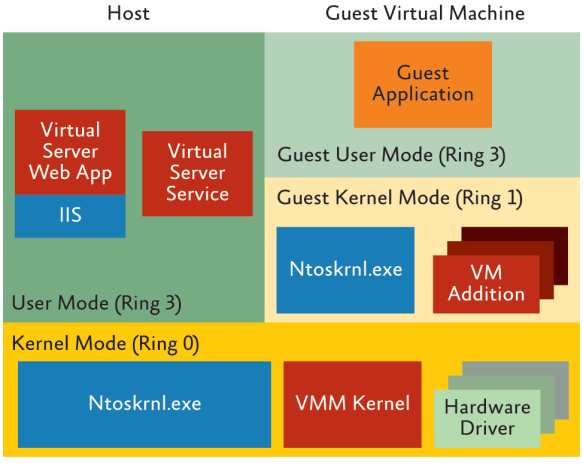

Figure 5
Hosted machine virtualisation

The hypervisor can partition the system into multiple VMs and treats the booting instance of Windows Server 2008 as the master, or root, partition, allowing it direct access to hardware devices such as the disk, networking adaptors and graphics processor. The hypervisor expects the root to perform power management and respond to hardware plug and play events. It intercepts hardware device I/O initiated in a child partition and routes it into the root, which uses standard Windows Server 2008 device drivers to perform hardware access. In this way, servers running Hyper-V can take full advantage of Windows support for hardware devices.

When you configure Windows Server 2008 with the Hyper-V server role, Windows sets the hypervisorimagelaunchtypeboot Boot Configuration Database (BCD) setting to auto and configures the Hvboot.sys device driver to start early in the boot process. If the option is configured, Hvboot.sys prepares the system for virtualisation and then loads either %Systemroot%\System32\Hvax64.exe or %Systemroot%\System32\Hvix64.exe into memory, depending on whether the system implements AMD-V or Intel VT CPU virtualisation extensions, respectively.

Once loaded, the hypervisor uses the virtualisation extensions to insert itself underneath Windows Server 2008. User-mode applications use the x64 processor's Ring 3 privilege level and kernel-mode code runs at Ring 0, so the hypervisor operates at conceptual privilege level Ring minus 1, since it can control the execution environment of code running in Ring 0.

When you use the Hyper-V management console to create or start a child partition, it communicates with the hypervisor using the %Systemroot%\System32\Drivers\Winhv.sys driver, which uses the publicly documented hypercall API to direct the hypervisor to create a new partition of specified physical-memory size and execution characteristics. The VM Service (%Systemroot%\System32\Vmms.exe) within the root is what creates

a VM Worker Process (%Systemroot%\System32\Vmwp.exe) for each child partition to manage the state of the child.

One way Windows improves the performance of child VM operating systems is that both Windows Server 2008 and Windows Vista implement enlightenments, which are code sequences that activate only when the operating system is running on a hypervisor that implements the Microsoft hypercall API. By directly requesting services of the hypervisor, the child VM avoids virtualisation code overhead that would result if the hypervisor had to guess the intent of the child operating system.

For example, a guest operating system that does not implement enlightenments for spinlocks, which execute low-level multiprocessor synchronisation, would simply spin in a tight loop waiting for a spinlock to be released by another virtual processor. The spinning might tie up one of the hardware CPUs until the hypervisor scheduled the second virtual processor. On enlightened operating systems, the spinlock code notifies the hypervisor via a hypercall when it would otherwise spin so that the hypervisor can immediately schedule another virtual processor and reduce wasted CPU usage.

Another way Windows Server 2008 improves VM performance is to accelerate VM access to devices. Performance is enhanced by installing a collection of components, collectively called the 'VM integration components', into the child operating system.

If you run a VM without installing integration components, the child operating system configures hardware device drivers for the emulated devices that hypervisor presents to it. The hypervisor must intervene when a device driver tries to touch a hardware resource in order to inform the Root partition, which performs device I/O using standard Windows device drivers on behalf of the child VM's operating system. Since a single high-level I/O operation, such as a read from a disk, may involve many discrete hardware accesses, it can cause many transitions, called intercepts, into the hypervisor and the root partition.

Windows Server 2008 minimises intercepts with three components: the Virtual Machine Bus Driver (%Systemroot%\System32\Drivers\Vmbus.sys), Virtual Service Clients (VSCs) and Virtual Service Providers (VSPs). When you install integration components into a VM with a supported operating system, VSCs take over the role of device drivers and use the services of the Vmbus.sys driver in the child VM to send high-level I/O requests to the Virtual Machine Bus Driver in the Root partition via the hypercall and memory-sharing services of the hypervisor. In the root partition, Vmbus.sys forwards the request to the corresponding VSP, which then initiates standard Windows I/O requests via the root's device drivers.

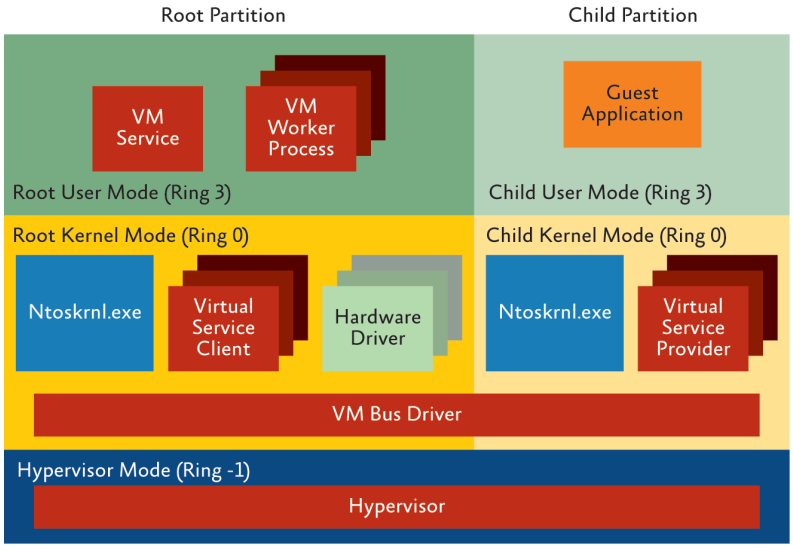

Figure 6 **Hyper-V architecture**

As you can see, Windows Server 2008 plays a key role in the Microsoft machine virtualisation strategy with its introduction of Hyper-V hypervisor-based virtualisation. Look for more information on these and other features in the next edition of my book, *Microsoft Windows Internals*, scheduled for publication later this year. ■

Mark Russinovich *is a Technical Fellow at Microsoft in the Platform and Services Division. He's coauthor of* Microsoft Windows Internals *(Microsoft Press, 2004) and a frequent speaker at IT and developer conferences, including Microsoft TechEd and the Professional Developer's Conference. Mark joined Microsoft with the acquisition of the company he co-founded, Winternals Software. He also created Sysinternals, where he published many popular utilities, including Process Explorer, Filemon and Regmon.*