Cover Feature

The Windows NT Kernel Architecture

Microsoft is putting the finishing touches on Windows NT 5.0, the largest release to date, with significant new operating system and network functionality being added to many areas of the system.

David A. Solomon
David Solomon
Expert Seminars

When Microsoft embarked upon Windows NT in 1989, several key requirements drove the project. NT was to be a true 32-bit, preemptive, reentrant, virtual memory operating system capable of running on multiple hardware architectures and platforms. NT was intended as a robust, distributed, client-server platform that would scale well on symmetric multiprocessing systems. Finally, NT was to meet government requirements for Posix 1003.1 compliance and for operating system security, be easily adaptable to the global market by supporting Unicode, and run most existing 16-bit MS-DOS and Microsoft Windows 3.1 applications.

To build a system to meet these requirements, developers agreed on the following design goals: portability, reliability and robustness, backward compatibility, performance, and, above all, extensibility. The code had to be written to grow and change as market requirements changed.

Now, some eight years later, Microsoft is putting the finishing touches on Windows NT 5.0, the largest release to date, with significant new operating system and network functionality being added to many areas of the system. It is not, however, a rewrite—many aspects of the kernel system architecture remain the same or are being extended. In other words, the fundamental operating system architecture is *not* changing in Windows NT 5.0.

This article is intended as a brief introduction to the NT architecture, including those features—notably plug and play, the job object, and 64-bit large memory support—that necessitated extensions to the executive and the kernel. (For further details, see the book *Inside Windows NT, 2nd edition*, Microsoft Press.) However, there are many other significant enhancements in the system. Two of these, the Active Directory and the distributed security extensions, are described in a sidebar. The companion articles describe major new extensions to the NT File System and the Microsoft Clustering Service. (Microsoft has posted reams of technical white papers describing the rest of the new features in Windows NT 5.0 at http://www.microsoft.com/windowsnt5.)

This article is based on the author's *Inside Windows NT, 2nd edition*, Microsoft Press, 1998. Used with permission of Microsoft Press.

SYSTEM ARCHITECTURE

Figure 1 is a diagram of the Windows NT system architecture and components. The boxes above the line represent user-mode processes, and the boxes below the line are kernel-mode OS services. User-mode threads execute in a protected process address space. While they are executing in kernel mode, however, they have access to system space. Thus, system processes, server processes (services), the environment subsystems, and user applications each have their own private process address space.

First we will examine the kernel mode components and then the user-mode components.

Kernel mode

Performance-sensitive OS components run in kernel mode, where they can interact with the hardware and with each other without incurring the overhead of context switches and mode transitions. For example, the memory manager, cache manager, object and security managers, network protocols, file systems (including network servers and redirectors), and all thread and process management run in kernel mode. All of these components are fully protected from errant applications, because applications don't have direct access to the code and data of the privileged part of the operating system.

The kernel-mode components of Windows NT also embody basic object-oriented design principles. For

