

Inside the Windows Vista Kernel

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 installments 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 (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 behavior 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.

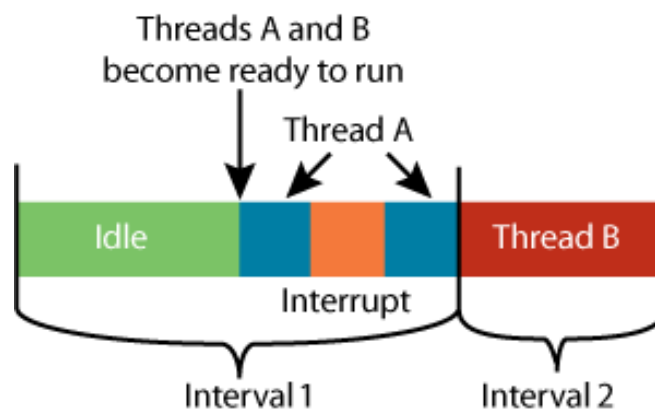


Figure 1 Unfair Thread Scheduling

Inside the Windows Vista Kernel

Mark Russinovich

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 behavior. 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.

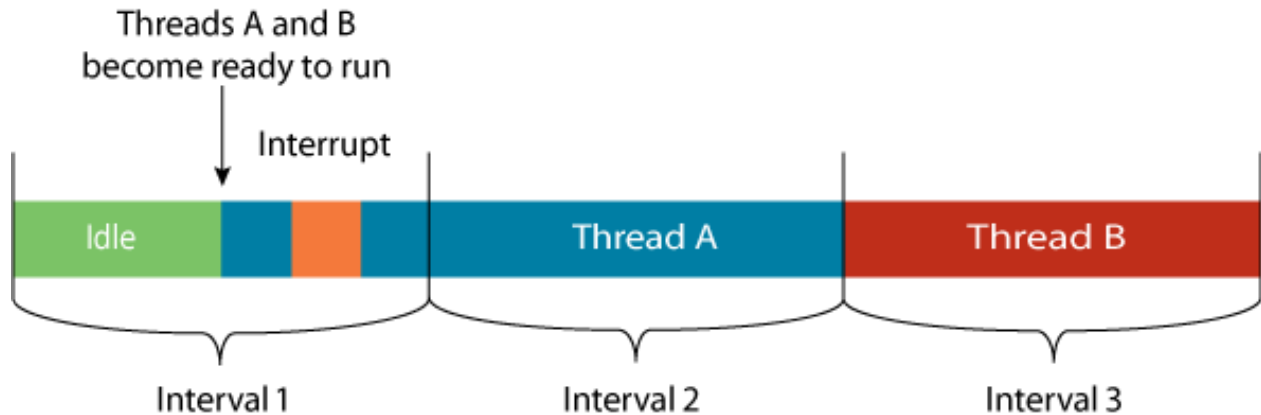


Figure 2 Windows Vista Cycle-Based Scheduling

The "Watching Process CPU Usage" sidebar illustrates how you can monitor process CPU cycle usage for yourself by 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\WindowsNT\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.

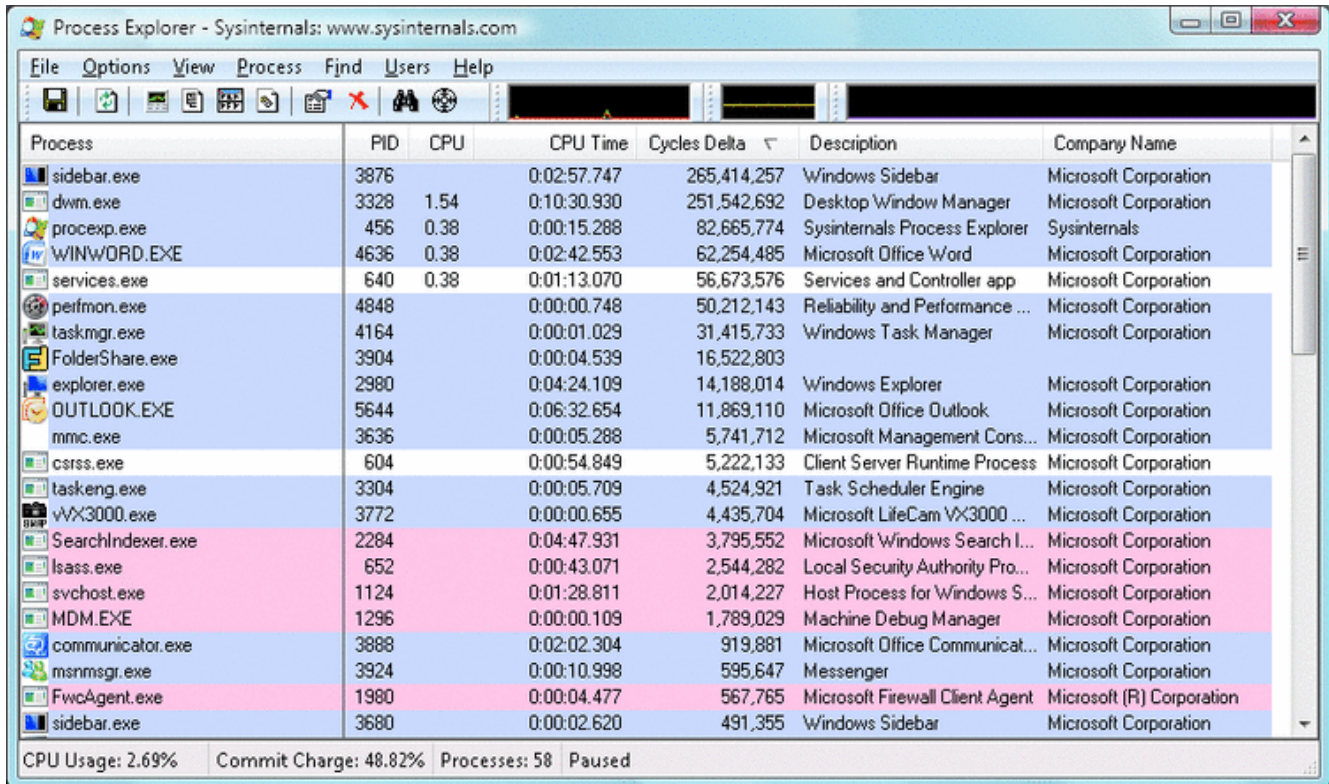
Watching Process CPU Usage

You can see the inaccuracy of the Windows standard clock-based time accounting using the Process Explorer utility from Sysinternals. Run Process Explorer on a Windows Vista system and add the

Inside the Windows Vista Kernel

Mark Russinovich

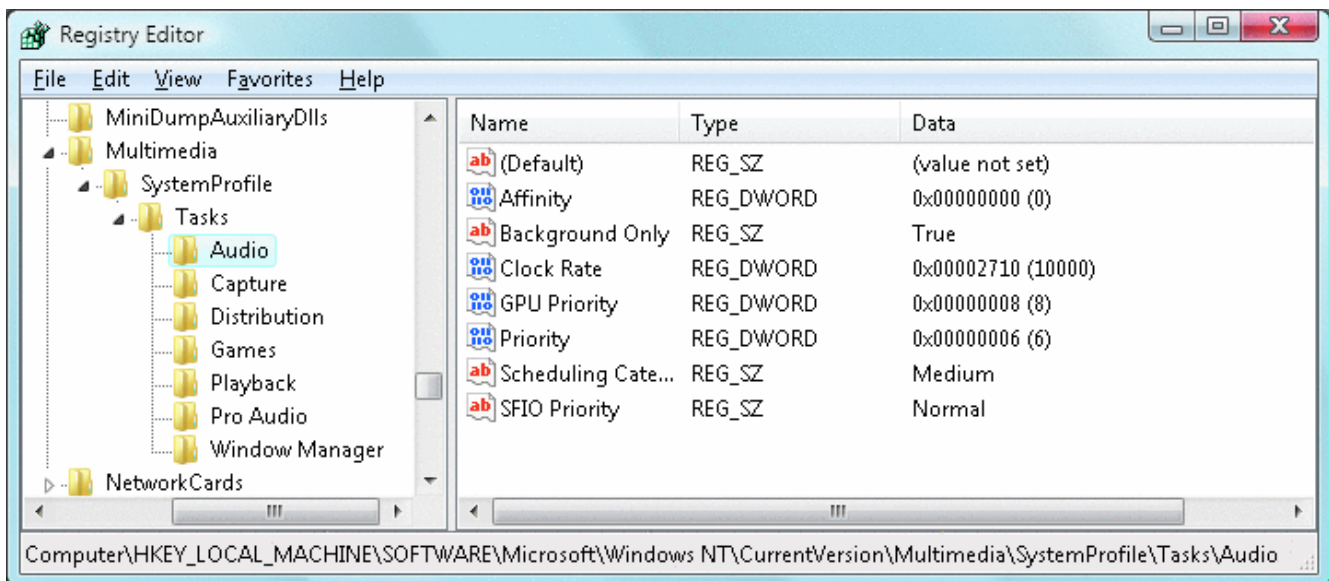
Cycles Delta column to the process view. Cycles Delta shows the number of cycles the threads of each process execute between Process Explorer updates. Because CPU time accounting is still based on the interval timer, if you also add the CPU Time column, then you'll see many processes that have threads consuming millions of CPU cycles and yet don't have their CPU time updated and don't show up in the CPU usage column.



The screenshot shows the Process Explorer window with the following columns: Process, PID, CPU, CPU Time, Cycles Delta, Description, and Company Name. The status bar at the bottom indicates CPU Usage: 2.69%, Commit Charge: 48.82%, Processes: 58, and Paused.

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
VX3000.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
FwAgent.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

Figure A Viewing CPU Time and Cycles Delta in Process Explorer



The screenshot shows the Registry Editor window with the following structure: Computer > HKEY_LOCAL_MACHINE > SOFTWARE > Microsoft > Windows NT > CurrentVersion > Multimedia > SystemProfile > Tasks > Audio. The right pane shows the following registry values:

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

Figure 3 Multimedia Class Scheduler Audio Task Definition

Inside the Windows Vista Kernel

Mark Russinovich

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.

Figure 4 MMCSS Thread Priorities

Scheduling Category	Boosted Thread Priority
High	23-26
Medium	16-23

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\WindowsNT\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 prioritized 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.

Inside the Windows Vista Kernel

Mark Russinovich

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.

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.

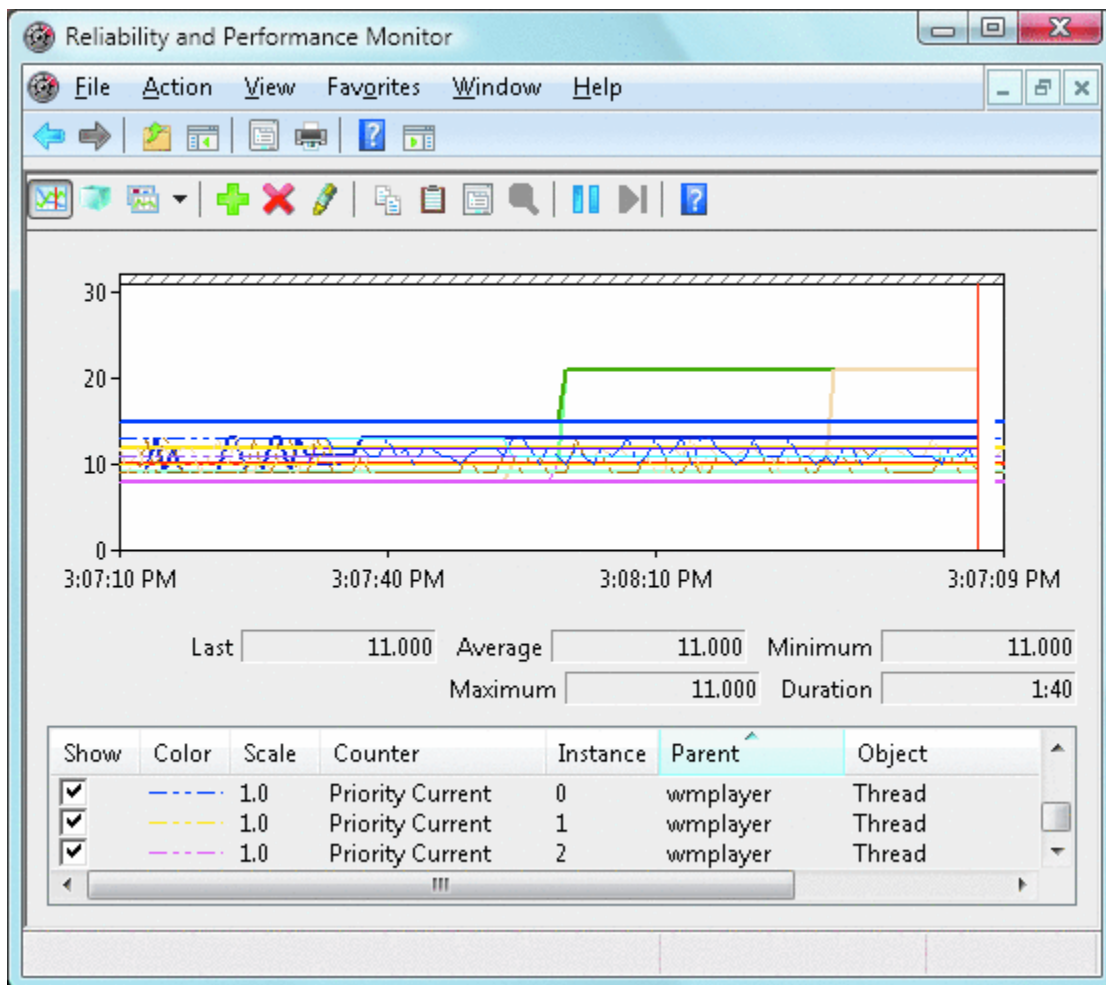
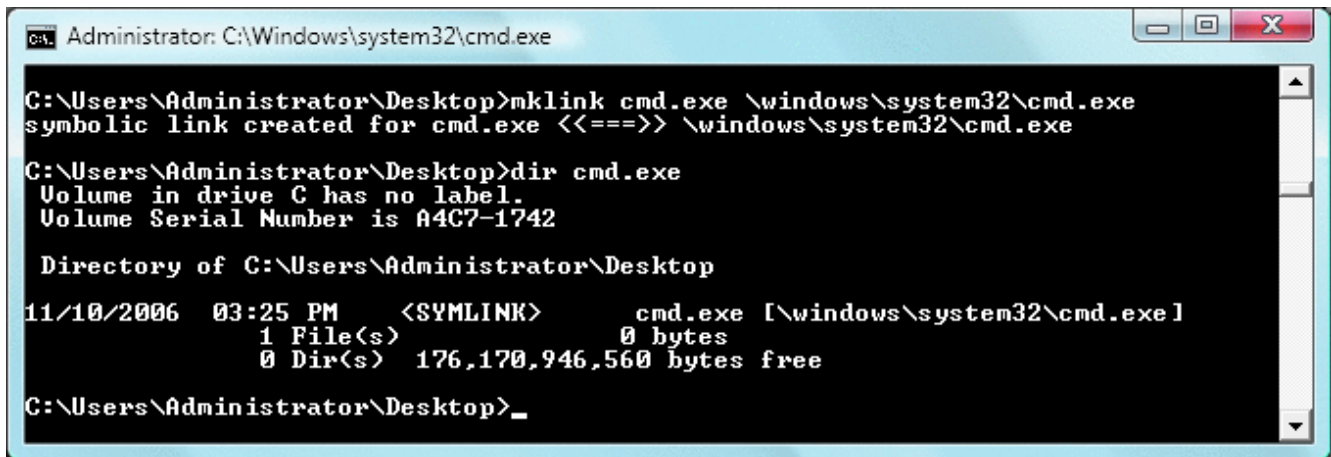


Figure B Thread Priority Boosting for Windows Media Player

Inside the Windows Vista Kernel

Mark Russinovich



```
Administrator: C:\Windows\system32\cmd.exe
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

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 synchronization 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 completion 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 canceling 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 canceled: 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 canceled, 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:

Inside the Windows Vista Kernel

Mark Russinovich

```
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 canceling 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 canceled. 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 prioritization 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 prioritization 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).

