At a glance:

Thread priority and scheduling

File-based symbolic links

Cancelling I/O operations

Inside the Windows Vista kernel: Part 1

Mark Russinovich

This is the first part of a series on what's new in the Windows Vista kernel. In this issue, I'll look at changes in the areas of processes and threads, and in I/O. Future instalments will cover memory management, startup and shutdown, reliability and recovery, and security.

The scope of this article comprises changes to the Windows Vista kernel only, specifically Ntoskrnl.exe and its closely associated components. Please remember that there are many other significant changes in Windows Vista that fall outside the kernel proper and therefore won't be covered. This includes improvements to the shell (such as integrated desktop search), networking (like the new IPv6 stack and two-way firewall), and the next-generation graphics model (such as Aero™ Glass, Windows Presentation Foundation, the Desktop Window Manager, and the new graphics driver model). Also not covered are the new Windows User-Mode and Kernel-Mode Driver Frameworks (UMDF and KMDF) since these are back-level installable on earlier versions of Windows.

CPU cycle counting

Windows Vista includes a number of enhancements in the area of processes and threads that include use of the CPU cycle counter for fairer CPU allocation and the new Multimedia Class Scheduler Service

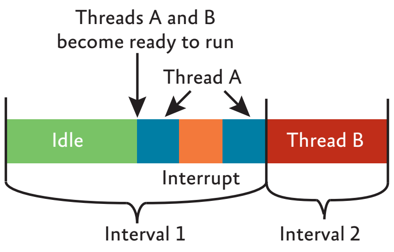


Figure 1 **Unfair thread scheduling**

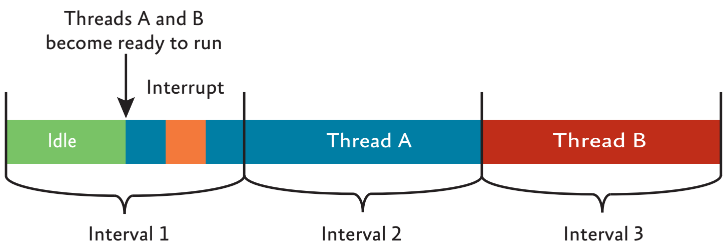


Figure 2 **Windows Vista cycle-based scheduling**

(MMCSS) that helps media applications deliver glitch-free playback.

All versions of Windows NT® up to and including Windows Vista program an interval-timer interrupt routine to execute approximately every 10 or 15 ms (milliseconds), depending on the hardware platform. The routine looks at what thread it interrupted and updates the thread's CPU usage statistics as if that thread had run for the entire interval, while in reality the thread might have started executing just before the interval's end. Further, the thread might have been technically assigned the CPU, but didn't get a chance to run because hardware and software interrupt routines executed instead.

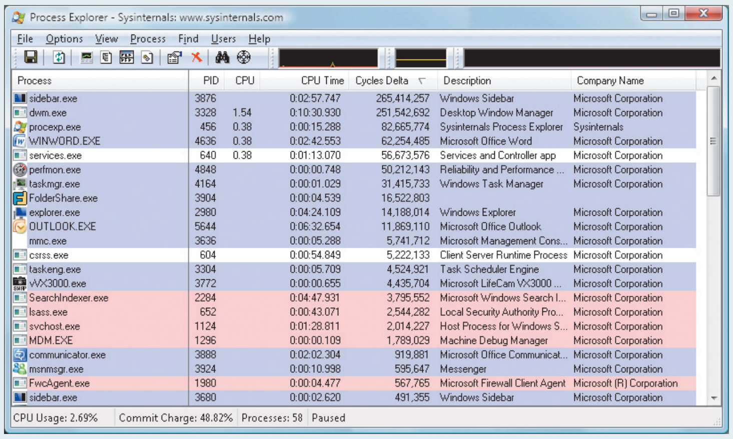
While clock-based time accounting might be OK for diagnostic tools that report thread and process CPU usage, use of that method by the thread scheduler can cause unfair CPU allocation. By default, on client versions of Windows threads are permitted to run up to 2 clock ticks (6 if in the foreground). However, the thread might get virtually no time on the CPU or up to 6 ticks (18 if in the foreground), depending on its behaviour and other activity on the system.

Figure 1 shows the unfairness that can occur when two threads that have the same priority become ready to run at the same time. Thread A runs until the next time-slice interval expiration when the scheduler assumes it has run for the entire interval and so decides that Thread A's turn is finished. Furthermore, Thread A gets unfairly charged for the interrupt that occurred during its turn. At the next interval, the scheduler picks Thread B to take over and it runs for a full interval.

