Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS

Mark Russinovich and David Solomon

This article assumes you're familiar with Win32 and C++

Level of Difficulty 1 2 3

SUMMARY The Windows XP kernel includes a number of improvements over Windows 2000 that promote better scalability and overall performance. This article covers these changes and explains how they improve startup time, increase registry size limits, and promote more efficient disk partitioning. Windows XP provides support for 64-bit processors, which is covered here along with a discussion of how side-by-side assemblies end DLL Hell. Also new in the Windows XP kernel is a facility that will roll back driver installations to the Last Known Good state of the registry, making driver installation safer. Other topics include the new volume shadow copy facility, which provides for more accurate backups and improvements in remote debugging.

lthough the number of changes to the Windows® XP kernel is small compared to the changes between Windows NT® 4.0 and Windows 2000 (the internal version number confirms this—Windows 2000 was originally Windows NT 5.0; Windows XP was version 5.1, not NT 6.0), there are a number of important changes that make Windows XP more reliable, more scalable, and more broadly compatible with existing applications.

This article focuses on the kernel changes made to achieve these improvements. It does not cover many other user-mode enhancements in areas such as usability (improved shell and remote assistance), consumer features (CD burner support and DirectX® 8.0), and other changes (terminal services with Windows XP Professional, fast user switching, and the new Windows XP Home Edition). Also, with the exception of System Restore, everything in this article applies to the upcoming Windows .NET Server family of products, since it will be based on the Windows XP kernel (with some additional features).

Scalability and Performance

Windows XP includes a number of kernel changes that were made to permit the system to scale better on multiprocessor systems and to support more users, larger applications, larger files, and larger system memory demands. The following sections describe these changes. Note that most of these improvements are more applicable to server environments, and thus will get more attention when Windows .NET Server is released.

Larger Mapped Files

Because of the way mapped files (internally called sections) were implemented, Windows 2000 had a limit of approximately 200GB on the total size of mapped files in use at any one time. This was because when a file was prepared for mapping (for instance, when the CreateFileMappingObject function was called), the memory manager allocated all the data structures (called prototype page table entries) needed to map the entire file, even though applications might only map views to small parts of the files at any one time. Since these structures were allocated from a finite resource (paged pool, which has a maximum size of 470MB), attempting to map large files would exhaust this resource. This restriction has been lifted in Windows XP (and also in Windows 2000 Service Pack 2); now the memory manager allocates these structures only when mapped views into the file are created.

A side benefit of this change is that it is now possible to perform file backup of any size file on any size system. Previously, you couldn't back up a 500GB file on a 32MB system because there wasn't enough paged pool to create the prototype page table entries for the entire section. Now that these structures are allocated for active views only, large file backups are supported on systems with small physical memory configurations.

Larger Device Drivers and System Space

Windows 2000 limited device drivers to 220MB (drivers were limited to 100MB on Windows NT 4.0). Device drivers on Windows XP can now be up to 960MB in size. This is because the number of available system page table entries (PTEs) has increased such that the 32-bit version of Windows XP can describe 1.3GB of system virtual address space, 960MB of which can be contiguous. Windows 2000 has a limit of 660MB of total system virtual address space (220MB of which could be contiguous). The limit for Windows NT 4.0 was 192MB, of which 100MB could be contiguous. The 64-bit version of Windows XP supports up to 128GB of system virtual address space.

Registry Limits Increased

Like some of the other changes in Windows XP, the Registry doesn't look different and it doesn't provide new

Win32® APIs, but there have been major changes under the hood. The biggest change is that its in-memory storage is no longer provided by paged pool; it's provided by the Cache Manager. The Configuration Manager (the kernel-mode subsystem that implements the Registry) now maps the Registry hives (the on-disk representation of the Registry) into memory much the same way a Win32-based application would memory-map a file.

Prior to Windows XP, the combined sizes of Registry hives could not exceed approximately 376MB (about 80 percent of the maximum paged pool size), but in Windows XP there is no hardcoded upper limit on total Registry size. There is a limit on the size of the System hive (the file that stores the HKEY_LOCAL_MACHINE\System key and its descendants) of 200MB, because of restrictions placed on the operating system boot loader by the environment in which it runs (the boot loader reads the System hive into memory very early in the boot process), but the limit for the System hive was just 12MB in previous versions of Windows. Larger Registries are required to support Terminal Server systems with hundreds of active users.

The Configuration Manager has also been recoded to minimize lock contention on key control blocks, as well as the in-memory data structures that represent Registry keys, and to pack Registry data more tightly in memory as the system is running. Finally, to further minimize memory footprint, the Configuration Manager maintains a security cache in memory to store security descriptors that are used by more than one key.

Larger Minimum Memory Size for Large Pages

Windows 2000 made it easier to find bugs in device drivers by mapping the read-only portions of NTOSKRNL.EXE and HAL.DLL as read-only pages (in Windows NT 4.0 and earlier, these files were mapped entirely as read/write pages). By doing this, if a device driver attempted to modify a read-only part of the operating system, the system would crash immediately with the finger pointing at the buggy driver, instead of allowing the corruption to occur later and result in a system crash that is difficult to diagnose.

However, for performance reasons, Windows 2000 does not attempt to map any parts of the kernel and hardware abstraction layer (HAL) as read-only pages if more than 127MB of physical memory is present. This is because on such systems, these files (and other core operating system data such as initial nonpaged pool and the data structures that describe the state of each physical memory page) are mapped with 4MB "large pages" (as opposed to the normal 4KB page size on the *x*86 processor set). By mapping this core operating system code and data with 4MB pages, the first reference to any byte within each 4MB region results in the *x*86 memory management unit caching the address translation information to find any other byte within the 4MB region without having to resort to looking in page table data structures. This speeds address translation.

The Windows XP change was needed because of the continual increase in typical memory configurations of PC systems. The minimum memory to use large pages is now more than 255MB, instead of more than 127MB. As with Windows 2000, you can force the use of large pages by adding the registry DWORD value HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\Memory Management\LargePageMinimum and setting a very small large page minimum (or to disable large pages and force kernel write protection, use a large value, but this may have performance implications). Finally, note that large pages are never used if Driver Verifier is active.

More Efficient Trimming of the Working Set

The way Windows XP decides which pages to remove in working sets when the system needs to create additional free pages is greatly improved for multiprocessor systems. In Windows 2000, pages in working sets are aged, meaning the system increments a count every time it visits a page in a working set if the page has not been accessed since the last time it was scanned. In this way, when the system needs to trim a working set to create free pages, it can remove pages that have not been referenced the longest. However, this algorithm was performed on uniprocessor systems only. Windows XP page aging is done on multiprocessor systems as well.

