

09/08/2016 • 23 minutes to read

This article is the second part of a three part series. Click here to read [Part One](#). Click here to read [Part Three](#).

Windows Administration

Inside the Windows Vista Kernel: Part 2

Mark Russinovich

At a Glance:

- Memory management
- Startup and shutdown
- Power management

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](#)).

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-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.

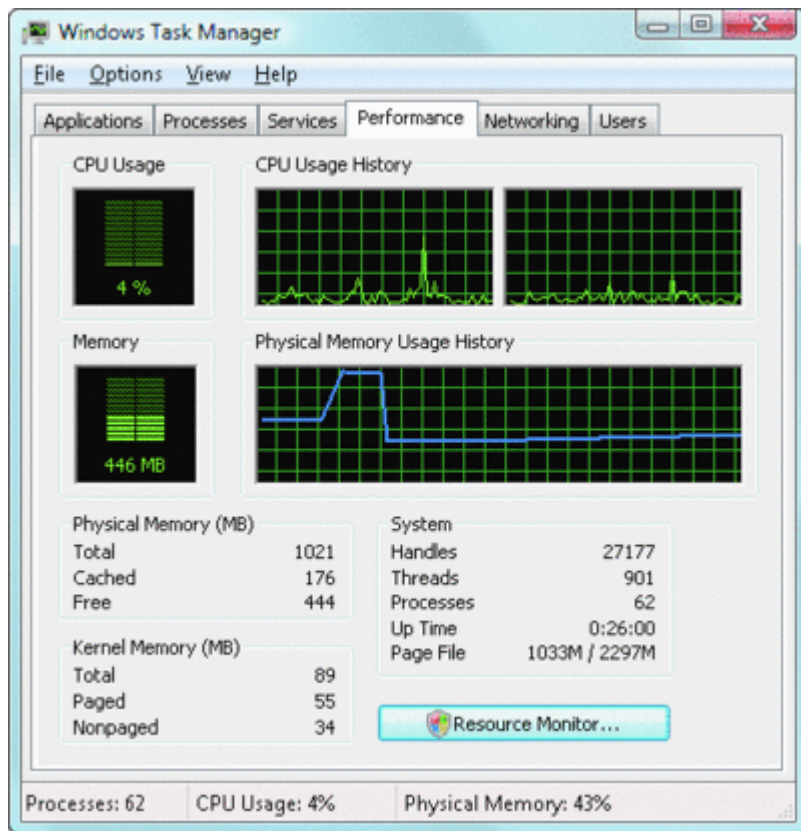
Whenever memory becomes free—for example, when an application exits or releases memory—SuperFetch asks the Memory Manager to fetch data and code that was recently evicted. This is done at a rate of a few pages per second with Very Low priority I/Os so that the preloading does not impact the user or other 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.

### 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(Click the image for a larger view)

## 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.

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.

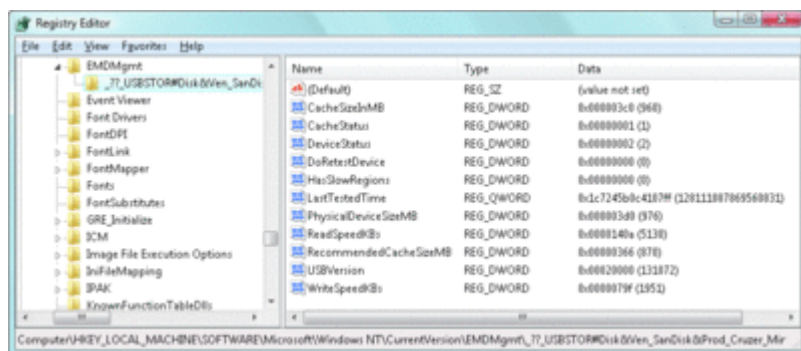


Figure 1\*\* ReadyBoost device test results in the registry \*\*(Click the image for a larger view)

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 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.

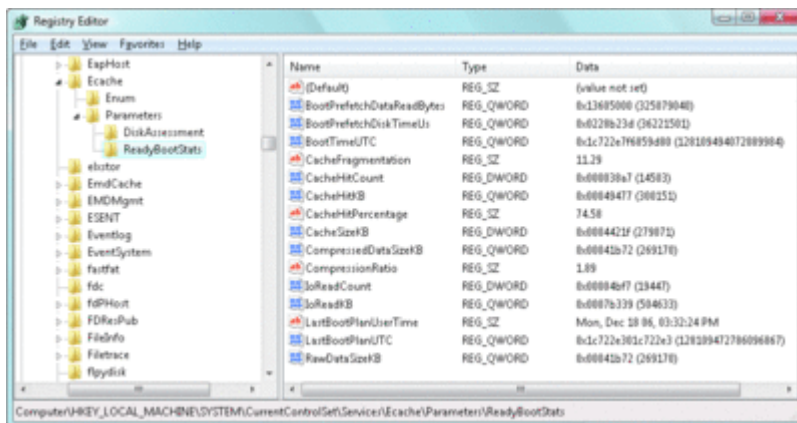


Figure 2\*\* ReadyBoot Performance statistics \*\* (Click the image for a larger view)