0018-9162/98/$10.00 © 1998 IEEE

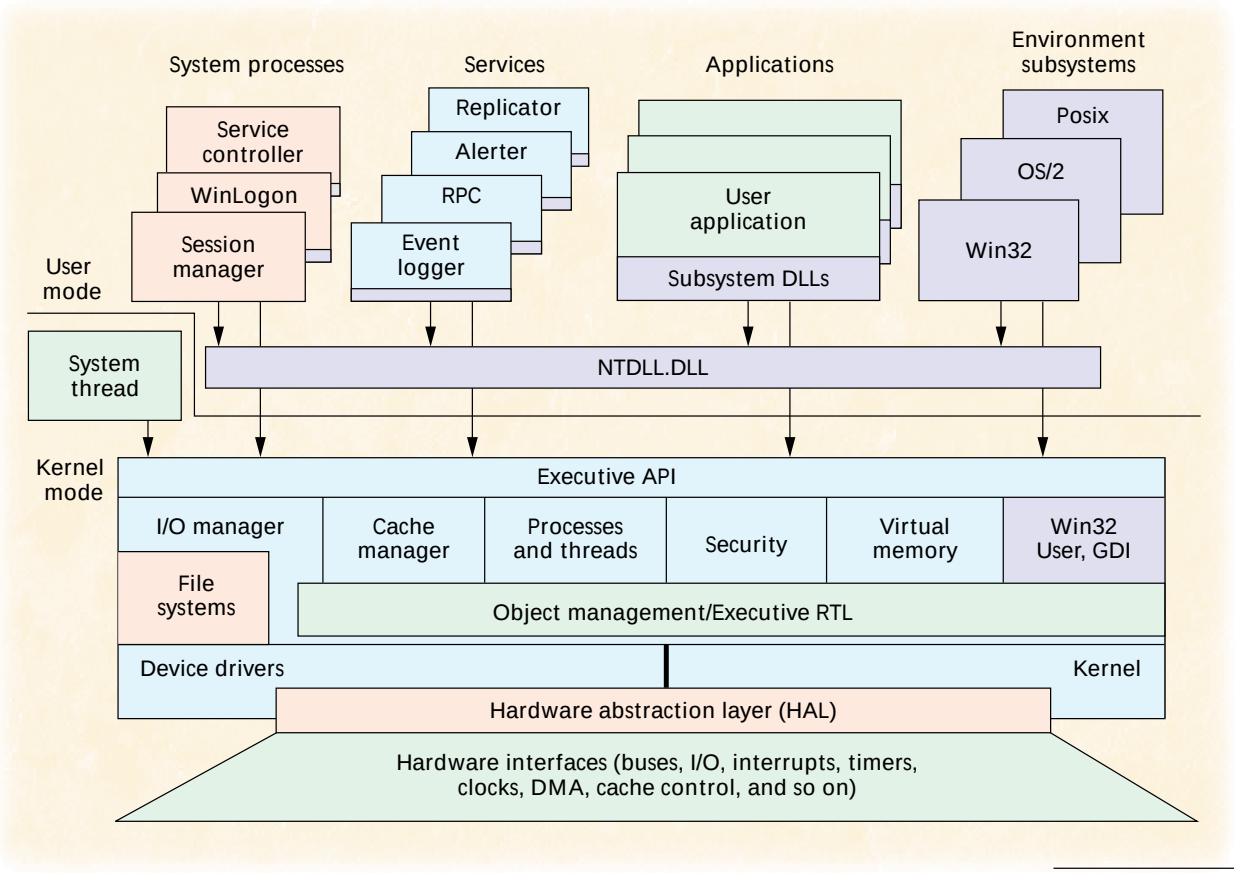

Figure 1. Windows NT system architecture and components. The boxes above the line represent user-mode processes, and the boxes below the line are kernel-mode OS services. User-mode threads execute in a protected process address space. While they are executing in kernel mode, however, they have access to system space.

example, they don't reach into one another's data structures to access information maintained by individual components. Instead, they use formal interfaces to pass parameters and access and/or modify data structures. (Despite its pervasive use of objects to represent shared system resources, Windows NT is not an object-oriented system in the strict sense. Most of the operating system code is written in C for both speed and portability.)

The OS components that run in kernel mode include these:

- The *Windows NT executive* contains the base OS services, such as memory management, process and thread management, security, I/O, and interprocess communication.
- The *Windows NT kernel* performs low-level OS functions, such as thread scheduling, interrupt and exception dispatching, and multiprocessor synchronization. It also provides a set of routines and basic objects that the rest of the executive uses to implement higher-level constructs.
- The *hardware abstraction layer* (HAL) is a layer of code that isolates the kernel, device drivers, and the Windows NT executive from platform-specific hardware differences.
- *Device drivers* include both file system and hardware device drivers that translate user I/O function calls into specific hardware device I/O requests.
- The *windowing and graphics system* implements the graphical user interface (GUI) functions, which deal with windows, controls, and drawing.

The next two sections expand on the roles of the executive, kernel, and HAL. The windowing system is described in the later section on environment subsystems.

Executive

The Windows NT executive is the upper layer of NTOSKRNL.EXE. (The kernel is the lower layer.) It contains the following major components:

- The process and thread manager creates and terminates processes and threads. The underlying

Active Directory

The Active Directory is one of the most important new features in Windows NT 5.0. It will greatly simplify the tasks involved in administering and managing large Windows NT networks, and it will improve the user's interaction with networked resources. The Active Directory is also the key underpinning that enables the improvements in distributed system security.

The Active Directory stores information about all resources on the network and makes this information easy for developers, administrators, and users to find and use. It provides a single, consistent, open set of interfaces for performing common administrative tasks, such as adding new users, managing printers, and locating resources throughout the distributed computing environment.