Reduced Lock Contention

A number of critical internal locks used to synchronize access to various internal memory management data structures have been either removed completely or optimized, resulting in much less contention. The following operations no longer involve acquiring locks: charging nonpaged and paged pool quotas, allocating and mapping system page table entries, charging commitment of pages, and allocating and mapping physical memory allocated through the address windowing extensions (AWE) functions.

Access to the lock that synchronizes access to the structures that describe physical memory (the PFN database) has been improved. These changes translate into greater parallelism and scalability on multiprocessor systems, since the number of times the memory manager may have to block while another CPU is making a change to a global structure has been reduced or eliminated.

Push Locks

Windows XP introduces a new locking mechanism, called push locks, as a synchronization mechanism for protecting pageable data in kernel mode. The advantages of push locks over the alternative mechanisms used prior to Windows XP are that they require only the size of a pointer in storage (4 bytes on 32-bit Windows XP and 8 bytes on 64-bit Windows XP), and acquisition and release are performed without the use of kernel mode spin locks if there is no contention on the push lock. This new approach to locking improves both performance and scalability in a multiprocessor environment.

Push locks cannot be acquired from interrupt request levels (IRQLs) of DISPATCH_LEVEL or higher, which limits their application to the same situations where mutexes, semaphores, and Executive resources are used. However, these other synchronization objects are larger and utilize spin locks during acquisition and release (which lock a multiprocessor's bus for an instant, which reduces scalability), making push locks attractive. The areas of the operating system where push locks have been retrofitted include the Object Manager, where they protect global Object Manager data structures and object security descriptors, and the Memory Manager, where they protect AWE data structures. The place where they have the most impact on system performance, however, is in their use to protect handle table entries in the Executive.

Fast System Calls

One other performance improvement is in the area of system call dispatching. Those familiar with the internals of Windows NT associate the assembly language instruction "INT 0x2E" with system calls, since it's with this instruction that Windows NT and Windows 2000 transition from user mode to the kernel-mode system call interface where the native API is implemented. Many Win32 APIs invoke system calls. Windows XP uses the SYSENTER/SYSEXIT pair of instructions to transition into and out of kernel-mode for system calls if it's running on a Pentium II or higher. This instruction sequence requires fewer clock cycles to execute, improving the speed of system calls.

Faster System and Application Startup

One of the goals of Windows XP was to improve the user experience. Users consider boot and application startup time to be a big part of the experience. Therefore, developers at Microsoft have spent a great deal of effort on improving the performance of the boot process and application startup. They've addressed this in several ways. Now serial and networking device drivers initialize in parallel, unlike in Windows 2000 where they initialize serially. Logons are allowed sooner, laptops can hibernate and resume more quickly, and applications start faster.

Faster Logon

If your account doesn't depend on a roaming profile, and a domain policy that affects logon hasn't changed since your last logon, Winlogon doesn't wait on the workstation service (which waits on networking services) to start before presenting the logon dialog and allowing a user to log on. This means disconnected laptop users with domain accounts won't be held up during logon as their system times out to look for a domain controller.

Faster Hibernate and Resume

The implementation of hibernation has been revamped for better performance. When the operating system hibernates, it informs device drivers to stop operations on their devices. Then it saves the contents of the computer's memory to a disk file (\hiberfil.sys on Windows 2000 and Windows XP) and powers off the computer. When the computer resumes, the operating system loader reads the contents of the hibernation file into memory and tells device drivers to restart their devices, after which the computer is back to the state it was in before the power-off.

The hibernation improvements come in several areas. The Power Manager compression algorithm, which it uses to compress the contents of memory before writing it to disk, has been improved to both run faster and obtain better compression ratios than in Windows 2000. Second, if the hibernation file is on an IDE drive, the Power Manager uses DMA disk I/O to perform the disk writes. With DMA, disk I/O occurs asynchronously, allowing the Power Manager to overlap compression and disk I/O.

Other changes help resume-from-standby and hibernation performance. For example, the resume code in NTLDR, the component that reads a hibernation file's contents into memory, has been streamlined to perform larger, more sequential reads. The Power Manager's dispatch of system power I/O request packets (IRPs) to drivers has also changed to maximize parallelism. Finally the PCMCIA, mouse, and keyboard class drivers all initialize asynchronously.

In addition to these power management-related performance improvements, Windows XP now supports Intel SpeedStep, AMD PowerNow!, and Transmeta LongRun processor power management. When a system is running on battery (DC power), the Power Manager automatically adjusts the processor's clock rate to accommodate the processing demands of applications, throttling back the speed during idle periods to save power.

Prefetch

All versions of Windows except real-mode Windows 3x are demand-paged operating systems, where file data and code is faulted into memory from disk as an application attempts to access it. Data and code is faulted in page-granular chunks where a page's size is dictated by the CPU's memory management hardware. A page is 4KB on the *x*86. Prefetching is the process of bringing data and code pages into memory from disk before it's demanded.

In order to know what it should prefetch, the Windows XP Cache Manager monitors the page faults, both those that require that data be read from disk (hard faults) and those that simply require that data already in memory be added to a process's working set (soft faults), that occur during the boot process and application startup. By default it traces through the first two minutes of the boot process, 60 seconds following the time when all Win32 services have finished initializing, or 30 seconds following the start of the user's shell (typically Microsoft Internet Explorer), whichever of these three events occurs first. The Cache Manager also monitors the first 10 seconds of application startup. After collecting a trace that's organized into faults taken on the NTFS Master File Table (MFT) metadata file (if the application accesses files or directories on NTFS volumes), the files referenced, and the directories referenced, it notifies the prefetch component of the Task Scheduler by signaling a named event object.

The Task Scheduler then performs a call to the internal NtQuerySystemInformation system call requesting the trace data. After performing post-processing on the trace data, the Task Scheduler writes it out to a file in the \Windows\Prefetch folder. The file's name is the name of the application to which the trace applies followed by a dash and the hexadecimal representation of a hash of the file's path. The file has a .pf extension, so an example would be NOTEPAD.EXE-AF43252301.PF.

An exception to the file name rule is the file that stores the boot's trace, which is always named NTOSBOOT-B00DFAAD.PF (a convolution of the hexadecimal-compatible word BAADF00D, which programmers often use to represent uninitialized data). Only after the Cache Manager has finished the boot trace (the time of which was defined earlier) does it collect page fault information for specific applications.

When the system boots or an application starts, the Cache Manager is called to give it an opportunity to perform prefetching. The Cache Manager looks in the prefetch directory to see if a trace file exists for the prefetch scenario in question. If it does, the Cache Manager calls NTFS to prefetch any MFT metadata file references, reads in the contents of each of the directories referenced, and finally opens each file referenced. It then calls the Memory Manager to read in any data and code specified in the trace that's not already in memory. The Memory Manager initiates all of the reads asynchronously and then waits for them to complete before letting an application's startup continue.