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

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 applications 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.

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 installment).

Inside the Windows Vista Kernel

Mark Russinovich

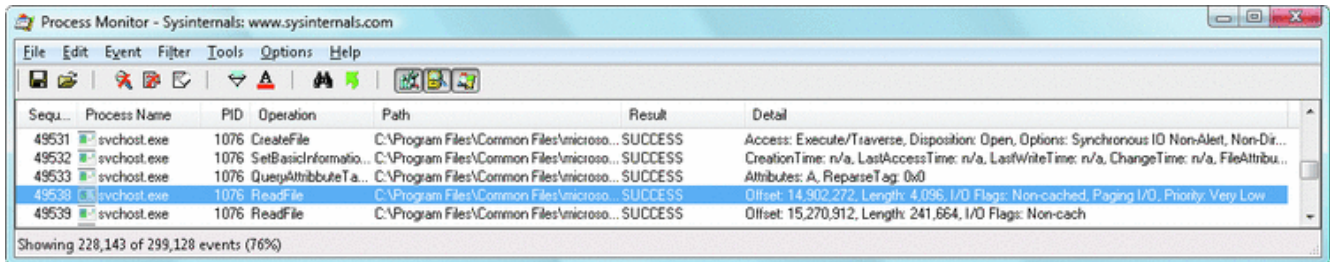


Figure C Viewing Very Low I/O Priority in Proces Monitor

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 installment, I'll continue introducing you to the new kernel by discussing memory management along with system start up and shutdown.

Last month, in the first installment of this three-part series, I looked at Windows Vista kernel enhancements in the areas of processes and I/O.

This time I'll cover advances in the way Windows Vista manages memory, as well as major improvements to system startup, shutdown, and power management (Part One).

Inside the Windows Vista Kernel

Mark Russinovich

Every release of Windows® improves scalability and performance, and Windows Vista™ is no different. The Windows Vista Memory Manager includes numerous enhancements, like more extensive use of lock-free synchronization techniques, finer-grained locking, tighter data-structure packing, larger paging I/Os, support for modern GPU memory architectures, and more efficient use of the hardware Translation Lookaside Buffer. Plus, Windows Vista memory management now offers dynamic address space allocation for the requirements of different workloads.

Four performance-enhancing features that use new technologies make their operating system debut on Windows Vista: SuperFetch, ReadyBoost, ReadyBoot, and ReadyDrive. I'll discuss them in detail later in this article.

Dynamic Kernel Address Space

Windows and the applications that run on it have bumped their heads on the address space limits of 32-bit processors. The Windows kernel is constrained by default to 2GB, or half the total 32-bit virtual address space, with the other half reserved for use by the process whose thread is currently running on the CPU. Inside its half, the kernel has to map itself, device drivers, the file system cache, kernel stacks, per-session code data structures, and both non-paged (locked-in physical memory) and paged buffers allocated by device drivers. Prior to Windows Vista, the Memory Manager determined at boot time how much of the address space to assign to these different purposes, but this inflexibility sometimes led to situations where one of the regions became full while others still had plenty of available space. The exhaustion of an area can lead to application failures and prevent device drivers from completing I/O operations.

In 32-bit Windows Vista, the Memory Manager dynamically manages the kernel's address space, allocating and deallocating space to various uses as the demands of the workload require. Thus, the amount of virtual memory used to store paged buffers can grow when device drivers ask for more, and it can shrink when the drivers release it. Windows Vista will therefore be able to handle a wider variety of workloads and likewise the 32-bit version of the forthcoming Windows Server® code-named "Longhorn," will scale to handle more concurrent Terminal Server users.

Of course, on 64-bit Windows Vista systems, address space constraints are not currently a practical limitation and therefore require no special treatment as they are configured to their maximums.

Memory Priorities

Just as Windows Vista adds I/O priorities (as I discussed in the last installment), it also implements memory priorities. Understanding how Windows uses memory priorities requires grasping how the Memory Manager implements its memory cache, called the Standby List. On all versions of Windows prior to Windows Vista, when a physical page (which is typically 4KB in size) that's owned by a process was reclaimed by the system, the Memory Manager typically placed the page at the end of the Standby List. If the process wanted to access the page again, the Memory Manager took the page from the Standby List and reassigned it to the process. When a process wanted to use a new page of physical memory and no free memory was available, the Memory Manager gave it the page at the front the Standby List. This scheme treated all pages on the standby essentially as equals, using only the time they were placed on the list to sort them.

On Windows Vista, every page of memory has a priority in the range of 0 to 7, and so the Memory Manager divides the Standby List into eight lists that each store pages of a particular priority. When the Memory Manager wants to take a page from the Standby List, it takes pages from low-priority lists first. A page's priority usually reflects that of the thread that first causes its allocation. (If the page is shared, it reflects the highest of memory priorities of the sharing threads.) A thread inherits its page-

Inside the Windows Vista Kernel

Mark Russinovich

priority value from the process to which it belongs. The Memory Manager uses low priorities for pages it reads from disk speculatively when anticipating a process's memory accesses.

By default, processes have a page-priority value of 5, but functions allow applications and the system to change process and thread page-priority values. The real power of memory priorities is realized only when the relative priorities of pages are understood at a macro-level, which is the role of SuperFetch.

SuperFetch

A significant change to the Memory Manager is in the way that it manages physical memory. The Standby List management used by previous versions of Windows has two limitations. First, the prioritization of pages relies only on the recent past behavior of processes and does not anticipate their future memory requirements. Second, the data used for prioritization is limited to the list of pages owned by a process at any given point in time. These shortcomings can result in scenarios like the "after lunch syndrome," where you leave your computer for a while and a memory-intensive system application runs (such as an antivirus scan or disk defragmentation). This application forces the code and data that your active applications had cached in memory to be overwritten by the memory-intensive activities. When you return, you experience sluggish performance as applications have to request their data and code from disk.

Windows XP introduced prefetching support that improved boot and application startup performance by performing large disk I/Os to preload memory with code and file system data that it expected, based on previous boots and application launches. Windows Vista goes a big step further with SuperFetch, a memory management scheme that enhances the least-recently accessed approach with historical information and proactive memory management.

SuperFetch is implemented in %SystemRoot%\System32\Sysmain.dll as a Windows service that runs inside a Service Host process (%SystemRoot%\System32\Svchost.exe). The scheme relies on support from the Memory Manager so that it can retrieve page usage histories as well as direct the Memory Manager to preload data and code from files on disk or from a paging file into the Standby List and assign priorities to pages. The SuperFetch service essentially extends page-tracking to data and code that was once in memory, but that the Memory Manager has reused to make room for new data and code. It stores this information in scenario files with a .db extension in the %SystemRoot%\Prefetch directory alongside standard prefetch files used to optimize application launch. Using this deep knowledge of memory usage, SuperFetch can preload data and code when physical memory becomes available.

Whenever memory becomes free—for example, when an application exits or releases memory—SuperFetch asks the Memory Manager to fetch data and code that was recently evicted. This is done at a rate of a few pages per second with Very Low priority I/Os so that the preloading does not impact the user or other active applications. Therefore, if you leave your computer to go to lunch and a memory-intensive background task causes the code and data from your active applications to be evicted from memory while you're gone, SuperFetch can often bring all or most of it back into memory before you return. SuperFetch also includes specific scenario support for hibernation, standby, Fast User Switching (FUS), and application launch. When the system hibernates, for example, SuperFetch stores data and code in the hibernation file that it expects (based on previous hibernations) will be accessed during the subsequent resume. In contrast, when you resume Windows XP, previously cached data must be reread from the disk when it is referenced.

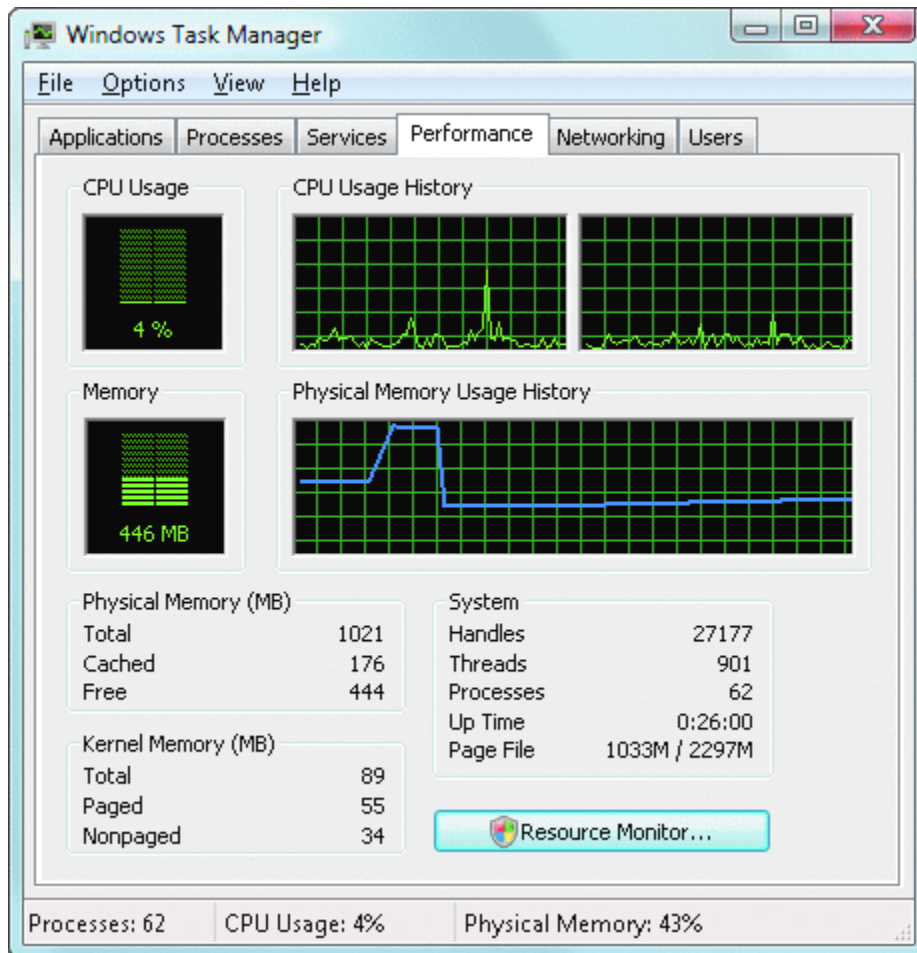
See the sidebar "Watching SuperFetch" for a glimpse of how SuperFetch impacts available memory.

Inside the Windows Vista Kernel

Mark Russinovich

Watching SuperFetch

After you've used a Windows Vista system a while, you'll see a low number for the Free Physical Memory counter on Task Manager's Performance page. That's because SuperFetch and standard Windows caching make use of all available physical memory to cache disk data. For example, when you first boot, if you immediately run Task Manager you should notice the Free Memory value decreasing as Cached Memory number rises. Or, if you run a memory-hungry program and then exit it (any of the freeware "RAM optimizers" that allocate large amounts of memory and then release the memory will work), or just copy a very large file, the Free number will rise and the Physical Memory Usage graph will drop as the system reclaims the deallocated memory. Over time, however, SuperFetch repopulates the cache with the data that was forced out of memory, so the Cached number will rise and the Free number will decline.



Watching Memory

ReadyBoost

The speed of CPUs and memory are fast outpacing that of hard disks, so disks are a common system performance bottleneck. Random disk I/O is especially expensive because disk head seek times are on the order of 10 milliseconds—an eternity for today's 3GHz processors. While RAM is ideal for caching disk data, it is relatively expensive. Flash memory, however, is generally cheaper and can service random reads up to 10 times faster than a typical hard disk. Windows Vista, therefore, includes a feature called ReadyBoost to take advantage of flash memory storage devices by creating an intermediate caching layer on them that logically sits between memory and disks.

Inside the Windows Vista Kernel

Mark Russinovich

ReadyBoost consists of a service implemented in %SystemRoot%\System32\Emdmgmt.dll that runs in a Service Host process, and a volume filter driver, %SystemRoot%\System32\Drivers\Ecache.sys. (Emd is short for External Memory Device, the working name for ReadyBoost during its development.) When you insert a flash device like a USB key into a system, the ReadyBoost service looks at the device to determine its performance characteristics and stores the results of its test in HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\Currentversion\Emdmgmt, seen in Figure 1.

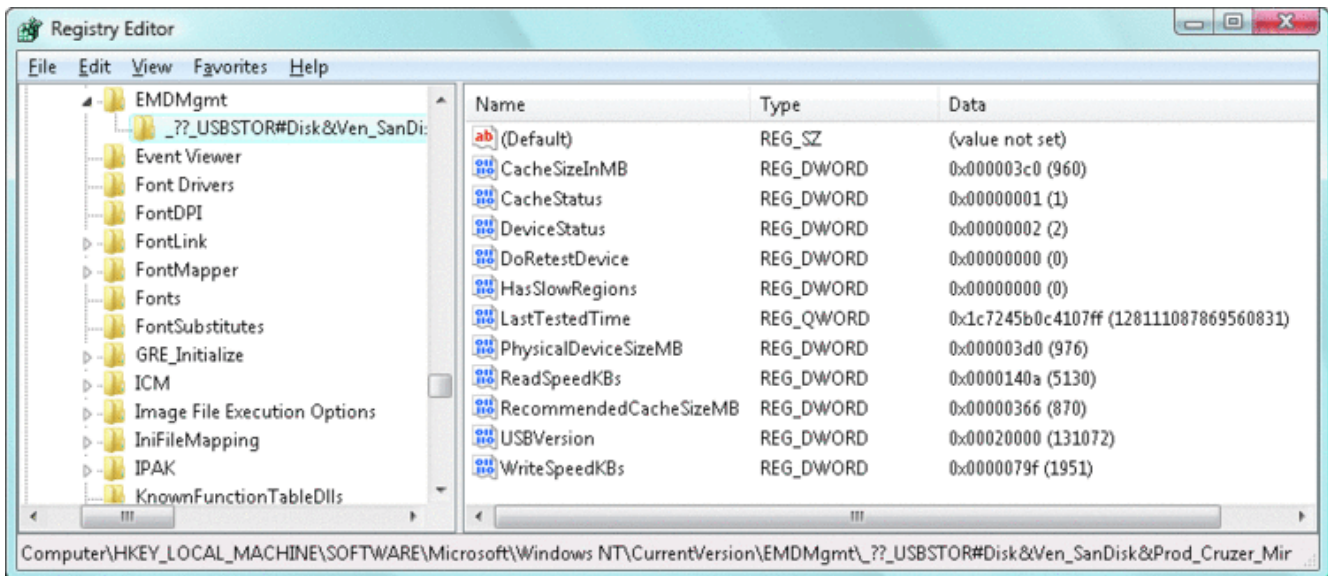


Figure 1 ReadyBoost Device Test Results In The Registry

If you aren't already using a device for caching, and the new device is between 256MB and 32GB in size, has a transfer rate of 2.5MB/s or higher for random 4KB reads, and has a transfer rate of 1.75MB/s or higher for random 512KB writes, then ReadyBoost will ask if you'd like to dedicate up to 4GB of the storage for disk caching. (Although ReadyBoost can use NTFS, it limits the maximum cache size to 4GB to accommodate FAT32 limitations.) If you agree, then the service creates a caching file named ReadyBoost.sfcache in the root of the device and asks SuperFetch to prepopulate the cache in the background.