In Windows Vista, the scheduler uses the cycle counter register of modern processors to track precisely how many CPU cycles a thread executes. By estimating how many cycles the CPU can execute in a clock interval, it can more accurately dole out turns on the CPU. In addition, the Windows Vista scheduler does not count interrupt execution against a thread's turn. This means that on Windows Vista a thread will always get at least its turn on the CPU and never more than an extra clock interval of execution, resulting in greater fairness and more deterministic app behaviour. **Figure 2** shows how Windows Vista responds to the scenario shown in **Figure 1** by giving both threads at least one time slice interval of execution.

Watching process CPU usage

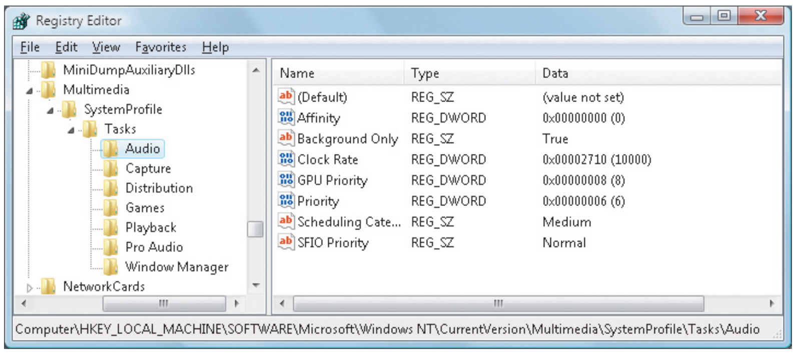
You can see the inaccuracy of the Windows standard clock-based time accounting using the Process Explorer utility from Sysinternals (microsoft.com/technet/sysinternals). Run Process Explorer on a Windows Vista system and add the Cycles Delta column to the process view. Cycles Delta shows the number of cycles the threads of each process execute between Process Explorer updates. Because CPU time accounting is still based on the interval timer, if you also add the CPU Time column, then you'll see many processes that have threads consuming millions of CPU cycles and yet don't have their CPU time updated and don't show up in the CPU usage column.


Process Explorer - Sysinternals: www.sysinternals.com
File Options View Process Find Users Help

Process	PID	CPU	CPU Time	Cycles Delta	Description	Company Name
sidebar.exe	3876		0:02:57.747	265,414,257	Windows Sidebar	Microsoft Corporation
dwm.exe	3328	1.54	0:10:30.930	251,542,692	Desktop Window Manager	Microsoft Corporation
procexp.exe	456	0.38	0:00:15.288	82,665,774	Sysinternals Process Explorer	Sysinternals
WINWORD.EXE	4636	0.38	0:02:42.553	62,254,485	Microsoft Office Word	Microsoft Corporation
services.exe	640	0.38	0:01:13.070	56,673,576	Services and Controller app	Microsoft Corporation
perfmon.exe	4848		0:00:00.748	50,212,143	Reliability and Performance ...	Microsoft Corporation
taskmgr.exe	4164		0:00:01.029	31,415,733	Windows Task Manager	Microsoft Corporation
FolderShare.exe	3904		0:00:04.539	16,522,803		
explorer.exe	2980		0:04:24.109	14,188,014	Windows Explorer	Microsoft Corporation
OUTLOOK.EXE	5644		0:06:32.654	11,869,110	Microsoft Office Outlook	Microsoft Corporation
mmc.exe	3636		0:00:05.288	5,741,712	Microsoft Management Cons...	Microsoft Corporation
csrss.exe	604		0:00:54.849	5,222,133	Client Server Runtime Process	Microsoft Corporation
taskeng.exe	3304		0:00:05.709	4,524,921	Task Scheduler Engine	Microsoft Corporation
vVX3000.exe	3772		0:00:00.655	4,435,704	Microsoft LifeCam VX3000 ...	Microsoft Corporation
SearchIndexer.exe	2284		0:04:47.931	3,795,552	Microsoft Windows Search I...	Microsoft Corporation
lsass.exe	652		0:00:43.071	2,544,282	Local Security Authority Pro...	Microsoft Corporation
svchost.exe	1124		0:01:28.811	2,014,227	Host Process for Windows S...	Microsoft Corporation
MDM.EXE	1296		0:00:00.109	1,789,029	Machine Debug Manager	Microsoft Corporation
communicator.exe	3888		0:02:02.304	919,881	Microsoft Office Communicat...	Microsoft Corporation
msnmsgr.exe	3924		0:00:10.998	595,647	Messenger	Microsoft Corporation
FwcAgent.exe	1980		0:00:04.477	567,765	Microsoft Firewall Client Agent	Microsoft (R) Corporation
sidebar.exe	3680		0:00:02.620	491,355	Windows Sidebar	Microsoft Corporation