How does this scheme provide a performance benefit? The answer lies in the fact that during typical system boot or application startup, the order of faults is such that some pages are brought in from one part of a file, then from another part of the same file, then pages are read from a different file, then perhaps from a directory, and so on. This jumping around results in moving the heads around on the disk. Microsoft has learned through analysis that this slows boot and application startup times. By prefetching data from a file or directory all at once before accessing another one, this scattered seeking for data on the disk is greatly reduced or eliminated, thus improving the overall time for system and application startup.

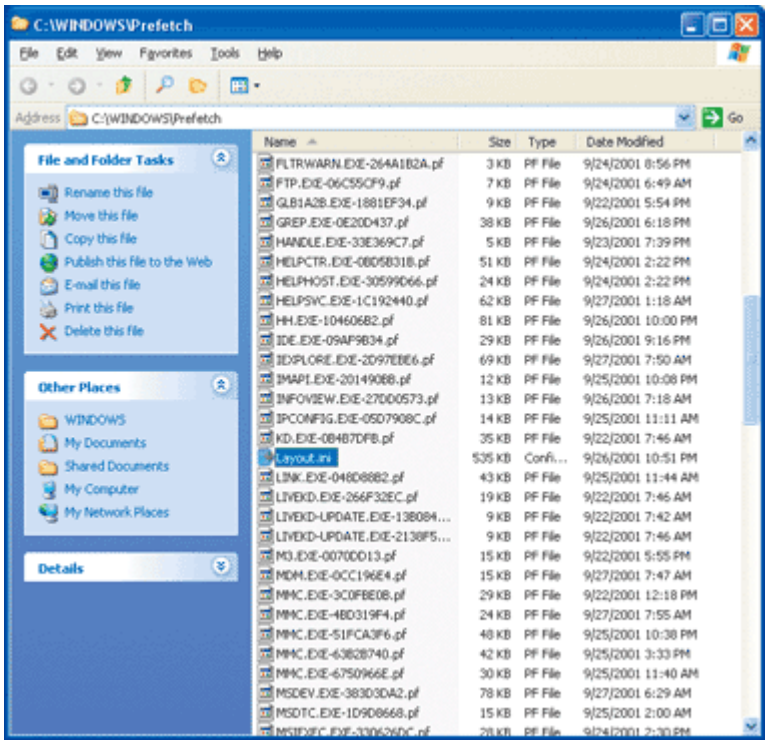

Figure 1 Prefetch Directory

To minimize seeking even further, every three days or so, during system idle periods, the Task Scheduler organizes a list of files and directories in the order that they are referenced during a boot or application start, and stores the list in a file named \Windows\Prefech\Layout.ini. **Figure 1** shows the contents of a prefetch directory, highlighting the layout file. Then it launches the system defragmenter with a command-line option that tells the defragmenter to defragment based on the contents of the file instead of performing a full defrag. The defragmenter finds a contiguous area on each volume large enough to hold all the listed files and directories that reside on that volume and then moves them in their entirety into that area so that they are stored one after the other. Thus, future prefetch operations will even be more efficient because all the data to be read in is now stored physically on the disk in the order it will be read. Since the number of files defragmented for prefetching is usually only in the hundreds, this defragmentation is much faster than full defragmentations.

64-bit Itanium Support

Windows XP 64-bit edition is the first released truly 64-bit version of Windows NT. It runs on the new Intel Itanium processor (at one time internal builds of Windows 2000 were running in native 64-bit mode on the Compaq Alpha AXP processor, but this was never released).

To port Windows XP to Itanium was a major development effort. First, the architecture-specific code in the kernel, memory manager, and HAL had to be written from scratch. This includes support for trap dispatching, context switching, and the new three-level page table structure. Then, thousands of changes were required to get the millions of lines of code that comprise Windows XP to compile and run properly using the native 64-bit compiler and data types. However, the end result is a system that feels like its 32-bit counterpart. In fact, there are virtually no visible differences to the user or administrator other than text on the system properties page and various system display utilities that report processor type, and the fact that the new Visual Styles, like the Luna theme, are not supported on 64-bit Windows; only classic-style Windows is supported.

Since much information about 64-bit support is already available in the MSDN® Library (such as the data type model and how to port Win32 applications), this section focuses on the internal changes to the operating system to support these systems.

64-bit Address Space

The most significant change is, of course, the fact that the virtual address space is huge compared to 32-bit Windows. While 32 bits provides 4GB of address space, 64 bits means over 17 billion GB (16 exabytes) of available address space.

However, the way this address space is divided and laid out is quite different. Whereas 32-bit Windows divides the address space in half—2GB for user processes and 2GB for system space—64-bit Windows provides

7152GB to each user process (.0000416 percent of the total) and uses the rest for page tables, operating system code, and data (see **Figure 2**).

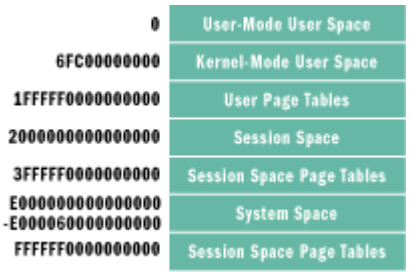

Figure 2 64-bit Address Space Layout

This larger virtual address space means applications can process vast amounts of data in a flat address space without resorting to mapping tricks like the AWE introduced in Windows 2000 that allow 32-bit applications to utilize more than 2GB of memory. Also, since the address space for the operating system is much larger, key system memory pools can be much larger now. This translates to the ability for the system to run more (and bigger) programs, load more (and bigger) device drivers, and cache more data. These larger limits are detailed in **Figure 3**.

64-bit Disk Partitioning

Itanium runs firmware that's compliant with the new Extensible Firmware Interface (EFI), a specification that is maintained by a consortium of companies. Part of EFI is the definition of a disk partitioning scheme called the GUID Partition Table (GPT). The 64-bit edition of Windows XP partitions disks during installation using GPT rather than Master Boot Record (MBR), which is the format Windows uses for *x*86 disks.

GPT offers some advantages over MBR partitioning. For example, all disk offsets in the partition table are 64-bit instead of 32-bit quantities, and the partition table information is mirrored at the start and end of a disk. Furthermore, there's no nesting of partitions as is required in MBR partitioning when there are more than four partitions on a disk.

Just as with the 32-bit version of the OS, the 64-bit edition can be configured to make the Logical Disk Manager (LDM) service manage a disk's partition so that you can define multipartition volumes (volume sets, RAID-5, and so on). LDM preserves the original partitioning scheme (MBR or GPT), but internally manages all the space on a disk not included in the boot partition or the partition table information, thereby creating soft partitions (as opposed to hard partitions defined by the disks main partitioning format). The host partition table defines the LDM spaces as partitions of type LDM.