After the ReadyBoost service initializes caching, the Ecache.sys device driver intercepts all reads and writes to local hard disk volumes (C:\, for example), and copies any data being written into the caching file that the service created. Ecache.sys compresses data and typically achieves a 2:1 compression ratio so a 4GB cache file will usually contain 8GB of data. The driver encrypts each block it writes using Advanced Encryption Standard (AES) encryption with a randomly generated per-boot session key in order to guarantee the privacy of the data in the cache if the device is removed from the system. When ReadyBoost sees random reads that can be satisfied from the cache, it services them from there, but because hard disks have better sequential read access than flash memory, it lets reads that are part of sequential access patterns go directly to the disk even if the data is in the cache.

ReadyBoot

Windows Vista uses the same boot-time prefetching as Windows XP did if the system has less than 512MB of memory, but if the system has 700MB or more of RAM, it uses an in-RAM cache to

Inside the Windows Vista Kernel

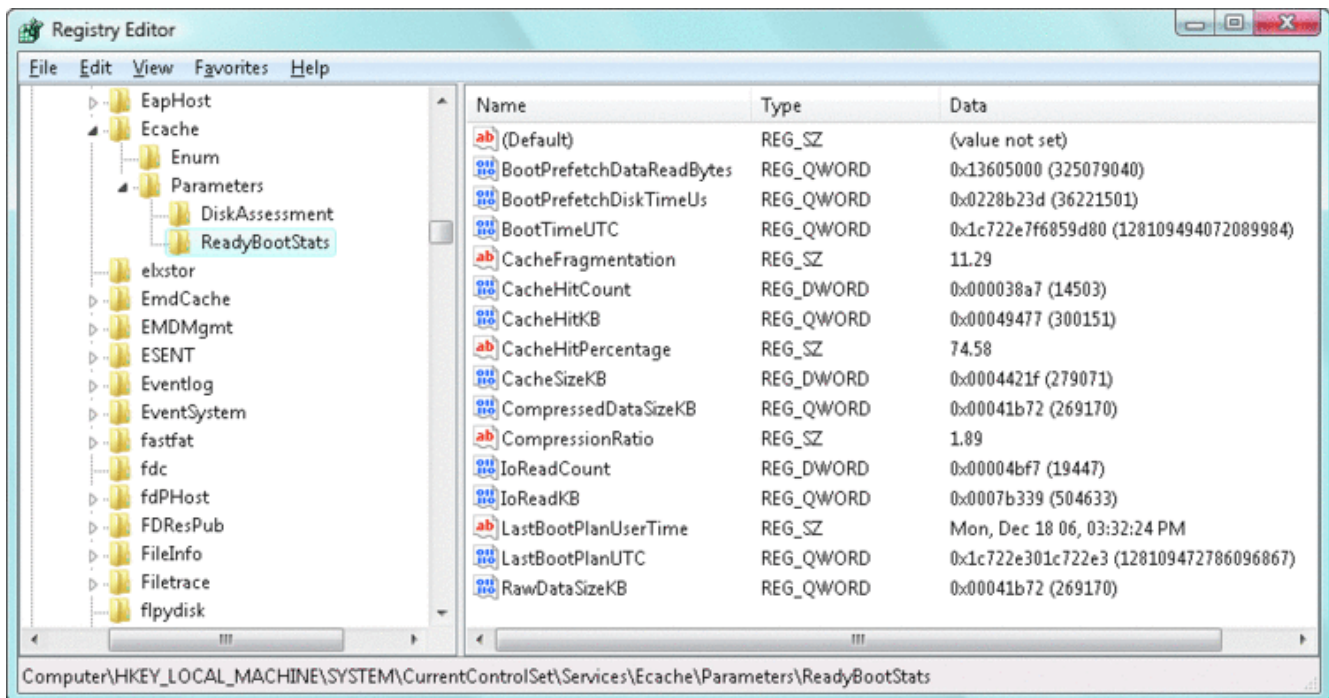
Mark Russinovich

optimize the boot process. The size of the cache depends on the total RAM available, but is large enough to create a reasonable cache and yet allow the system the memory it needs to boot smoothly. After every boot, the ReadyBoost service (the same service that implements the ReadyBoost feature just described) uses idle CPU time to calculate a boot-time caching plan for the next boot. It analyzes file trace information from the five previous boots and identifies which files were accessed and where they are located on disk. It stores the processed traces in %SystemRoot%\Prefetch\Readyboot as .fx files and saves the caching plan under:

HKLM\System\CurrentControlSet\Services\Ecache\Parameters

in REG_BINARY values named for internal disk volumes they refer to.

The cache is implemented by the same device driver that implements ReadyBoost caching (Ecache.sys), but the cache's population is guided by the ReadyBoost service as the system boots. While the boot cache is compressed like the ReadyBoost cache, another difference between ReadyBoost and ReadyBoot cache management is that while in ReadyBoot mode, other than the ReadyBoost service's updates, the cache doesn't change to reflect data that's read or written during the boot. The ReadyBoost service deletes the cache 90 seconds after the start of the boot, or if other memory demands warrant it, and records the cache's statistics in HKLM\System\CurrentControlSet\Services\Ecache\Parameters\ReadyBootStats, as shown in Figure 2. Microsoft performance tests show that ReadyBoot provides performance improvements of about 20 percent over the legacy Windows XP prefetcher.



The screenshot shows the Windows Registry Editor window. The left pane displays the tree structure with 'ReadyBootStats' selected under 'Parameters'. The right pane shows a list of registry values with their names, types, and data.

Name	Type	Data
(Default)	REG_SZ	(value not set)
BootPrefetchDataReadBytes	REG_QWORD	0x13605000 (325079040)
BootPrefetchDiskTimeUs	REG_QWORD	0x0228b23d (36221501)
BootTimeUTC	REG_QWORD	0x1c722e7f6859d80 (128109494072089984)
CacheFragmentation	REG_SZ	11.29
CacheHitCount	REG_DWORD	0x000038a7 (14503)
CacheHitKB	REG_QWORD	0x00049477 (300151)
CacheHitPercentage	REG_SZ	74.58
CacheSizeKB	REG_DWORD	0x0004421f (279071)
CompressedDataSizeKB	REG_QWORD	0x00041b72 (269170)
CompressionRatio	REG_SZ	1.89
IoReadCount	REG_DWORD	0x00004bf7 (19447)
IoReadKB	REG_QWORD	0x0007b339 (504633)
LastBootPlanUserTime	REG_SZ	Mon, Dec 18 06, 03:32:24 PM
LastBootPlanUTC	REG_QWORD	0x1c722e301c722e3 (128109472786096867)
RawDataSizeKB	REG_QWORD	0x00041b72 (269170)

Figure 2 ReadyBoot Performance Statistics

ReadyDrive

ReadyDrive is a Windows Vista feature that takes advantage of new hybrid hard disk drives called H-HDDs. An H-HDD is a disk with embedded nonvolatile flash memory (also known as NVRAM). Typical H-HDDs include between 50MB and 512MB of cache, but the Windows Vista cache limit is 2TB.

Inside the Windows Vista Kernel

Mark Russinovich

Windows Vista uses ATA-8 commands to define the disk data to be held in the flash memory. For example, Windows Vista will save boot data to the cache when the system shuts down, allowing for faster restarting. It also stores portions of hibernation file data in the cache when the system hibernates so that the subsequent resume is faster. Because the cache is enabled even when the disk is spun down, Windows can use the flash memory as a disk-write cache, which avoids spinning up the disk when the system is running on battery power. Keeping the disk spindle turned off can save much of the power consumed by the disk drive under normal usage.

Boot Configuration Database

Windows Vista has enhanced several aspects of startup and shutdown. Startup has improved with the introduction of the Boot Configuration Database (BCD) for storing system and OS startup configuration, a new flow and organization of system startup processes, new logon architecture, and support for delayed-autostart services. Windows Vista shutdown changes include pre-shutdown notification for Windows services, Windows services shutdown ordering, and a significant change to the way the OS manages power state transitions.

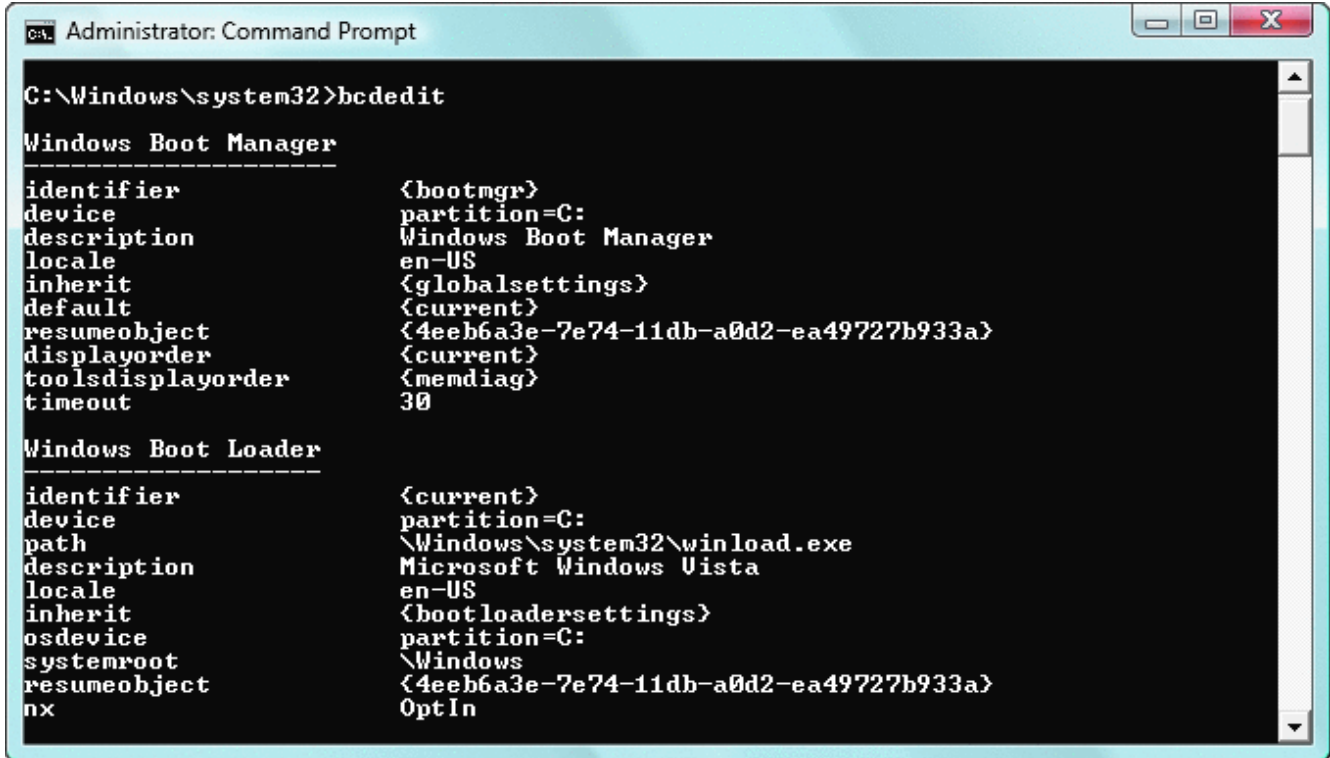
One of the most visible changes to the startup process is the absence of Boot.ini from the root of the system volume. That's because the boot configuration, which on previous versions of Windows was stored in the Boot.ini text file, is now stored in the BCD. One of the reasons Windows Vista uses the BCD is that it unifies the two current boot architectures supported by Windows: Master Boot Record (MBR) and Extensible Firmware Interface (EFI). MBR is generally used by x86 and x64 desktop systems, while EFI is used by Itanium-based systems (though desktop PCs are likely to ship with EFI support in the near future). The BCD abstracts the firmware and has other advantages over Boot.ini, like its support for Unicode strings and alternate pre-boot executables.

The BCD is actually stored on disk in a registry hive that loads into the Windows registry for access via registry APIs. On PCs, Windows stores it in \Boot\Bcd on the system volume. On EFI systems, it's on the EFI system partition. When the hive is loaded, it appears under HKLM\Bcd00000000, but its internal format is undocumented so editing it requires the use of a tool like %SystemRoot%\System32\Bcdedit.exe. Interfaces for manipulating the BCD are also made available for scripts and custom editors through Windows Management Instrumentation (WMI) and you can use the Windows System Configuration Utility (%SystemRoot%\System32\Msconfig.exe) to edit or add basic parameters, like kernel debugging options.

The BCD divides platform-wide boot settings, like the default OS selection and the boot menu timeout, from OS-specific settings such as OS boot options and the path to the OS boot loader. For example, Figure 3 shows that when you run Bcdedit with no command-line options, it displays platform settings in the Windows Boot Manager section at the top of the output, followed by OS-specific settings in the Windows Boot Loader section.

Inside the Windows Vista Kernel

Mark Russinovich



```
Administrator: Command Prompt
C:\Windows\system32>bcdedit

Windows Boot Manager
-----
identifier          {bootmgr}
device              partition=C:
description         Windows Boot Manager
locale              en-US
inherit             {globalsettings}
default             {current}
resumeobject        {4eeb6a3e-7e74-11db-a0d2-ea49727b933a}
displayorder        {current}
toolsdisplayorder  {memdiag}
timeout             30

Windows Boot Loader
-----
identifier          {current}
device              partition=C:
path                \Windows\system32\winload.exe
description         Microsoft Windows Vista
locale              en-US
inherit             {bootloadersettings}
osdevice            partition=C:
systemroot          \Windows
resumeobject        {4eeb6a3e-7e74-11db-a0d2-ea49727b933a}
nx                  OptIn
```

Figure 3 Settings displayed in BCDEdit

When you boot a Windows Vista installation, this new scheme divides the tasks that were handled by the operating system loader (Ntldr) on previous versions of Windows into two different executables: \BootMgr and %SystemRoot%\System32\Winload.exe. Bootmgr reads the BCD and displays the OS boot menu, while Winload.exe handles operating-system loading. If you're performing a clean boot, Winload.exe loads boot-start device drivers and core operating system files, including Ntoskrnl.exe, and transfers control to the operating system; if the system is resuming from hibernation, then it executes %SystemRoot%\System32\Winresume.exe to load the hibernation data into memory and resume the OS.