The Active Directory data model has many concepts similar to X.500. The directory holds objects that represent various resources, which are described by attributes. The universe of objects that can be stored in the directory is defined in the schema. For each object class, the schema defines what attributes an instance of the class must have, what additional attributes it can have, and what object class can be a parent of the current object class. This directory structure has the following key features:

- Flexible hierarchical structure.
- Efficient multimaster replication.
- Granular security delegation.
- Extensible storage of new classes of objects and properties.
- Interoperability through Lightweight Directory Access Protocol (LDAP) version 3 support.
- Scalability to millions of objects per store.
- Integrated dynamic Domain Name System (DNS) server.
- A programmable class store.

Programmability and extensibility are significant capabilities of the Active Directory. Developers and administrators deal with a single set of directory service interfaces, regardless of the installed directory service(s). The programming interface, called the Active Directory Service Interfaces (ADSI), is accessible by any language. You can also access the directory using the LDAP API. The LDAP C API, defined in RFC 1823, is a lower-level interface available to C programmers.

support for processes and threads is implemented in the Windows NT kernel; the executive adds additional semantics and functions to these lower-level objects.
- The virtual memory manager implements virtual memory, a memory management scheme that provides a large, private address space for each process and protects each process's address space from other processes. The memory manager also provides the underlying support for the cache manager.
- The security reference monitor enforces security policies on the local computer. It guards OS resources, performing runtime object protection and auditing.
- The I/O system implements device-independent input/output and is responsible for dispatching to the appropriate device drivers for further processing.
- The cache manager improves the performance of file-based I/O by causing recently referenced disk data to reside in main memory for quick access (and by deferring disk writes by holding the updates in memory for a short time before sending them to the disk).

In addition, the executive contains four main groups of support functions that are used by the executive components just listed:

- The object manager, which creates, manages, and deletes Windows NT executive objects and abstract data types that are used to represent OS resources such as processes, threads, and the various synchronization objects.
- The LPC facility passes messages between a client process and a server process on the same computer. LPC is a flexible, optimized version of remote procedure call (RPC), an industry-standard communication facility for client-server processes across a network.
- A broad set of common runtime library functions, such as string processing, arithmetic operations, data type conversion, and security structure processing.
- Executive support routines, such as system memory allocation (paged and nonpaged pool), interlocked memory access, as well as two special types of synchronization objects: resources and fast mutexes.

Kernel

The kernel performs the most fundamental operations in Windows NT, determining how the OS uses the processor or processors and ensuring that they are used prudently. It is the lowest layer in NTOSKRNL .EXE. These are the primary functions the kernel provides

- Thread scheduling and dispatching.
- Trap handling and exception dispatching.
- Interrupt handling and dispatching.
- Multiprocessor synchronization.
- Providing the base kernel objects that are used (and in some cases exported to user mode) by the executive.

Distributed Security Extensions

Windows NT 5.0 Distributed Security Services has many new features to simplify domain administration, improve performance, and integrate Internet security technology based on public-key cryptography.

The Active Directory provides the store for all domain security policy and account information and is available for remote administration. It supports a multilevel hierarchical tree namespace for user, group, and machine account information. Accounts can be grouped by organizational units rather than with the flat-domain account namespace provided by earlier versions of Windows NT. Management of trust relationships among domains is simplified through treewide transitive trust throughout the domain tree. Finally, administrator rights to create and manage user or group accounts can be delegated to the level of organizational units. Access rights can be granted to individual properties on user objects to allow, for example, a specific individual or group to have the right to reset passwords but not to modify other account information.

Windows NT security includes new authentication based on Internet standard security protocols, including Kerberos version 5 and transport layer security (TLS) for distributed security protocols, in addition to supporting Windows NT LAN Manager authentication protocols for compatibility. The implementation of secure channel security protocols supports strong client authentication by mapping user credentials in the form of public-key certificates to existing Windows NT accounts. Common administration tools are used to manage account information and access control, whether you are using shared secret authentication or public-key security.

Windows NT provides the Microsoft Certificate Server for organizations to issue X.509 version 3 certificates to their employees or business partners. Windows NT 5.0 introduces CryptoAPI certificate management APIs and modules to handle public-key certificates, including standard format certificates issued by either a commercial certificate authority, third-party certificate authority, or the Microsoft Certificate Server included in Windows NT.