Interestingly, the Compaq Alpha processor's firmware was programmed to understand the relatively simple FAT file system format, which meant that at least one partition on the Alpha boot disk had to be formatted as FAT. EFI firmware also only understands FAT, so the 64-bit edition of Windows XP boot disk always contains at least one small FAT partition that includes the Windows boot loader file.

Recovery and Reliability

Windows 2000 was already a reliable platform. Windows XP builds on that foundation by adding a number of significant recovery capabilities and reliability improvements such as System Restore, Driver Rollback, Volume Shadow Copy, a more reliable service infrastructure, and new Driver Verifier options. We'll describe these options in detail.

System Restore

System Restore, which originally appeared in a more rudimentary form in Windows Me, provides a way to restore the system to a previously known state that would otherwise require you to reinstall an application or even the entire operating system. For example, if you install one or more applications or make other system file or registry changes that cause applications to fail, you can use System Restore to revert the system files and the Registry to the state it had before the change occurred. System Restore is especially useful when you install an application that makes changes that you would like to undo. Setup applications that are compatible with Windows XP integrate with System Restore to create a restore point before an installation begins.

System Restore's core lies in a service named SrService, which executes from a DLL (\Windows\System32\Srsvc.dll) running in an instance of the Service Host (\Windows\System32\Svchost.exe) process. The service's role is to both automatically create restore points and to export an API so that other applications—such as setup programs—can manually initiate restore point creation. System Restore reads its configuration parameters from HKLM\Software\Microsoft\System Restore, including those that specify how much disk space

must be available for it to operate and at what time interval automated restore-point creation occurs. By default, the service creates a restore point every 24 hours while the system is up, and when the system is off or running on batteries (when automated restore points creation is disabled), it tries to ensure that the latest restore point is no older than 24 hours.

When the System Restore service creates a new restore point it first creates a restore point directory, then snapshots a set of critical system files, including the system and user-profile Registry hives, WMI configuration information, the Microsoft® Internet Information Services (IIS) metabase file (if IIS is installed), and the COM registration database. Then the system restore driver, \Windows\System32\Drivers\Sr.sys, begins to track changes to files and directories, saving copies of files that are being deleted or modified in the restore point, and noting other changes, such as directory creation and deletion, in a restore point change log.

Restore point data is maintained on a per-volume basis, so change logs and saved files are stored under the \System Volume Information_restore{*XX-XXX-XXX*} directory, where the Xs represent the computer's system-assigned GUID of a file's original volume. The restore directory contains restore-point subdirectories having names in the form RPn, where *n* is a restore point's unique identifier. Files that make up a restore point's initial snapshot are stored under a restore point's Snapshot directory.

Backup files copied by the System Restore driver are given unique names such as A0000135.dll in an appropriate restore-point directory that reflect the assignment of an identifier and the preservation of the file's original extension. A restore point can have multiple change logs, each having a name like change.log.*N*, where *N* is a unique change log ID. A change log contains records that store enough information regarding a change to a file or directory so that the change can be undone. For example, if a file was deleted, the change log entry for that operation would store the copy's name in the restore point (A0000135.dll for example) and the file's original long and short file names. The System Restore service starts a new change log when a current one grows larger than 1MB or a certain time has passed. **Figure 4** depicts the flow of file system requests as the System Restore driver updates a restore point in response to modifications.

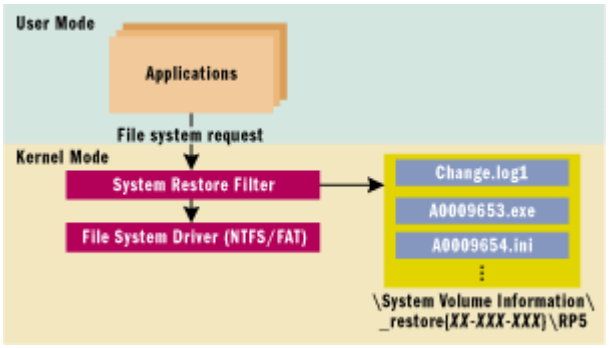

Figure 4 Flow of File System Request

Figure 5 shows a screenshot of a System Restore directory, which includes several restore point subdirectories, as well as the contents of the subdirectory corresponding to restore point 5. Note that the \System Volume Information directories are not accessible to either user or administrator accounts, but are accessible to the Local System account. To see this folder, open an instance of the command prompt running under the Local System account by using the "at" command to run cmd.exe interactively. For example, the command "at 13:44 /interactive cmd.exe" (replace the time with a time in the near future) will, at the designated time, open a command-prompt window that's running in the Local System Account.

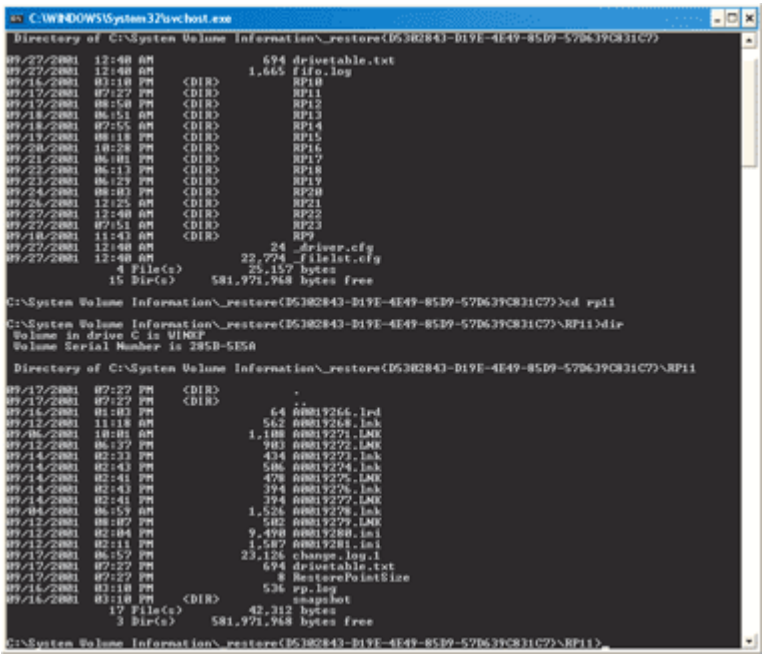

Figure 5 System Restore Directory

The restore point directory on the system volume also stores a file named _filelist.cfg that includes the extensions of files for which changes should be stored in a restore point. This list, which is documented in the Platform SDK, directs System Restore to only track non-data files. For example, you wouldn't want an important Microsoft Word document to be deleted just because you rolled back the system to correct an application configuration problem.