CPU Usage: 2.69% | Commit Charge: 48.82% | Processes: 58 | Paused

Figure A **Viewing CPU time and Cycles Delta in Process Explorer**



```
MiniDumpAuxiliaryDlls
Multimedia
  SystemProfile
    Tasks
      Audio
      Capture
      Distribution
      Games
      Playback
      Pro Audio
      Window Manager
NetworkCards
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Affinity	REG_DWORD	0x00000000 (0)
Background Only	REG_SZ	True
Clock Rate	REG_DWORD	0x00002710 (10000)
GPU Priority	REG_DWORD	0x00000008 (8)
Priority	REG_DWORD	0x00000006 (6)
Scheduling Cate...	REG_SZ	Medium
SFIO Priority	REG_SZ	Normal

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Audio
```

Figure 3 **Multimedia Class Scheduler audio task definition**

The 'Watching process CPU usage' sidebar illustrates how you can monitor process CPU cycle usage for yourself using the Process Explorer utility.

Multimedia Class Scheduler Service

Users expect multimedia applications, including music and video players, to offer a seamless playback experience. However, demand for the CPU by other concurrently running applications, like antivirus, content indexing, or even the mail client, can result in unpleasant hiccups. To provide a better playback experience, Windows Vista introduces MMCSS to manage the CPU priorities of multimedia threads.

A multimedia app like Windows Media® Player 11 registers with MMCSS using new APIs that indicate its multimedia characteristics, which must match one of those listed by name under the following registry key:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\
Currentversion\Multimedia\SystemProfile\Tasks
```

The various task keys specify how much preference threads associated with different multimedia types get for CPU and graphics processor resources (though graphics processor resource management is not implemented in Windows Vista). **Figure 3** shows the contents of one of the task registry keys after a clean Windows Vista installation, though third-party developers can add their own task definitions.

Figure 4 **MMCSS thread priorities**

Scheduling Category	Boosted thread priority
High	23-26
Medium	16-23

MMCSS, which is implemented in %SystemRoot%\System32\Mmcss.dll and runs in a Service Host (Svchost.exe) process, has a priority-management thread that runs at priority 27. (Thread priorities in Windows range from 0 to 31.) This thread boosts the priority of registered multimedia threads into the range associated with the Scheduling Category value of their task's registry key as listed in **Figure 4**. In Windows, thread priorities 16 and higher are in the real-time priority range and higher than all other threads on a system (with the exception of the kernel's Memory Manager worker threads, which run at priorities 28 and 29). Only administrative accounts, like the Local System account in which MMCSS executes, have the Increase Priority privilege that's required to set real-time thread priorities.

When you play an audio file, Windows Media Player registers Audio task threads, and when you play a video, it registers Playback task threads. The MMCSS service boosts all threads that have indicated that they are delivering a stream at the same time when they are running in the process that owns the foreground window and when they have the BackgroundOnly value set to True in their task's definition key.

But while MMCSS wants to help multimedia threads get the CPU time they need, it also wants to ensure that other threads get at least some CPU time so that the system and other applications remain responsive. MMCSS therefore reserves a percentage of CPU time for other activity, specified in the following registry value:

```
HKLM\Software\Microsoft\Windows NT\Currentversion\
Multimedia\SystemProfile\SystemResponsiveness
```

By default, this is 20 percent; MMCSS monitors CPU usage to ensure that multimedia threads aren't boosted for more than 8 ms over a 10 ms period if other threads want the CPU. To get the multimedia threads out of the way for the remaining 2 ms, the scheduler drops their priorities into the 1-7 range.

You can see how MMCSS boosts thread priority by reading the 'Watching MMCSS priority boosting' sidebar.

File-based symbolic links

The Windows Vista I/O-related changes include file-based symbolic links, more efficient I/O completion processing, comprehensive support for I/O cancellation and prioritised I/O.

A file system feature many have considered missing from NTFS, the symbolic file link (or as it's called in UNIX, the soft link) finally arrives in Windows Vista. The Windows 2000 version of NTFS introduced symbolic directory links, called directory junctions, which allow you to create a directory that points at a different directory, but until the Windows Vista version, NTFS has only supported hard links for files.

A major difference in the way Windows resolves symbolic links and directory junctions is where the processing takes place. Windows processes symbolic links on the local system, even when they reference a location on a remote file server. Windows processes directory junctions that reference a remote file server on the server itself. Symbolic links on a server can therefore refer to locations that are only accessible from a client, like other client volumes, whereas directory junctions cannot. To address this, Windows Vista supports the new symbolic link type for both files and directories.

Many file system commands have been updated to understand the implications of symbolic links. For example, the Delete command knows not to follow links, which would result in deletion of the target, but to delete the link instead. However, because not all applications may handle symbolic links correctly, creating a symbolic link requires the new Create Symbolic Link privilege that only administrators have by default.

You can create a symbolic link from a command prompt with the Mklink command. The command prompt's built-in directory command identifies a symbolic link by flagging it with <SYMLINK> and showing you the target in brackets, as shown in **Figure 5**. Windows Explorer is also symbolic-link-aware and shows them with the short-cut arrow. You can see the target of a link in Explorer by adding the Link Target column to the browsing window.

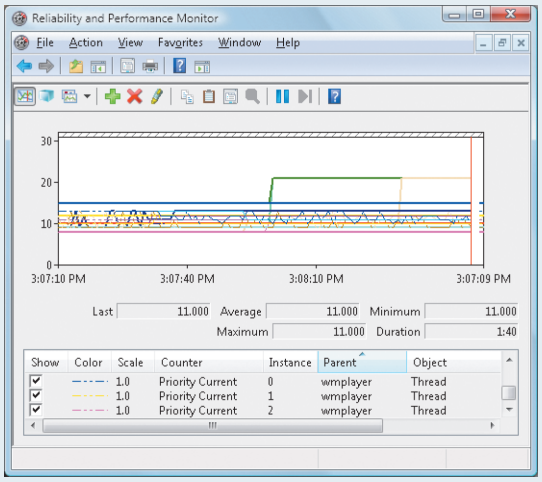
I/O completion and cancellation

There are a number of under-the-hood changes to the I/O system that can improve the performance of server applications. These applications commonly use a synchronisation object called a completion port to wait for the completion of asynchronous I/O requests. Prior to Windows Vista, when such an I/O completed, the thread that issued the I/O would execute I/O completion work, causing a switch to the process the thread belongs to and interrupting whatever else was going on. Then the I/O system would update the completion port status to wake up a thread waiting for it to change.

On Windows Vista, the I/O completion processing is performed not necessarily by the thread that issued the I/O, but instead by the one that is waiting for the comple-

Watching MMCSS priority boosting

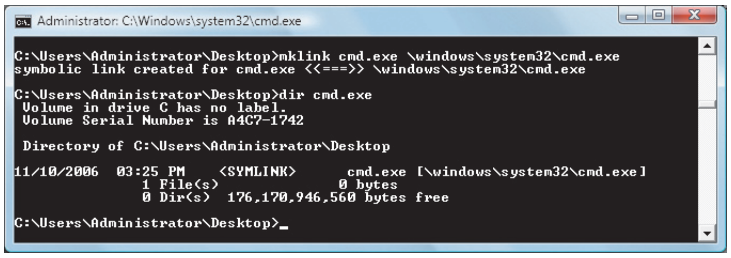
You can witness the thread boosting that the MMCSS service applies to Windows Media Player threads by playing a video or audio clip, running the Performance Monitor, setting the graph scale to 31 (the highest Windows thread priority), and adding the Priority Current counter for all instances of the Windows Media Player (Wmplayer.exe) thread objects to the display. One or more threads will run at priority 21.


Reliability and Performance Monitor
File Action View Favorites Window Help
30
20
10
0
3:07:10 PM
3:07:40 PM
3:08:10 PM
3:07:09 PM

Last 11.000 Average 11.000 Minimum 11.000
Maximum 11.000 Duration 1:40

Show	Color	Scale	Counter	Instance	Parent	Object
✓	- - -	1.0	Priority Current	0	wmplayer	Thread
✓	- - -	1.0	Priority Current	1	wmplayer	Thread
✓	- - -	1.0	Priority Current	2	wmplayer	Thread

Figure B **Thread priority boosting for Windows Media Player**