The kernel is different from the rest of the executive in several ways. Unlike other parts of the executive, the bulk of the kernel is never paged out of memory. Similarly, although the kernel can be interrupted to execute an interrupt service routine, its execution is never preempted by another running thread. The kernel always runs in kernel mode and does not probe accessibility of parameters, since it assumes that its callers know what they are doing. The kernel code is written primarily in C, with assembly code reserved for those tasks that require the fastest possible code or that rely heavily on the capabilities of the processor.

Kernel objects. One goal for the kernel was to provide a low-level base of well-defined, predictable OS primitives and mechanisms that would allow higher-level components of the executive to do what they need to do. The kernel separates itself from the rest of the executive by implementing operating system mechanisms and avoiding policy making. It leaves nearly all policy decisions to the executive, with the exception of thread scheduling and dispatching, which the kernel implements.

The kernel implements a set of simpler objects, called kernel objects, that help the kernel control central processing and support the creation of executive objects. Most executive-level objects encapsulate one or more kernel objects, incorporating their kernel-defined attributes. One set of kernel objects, called control objects, includes the kernel process object, the APC object, the deferred procedure call (DPC) object, and several objects used by the I/O system, such as the interrupt object. Another set of kernel objects, known as dispatcher objects, incorporates synchronization capabilities that alter or affect thread scheduling. The dispatcher objects include the kernel thread, mutex (called mutant internally), event, kernel event pair, semaphore, timer, and waitable timer.

Hardware support. The other major job of the kernel is to abstract or isolate the executive and device drivers from variations between the hardware architectures supported by Windows NT: the x86 and Alpha AXP. The two key components that provide operating system portability are the kernel and the HAL (described in the next section). Functions that are architecture-specific (such as thread context switching) are implemented in the kernel. Functions that can differ from machine to machine within the same architecture are implemented in the HAL.

The kernel supports a set of interfaces that are both portable and semantically identical across architectures. Some of these interfaces are implemented differently on different architectures, however, and others are partially implemented with architecture-specific code.

These architecturally independent interfaces can be called on any machine, and the semantics of the interface will be the same whether or not the code varies by architecture. Some kernel interfaces, such as the SMP spinlock synchronization routines, are actually implemented in the HAL because their implementation can vary for systems within the same architecture family. Other examples of architecture-specific code in the kernel include the interface to provide translation buffer and CPU cache support. This support requires different code for the different architectures because of the way caches are implemented.

Hardware abstraction layer (HAL)

The HAL is a loadable kernel-mode module that provides the low-level interface to the hardware platform on which Windows NT is running. It hides hard-

Rather than access hardware directly, Windows NT internal components as well as user-written device drivers maintain portability by calling the HAL routines.

ware-dependent details such as I/O interfaces, interrupt controllers, and multiprocessor communication mechanisms—any functions that are architecture-specific and machine-dependent.

Rather than access hardware directly, Windows NT internal components as well as user-written device drivers maintain portability by calling the HAL routines when they need platform-dependent information.

Device drivers. Device drivers are loadable kernel-mode modules (typically ending in .SYS) that interface between the I/O system and the relevant hardware. Drivers are typically written in C (sometimes C++) and therefore, with proper use of HAL routines, can be source-code portable across the CPU architectures supported by Windows NT and binary portable within an architecture family.

There are several types of device drivers:

- Hardware device drivers manipulate hardware (using the HAL) to write output to or retrieve input from a physical device or network.
- File system drivers are Windows NT drivers that accept file-oriented I/O requests and translate them into I/O requests bound for a particular device.
- Filter drivers, such as those that perform disk mirroring and encryption, intercept I/Os and perform some added-value processing before passing the I/O to the next layer.
- Network redirectors and servers are file system drivers that transmit remote I/O requests to a machine on the network and receive such requests, respectively.

Because installing a device driver is the only way to add user-written kernel-mode code to the system, some programmers have written device drivers simply as a way to access internal operating system functions or data structures that are not accessible from user mode.

User processes

The four basic types of user processes are described in the following list:

- *Special system support processes*, such as the logon process and the session manager, which are not Windows NT services.
- *Server processes* that are Windows NT services (the equivalent of Unix daemon processes), such as the Event Logger. Many add-on server applications, such as Microsoft SQL Server and Microsoft Exchange Server, also include components that run as Windows NT services.
- *Environment subsystems*, which expose the native OS services to user applications and thus provide an OS environment, or personality. Windows NT ships with three environment subsystems: Win32, Posix, and OS/2 1.2.
- *User applications*, which can be one of five types: Win32, Windows 3.1, MS-DOS, Posix, or OS/2 1.2.