When the user directs the system to perform a restore, the System Restore wizard (\Windows\System32\Restore\Rstrui.exe) creates a DWORD value named RestoreInProgress under the System Restore parameters key and sets it to 1. Then it initiates a system shutdown with reboot by calling the ExitWindowsEx Win32 API. After the reboot, the WinLogon process (\Windows\System32\Winlogon.exe) realizes that it should perform a restore and copies saved files from the Restore Point's directory to their original locations and uses the log files to undo file system changes to files and directories. When the process is complete, the boot continues. Besides making restores safer, the reboot is necessary to activate restored Registry hives.

The Platform SDK documents two System Restore APIs, SRSetRestorePoint and SRRemoveRestorePoint, for use by installation programs. Developers should examine the file extensions that their applications use in light of System Restore. Files that store user data should not have extensions matching those protected by System Restore, because otherwise users could lose data when rolling back to a restore point.

Driver Rollback, Protection, and Last Known Good

Another area where Microsoft has added a recovery capability to improve system reliability is in driver installation. To protect you from the situation where you install a third-party vendor's driver update that introduces problems, the Hardware Installation Wizard (HIW) keeps backup copies of replaced drivers. When you update a driver, the HIW creates a system restore point if the driver is unsigned and then saves the driver being replaced and its driver installation file (INF) in a unique directory with a name having the form \Windows\System32\Reinstallbackups\ *nnnn*\DriverFiles, where the trailing directory is a unique 4-digit decimal identifier. It then creates a corresponding Registry key named HKLM\Software\Microsoft\Windows\CurrentVersion\Reinstall*nnnn* (*nnnn* is an incremented value) where it stores values that identify the replaced driver. If you update the same driver again, the HIW will create a new backup and delete the previous one, thus keeping only the most recent backup. A driver's property page in the Device Manager has a button that lets you roll back the driver to the previous version, as seen in **Figure 6**.

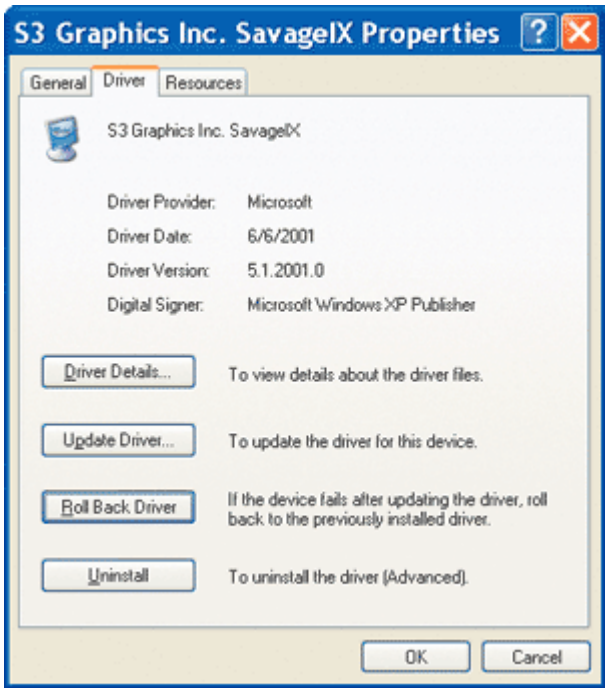

Figure 6 Rolling Back the Driver

The Last Known Good recovery option, with which you access through the boot menu, allows you to revert the system to the last copy of the HKEY_LOCAL_MACHINE\System\CurrentControlSet subkey with which the system successfully booted and a user logged on. It has been integrated with the HIW to make recovery even more likely. One of the most common uses for Last Known Good is to return a system to a bootable state after you've installed a driver that prevents the system from booting successfully—the previous copy of the CurrentControlSet won't have the Registry settings that enable the new driver. However, if you update an existing driver, Last Known Good wouldn't have helped you—prior to Windows XP, that is. Now, when you update a driver the HIW saves the previous version used to boot successfully in \Windows\LastGood\System32\Drivers, and when you select the Last Known Good boot option the updated driver is replaced with the old version.

Windows XP has the same driver-signing policy support as Windows 2000 where you can configure the system to warn you about, prevent, or silently allow the installation of device drivers that haven't been signed by Microsoft (and therefore haven't passed Microsoft driver testing). Windows XP adds to this a new feature called Driver Protection, which consists of a database of drivers that are known to crash systems. When in user mode, the Plug and Play Manager service blocks the installation of any driver listed in the database stored at \Windows\Drvmain.sdb and instead opens a Web page at the Microsoft site that reports the reason why the installation was blocked and where to find updates, if available.

Volume Shadow Copy Service

A limitation of many backup utilities relates to open files. If an application has a file open for exclusive access, a backup utility can't gain access to the file's contents. Even if the backup utility can access an open file, an inconsistent backup could be created. Consider an application that updates a file at its beginning and again at its end. A backup utility that saves the file during this operation might record an image of the file that reflects the start of the file before the application's modification and the end after the modification. If the file is later restored, the application may deem the entire file corrupt, since it might be prepared to handle the case where the beginning has been modified and not the end, but not vice versa. These two problems illustrate why most backup utilities skip open files altogether.

A new facility in Windows XP, called volume shadow copy, allows the built-in backup utility to record consistent views of all files, including open ones. The shadow copy driver is a type of driver, called a storage filter driver, that layers between file system drivers and volume drivers (the drivers that present views of the disk sectors that represent a logical drive) so that it can see the I/O directed at a volume. When the backup utility starts a backup operation it directs the volume shadow copy driver (\Windows\System32\Drivers\Volsnap.sys) to create a volume shadow copy for the volumes that include files and directories being recorded. The volume shadow copy driver freezes I/O to the volumes in question and creates a shadow volume for each. For example, if a volume's name in the Object Manager namespace is \Device\HarddiskVolume0, the shadow volume might be named \Device\HarddiskVolumeShadowCopy*N*, where *N* is a unique ID.

Instead of opening files to back up on the original volume, the backup utility opens them on the shadow volume. A shadow volume represents a point-in-time view of a volume, so whenever the volume shadow copy driver sees a write operation directed at an original volume, it reads a copy of the sectors that will be overwritten into a paging file-backed memory section that's associated with the corresponding shadow volume. It services read operations directed at the shadow volume of modified sectors from this memory section, and services reads to non-modified areas by reading from the original volume. Because the backup utility won't save the paging file or the contents of the system-managed \System Volume Information directory located on every volume, the snapshot driver uses the defragmentation API to determine the location of these files and directories, and does not record changes to them. By relying on the shadow copy facility, the Windows XP backup utility overcomes both of the backup problems related to open files.

The shadow copy driver is actually only an example of a shadow copy provider that plugs into the shadow copy service (\Windows\System32\Vssvc.exe). The shadow copy service acts as the command center of an extensible backup core that enables ISVs to plug in writers and providers. A writer is a software component that enables shadow copy-aware applications to receive freeze and thaw notifications in order to ensure that backup copies of their data files are internally consistent, whereas providers allow ISVs with unique storage schemes to integrate with the shadow copy service. For instance, an ISV with mirrored storage devices might define a shadow copy as the frozen half of a split-mirrored volume. **Figure 7** shows the relationship between the shadow copy service, writers, and providers.