Bootmgr also includes support for additional pre-boot executables. Windows Vista comes with the Windows Memory Diagnostic (\Boot\Memtest.exe) pre-configured as an option for checking the health of RAM, but third parties can add their own pre-boot executables as options that will display in Bootmgr's boot menu.

Startup Processes

In previous versions of Windows, the relationship between various system processes was unintuitive. For example, as the system boots, the interactive logon manager (%SystemRoot%\System32\Winlogon.exe) launches the Local Security Authority Subsystem Service (Lsass.exe) and the Service Control Manager (Services.exe). Further, Windows uses a namespace container called a Session to isolate processes running in different logon sessions. But prior to Windows Vista, the user logged into the console shared Session 0, the session used by system processes, which created potential security issues. One such issue was introduced, for example, when a poorly written Windows service running in Session 0 displayed a user interface on the interactive console, allowing malware to attack the window through techniques like shatter attacks and possibly gain administrative privileges.

Inside the Windows Vista Kernel

Mark Russinovich

To address these problems, several system processes were re-architected for Windows Vista. Session Manager (Smss.exe) is the first user-mode process created during the boot as in previous versions of Windows, but on Windows Vista the Session Manager launches a second instance of itself to configure Session 0, which is dedicated solely to system processes. The Session Manager process for Session 0 launches the Windows Startup Application (Wininit.exe), a Windows subsystem process (Csrss.exe) for Session 0, and then it exits. The Windows Startup Application continues by starting the Service Control Manager, the Local Security Authority Subsystem, and a new process, Local Session Manager (Lsm.exe), which manages terminal server connections for the machine.

When a user logs onto the system, the initial Session Manager creates a new instance of itself to configure the new session. The new Smss.exe process starts a Windows subsystem process and Winlogon process for the new session. Having the primary Session Manager use copies of itself to initialize new sessions doesn't offer any advantages on a client system, but on Windows Server "Longhorn" systems acting as terminal servers, multiple copies can run concurrently to allow for faster logon of multiple users.

With this new architecture, system processes, including Windows services, are isolated in Session 0. If a Windows service, which runs in Session 0, displays a user interface, the Interactive Services Detection service (%SystemRoot%\System32\UI0Detect.exe) notifies any logged-on administrator by launching an instance of itself in the user's security context and displaying the message shown in Figure 4. If the user selects the "Show me the message" button, the service switches the desktop to the Windows service desktop, where the user can interact with the service's user interface and then switch back to their own desktop. For more on what happens at startup, see the sidebar "Viewing Startup Process Relationships."

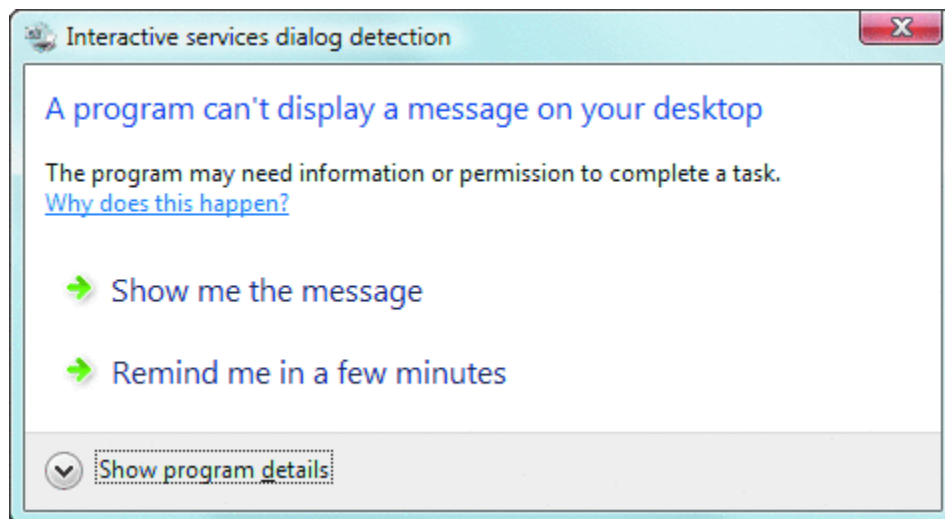


Figure 4 Service has Displayed A Window

Viewing Startup Process Relationships

You can use Process Explorer from Sysinternals (microsoft.com/technet/sysinternals) to see the process startup tree of Windows Vista.

The screenshot includes the Session column, which you can add through Process Explorer's column dialog. The highlighted process is the initial Smss.exe. Below it is the Session 0 Csrss.exe and Wininit.exe, which are left-justified because their parent process, the instance of Smss.exe that configured Session 0, has exited. Wininit's three children are Services.exe, Lsass.exe, and Lsm.exe.

Inside the Windows Vista Kernel

Mark Russinovich

Process Explorer identifies a set of processes as running in Session 1 and that's the session I'm logged into through a Remote Desktop connection. Process Explorer displays processes running in the same account as itself with a blue highlight color. Finally, Session 2 was initialized to prepare for a user logging into the console and creating a new logon session. It's in that session that Winlogon is running and using LogonUI to ask a new console user to "Press Ctrl+Alt+DELETE to Log on", and in which Logonui.exe will ask the user for his credentials.

Process	PID	Se...	Description	Company Name
System Idle Process	0			
Interrupts	n/a	0	Hardware Interrupts	
DPCs	n/a	0	Deferred Procedure Calls	
System	4	0		
smss.exe	376	0	Windows Session Manager	Microsoft Corporation
csrss.exe	444	0	Client Server Runtime Process	Microsoft Corporation
wininit.exe	496	0	Windows Start-Up Application	Microsoft Corporation
services.exe	572	0	Services and Controller app	Microsoft Corporation
lsass.exe	584	0	Local Security Authority Process	Microsoft Corporation
lsm.exe	592	0	Local Session Manager Service	Microsoft Corporation
csrss.exe	2012	2	Client Server Runtime Process	Microsoft Corporation
winlogon.exe	3764	2	Windows Logon Application	Microsoft Corporation
LogonUI.exe	3768	2	Windows Logon User Interface Host	Microsoft Corporation
csrss.exe	2796	1	Client Server Runtime Process	Microsoft Corporation
winlogon.exe	2100	1	Windows Logon Application	Microsoft Corporation
explorer.exe	2464	1	Windows Explorer	Microsoft Corporation
MSASCui.exe	4024	1	Windows Defender User Interface	Microsoft Corporation
GrooveMonitor.exe	3684	1	GrooveMonitor Utility	Microsoft Corporation
jusched.exe	1784	1	Java(TM) Platform SE binary	Sun Microsystems, Inc.
Realmon.exe	3308	1		Computer Associates Inter
procexp.exe	2616	1	Sysinternals Process Explorer	Sysinternals

CPU Usage: 6.80% Commit Charge: 36.49% Processes: 47

Startup Process and Session Information

Credential Providers

Even the logon architecture is changed on Windows Vista. On previous versions of Windows, the Winlogon process loaded the Graphical Identification and Authentication (GINA) DLL specified in the registry to display a logon UI that asked users for their credentials. Unfortunately, the GINA model suffers from several limitations, including the fact that only one GINA can be configured, writing a complete GINA is difficult for third parties, and custom GINAs that have non-standard user interfaces change the Windows user experience.

Instead of a GINA, Windows Vista uses the new Credential Provider architecture. Winlogon launches a separate process, the Logon User Interface Host (Logonui.exe), that loads credential providers that are configured in:

Inside the Windows Vista Kernel

Mark Russinovich

`HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\Currentversion\Authentication\Credential Providers`

Logonui can host multiple credential providers concurrently; in fact, Windows Vista ships with interactive (Authui.dll) and smartcard (Smart-cardcredentialprovider.dll) providers. To ensure a uniform user experience, LogonUI manages the user interface that is displayed to end users, but it also allows credential providers to specify custom elements like text, icons, and edit controls.

Delayed-Autostart Services

If you've ever logged onto a Windows system immediately after it starts, you've probably experienced delays before your desktop is fully configured and you can interact with the shell and any applications you launch. While you're logging on, the Service Control Manager is starting the many Windows services that are configured as automatic start services and therefore activate at boot time. Many services perform CPU and disk-intensive initializations that compete with your logon activities. To accommodate this, Windows Vista introduces a new service start type called delayed automatic start, which services can use if they don't have to be active immediately after Windows boots.

The Service Control Manager starts services configured for delayed automatic start after the automatic-start services have finished starting and it sets the priority of their initial thread to `THREAD_PRIORITY_LOWEST`. This priority level causes all the disk I/O the thread performs to be Very Low I/O priority. After a service finishes initializing, the Service Control Manager sets its priority to normal. The combination of the delayed start, low CPU and memory priority, and background disk priority greatly reduce interference with a user's logon. Many Windows services, including Background Intelligent Transfer, Windows Update Client, and Windows Media® Center, use the new start type to help improve the performance of logons after a boot.

Shutdown

A problem that's plagued Windows service writers is that during a Windows shutdown they have, by default, a maximum of twenty seconds to perform cleanup. Versions of Windows prior to Windows Vista haven't supported a clean shutdown that waits for all services to exit because a buggy service can hold up a shutdown indefinitely. Some services, like those that have network-related shutdown operations or have to save large amounts of data to disk, might require more time and so Windows Vista allows a service to request pre-shutdown notification.

When Windows Vista shuts down, the Service Control Manager first notifies those services asking for pre-shutdown notification. It will wait indefinitely for these services to exit, but if they have a bug and don't respond to queries, the Service Control Manager gives up and moves on after three minutes. Once all those services have exited or the timeout has expired, the Service Control Manager proceeds with legacy-style services shutdown for the rest of the services. The Group Policy and Windows Update services register pre-shutdown notification in a fresh Windows Vista installation.

The Group Policy and Windows Update services also use another Windows Vista services feature: shutdown ordering. Services have always been able to specify startup dependencies that the Service Control Manager honors to start services in an order that satisfies them, but until Windows Vista they have been unable to specify shutdown dependencies. Now services that register for pre-shutdown notification can also insert themselves into the list stored at:

`HKLM\System\CurrentControlSet\Control\PreshutdownOrder`

and the Service Control Manager will shut them down according to their order. See the sidebar "Identifying a Delayed-Autostart and Pre-Shutdown Service" for more on these services.

Inside the Windows Vista Kernel

Mark Russinovich

Power Management

Sleep and hibernate are other forms of shutdown, and buggy power management in drivers and applications has been the curse of road warriors since Windows 2000 introduced power management to the Windows NT®-based line of Windows operating systems. Many users have expected their laptop system to suspend or hibernate when they closed the lid before embarking on a trip, only to arrive at their destination with a hot carrying case, a dead battery, and lost data. That's because Windows has always asked device drivers and applications for their consent to change power state and a single unresponsive driver or application could prevent a transition.

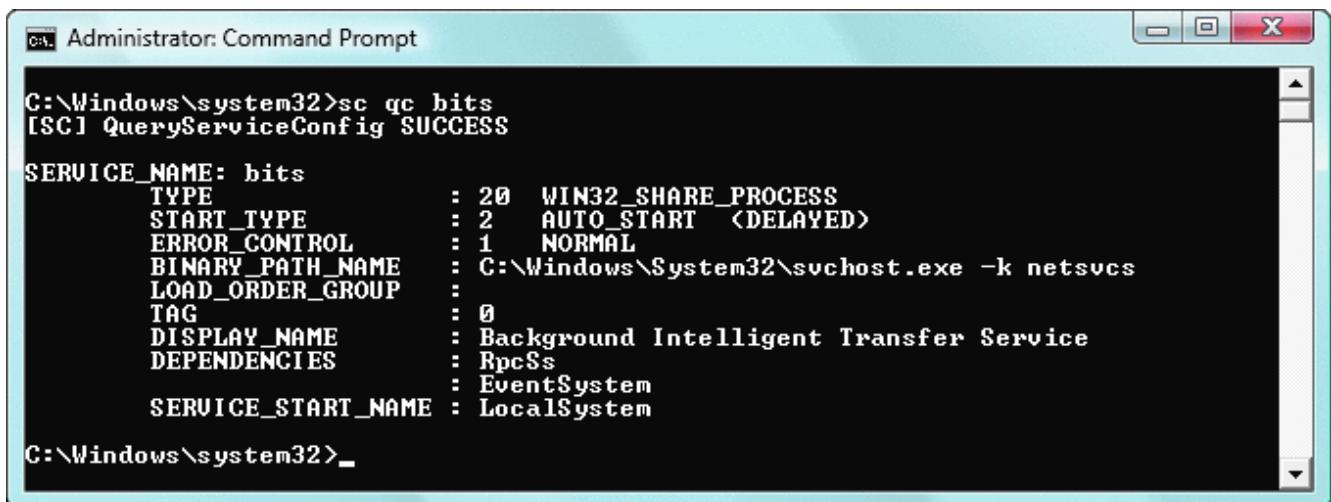
In Windows Vista, the kernel's Power Manager still informs drivers and applications of power-state changes so that they can prepare for them, but it no longer asks for permission. In addition, the Power Manager waits, at most, 20 seconds for applications to respond to change notifications, rather than the two minutes it waited on previous versions of Windows. As a result, Windows Vista users can be more confident that their systems are honoring hibernations and suspends.

Next Up

As mentioned earlier, this is the second installment in a three-part series. The first part covered Windows Vista kernel improvements in the areas of I/O and processes. This time, I looked at Windows Vista enhancements in memory management, startup, and shutdown. Next time, I'll conclude the series by describing changes to the kernel in the areas of reliability and security.