## 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.

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

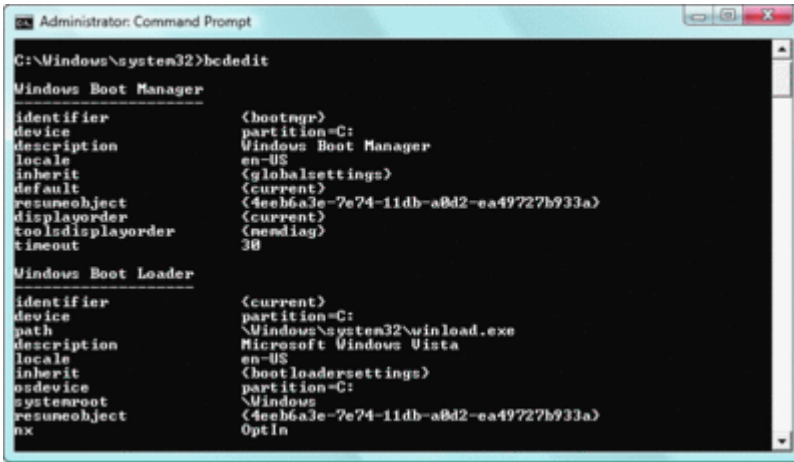
Windows Vista has enhanced several aspects of startup and shutdown. Startup has improved with the introduction of the Boot Configuration Database (BCD) for storing system and OS startup configuration, a new flow and organization of system startup processes, new logon architecture, and support for delayed-autostart 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.





```

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-a8d2-ea49727b933a)
displayorder               (current)
tooldisplayorder          (mendiag)
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-a8d2-ea49727b933a)
nx                         OptIn

```

Figure 3\*\* Settings displayed in BCDEdit \*\*[\(Click the image for a larger view\)](#)

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.

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."

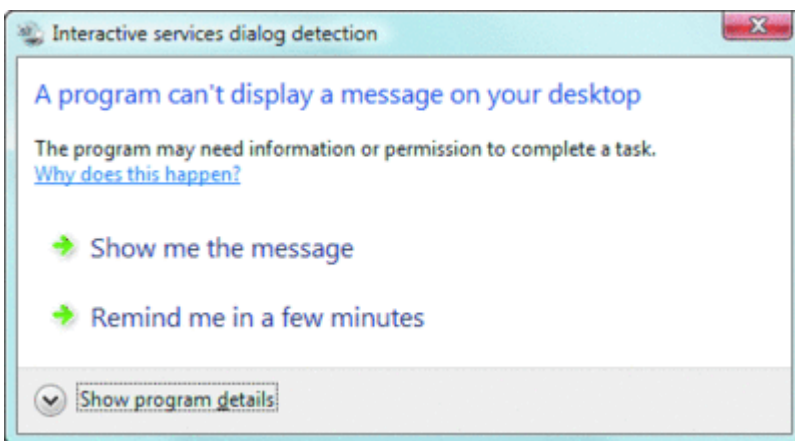


Figure 4\*\* Service has displayed a window \*\*(Click the image for a larger view)

Viewing Startup Process Relationships

You can use Process Explorer from Sysinternals ([microsoft.com/technet/sysinternals](http://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.

Process Explorer identifies a set of processes as running in Session 1 and that's the session I'm logged in 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
usched.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

Startup process and session information(Click the image for a larger view)

## 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

HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows

NT\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.

## 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.

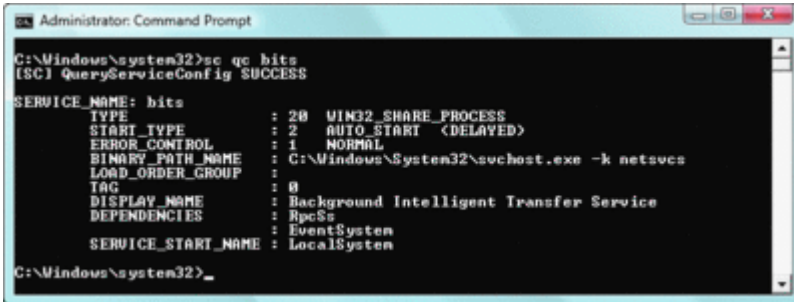
In Windows Vista, the kernel's Power Manager still informs drivers and applications of power-state changes so that they can prepare for them, but it no longer asks for permission. In addition, the Power Manager waits, at most, 20 seconds for applications to respond to change notifications, rather than 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:



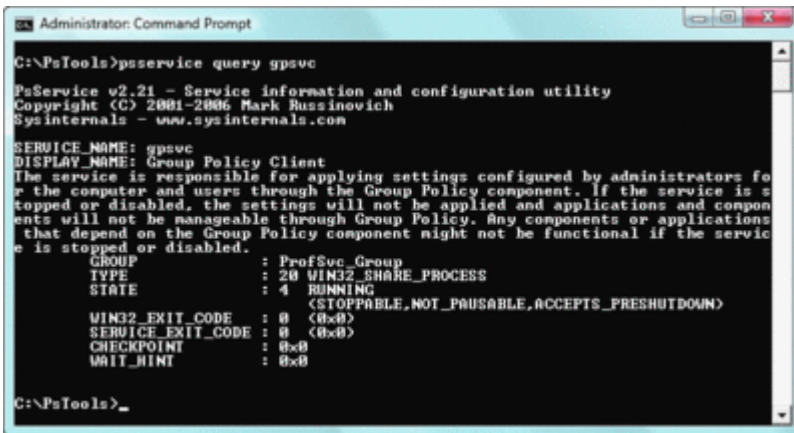
```
Administrator: Command Prompt
C:\Windows\system32>sc qc bits
[SC] QueryServiceConfig SUCCESS

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 netsvc
        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(Click the image for a larger view)

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:



```
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(Click the image for a larger view)

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

© 2008 Microsoft Corporation and CMP Media, LLC. All rights reserved; reproduction in part or in whole without permission is prohibited.