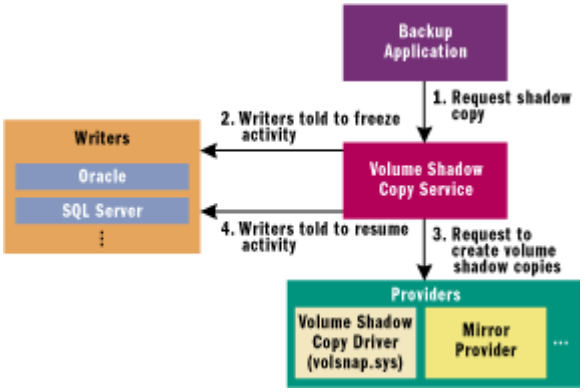

Figure 7 Shadow Copy Service, Writers, and Providers

The entire shadow copy API is currently only available to ISVs under NDA; however, file system drivers must properly handle two documented shadow copy-related I/O control requests (IOCTL): IOCTL_VOLSNAP_FLUSH_AND_HOLD_WRITES and IOCTL_VOLSNAP_RELEASE_WRITES. Their names are self-explanatory. The shadow copy API sends the IOCTLs to the logical drives for which snapshots are being taken so that all modifications initiated before the snapshot have completed when the shadow copy is taken, making the file data recorded from a shadow copy consistent in time.

Services Reliability

The last area of reliability improvements is in the area of the services infrastructure. Prior to Windows 2000, some services shared a process with other services and some ran in their own process. Windows 2000 introduced the generic service host process, Svchost.exe. The goal was to reduce system resources by consolidating the various processes hosting built-in operating system services into a single process. Or, it could permit the system administrator to configure the system to run certain services in their own processes, which would prevent one service from corrupting the private memory of other unrelated services (this capability is not documented or supported yet).

If you look at the Windows XP process list in Task Manager (see **Figure 8**), you will notice at least four Svchost.exe processes: two running under the SYSTEM account (sometimes referred to as LocalSystem) and two running under two new service accounts: NETWORK SERVICE and LOCAL SERVICE.

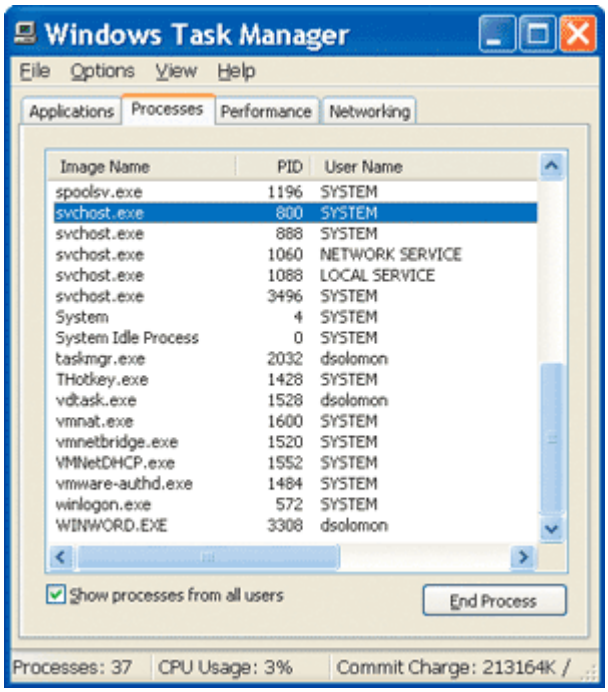

Figure 8 Windows XP Process List in Task Manager

One of the two Svchost processes running under SYSTEM hosts the bulk of the services, 29 of them in total. The second one hosts a single service, Remote Procedure Call (RPCSS). The reason this service needs to be in a separate process is that user-written DLLs are loaded into this process. By having RPC running in its own process, these DLLs cannot adversely affect the operation of the other built-in operating system services. The Svchost process running under NETWORK SERVICE hosts a single service, the DNS Client. The Svchost process running LOCAL SERVICE hosts the TCP/IP NetBIOS Helper, Remote Registry, Simple Service Discovery Protocol, and Web Client services.

The reason for the two new service accounts is to improve system security by reducing the privileges that services run with. LOCAL SERVICE is a built in account that doesn't need a password to log on. The account has only a few privileges, and is not a member of the local administrators group. So, if a service that is running under this account is compromised, it cannot take down the whole machine. LOCAL SERVICE also has no network credentials, so attempts to access a machine on the network will connect with the null session. The NETWORK SERVICE account has the same set of privileges as LOCAL SERVICE, but has access to the machine's credentials for outbound connections, similar to the SYSTEM account.

New Driver Verifier Options

Driver Verifier in Windows 2000 is credited with reducing the number of blue screens that customers faced with Windows NT 4.0 by helping driver writers find more bugs in their code during the testing process. Windows XP adds a few new verification options that increase the rigorous testing of driver operations.

- DMA verification detects improper use of DMA buffers, adapters, and map registers.
- Deadlock detection detects lock hierarchy violations with spinlocks, mutexes, and fast mutexes.
- SCSI verification monitors the interaction between a SCSI miniport driver and the port driver.
- Enhanced I/O Verification tests drivers' support for power management, WMI, and filters.

The user interface has also been improved to make it easier for the administrator to choose verification options, including a new option to automatically verify all unsigned drivers.

File Systems

Windows XP includes a number of enhancements in the area of file systems:

- Windows XP now supports DVD-RAM devices as CD, DVD, and rewritable disks. DVD-RAM media can be formatted with the FAT32 file system.
- The Encrypting File System (EFS) is no longer a separate driver in Windows XP, but instead is integrated into NTFS. EFS now supports multiple-user access to encrypted files, making the sharing of encrypted files possible.

- A new Win32 API, GetVolumePathNamesForVolumeName, returns the list of mount points that refer to a volume given any volume type.
- Defragmentation support is more sophisticated.
- NTFS can mount read-only media and has better security defaults.
- File system filter drivers can register callbacks to intercept fast I/O operations.

Defragmentation Improvements

Windows NT 4.0 introduced file system support for defragmenting files and Windows 2000 added a built-in defragmentation utility. Although there were some improvements to the underlying defragmentation engine in Windows 2000 (for example, support for defragmenting NTFS directories), the implementation had limitations, primarily on NTFS volumes, that prevented defragmentation utilities from being as effective as they otherwise could be.

For example, you could not defragment NTFS volumes with cluster sizes larger then 4KB. Another limitation prevented fine-grained movement of uncompressed NTFS file data—moving a single file cluster moved the 4KB chunk of the file containing the cluster as well. Perhaps the biggest limitation was that it did not defragment NTFS metadata files such as the Master File Table (MFT) or the metadata describing a directory's contents.