```
C:\Users\Administrator\Desktop>mklink cmd.exe \windows\system32\cmd.exe
symbolic link created for cmd.exe <<===>> \windows\system32\cmd.exe

C:\Users\Administrator\Desktop>dir cmd.exe
 Volume in drive C has no label.
 Volume Serial Number is A4C7-1742

 Directory of C:\Users\Administrator\Desktop

11/10/2006  03:25 PM    <SYMLINK>      cmd.exe [\windows\system32\cmd.exe]
               1 File(s)              0 bytes
               0 Dir(s)  176,170,946,560 bytes free

C:\Users\Administrator\Desktop>_
```

Figure 5 **Using Mklink to create a symbolic link**

tion port to wake it up. This relatively minor change avoids needless thread scheduling and context switches that can degrade the application's and the system's overall performance. To improve performance further, a server can retrieve the results of multiple I/O operations from a completion in one request, avoiding transitions to kernel mode.

Probably the most visible change in the I/O system from an end-user perspective is Windows Vista support for cancelling synchronous I/O operations. If you've ever performed a net view command or attempted to access a share to an off-line remote system using Windows XP or Windows Server 2003, you've experienced the problems with I/O operations that can't be cancelled: the command or file browser won't respond until a network timeout expires. An application has no choice but to wait until the operation fails because there's no way for it to tell the device driver executing the I/O that it doesn't care about the I/O anymore.

In Windows Vista most I/O operations can be cancelled, including the open file I/O that Net View and Explorer use. Applications have to be updated to respond to end-user requests to cancel I/O, however, and many of the Windows Vista utilities that interact with devices that have timeouts have the necessary support. The file open and save dialogs that are used by virtually every Windows application, including third-party applications, for example, now enable their Cancel button while trying to display the contents of a folder. The Net command also cancels its synchronous I/O when you press Ctrl+C.

You can see the benefits of I/O cancellation by opening a command prompt on Windows Vista and typing:

```
net view (\nonexistentmachine)
```

The command will hang while Windows tries to contact the nonexistent system, but you can type Ctrl+C to terminate it. In Windows XP, Ctrl+C has no effect and the command doesn't return until the network operation times out.

Another type of I/O that has caused users problems in past versions of Windows are those that device drivers didn't cancel properly because there was no easy way for them to know that they should. If you've ever terminated a process, but subsequently saw it lingering in process-viewing tools, then you've witnessed a device driver failing to respond to a process termination and cancelling I/O issued by the process that hadn't completed. Windows can't perform final process cleanup until all the process' I/O has either finished or been cancelled. In Windows Vista, device drivers easily register for notification of process terminations and so most of the un-killable process problems are gone.

I/O priority

While Windows has always supported prioritisation of CPU usage, it hasn't included the concept of I/O priority. Without I/O priority, background activities like search indexing, virus scanning and disk defragmenting can severely impact the responsiveness of foreground operations. A user launching an app or opening a document while another process is performing disk I/O, for example, experiences delays as the foreground task waits for disk access. The same interference also affects the streaming playback of multimedia content like songs from a hard disk.

Windows Vista introduces two new types of I/O prioritisation in order to help make foreground I/O operations get preference: priority on individual I/O operations and I/O bandwidth reservations. The Windows Vista I/O system internally includes support for five I/O priorities as shown in **Figure 6**, but only four of the priorities are used (future versions of Windows may support High).

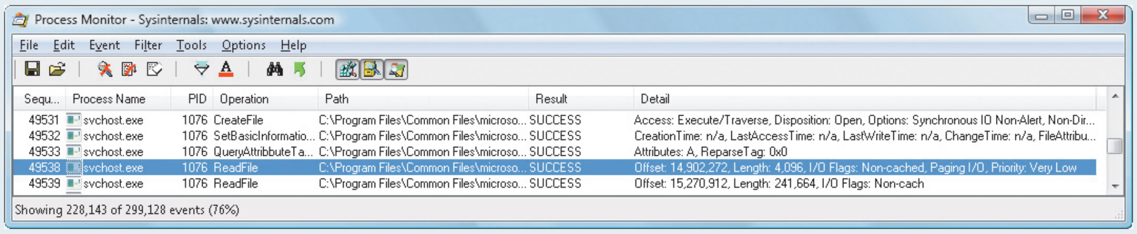
I/O has a default priority of Medium and the Memory Manager uses Critical when it wants to write dirty memory data out to disk under low memory situations to make room in RAM for other data and code. The Windows Task Scheduler sets the I/O priority for tasks that have the default task priority to Low, and the priority specified by appli-

Figure 6 **Windows Vista I/O priorities**

I/O priority	Usage
Critical	Memory Manager
High	Unused
Normal	Default priority
Low	Default task priority
Very low	Background activity

Seeing Very Low I/O priority

Process Monitor, a real-time file system and Registry monitoring utility from Sysinternals, collects detailed information for read and write file system operations, including their I/O priorities on Windows Vista. The highlighted line shows an example of a Very Low priority I/O that was issued by SuperFetch, (which I'll discuss in my next instalment).



Sequ...	Process Name	PID	Operation	Path	Result	Detail
49531	svchost.exe	1076	CreateFile	C:\Program Files\Common Files\microso...	SUCCESS	Access: Execute/Traverse, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Dir...
49532	svchost.exe	1076	SetBasicInformatio...	C:\Program Files\Common Files\microso...	SUCCESS	CreationTime: n/a, LastAccessTime: n/a, LastWriteTime: n/a, ChangeTime: n/a, FileAttribu...
49533	svchost.exe	1076	QueryAttribbuteTa...	C:\Program Files\Common Files\microso...	SUCCESS	Attributes: A, ReparseTag: 0x0
49538	svchost.exe	1076	ReadFile	C:\Program Files\Common Files\microso...	SUCCESS	Offset: 14,902,272, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Priority: Very Low
49539	svchost.exe	1076	ReadFile	C:\Program Files\Common Files\microso...	SUCCESS	Offset: 15,270,912, Length: 241,664, I/O Flags: Non-cach

Showing 228,143 of 299,128 events (76%)

Figure C **Viewing Very Low I/O priority in Process Monitor**

cations written for Windows Vista that perform background processing is Very Low. All of the Windows Vista background operations, including Windows Defender scanning and desktop search indexing, use Very Low I/O priority.

The system storage class device driver (%SystemRoot%\System32\Classpnp.sys) enforces I/O priorities and so they automatically apply to I/O directed at most storage devices. The class and other storage drivers insert Medium I/Os ahead of those that are Low and Very Low in their queues, but issue at least one waiting Low or Very Low I/O every second so that background processes can make forward progress. Data read using Very Low I/O also causes the Cache Manager to immediately write modifications to disk instead of doing it later, and to bypass its read-ahead logic for read operations that would otherwise preemptively read from the file being accessed. Take a look at the sidebar 'Seeing Very Low I/O priority' for an example of Very Low I/O priority using the Process Monitor utility.

The Windows Vista bandwidth reservation support is useful for media player applications and Windows Media Player uses it, along with MMCSS priority boosts, to deliver nearly glitch-free playback of local content. A media player application asks the I/O system to guarantee it the ability to read data at a specified rate and, if the device can deliver data at the requested rate and existing reservations allow it, it gives the app guidance as to how fast it should issue I/Os and how large the I/Os should be. The I/O system won't service other I/Os unless it can satisfy the requirements of apps that have made reservations on the target storage device.

One final change in the I/O system worth mentioning relates to the size of I/O operations. Since the first version of Windows NT, the Memory Manager and the I/O system have limited the amount of data processed by an individual storage I/O request to 64Kb. Thus, even if an application issues a much larger I/O request, it's broken into individual requests having a maximum size of 64Kb. Each I/O incurs an overhead for transitions to kernel-mode and initiating an I/O transfer on the storage device, so in Windows Vista storage I/O request sizes are no longer capped. Several Windows Vista user-mode components have been modified to take advantage of the support for larger I/Os, including Explorer's copy functionality and the command prompt's Copy command, which now issue 1Mb I/Os.

Next up

Now you've seen two areas in which the Windows Vista kernel has been enhanced. You can expect additional in-depth information in the next edition of my book, *Windows Internals* (coauthored with David Solomon), planned for release at the same time as the next version of Windows Server, code-named Longhorn. In my next instalment, I'll continue introducing you to the new kernel by discussing memory management along with system startup and shutdown. ■

Mark Russinovich *is a Technical Fellow at Microsoft in the Platform and Services Division. He is a coauthor of* Microsoft Windows Internals *(Microsoft Press, 2004) and a frequent speaker at IT and developer conferences. He joined Microsoft with the recent acquisition of the company he cofounded, Winternals Software. He also created Sysinternals, where he published the Process Explorer, Filemon and Regmon utilities.*