Environment subsystems and subsystem DLLs

As Figure 1 shows, Windows NT has three environment subsystems: Posix, OS/2, and Win32. OS/2 is available only for x86 systems. The Win32 subsystem is special in that Windows NT can't run without it.

Environment subsystems expose some subset of the base Windows NT executive system services to application programs. Each subsystem can provide access to different subsets of the native services in Windows NT. That means that some things can be done from an application built on one subsystem that can't be done by an application built on another subsystem. For example, a Win32 application can't use the Posix fork function.

Each executable image (.EXE) is bound to one and only one subsystem. When an image is run, the process creation code examines the subsystem type code in the image header so that it can notify the proper subsystem of the new process.

Under Windows NT, user applications do not call the native Windows NT OS services directly; rather, they go through one or more environment subsystem dynamic-link libraries (DLLs). The role of the environment subsystem DLLs is to translate a documented function into the appropriate undocumented Windows NT system service calls. These DLL libraries export the documented interface that the programs linked to that subsystem can call. For example, the Win32 subsystem DLLs implement the Win32 API functions. The Posix subsystem DLL implements the Posix 1003.1 API.

Win32 subsystem. The major components of the Win32 subsystem are the environment subsystem process and kernel-mode device driver. The environment subsystem process supports

- console (text) windows,
- the creation and deletion of processes and threads,
- portions of the support for 16-bit virtual DOS machine (VDM) processes, and
- other miscellaneous functions, such as GetTempFile, DefineDosDevice, ExitWindowsEx, and several natural-language support functions.

The kernel-mode device driver supports

- The window manager, which controls window displays, manages screen output, collects input

from keyboard, mouse, and other devices, and passes user messages to applications.
- The Graphical Device Interface (GDI), a library of functions for graphics output devices, with functions for line, text, and figure drawing and for graphics manipulation.
- Graphics device drivers, which are hardware-dependent graphics display drivers, printer drivers, and video miniport drivers.
- Several subsystem DLLs, which translate documented Win32 API functions into the appropriate undocumented kernel-mode system service calls to NTOSKRNL.EXE and WIN32K.SYS.

Applications call the standard user functions to create windows and buttons on the display. The window manager communicates these requests to the GDI, which passes them to the graphics device drivers, where they are formatted for the display device.

GDI provides a set of standard functions that let applications communicate with graphics devices, including displays and printers, without knowing anything about the devices. GDI interprets application requests for graphic output and sends them to graphics display drivers. It also provides a standard interface for applications to use varying graphics output devices. This interface enables application code to be independent of the hardware devices and their drivers.

NTDLL.DLL. NTDLL.DLL is a special system support library primarily for the use of subsystem DLLs. It contains two types of functions:

- System service dispatch stubs to Windows NT executive system services.
- Internal support functions used by subsystems, subsystem DLLs, and other native images.

The first group of functions provides the interface to the Windows NT executive system services that can be called from user mode. There are more than 200 such functions, such as NtCreateFile, NtSetEvent, and so on. For each of these functions, NTDLL contains an entry point with the same name. The code inside the function contains the architecture-specific instruction that causes a transition into kernel mode to invoke the actual kernel-mode system service that contains the real code inside NTOSKRNL.EXE.

NTDLL also contains many support functions, such as the image loader, the heap manager, and Win32 subsystem process communication functions, as well as general runtime library routines. It also contains the user-mode asynchronous procedure call (APC) dispatcher and exception dispatcher.

NEW FEATURES IN THE NT 5.0 KERNEL

Despite NT 5.0's extensible architecture, several of the new features in NT 5.0 entailed some important changes in the kernel architecture—namely, plug-and-play, power management, the job object, and very large memory extensions for Alpha.

Despite NT 5.0's extensible architecture, several of the new features in NT 5.0 entailed some important changes in the kernel architecture.

Plug-and-play

Plug-and-play (PnP) is a combination of hardware and software support that enables a computer system to recognize and adapt to hardware configuration changes with little or no user intervention. With PnP, a user can add or remove devices dynamically, without manual configuration and without any intricate knowledge of computer hardware. For example, a user can dock a portable computer and use the docking station's Ethernet card to connect to the network without changing the configuration. Later, the user can undock that same computer and use a modem to connect to the network—again without making any manual configuration changes.