Some of these limitations resulted from the fact that the NTFS defrag implementation on Windows NT and Windows 2000 piggy-backs on NTFS support for compressed files. The NTFS compressed file functions map files into memory in 4KB-granular chunks using Cache Manager interfaces, and therefore do not support cluster sizes larger than 4KB.

Microsoft virtually rewrote file system defrag support for Windows XP to remove the dependency on compressed-file routines and the Cache Manager. This means that data movement works at granularity of a single cluster for uncompressed files and that defragmentation works on NTFS volumes with cluster sizes larger than 4KB. Also, defragmentation is now supported on encrypted files.

The other big enhancement is support for online defragmentation of the MFT and most directory and file metadata. Finally, there are a number of odd special cases in the Windows 2000 defragmentation interface that made writing a defragmenter especially challenging. In Windows XP, while the defragmentation API interface has remained unchanged, the way you can use it has improved enormously, which means better defragmentation that will result in better system performance.

NTFS

Prior to Windows XP, NTFS would fail to mount volumes on read-only media, but now it does, and it returns the new FILE_READ_ONLY_VOLUME flag for the GetVolumeInformation API on such volumes. NTFS on Windows XP also introduces the SetFileShortName Win32 API, which allows applications to set a file or directory's short name independently of its long name on NTFS volumes. This functionality is required by backup applications that want to preserve original short names.

Another new Win32 API, SetFileValidData, enables applications to efficiently create files without incurring a performance penalty for zero-filling them. A file has a notion of a valid data length (VDL) and an end-of-file (EOF) where the VDL is always less than or equal to the EOF, and any space between the two is known as the file's tail. Any writes to the tail cause the VDL to grow to the end of the write, and any region between the previous VDL and the start of the write are zero-filled.

Some applications, usually clustered file systems and restore applications, use hardware devices to write data directly into files, bypassing the operating system and file system drivers. Such applications take advantage of the new API to set a file's VDL and then fill the file up to the VDL with data. In addition, some multimedia applications create very large files and don't want to pay the up-front cost of having a file initially zero-filled, which is what NTFS would normally do when a file is sized.

Because NTFS doesn't zero-fill the data up to the VDL defined by SetFileValidData, the API opens a security hole where an application may be able to see data of the file or files that previously occupied the clusters that NTFS assigns the grown file. SetFileValidData therefore requires that the caller have the new SeManageVolumePrivilege enabled, which is by default only assigned to administrators. In the future, this privilege will allow nonadministrators to perform other disk management functions.

An NTFS change that systems administrators interested in security will view as a positive move is the default access control list (ACL) that NTFS applies to new files in directories. Prior to Windows XP the default ACL allowed the Everyone group (to which all users are members) full control, but in Windows XP the default ACL only allows the Administrators group, the System account, and the owner full access—the Users group only has read access.

File System Filter Callbacks

A file system filter driver is a type of device driver that has been supported since the first version of Windows NT. These drivers are widely used in on-access virus scanners, encryption products, and licensing products that

need to see file system changes as they occur. Windows XP introduces changes that affect filter driver writers in its fast I/O routines, the special functions that a file system driver registers so that the Memory Manager, Cache Manager, and I/O system can execute file system I/O and interact with file system drivers without having to generate I/O Request Packets (IRPs).

There are a number of fast I/O routines for which file system filter drivers are not invoked. The routines for which filters are bypassed are those that the Memory Manager calls before and after creating a section backed by a file, ones the Cache Manager calls before and after flushing all or part of a file's modified cached data back to disk, and routines the Memory Manager executes before and after writing dirty mapped-file pages back to a file.

The reason that the system bypasses filters for these functions is that it doesn't trust filter drivers. If a filter doesn't pass these calls down to the underlying file system driver it causes file system data corruption and almost certainly a crash (however, there are a lot of things that a filter driver can do to cause crashes).

Windows XP introduces a new function, FsRtlRegisterFileSystemFilterCallbacks, that filter drivers use to register callbacks for these various operations. This makes it possible for file system filter drivers to examine these operations and even to fail them, and the runtime can ensure that the underlying file system driver is always invoked when appropriate. The Windows XP Installable File System Kit, which can be ordered through http://www.microsoft.com/ddk/ifskit/, provides documentation for this function.

Application Compatibility—Side-by-side Assemblies

A problem that has long plagued Windows users is DLL Hell. You enter DLL Hell when you install an application that replaces one or more core system DLLs, such as those for common controls, the Visual Basic® runtime, or MFC. Application installation programs do this in order to ensure that the application runs properly, but at the same time the updated DLL may have incompatibilities with other installed applications.

Windows 2000 partly addressed DLL Hell by preventing the modification of core system DLLs with the Windows File Protection feature, and by allowing applications to use private copies of these core DLLs. To use a private copy of a DLL instead of the one in the system directory, an application's installation must include a file named *Application*.exe.local (where "Application" is the name of the application's executable), which directs the loader to first look for DLLs in that directory. This type of DLL redirection avoids application/DLL incompatibility problems, but at the expense of not sharing DLLs, which is one of the points of DLLs in the first place. In addition, any DLLs loaded from the list of known DLLs (DLLs that are permanently mapped into memory), or that are loaded by those DLLs, can't be redirected using this mechanism.

To further address application and DLL compatibility while allowing sharing, Windows XP introduces shared assemblies. An assembly consists of a group of resources, including DLLs, and an XML manifest file that describes the assembly and its contents. An application references an assembly through the existence of its own XML manifest. The manifest can be a file in the application's installation directory that has the same name as the application with ".manifest" appended, for instance, *application*.exe.manifest, or it can be linked into the application as a resource. The manifest describes the application and its dependence on assemblies.

There are two types of assemblies, private and shared. The difference between the two is that shared assemblies are digitally signed so that corruption or modification of their contents can be detected, as well as stored under the \Windows\Winsxs directory whereas private assemblies are stored in an application's installation directory. Thus, shared assemblies also have an associated catalog file (.cat) that contains its digital signature information. Shared assemblies can be side-by-side assemblies because multiple versions of a DLL can reside on a system simultaneously, with applications dependent on a particular version using that version.

An assembly's manifest file typically has a name that includes the name of the assembly, version info, some text that represents a unique signature, and ".manifest" as the extension. The manifests are stored in \Windows\Winsxs\Manifests and the rest of the assembly's resources are stored in subdirectories of \Windows\Winsxs that have the same name as the corresponding manifest files, with the exception of the trailing .manifest extension.