Identifying a Delayed-Autostart and Pre-Shutdown Service

The built-in SC command is updated in Windows Vista to show services configured as delayed autostart services:



```
Administrator: Command Prompt
C:\Windows\system32>sc qc bits
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: bits
        TYPE               : 20    WIN32_SHARE_PROCESS
        START_TYPE          : 2     AUTO_START <DELAYED>
        ERROR_CONTROL       : 1     NORMAL
        BINARY_PATH_NAME    : C:\Windows\System32\svchost.exe -k netsvcs
        LOAD_ORDER_GROUP    :
        TAG                 : 0
        DISPLAY_NAME        : Background Intelligent Transfer Service
        DEPENDENCIES        : RpcSs
                          : EventSystem
        SERVICE_START_NAME  : LocalSystem

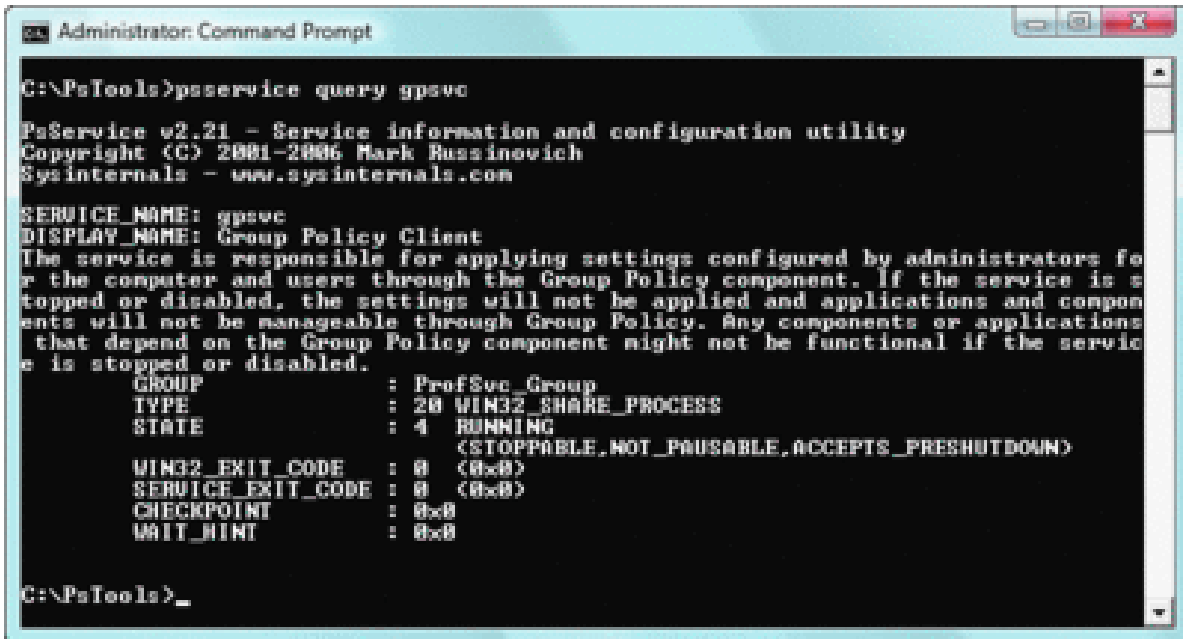
C:\Windows\system32>_
```

Using SC to Display Start Type

Unfortunately, the SC command does not report services that have requested pre-shutdown notification, but you can use the PsService utility from Sysinternals to see that a service accepts pre-shutdown notification:

Inside the Windows Vista Kernel

Mark Russinovich



```
Administrator: Command Prompt
C:\PsTools>pservice query gpssc

PsService v2.21 - Service information and configuration utility
Copyright (C) 2001-2006 Mark Russinovich
Sysinternals - www.sysinternals.com

SERVICE_NAME: gpssc
DISPLAY_NAME: Group Policy Client
The service is responsible for applying settings configured by administrators for the computer and users through the Group Policy component. If the service is stopped or disabled, the settings will not be applied and applications and components will not be manageable through Group Policy. Any components or applications that depend on the Group Policy component might not be functional if the service is stopped or disabled.
GROUP           : ProfSvc_Group
TYPE            : 20 WIN32_SHARE_PROCESS
STATE           : 4 RUNNING
                (STOPPABLE,NOT_PAUSABLE,ACCEPTS_PRESHUTDOWN)
WIN32_EXIT_CODE : 0 (0x0)
SERVICE_EXIT_CODE : 0 (0x0)
CHECKPOINT      : 0x0
WAIT_HINT      : 0x0

C:\PsTools>_
```

Viewing Pre-shutdown Status

This series has so far covered Windows Vista kernel enhancements related to processes, I/O, memory management, system startup, shutdown, and power management. In this third and final installment, I take a look at features and improvements in the areas of reliability, recovery, and security.

One feature I'm not covering in this series is User Account Control (UAC), which comprises several different technologies, including file system and registry virtualization for legacy applications, elevation consent for accessing administrative rights, and the Windows® Integrity Level mechanism for isolating processes running with administrative rights from less-privileged processes running in the same account. Look for my in-depth coverage of UAC internals in a future issue of TechNet Magazine.

Windows Vista™ improves the reliability of your system and your ability to diagnose system and application problems through a number of new features and enhancements. For example, the kernel Event Tracing for Windows (ETW) logger is always active, generating trace events for file, registry, interrupt, and other types of activity into a circular buffer. When a problem occurs, the new Windows Diagnostic Infrastructure (WDI) can capture a snapshot of the buffer and analyze it locally or upload it to Microsoft support for troubleshooting.

The new Windows Performance and Reliability Monitor helps users correlate errors, such as crashes and hangs, with changes that have been made to system configuration. The powerful System Repair Tool (SRT) replaces the Recovery Console for off-line recovery of unbootable systems.

There are three areas that rely on kernel-level changes to the system and so merit a closer look in this article: Kernel Transaction Manager (KTM), improved crash handling, and Previous Versions.

Kernel Transaction Manager

One of the more tedious aspects of software development is handling error conditions. This is especially true if, in the course of performing a high-level operation, an application has completed one or more subtasks that result in changes to the file system or registry. For example, an application's

Inside the Windows Vista Kernel

Mark Russinovich

software updating service might make several registry updates, replace one of the application's executables, and then be denied access when it attempts to update a second executable. If the service doesn't want to leave the application in the resulting inconsistent state, it must track all the changes it makes and be prepared to undo them. Testing the error recovery code is difficult and consequently often skipped, so errors in the recovery code can negate the effort.

Applications written for Windows Vista can, with very little effort, gain automatic error recovery capabilities by using the new transactional support in NTFS and the registry with the Kernel Transaction Manager. When an application wants to make a number of related changes, it can either create a Distributed Transaction Coordinator (DTC) transaction and a KTM transaction handle, or create a KTM handle directly and associate the modifications of the files and registry keys with the transaction. If all the changes succeed, the application commits the transaction and the changes are applied, but at any time up to that point the application can roll back the transaction and the changes are then discarded.

As a further benefit, other applications don't see changes made in a transaction until the transaction commits, and applications that use the DTC in Windows Vista and the forthcoming Windows Server®, code-named "Longhorn," can coordinate their transactions with SQL Server™, Microsoft® Message Queue Server (MSMQ), and other databases. An application updating service that uses KTM transactions will therefore never leave the application in an inconsistent state. This is why both Windows Update and System Restore use transactions.

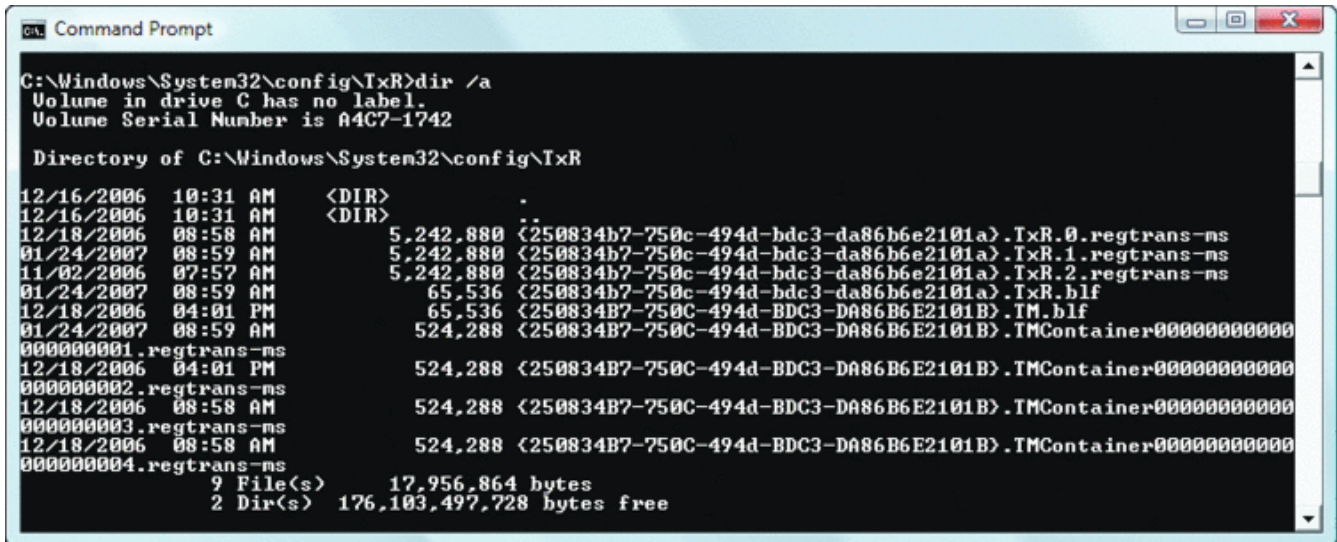
As the heart of transaction support, KTM allows transactional resource managers such as NTFS and the registry to coordinate their updates for a specific set of changes made by an application. In Windows Vista, NTFS uses an extension to support transactions called TxF. The registry uses a similar extension called TxR. These kernel-mode resource managers work with the KTM to coordinate the transaction state, just as user-mode resource managers use DTC to coordinate transaction state across multiple user-mode resource managers. Third parties can also use KTM to implement their own resource managers.

TxF and TxR both define a new set of file system and registry APIs that are similar to existing ones, except that they include a transaction parameter. If an application wants to create a file within a transaction, it first uses KTM to create the transaction, then passes the resulting transaction handle to the new-file creation API.

TxF and TxR both rely on the high-speed file system logging functionality of the Common Log File System or CLFS (%SystemRoot%\System32\Cdfs.sys) that was introduced in Windows Server 2003 R2. TxR and TxF use CLFS to durably store transactional state changes before they commit a transaction. This allows them to provide transactional recovery and assurances even if power is lost. In addition to the CLFS log, TxR creates a set of related log files to track transaction changes to the system's registry file in %Systemroot%\System32\Config\Txr, as seen in Figure 1, as well as separate sets of log files for each user registry hive. TxF stores transactional data for each volume in a hidden directory on the volume that's named \ \$Extend\ \$RmMetadata.

Inside the Windows Vista Kernel

Mark Russinovich



```
ca: Command Prompt
C:\Windows\System32\config\TxR>dir /a
Volume in drive C has no label.
Volume Serial Number is 04C7-1742

Directory of C:\Windows\System32\config\TxR

12/16/2006  10:31 AM    <DIR>
12/16/2006  10:31 AM    <DIR>
12/18/2006  08:58 AM           5,242,880 <250834b7-750c-494d-bdc3-da86b6e2101a>.TxR.0.regtrans-ms
01/24/2007  08:59 AM           5,242,880 <250834b7-750c-494d-bdc3-da86b6e2101a>.TxR.1.regtrans-ms
11/02/2006  07:57 AM           5,242,880 <250834b7-750c-494d-bdc3-da86b6e2101a>.TxR.2.regtrans-ms
01/24/2007  08:59 AM             65,536 <250834b7-750c-494d-bdc3-da86b6e2101a>.TxR.b1f
12/18/2006  04:01 PM             65,536 <250834B7-750C-494D-BDC3-DA86B6E2101B>.TM.b1f
01/24/2007  08:59 AM           524,288 <250834B7-750C-494D-BDC3-DA86B6E2101B>.TMContainer0000000000
00000001.regtrans-ms
12/18/2006  04:01 PM           524,288 <250834B7-750C-494D-BDC3-DA86B6E2101B>.TMContainer0000000000
00000002.regtrans-ms
12/18/2006  08:58 AM           524,288 <250834B7-750C-494D-BDC3-DA86B6E2101B>.TMContainer0000000000
00000003.regtrans-ms
12/18/2006  08:58 AM           524,288 <250834B7-750C-494D-BDC3-DA86B6E2101B>.TMContainer0000000000
00000004.regtrans-ms
          9 File(s)      17,956,864 bytes
          2 Dir(s)     176,103,497,728 bytes free
```

Figure 1 System Registry Hive TxR Logging Files

Enhanced Crash Support

When Windows encounters an unrecoverable kernel-mode error—whether due to a buggy device driver, faulty hardware, or the operating system—it tries to prevent corruption of on-disk data by halting the system after displaying the notorious "blue screen of death" and, if configured to do so, writing the contents of some or all of physical memory to a crash dump file. Dump files are useful because when you reboot from a crash, the Microsoft Online Crash Analysis (OCA) service offers to analyze them to look for the root cause. If you like, you can also analyze them yourself using the Microsoft Debugging Tools for Windows).

In previous versions of Windows, however, support for crash dump files wasn't enabled until the Session Manager (%Systemroot%\System32\Smss.exe) process initialized paging files. This meant that any critical errors before that point result in a blue screen, but no dump file. Since the bulk of device driver initialization occurs before Smss.exe starts, early crashes would never result in crash dumps, thus making diagnosis of the cause extremely difficult.

Windows Vista reduces the window of time where no dump file is generated by initializing dump file support after all the boot-start device drivers are initialized but before loading system-start drivers. Because of this change, if you do experience a crash during the early part of the boot process, the system can capture a crash dump, allowing OCA to help you resolve the problem. Further, Windows Vista saves data to a dump file in 64KB blocks, whereas previous versions of Windows wrote them using 4KB blocks. This change results in large dump files being written up to 10 times faster.

Application crash handling is also improved in Windows Vista. On previous versions of Windows, when an application crashed it executed an unhandled exception handler. The handler launched the Microsoft Application Error Reporting (AER) process (%Systemroot%\System32\Dwwin.exe) to display a dialog indicating that the program has crashed and asking whether you wanted to send an error report to Microsoft. However, if the stack of the process's main thread was corrupted during the crash, the unhandled exception handler crashed when it executed, resulting in termination of the process by the kernel, the instant disappearance of the program's windows, and no error reporting dialog.

Inside the Windows Vista Kernel

Mark Russinovich

Windows Vista moves error handling out of the context of the crashing process into to a new service, Windows Error Reporting (WER). This service is implemented by a DLL (%Systemroot%\System32\Wersvc.dll) inside a Service Hosting process. When an application crashes, it still executes an unhandled exception handler, but that handler sends a message to the WER service and the service launches the WER Fault Reporting process (%Systemroot%\System32\Werfault.exe) to display the error reporting dialog. If the stack is corrupted and the unhandled exception handler crashes, the handler executes again and crashes again, eventually consuming all the thread's stack (scratch memory area), at which point the kernel steps in and sends the crash notification message to the service.

You can see the contrast in these two approaches in Figures 2 and 3, which show the process relationship of Accvio.exe, a test program that crashes, and the error reporting processes highlighted in green, on Windows XP and Windows Vista. The new Windows Vista error handling architecture means that programs will no longer silently terminate without offering the chance for Microsoft to obtain an error report and help software developers improve their applications.