The evolution of PnP. Support for PnP was first provided in Windows 95; since that time, however, PnP has evolved dramatically. This evolution is largely a result of the OnNow design initiative, which defines a comprehensive, systemwide approach to controlling system and device configuration and power management. One product of the OnNow initiative is the Advanced Configuration and Power Interface (ACPI) version 1.0 specification, which defines a new system board and BIOS interface that extends PnP data to include power management and other new configuration capabilities, all under complete control of the operating system.

The ACPI methods defined are independent of the operating system or the CPU. ACPI specifies a register-level interface to core PnP and power management functions and defines a descriptive interface for additional hardware features. This arrangement gives system designers the ability to implement a range of PnP and power management features with different hardware designs while using the same operating system driver. ACPI also provides a generic system-event mechanism for PnP and power management.

Windows NT 5.0 implementation. The Windows NT 5.0 PnP architecture is designed to meet the following two goals:

- Extend the existing Windows NT input/output infrastructure to support PnP and power management while also supporting industry hardware standards for PnP.
- Achieve common device driver interfaces that support PnP and power management for many device classes under Windows NT 5.0 and Windows 98.

Windows NT 5.0 support for PnP includes initial system installation, recognition of PnP hardware changes that occur between system boots, and response to runtime hardware events such as dock/undock and device insertion/removal.

Drivers for PnP devices do not assign their own resources. Instead, the required resources for a device are identified when the operating system enumerates the device. The PnP Manager retrieves the requirements for each device during resource allocation. Based on the resource requests each device makes, the PnP Manager assigns the appropriate hardware resources such as I/O ports, IRQs, DMA channels, and memory locations. The PnP Manager reconfigures resource assignments when needed, such as when a device is added to the system and requests resources that are already in use. The PnP Manager determines which drivers are required to support a particular device and loads those drivers.

One of the key features of both PnP and power management is dynamic handling of events. The addition or removal of a device is an example of such a dynamic event, as is the ability to awaken a device or put it to sleep. PnP and power management both use WDM-based functions and have similar methods for responding to dynamic events.

Driver changes. The native Windows NT 5.0 PnP support results in the following changes for developers who previously created drivers under the Windows NT 4.0 device driver model:

- *Bus drivers separate from the HAL*. Bus drivers control an I/O bus, including per-slot functionality that is device-independent. In the new architecture, bus drivers have moved out of the hardware abstraction layer (HAL) to coordinate with changes and extensions made to existing kernel-mode components, such as the executive, device drivers, and the HAL. (Microsoft generally provides bus drivers.)
- *New methods and capabilities to support device installation and configuration*. The new design includes changes and extensions to existing user-mode components, such as the Spooler, class installers, Control Panel applications, and Setup. In addition, new kernel-mode and user-mode PnP-enabled components have been added.
- *New PnP APIs to read and write information from the registry*. For the new design, changes and extensions were made to the registry structure. This structure supports PnP and allows the registry to be enhanced in future versions of Windows NT, while also providing backward compatibility.

Windows NT 5.0 will support legacy Windows NT drivers, but since these don't support PnP and power management, systems running these drivers will have reduced capabilities in these two areas. Manufacturers who want to support complete PnP capabilities for their devices and who want the same drivers to function on both Windows NT and Windows 98 will need to develop new drivers that integrate the latest PnP and power management functionality.

Job object

Windows NT 5.0 contains an extension to the process model called a *job*. A job object is a nameable, securable, sharable object that controls certain attributes of processes associated with the job. A job object's basic function is to allow groups of processes to be managed and manipulated as a unit. In some ways, the job object compensates for the lack of a structured process tree on Windows NT (yet in many ways is more powerful than a Unix-style process tree). The job object also records basic accounting information for all processes associated with the job and for all processes that were associated with the job but have since terminated.

A job object can contain limits that are forced on each process associated with the job, limits such as these:

- Jobwide user-mode CPU time limit.
- Per-process user-mode CPU time limit.
- Maximum number of active processes.
- Job process priority class.

Jobs can be set to queue an entry to an I/O completion port object, which other threads might be waiting on with the Win32 GetQueuedCompletion Status function. You can also place security limits on processes in a job. Finally, you can also place user interface limits on processes in a job. Such limits include being able to restrict processes from opening handles to windows owned by threads outside the job, reading and/or writing to the clipboard, and changing the many user interface system parameters via the Win32 SystemParametersInfo function.