An example of a shared assembly is version 6.0 of the Windows common controls DLL, comctl32.dll, which is new to Windows XP. Its manifest file is named \Windows\Winsxs\Manifest\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.0.0_x-ww_1382d70a.manifest. It has an associated catalog file (the same name with the .cat extension), and a subdirectory of Winsxs that includes comctl32.dll.

Version 6.0 of Comctl32.dll includes integration with Windows XP themes, and because applications not written with themes support in mind might not appear correctly with the new DLL, it's only available to applications that explicitly reference the shared assembly containing it—the version of Comctl32.dll installed in \Windows\System32 is an instance of version 5.*x*, which is not theme-aware. When an application loads, the loader looks for the application's manifest, and if one exists, loads the DLLs from the assemblies specified. DLLs not included in assemblies referenced in the manifest are loaded in the traditional way. Legacy applications therefore link against the version in \Windows\System32, whereas theme-aware applications can

specify the new version in their manifest.

A final advantage that shared assemblies bring is that a publisher can issue a publisher configuration, which can redirect all applications that use a particular assembly to use an updated version. They would do this if they were preserving backward compatibility while addressing bugs. Ultimately, however, because of the flexibility inherent in the assembly model, an application could decide to override the new setting and continue to use an older version.

A number of other enhancements don't fall into the categories already covered. They will be described next.

Debugging Improvements

Windows XP has a number of improvements aimed at facilitating debugging of system and application problems. You can now detach the debugger from a process being debugged without causing the target process to terminate (see the new Win32 DebugActiveProcessStop function). For example, you can attach to a process that is experiencing problems but is still running (perhaps serving multiple remote client users), dump the process memory space (with the .dump command), and detach the debugger. The process dump can be analyzed without interrupting the execution of the target process. Debuggers can also use the new DebugSetProcessKillOnExit Win32 function to set a target process to not be killed upon debugger exit.

On systems running Terminal Services, you can now debug applications running in one terminal server session from another session. You can also now use an IEEE 1394 Firewire port as a connection vehicle for kernel debugging, which is much faster than debugging over a serial port connection.

The kernel debugger can now attach to the local system (see the -kl switch on Windbg and Kd) on which it is running (instead of requiring a separate target system). While you can't set breakpoints, you can use the kernel debugger to view internal system state with the many commands that dump internal data structures.

When a device driver is loaded on a system being debugged with the kernel debugger, the system will query the host kernel debugger to see if an updated driver image should be copied from the host through the debug port. This makes it much easier for driver developers, since they don't have to reboot the target into a working system to copy over updated driver images.

Boot and Execute from ROM

Windows XP supports booting and executing applications from ROM. While the operating system and device drivers are copied from ROM into RAM and then executed, user applications can be executed directly from ROM. This support was added to enhance capabilities of the upcoming embedded Windows NT release based on the Windows XP kernel.

New Authorization APIs

Windows XP doesn't introduce any significant changes to its kernel-mode security subsystem, but there's a new user-mode API, called AuthZ. AuthZ can be used by resource management applications such as database servers that protect objects using the standard Windows NT security model, but that require high performance. Instead of representing security contexts with kernel-based access tokens, AuthZ represents them with user-mode data structures. This new approach provides the advantage that security access checks do not require kernel-mode system calls. In addition, an AuthZ client can cache the results of access checks for even better performance.

Winsock

Although Winsock (Windows Sockets) didn't undergo any significant change with the release of Windows 2000, Windows XP introduces a handful of new APIs. For example, getaddrinfo and getnameinfo provide enhanced host name-to-address translations, making gethostbyname and gethostbyaddr obsolete.

The most interesting of the new APIs, ConnectEx, DisconnectEx, and TransmitPackets, provide a benefit for high-performance network servers. ConnectEx enables an application to establish a connection on a socket and optionally send an initial message, and DisconnectEx both disconnects a socket and prepares it for reuse. Both of these calls execute multiple operations in a single system call. TransmitPackets is an extension of the Winsock TransmitFile function that existed before Windows XP. Like TransmitFile, applications use TransmitPackets to send file data directly from the file system cache over the network, thus avoiding memory copy operations. Unlike TransmitFile, TransmitPackets can send any number of buffers in a single invocation, and each can either be a memory buffer or a portion of a file, which makes TransmitPackets much more flexible and gives it scatter/gather characteristics.

The Microsoft Winsock-extensions DLL, \Windows\System32\Mswsock.dll, exposes the new performance-related APIs while the rest are in the Winsock 2 DLL, \Windows\System32\Ws2_32.dll. The Winsock helper driver, \Windows\System32\Drivers\Afd.sys, implements support for the new APIs in kernel-mode. All of the APIs are documented in the latest releases of the Platform SDK.

Another network-related Windows XP feature is that the TCP/IP stack supports IPv6 (in addition to IPv4),

making Windows XP ready for changes as the Internet moves from 4-byte to 16-byte addresses. In fact, besides providing an improved translation interface, part of the purpose of the new getaddrinfo is to provide IPv6 support, since the legacy gethostbyname function doesn't support IPv6 addresses.

Conclusion

Windows XP has a number of improvements to the kernel components that improve system reliability, scalability, and performance. While the changes were not as large-scale as the changes from Windows NT 3.51 to Windows NT 4.0, or from Windows NT 4.0 to Windows 2000, Windows XP builds on the proven reliability of the Windows NT operating system. Many of these changes will be even more important when the Windows .NET Server family (based on the Windows XP kernel) comes out next year. But right now, Windows XP offers this rock-solid operating system foundation to the PC consumer and home user.

For related articles see:
Microsoft Windows XP: What's in It for Developers?
How To Build and Service Isolated Applications and Side-by-Side Assemblies for Windows XP
For background information see:
Inside Windows 2000, 3rd Edition (Microsoft Press, 2000)

Mark Russinovich is chief software architect and cofounder of Winternals Software (http://www.winternals.com). He is coauthor of *Inside Windows 2000*, *3rd Edition* (Microsoft Press, 2000), and founder of the SysInternals Web site (http://www.sysinternals.com).

David Solomon teaches classes on Windows internals to companies worldwide, including Microsoft (see http://www.solsem.com). He is coauthor of *Inside Windows 2000*, *3rd Edition* (Microsoft Press, 2000) and other books on Windows. David also speaks regularly at conferences such as TechEd, WinDev, and the Microsoft PDC.

Figure 3 Size Limits

Resource	32-bit Maximum	64-bit Maximum
Physical Memory	64GB*	128GB
Nonpaged pool	256MB	128GB
Paged pool	470MB	128GB
System Cache	960MB	1024GB (1TB)
User address space	2 or 3GB	7152GB (6.9TB)
System space	2GB	128GB
Single Page file	16TB	32TB
*Physical memory in 32-bit Windows is actually limited by the hardware page table entry format.		

From the December 2001 issue of MSDN Magazine.
Get it at your local newsstand, or better yet, *subscribe*.