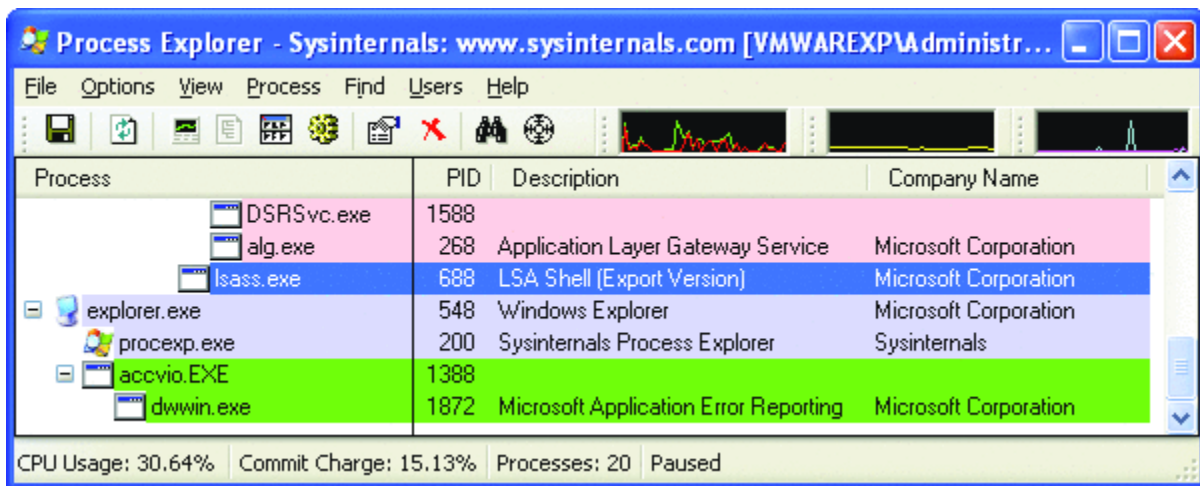


Figure 2a Application Error Handling in Windows XP

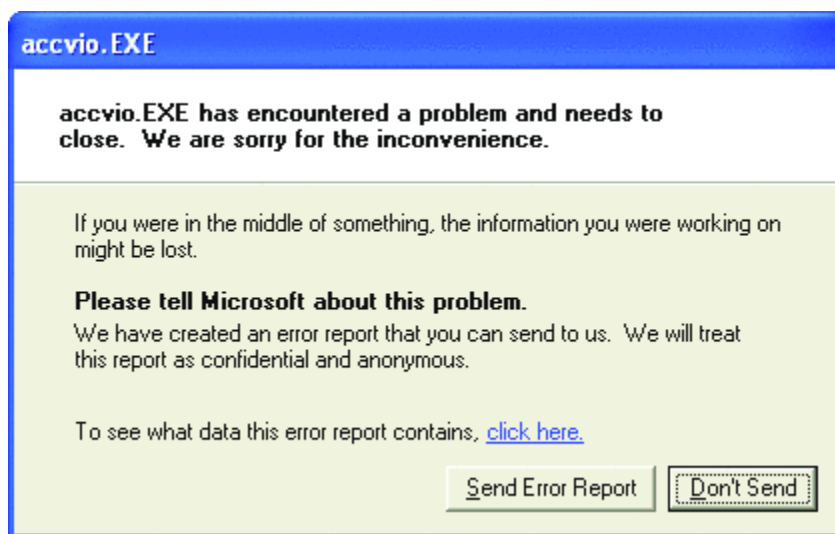


Figure 2b

Inside the Windows Vista Kernel

Mark Russinovich

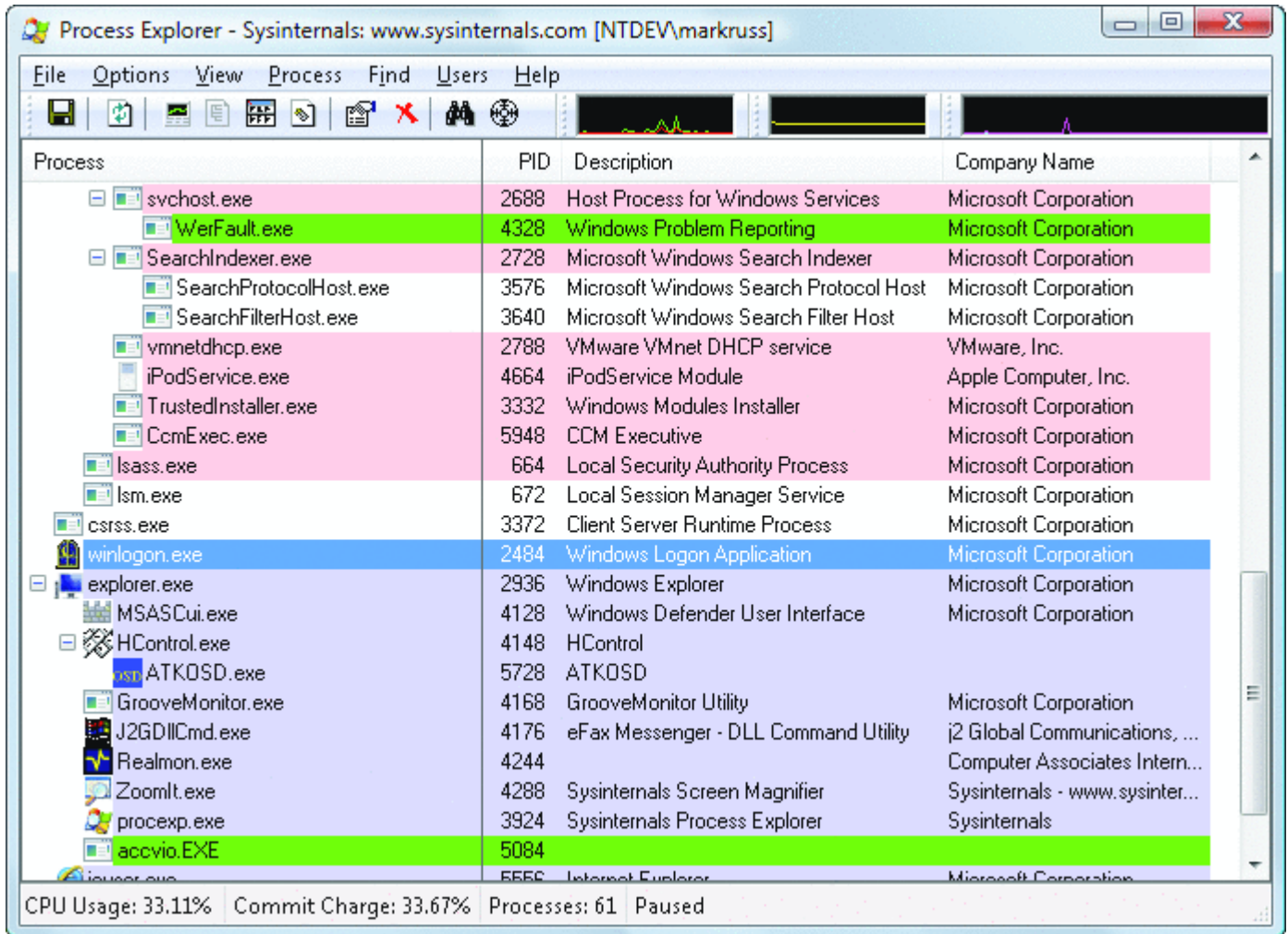


Figure 3a Application Error Handling in Windows Vista

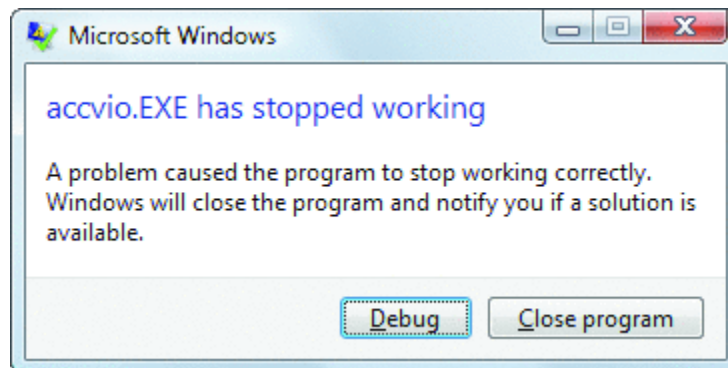


Figure 3b

Volume Shadow Copy

Windows XP introduced a technology called Volume Shadow Copy to make point-in-time snapshots of disk volumes. Backup applications can use these snapshots to make consistent backup images,

Inside the Windows Vista Kernel

Mark Russinovich

but the snapshots are otherwise hidden from view and kept only for the duration of the backup process.

The snapshots are not actually full copies of volumes. Rather, they are views of a volume from an earlier point that comprise the live volume data overlaid with copies of volume sectors that have changed since the snapshot was taken. The Volume Snapshot Provider driver (%Systemroot%\%System32\Drivers\Volsnap.sys) monitors operations aimed at volumes and makes backup copies of sectors before allowing them to change, storing the original data in a file associated with the snapshot in the System Volume Information directory of the volume.

Windows Server 2003 exposed snapshot management to administrators on the server and to users on client systems with its Shadow Copies for Shared Folders. This feature enabled persistent snapshots that users could access via a Previous Versions tab on the Explorer properties dialog boxes for their folders and files located on the server's file shares.

The Windows Vista Previous Versions feature brings this support to all client systems, automatically creating volume snapshots, typically once per day, that you can access through Explorer properties dialogs using the same interface used by Shadow Copies for Shared Folders. This enables you to view, restore, or copy old versions of files and directories that you might have accidentally modified or deleted. While technically not new technology, the Windows Vista Previous Versions implementation of Volume Shadow Copy optimizes that of Windows Server 2003 for use in client desktop environments.

Windows Vista also takes advantage of volume snapshots to unify user and system data protection mechanisms and avoid saving redundant backup data. When an application installation or configuration change causes incorrect or undesirable behaviors, you can use System Restore, a feature introduced into the Windows NT® line of operating systems in Windows XP, to restore system files and data to their state as it existed when a restore point was created.

In Windows XP, System Restore uses a file system filter driver—a type of a driver that can see changes at the file level—to make backup copies of system files at the time they change. On Windows Vista, System Restore uses volume snapshots. When you use the System Restore user interface in Windows Vista to go back to a restore point, you're actually copying earlier versions of modified system files from the snapshot associated with the restore point to the live volume.

BitLocker

Windows Vista is the most secure version of Windows yet. In addition to the inclusion of the Windows Defender antispyware engine, Windows Vista introduces numerous security and defense-in-depth features, including BitLocker™ full-volume encryption, code signing for kernel-mode code, protected processes, Address Space Load Randomization, and improvements to Windows service security and User Account Control.

An operating system can only enforce its security policies while it's active, so you have to take additional measures to protect data when the physical security of a system can be compromised and the data accessed from outside the operating system. Hardware-based mechanisms such as BIOS passwords and encryption are two technologies commonly used to prevent unauthorized access, especially on laptops, which are most likely to be lost or stolen.

Windows 2000 introduced the Encrypting File System (EFS) and, in its Windows Vista incarnation, EFS includes a number of improvements over previous implementations, including performance enhancements, support for encrypting the paging file, and storage of user EFS keys on smart cards. However, you can't use EFS to protect access to sensitive areas of the system, such as the registry

Inside the Windows Vista Kernel

Mark Russinovich

hive files. For example, if Group Policy allows you to log onto your laptop even when you're not connected to a domain, then your domain credential verifiers are cached in the registry, so an attacker could use tools to obtain your domain account password hash and use that to try to obtain your password with a password cracker. The password would gain them access to your account and EFS files (assuming you didn't store the EFS key on a smart card).

To make it easy to encrypt the entire boot volume (the volume with the Windows directory), including all its system files and data, Windows Vista introduces a full-volume encryption feature called Windows BitLocker Drive Encryption. Unlike EFS, which is implemented by the NTFS file system driver and operates at the file level, BitLocker encrypts at the volume level using the Full Volume Encryption (FVE) driver (%Systemroot%\System32\Drivers\Fvevol.sys) as diagrammed in Figure 4.

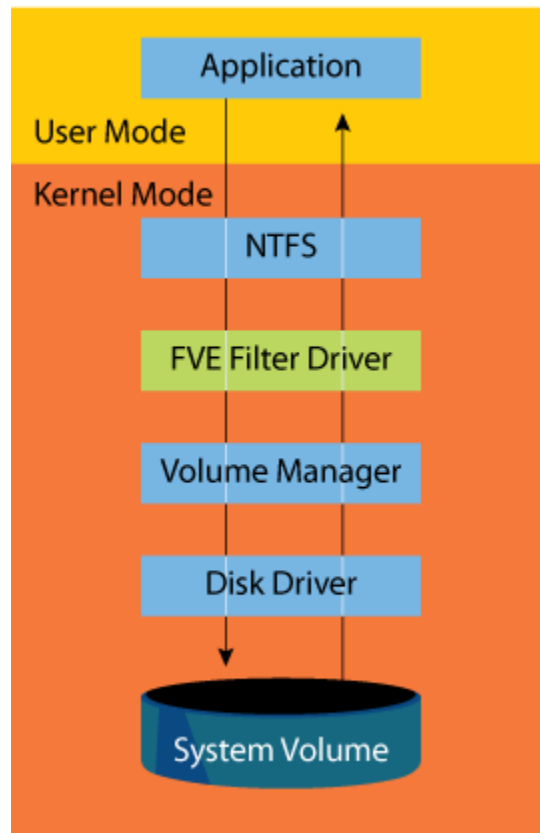


Figure 4 BitLocker FVE Filter Driver

FVE is a filter driver so it automatically sees all the I/O requests that NTFS sends to the volume, encrypting blocks as they're written and decrypting them as they're read using the Full Volume Encryption Key (FVEK) assigned to the volume when it's initially configured to use BitLocker. By default, volumes are encrypted using a 128-bit AES key and a 128-bit diffuser key. Because the encryption and decryption happen beneath NTFS in the I/O system, the volume appears to NTFS as if it's unencrypted and NTFS does not even need to be aware that BitLocker is enabled. If you attempt to read data from the volume from outside of Windows, however, it appears to be random data.

The FVEK is encrypted with a Volume Master Key (VMK) and stored in a special metadata region of the volume. When you configure BitLocker, you have a number of options for how the VMK will be protected, depending on the system's hardware capabilities. If the system has a Trusted Platform Module (TPM) that conforms to v1.2 of the TPM specification and has associated BIOS support, then

Inside the Windows Vista Kernel

Mark Russinovich

you can either encrypt the VMK with the TPM, have the system encrypt the VMK using a key stored in the TPM and one stored on a USB flash device, or encrypt the key using a TPM-stored key and a PIN you enter when the system boots. For systems that don't have a TPM, BitLocker offers the option of encrypting the VMK using a key stored on an external USB flash device. In any case you'll need an unencrypted 1.5GB NTFS system volume, the volume where the Boot Manager and Boot Configuration Database (BCD) are stored.

The advantage of using a TPM is that BitLocker uses TPM features to ensure that it will not decrypt the VMK and unlock the boot volume if the BIOS or the system boot files have changed since BitLocker was enabled. When you encrypt the system volume for the first time, and each time you perform updates to any of the components mentioned, BitLocker calculates SHA-1 hashes of these components and stores each hash, called a measurement, in different Platform Configuration Registers (PCR) of the TPM with the help of the TPM device driver (%Systemroot%\System32\Drivers\Tpm.sys). It then uses the TPM to seal the VMK, an operation that uses a private key stored in the TPM to encrypt the VMK and the values stored in the PCRs along with other data BitLocker passes to the TPM. BitLocker then stores the sealed VMK and encrypted FVEK in the volume's metadata region.

When the system boots, it measures its own hashing and PCR loading code and writes the hash to the first PCR of the TPM. It then hashes the BIOS and stores that measurement in the appropriate PCR. The BIOS in turn hashes the next component in the boot sequence, the Master Boot Record (MBR) of the boot volume, and this process continues until the operating system loader is measured. Each subsequent piece of code that runs is responsible for measuring the code that it loads and for storing the measurement into the appropriate register in the TPM. Finally, when the user selects which operating system to boot, the Boot Manager (Bootmgr) reads the encrypted VMK from the volume and asks the TPM to unseal it. Only if all the measurements are the same as when the VMK was sealed, including the optional PIN, will the TPM successfully decrypt the VMK.