Very large memory on Alpha

Windows NT 5.0 adds support for accessing up to 28 Gbytes of memory on Digital Alpha AXP systems. This extension is called VLM, for *Very Large Memory*. VLM won't be supported on the x86 family of processors. However, Microsoft is building a true 64-bit version of Windows NT that will run on the Alpha AXP and the upcoming Intel IA64 architecture. (See the "64-bit Windows NT" sidebar.)

Windows NT 4.0 has always supported 64-bit offsets for file I/O operations but has limited each process to a private 2-Gbyte (or on Windows NT Server, Enterprise Edition, a 3-Gbyte) address space. VLM

64-bit Windows NT

In 1997, Microsoft announced plans to implement a full 64-bit version of Windows NT. The target processor architectures are the existing 64-bit Alpha AXP platform and the upcoming Intel 64-bit (IA64) processors (the first implementation code-named Merced). A full 64-bit Windows NT means that each 64-bit process will have a large, flat address space (initially at least 512 Gbytes in size). The reason for supporting this platform is the same reason Microsoft moved from a 16-bit to a 32-bit address space—ever increasing requirements for storing and processing huge amounts of data in memory.

Although the VLM extensions in Windows NT 5.0 alleviate to some degree the address space limitations in 32-bit Windows NT, they present a variation of the process address space that requires special application support. Also, processes are limited to the storing of data in the large memory area. Finally, only the VLM-enabled Win32 API functions support 64-bit pointers—many other system functions don't. In 64-bit Windows NT, all APIs that accept pointers will accept 64-bit pointers.

In 64-bit Windows NT, the Win32 API will be extended to a true 64-bit programming interface called *Win64*. Various parameters that currently reference 32-bit memory addresses will be widened to 64-bit pointers. This new API will be designed to make porting from Win32 as straightforward as possible and will allow a vendor to maintain a single source that can be compiled to produce both a Win32 and Win64 binary. The 64-bit Windows NT will run existing 32-bit applications on both Alpha and Merced and new 64-bit applications in native 64-bit mode. However, mixing of the two within the same process won't be permitted.

allows an application to use 64-bit pointers and therefore directly access an address space that is much larger than 2 Gbytes. This extension is being implemented to meet the needs of data-intensive applications, such as database management systems, in which the application needs to keep a large amount of information in physical memory. Restricting that information to fit within the 2-Gbyte (or even the 3-Gbyte) address space isn't acceptable for such applications.

With VLM, applications on Alpha systems can address up to 28 Gbytes of memory beyond their 2-Gbyte private address space. To provide access to the large memory area, a 64-bit pointer data type is required. (See __ptr64 added in Visual C++ 5.0 for Alpha.) A small set of new Win32 APIs is being added to operate on memory using 64-bit pointers.

VLM has an important restriction on the virtual addresses that can be addressed by 64-bit pointers, namely that such addresses must be backed by locked-down physical memory. In other words, all committed VLM memory must be backed by physical memory that must be available at the time the address space is committed. The virtual address space used to map this memory can be reserved, but the actual memory committed must be backed by physical memory. Page faults are never taken on VLM addresses unless these addresses are being used to map an actual data file; hence, paging files are never used for any of this memory. Page faults are taken only on mapped data files, and each page will be faulted only once (when first accessed) and then effectively locked into memory, never to be removed.

Despite these restrictions, VLM does meet the need for the class of applications described above that demand high-speed access to large portions of memory.

The architecture of Windows NT reflects a modern, 32-bit operating system design. With Windows NT 5.0, it continues to evolve to meet everchanging requirements of being both a desktop/workstation operating system as well as one for running high-end server applications. ❖

David A. Solomon, president of David Solomon Expert Seminars, teaches seminars on Windows NT internals and systems programming. He is the author of Inside Windows NT *(Microsoft Press) and* Windows NT for OpenVMS Professionals *(Digital Press/Butterworth Heinemann). Formerly a consulting software engineer at Digital Equipment Corporation, he worked for more than nine years as a project leader and developer in the VMS operating system development group. Solomon has served as technical chair for several Windows NT conferences.*

Contact Solomon at David Solomon Expert Seminars, 5 Partridge Trail, Sherman, CT 06784; e-mail daves@solsem.com.