You can think of this scheme as a verification chain, where each component in the boot sequence describes the next component to the TPM. Only if all the descriptions match the original ones given to it will the TPM divulge its secret. BitLocker therefore protects the encrypted data even when the disk is removed and placed in another system, the system is booted using a different operating system, or the unencrypted files on the boot volume are compromised.

Code Integrity Verification

Malware that is implemented as a kernel-mode device driver, including rootkits, runs at the same privilege level as the kernel and so is the most difficult to identify and remove. Such malware can modify the behavior of the kernel and other drivers so as to become virtually invisible. The Windows Vista code integrity for kernel-mode code feature, also known as kernel-mode code signing (KMCS), only allows device drivers to load if they are published and digitally signed by developers who have been vetted by one of a handful of certificate authorities (CAs). KMCS is enforced by default on Windows Vista for 64-bit systems.

Because certificate authorities charge a fee for their services and perform basic background checks, such as verifying a business identity, it's harder to produce anonymous kernel-mode malware that runs on 64-bit Windows Vista. Further, malware that does manage to slip through the verification process can potentially leave clues that lead back to the author when the malware is discovered on a compromised system. KMCS also has secondary uses, like providing contact information for the Windows Online Crash Analysis team when a driver is suspected of having a bug that's crashing customer systems, and unlocking high-definition multimedia content, which I'll describe shortly.

Inside the Windows Vista Kernel

Mark Russinovich

KMCS uses public-key cryptography technologies that have been employed for over a decade by Windows and requires that kernel-mode code include a digital signature generated by one of the trusted certificate authorities. If a publisher submits a driver to the Microsoft Windows Hardware Quality Laboratory (WHQL) and the driver passes reliability testing, then Microsoft serves as the certificate authority that signs the code. Most publishers will obtain signatures via WHQL, but when a driver has no WHQL test program, the publisher doesn't want to submit to WHQL testing, or the driver is a boot-start driver that loads early in system startup, the publishers must sign the code themselves. To do so, they must first obtain a code-signing certificate from one of the certificate authorities that Microsoft has identified as trusted for kernel-mode code signing. The author then digitally hashes the code, signs the hash by encrypting it with a private key, and includes the certificate and encrypted hash with the code.

When a driver tries to load, Windows decrypts the hash included with the code using the public key stored in the certificate, then verifies that the hash matches the one included with the code. The authenticity of the certificate is checked in the same way, but using the certificate authority's public key, which is included with Windows.

Windows also checks the associated certificate chains up to one of the root authorities embedded in the Windows boot loader and operating system kernel. Attempts to load an unsigned 64-bit driver should never occur on a production system, so unlike the Plug and Play Manager, which displays a warning dialog when it's directed to load a driver that doesn't have a signature confirming that it's been through WQHL testing, 64-bit Windows Vista silently writes an event to the Code Integrity application event log, like the one shown in Figure 5, anytime it blocks the loading of an unsigned driver. 32-bit Windows Vista also checks driver signatures, but allows unsigned drivers to load. Blocking them would break upgraded Windows XP systems that require drivers that were loaded on Windows XP, and also allows support for hardware for which only Windows XP drivers exist. However, 32-bit Windows Vista also writes events to the Code Integrity event log when it loads an unsigned driver.

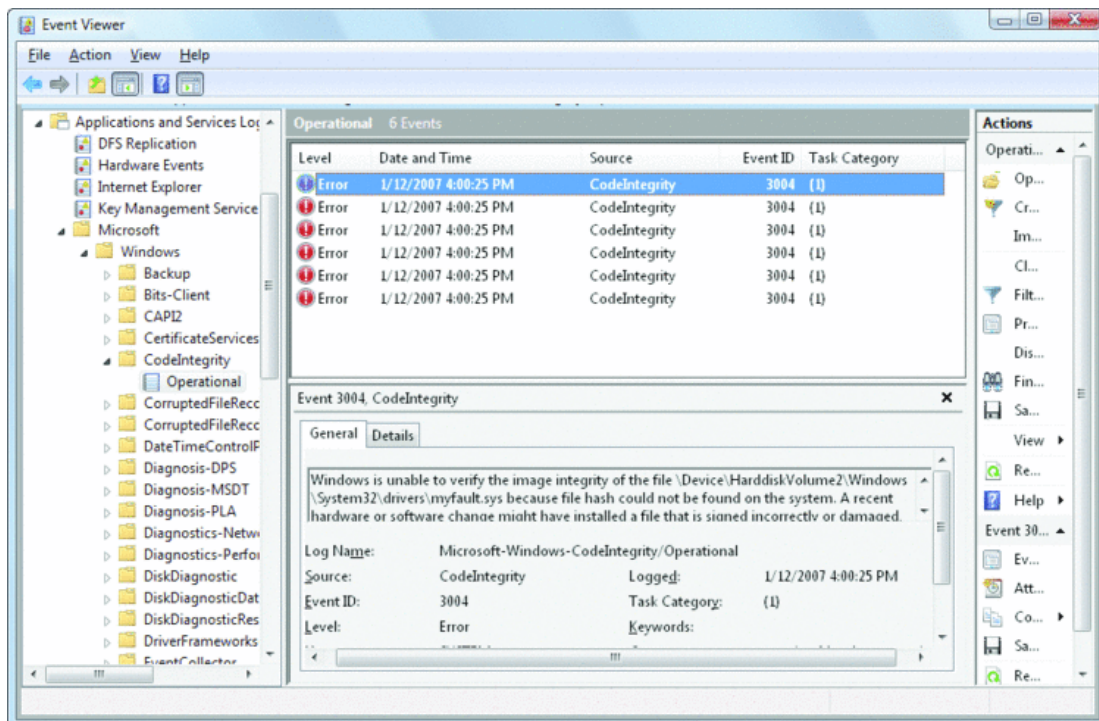


Figure 5 Unsigned Driver Load Attempt Events

Inside the Windows Vista Kernel

Mark Russinovich

Because code signing is commonly used to label code as a rigorously tested official release, publishers typically don't want to sign test code. Windows Vista therefore includes a test-signing mode you can enable and disable with the Bcdedit tool (described in my March 2007 TechNet Magazine article), where it will load kernel-mode drivers digitally signed with a test certificate generated by an in-house certificate authority. This mode is designed for use by programmers while they develop their code. When Windows is in this mode, it displays markers on the desktop like the one in Figure 6.

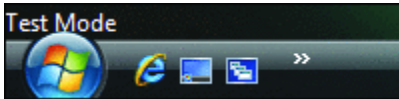


Figure 6 Windows Vista test-Signing Mode

Protected Processes

Next-generation multimedia content, like HD-DVD, BluRay, and other formats licensed under the Advanced Access Content System (AACs), will become more common over the next few years. Windows Vista includes a number of technologies, collectively called Protected Media Path (PMP), that are required by the AACs standard for such content to be played. PMP includes Protected User-Mode Audio (PUMA) and Protected Video Path (PVP) that together provide mechanisms for audio and video drivers, as well as media player applications, to prevent unauthorized software or hardware from capturing content in high-definition form.

PUMA and PVP define interfaces and support specific to audio and video players, device drivers, and hardware, but PMP also relies on a general kernel mechanism introduced in Windows Vista called a protected process. Protected processes are based on the standard Windows process construct that encapsulates a running executable image, its DLLs, security context (the account under which the process is running and its security privileges), and the threads that execute code within the process, but prevent certain types of access.

Standard processes implement an access control model that allows full access to the owner of the process and administrative accounts with the Debug Programs privilege. Full access allows a user to view and modify the address space of the process, including the code and data mapped into the process. Users can also inject threads into the process. These types of access are not consistent with the requirements of PMP because they would allow unauthorized code to gain access to high-definition content and Digital Rights Management (DRM) keys stored in a process that is playing the content.

Protected processes restrict access to a limited set of informational and process management interfaces that include querying the process's image name and terminating or suspending the process. But the kernel makes diagnostic information for protected processes available through general process query functions that return data regarding all the processes on a system and so don't require direct access to the process. Accesses that could compromise media are allowed only by other protected processes.

Further, to prevent compromise from within, all executable code loaded into a protected process, including its executable image and DLLs, must be either signed by Microsoft (WHQL) with a Protected Environment (PE) flag, or if it's an audio codec, signed by the developer with a DRM-signing certificate obtained from Microsoft. Because kernel-mode code can gain full access to any process, including protected processes, and 32-bit Windows allows unsigned kernel-mode code to load, the

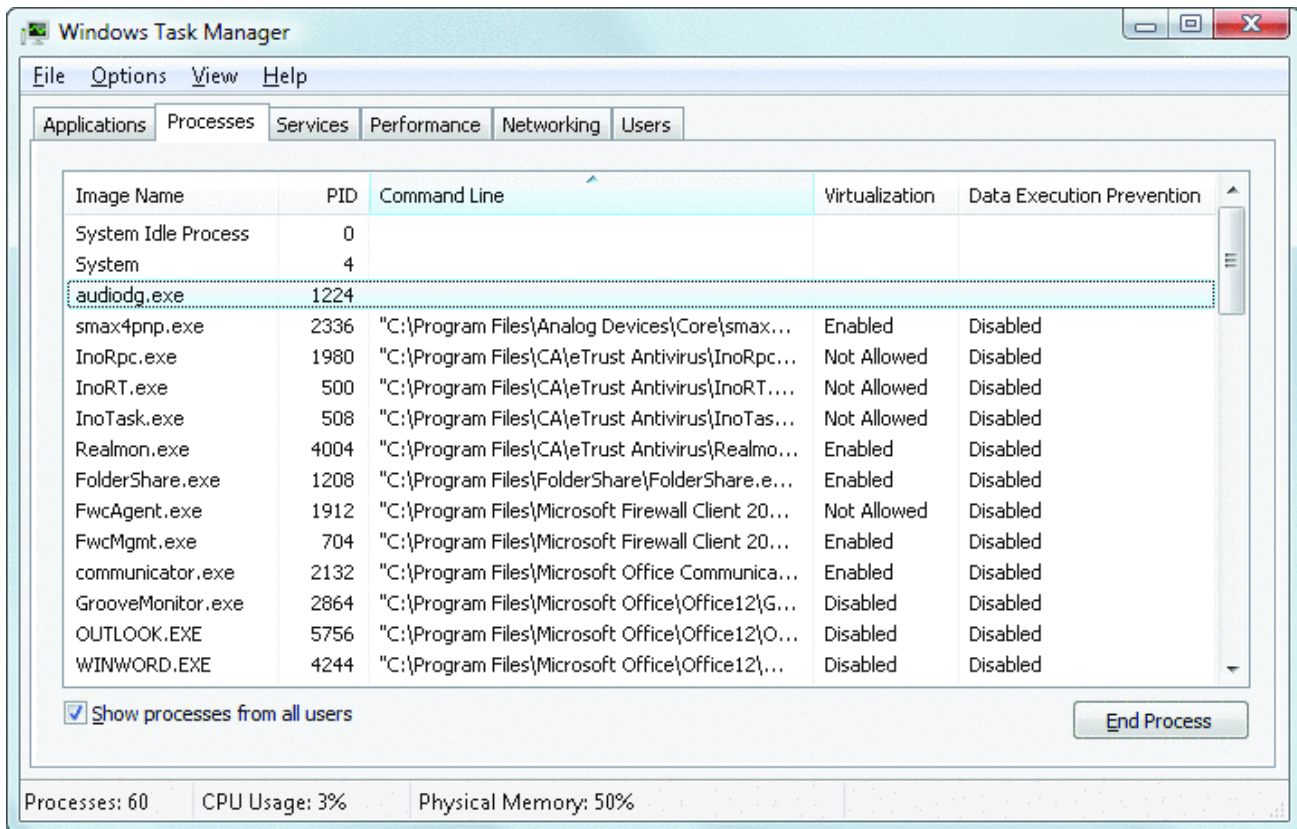
Inside the Windows Vista Kernel

Mark Russinovich

kernel provides an API for protected processes to query the "cleanliness" of the kernel-mode environment and use the result to unlock premium content only if no unsigned code is loaded.

Identifying a Protected Process

There are no APIs that specifically identify protected processes, but you can indirectly identify them based on the limited information available for them and the inability to debug them even from an administrative account. The Audio Device Graph Isolation process (%Systemroot%\System32\Audiodg.exe) is used to play Content Scramble System (CSS)-encoded DVDs and is identifiable as a protected process in the Task Manager pane by the fact that Task Manager can't obtain its command line, virtualization, and Data Execution Prevention status even when it is run with administrative rights.



Task Manager Viewing the Audiodg Protected Process

Address Space Load Randomization

Despite measures like Data Execution Prevention and enhanced compiler error checking, malware authors continue to find buffer overflow vulnerabilities that allow them to infect network-facing processes like Internet Explorer®, Windows services, and third-party applications to gain a foothold on a system. Once they have managed to infect a process, however, they must use Windows APIs to accomplish their ultimate goal of reading user data or establishing a permanent presence by modifying user or system configuration settings.

Connecting an application with API entry points exported by DLLs is something usually handled by the operating system loader, but these types of malware infection don't get the benefit of the loader's services. This hasn't posed a problem for malware on previous versions of Windows because for any

Inside the Windows Vista Kernel

Mark Russinovich

given Windows release, system executable images and DLLs always load at the same location, allowing malware to assume that APIs reside at fixed addresses.

The Windows Vista Address Space Load Randomization (ASLR) feature makes it impossible for malware to know where APIs are located by loading system DLLs and executables at a different location every time the system boots. Early in the boot process, the Memory Manager picks a random DLL image-load bias from one of 256 64KB-aligned addresses in the 16MB region at the top of the user-mode address space. As DLLs that have the new dynamic-relocation flag in their image header load into a process, the Memory Manager packs them into memory starting at the image-load bias address and working its way down.

Executables that have the flag set get a similar treatment, loading at a random 64KB-aligned point within 16MB of the base load address stored in their image header. Further, if a given DLL or executable loads again after being unloaded by all the processes using it, the Memory Manager reselects a random location at which to load it. Figure 7 shows an example address-space layout for a 32-bit Windows Vista system, including the areas from which ASLR picks the image-load bias and executable load address.

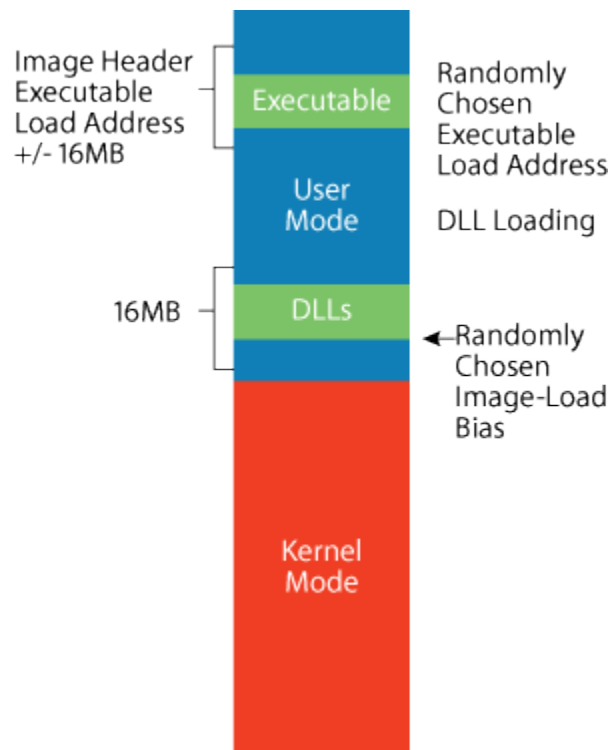


Figure 7 ASLR's Effect on Executable and DLL Load Addresses

Only images that have the dynamic-relocation flag, which includes all Windows Vista DLLs and executables, get relocated because moving legacy images could break internal assumptions that developers have made about where their images load. Visual Studio® 2005 SP1 adds support for setting the flag so that third-party developers can take full advantage of ASLR.

Randomizing DLL load addresses to one of 256 locations doesn't make it impossible for malware to guess the correct location of an API, but it severely hampers the speed at which a network worm can propagate and it prevents malware that only gets one chance at infecting system from working reliably.

Inside the Windows Vista Kernel

Mark Russinovich

In addition, ASLR's relocation strategy has the secondary benefit that address spaces are more tightly packed than on previous versions of Windows, creating larger regions of free memory for contiguous memory allocations, reducing the number of page tables the Memory Manager allocates to keep track of address-space layout, and minimizing Translation Lookaside Buffer (TLB) misses.

Service Security Improvements

Windows services make ideal malware targets. Many offer network access to their functionality, possibly exposing remotely exploitable access to a system, and most run with more privilege than standard user accounts, offering the chance to elevate privileges on a local system if they can be compromised by malware. For this reason, Windows started evolving with changes made in Windows XP SP2 that reduced the privileges and access assigned to services to just those needed for their roles. For example, Windows XP SP2 introduced the Local Service and Network Service accounts that include only a subset of the privileges available to the account in which services always previously ran, Local System. This minimizes the access an attacker gains when exploiting a service.

Seeing ASLR in Action

You can easily see the effects of ASLR by comparing the DLL load addresses for a process in two different boot sessions using a tool like Process Explorer from Sysinternals. In these two screenshots, taken from two different sessions, Ntdll.dll loaded into Explorer first at address 0x77A30000 and then at address 0x77750000.

The screenshot shows Process Explorer with Explorer.exe selected. The DLL list shows ntdll.dll loaded at base address 0x77A30000. The status bar at the bottom indicates CPU Usage: 3.57%, Commit Charge: 25.49%, and Processes: 60.

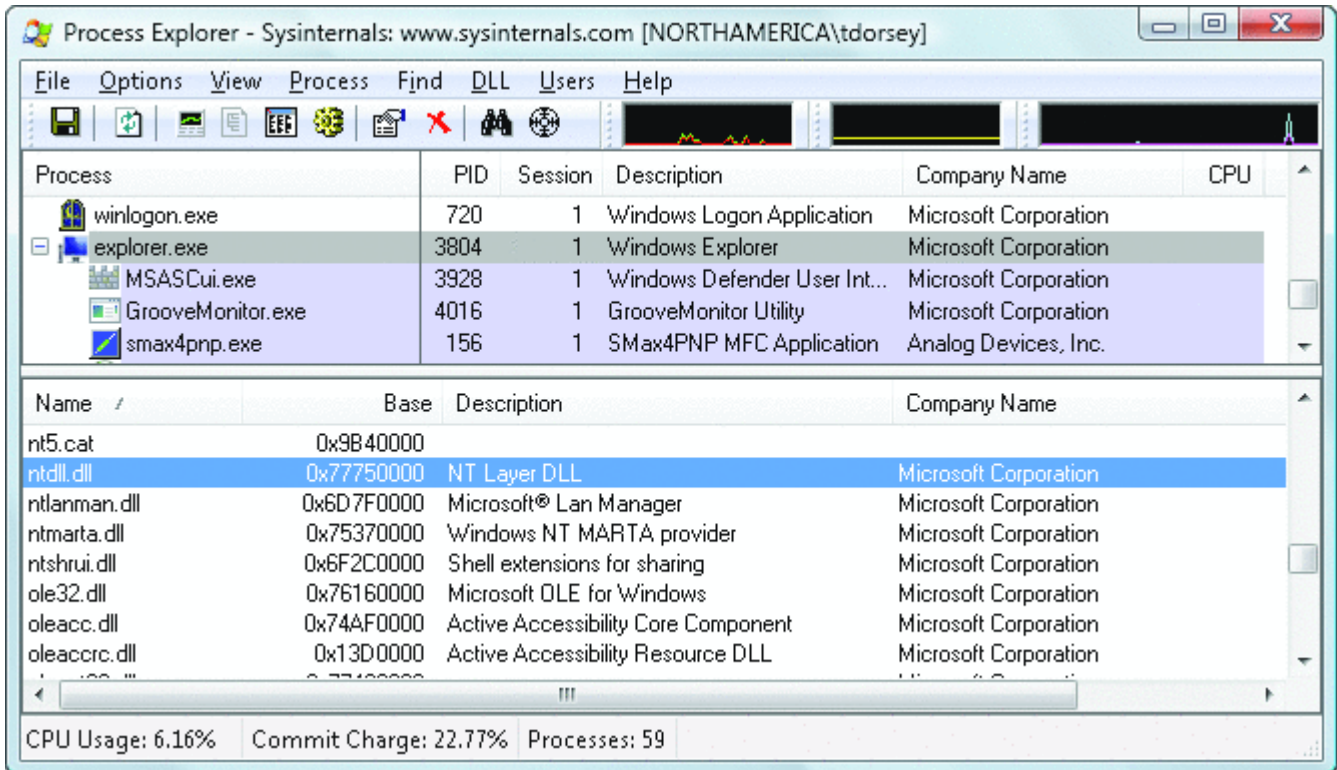
Process	PID	Session	Description	Company Name	CPU
winlogon.exe	736	1	Windows Logon Application	Microsoft Corporation	
explorer.exe	3672	1	Windows Explorer	Microsoft Corporation	
MSASCui.exe	3564	1	Windows Defender User Int...	Microsoft Corporation	
GrooveMonitor.exe	2864	1	GrooveMonitor Utility	Microsoft Corporation	
smax4pnp.exe	2336	1	SMax4PNP MFC Application	Analog Devices, Inc.	

Name	Base	Description	Company Name
nsi.dll	0x769A0000	NSI User-mode interface DLL	Microsoft Corporation
ntdll.dll	0x77A30000	NT Layer DLL	Microsoft Corporation
ntlanman.dll	0x6DA90000	Microsoft® Lan Manager	Microsoft Corporation
ntmarta.dll	0x75660000	Windows NT MARTA provider	Microsoft Corporation
ntshrui.dll	0x6D9D0000	Shell extensions for sharing	Microsoft Corporation
ole32.dll	0x763C0000	Microsoft OLE for Windows	Microsoft Corporation
oleacc.dll	0x74DE0000	Active Accessibility Core Component	Microsoft Corporation
oleaccrc.dll	0x1290000	Active Accessibility Resource DLL	Microsoft Corporation

Different Base Addresses for ntdll.dll

Inside the Windows Vista Kernel

Mark Russinovich



Different Base Addresses for ntdll.dll

In my previous article, I described how services run isolated from user accounts in their own session, but Windows Vista also expands its use of the principle of least privilege by further reducing the privileges and access to the files, registry keys, and firewall ports it assigns to most services. Windows Vista defines a new group account, called a service Security Identifier (SID), unique to each service. The service can set permissions on its resources so that only its service SID has access, preventing other services running in the same user account from having access if a service becomes compromised. You can see a service's SID by using the `sc showsid` command followed by the service name, as seen in Figure 8.

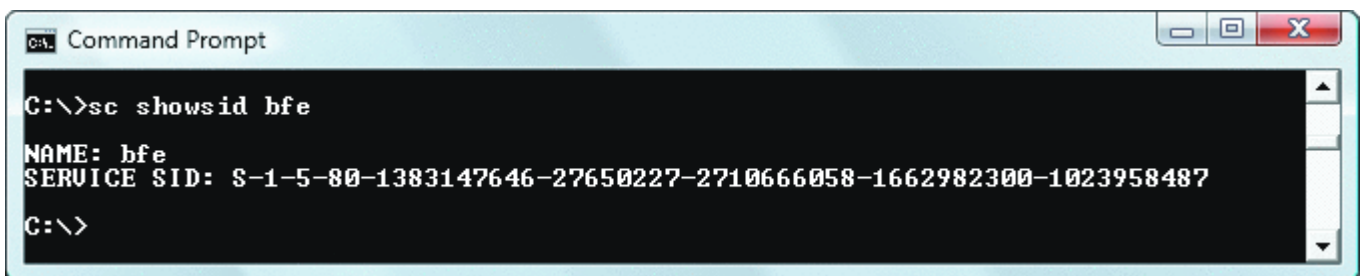


Figure 8 Viewing a service SID

Service SIDs protect access to resources owned by a particular service, but by default services still have access to all the objects that the user account in which they run can access. For example, a service running in the Local Service account might not be able to access resources created by another service running as Local Service in a different process that has protected its objects with

Inside the Windows Vista Kernel

Mark Russinovich

permissions referencing a service SID, however, it can still read and write any objects to which Local Service (and any groups to which Local Service belongs, like the Service group) has permissions.

Windows Vista therefore introduces a new restricted service type called a write-restricted service that permits a service write access only to objects accessible to its service SID, the Everyone group, and the SID assigned to the logon session. To accomplish this, it uses restricted SIDs, a SID type introduced back in Windows 2000. When the process opening an object is a write-restricted service, the access-check algorithm changes so that a SID that has not been assigned to a process in both restricted and unrestricted forms cannot be used to grant the process write access to an object. You can see if a service is restricted with the following command:

```
sc qsidtype [service]
```

Another change makes it easy for a service to prevent other services running in the same account from having access to the objects it creates. In previous versions of Windows, the creator of an object is also the object's owner, and owners have the ability to read and change the permissions of their objects, allowing them full access to their own objects. Windows Vista introduces the new Owner Rights SID, which, if present in an object's permissions, can limit the accesses an owner has to its own object, even removing the right to set and query the permissions.

A further enhancement to the service security model in Windows Vista enables a service developer to specify exactly what security privileges the service needs to operate when the service registers on a system. For example, if the service needs to generate audit events it could list the Audit privilege.

When the Service Control Manager starts a process that hosts one or more Windows services, it creates a security token (the kernel object that lists a process user account, group memberships, and security privileges) for the process that includes only the privileges required by the services in the process. If a service specifies a privilege that is not available to the account in which it runs, then the service fails to start. When none of the services running in a Local Service account process need the Debug Programs privilege, for example, the Service Control Manager strips that privilege from the process's security token. Thus, if the service process is compromised, malicious code cannot take advantage of privileges that were not explicitly requested by the services running in the process. The sc qprivs command reports the privileges that a service has requested.

Conclusion

This concludes my three-part look at Windows Vista kernel changes. There are features and improvements I didn't cover or mention, like a new worker thread pool for application developers, new synchronization mechanisms such as shared reader/writer locks, service thread tagging, support for online NTFS disk checking and volume resizing, and a new kernel IPC mechanism called Advanced Local Procedure Call (ALPC). Look for more information on these and other features in the next edition of Windows Internals, scheduled for publication by the end of 2007.

Viewing Write-Restricted Service

Only one service-hosting process on Windows Vista hosts restricted services and you can identify it with a process-viewing tool like Process Explorer as the one that has the command line:

```
svchost -k LocalServiceNoNetwork
```

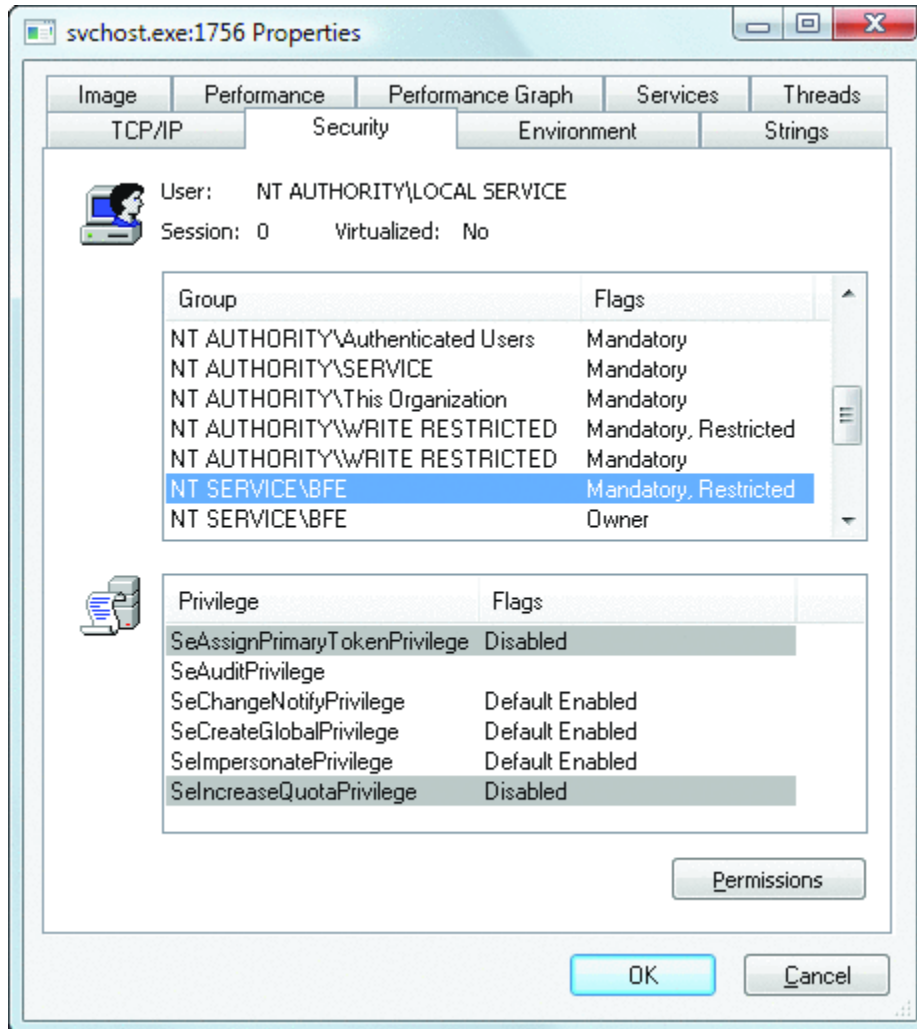
Services configured to run in this process include Base Filtering Engine, Diagnostic Policy Service, Windows Firewall, Performance Logs and Alerts, and Windows Media® Center Service Starter.

Inside the Windows Vista Kernel

Mark Russinovich

This screen shows the textual form of the Base Filtering Engine's service SID, NT SERVICE\BFE, listed once with the Restricted flag and again without it, so the process has access to resources accessible to that account. It doesn't necessarily have access to other objects normally accessible to the Local Service account, however. For instance, because the NT AUTHORITY\SERVICE account doesn't appear in the process token with the restricted flag, the process can't modify objects that grant write access only to that account but not to other accounts in the token that have the restricted flag.

The services running in this process also limit their privileges, because the privileges listed at the bottom of the properties dialog are a subset of those available to the Local Service account.



SID Flags on a Service

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 Tech•Ed and the PDC. He 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.