Sample Chapter

Inside Windows NT(R), Second Edition

David A. Solomon,
based on the original edition by Helen Custer

ISBN: 1-57231-677-2

Chapter 2: System Architecture

System Architecture

Now that we’ve covered the terms, concepts, and tools you need to be familiar with, we’re ready tc
exploration of the internal design goals and structure of Microsoft Windows NT. This chapter expl
overall architecture of the system--the key components, how they interact with each other, and the
in which they run. To provide a framework for understanding the internals of Windows NT, let’s fi
review the requirements and goals that shaped the original design and specification of the system.

Requirements and Design Goals

The following requirements drove the specification of Windows NT back in 1989:

- Provide a true 32-bit, preemptive, reentrant, virtual memory operating system
- Run on multiple hardware architectures and platforms
- Run and scale well on symmetric multiprocessing systems
- Be a great distributed computing platform, both as a network client and a server
- Run most existing 16-bit MS-DOS and Microsoft Windows 3.1 applications
- Meet government requirements for POSIX 1003.1 compliance
- Meet government and industry requirements for operating system security
- Be easily adaptable to the global market by supporting Unicode

To guide the thousands of decisions that had to be made to create a system that met these requirem
Windows NT design team adopted the following design goals at the beginning of the project:

- **Extensibility** The code must be written to comfortably grow and change as market requiren
  change.

- **Portability** The system must be able to run on multiple hardware architectures and must be
  move with relative ease to new ones as market demands dictate.

- **Reliability and robustness** The system should protect itself from both internal malfunction
  external tampering. Applications should not be able to harm the operating system or other ru
  applications.

- **Compatibility** Although Windows NT should extend existing technology, its user interface
  application programming interfaces (APIs) should be compatible with older versions of Wir
  well as older operating systems such as MS-DOS. It should also interoperate well with othe
  such as UNIX, OS/2, and NetWare.

- **Performance** Within the constraints of the other design goals, the system should be as fast
  responsive as possible on each hardware platform.

As we explore the details of the internal structure and operation of Windows NT, you'll see how th
design goals and market requirements were woven successfully into the construction of the system
before we start that exploration, let's examine the overall design model for Windows NT and comp
other modern operating systems.

Operating System Models

In most operating systems, applications are separated from the operating system itself--the operatin
code runs in a privileged processor mode (referred to as *kernel mode* in this book), with access to s
data and to the hardware; applications run in a nonprivileged processor mode (called *user mode*), w
limited set of interfaces available and with limited access to system data. When a user-mode progra
system service, the processor traps the call and then switches the calling thread to kernel mode. Wh
system service completes, the operating system switches the thread context back to user mode and
the caller to continue.

The design of the internal structure of the kernel-mode portion of such systems varies widely. For e
traditional operating systems were monolithic in nature, as illustrated in Figure 2-1. The system wa
constructed as a single, large software system with many dependencies among internal components
interdependency meant that extensions to the system might require many changes across the entire
base. Also, in a monolithic operating system, the bulk of the operating system code runs in the sam
memory space, which means that any operating system component could corrupt data being used b
components.

Click to view graphic (12 KB)

Figure 2-1
Monolithic operating system

A different structuring approach divides the operating system into modules and layers them one on
the other. Each module provides a set of functions that other modules can call. Code in any particu
calls code only in lower layers. On some systems, such as the Digital Equipment Corporation (DE
OpenVMS or the old Multics operating system, hardware even enforces the layering (using multipl
hierarchical processor modes). One advantage of a layered operating system structure is that becau
layer of code is given access to only the lower-level interfaces (and data structures) it requires, the
of code that wields unlimited power is limited. This structure also allows the operating system to b
debugged starting at the lowest layer, adding one layer at a time until the whole system works corr
Layering also makes it easier to enhance the operating system because individual layers can be mo
replaced without affecting other parts of the system.

Another approach to structuring an operating system is the client/server microkernel model. The
architecture in this approach divides the operating system into several server processes, each of wh
implements a single set of services--for example, memory management services, process creation s
or processor scheduling services. Each *server* runs in user mode, waiting for a client request for on
services. The *client,* which can be either another operating system component or an application pro
requests a service by sending a message to the server. An operating system microkernel running in
mode delivers the message to the server; the server performs the operation; and the kernel returns t
to the client in another message, as illustrated in Figure 2-2.

NOTE:
The client/server model of networking is distinctly different from the client/server model of
processing. In client/server networking, a server provides resources (such as files, printer, a
storage space) to the clients. Client/server processing is a method of distributing the
processing load required by an application to best suit the capabilities of network, server, an
client so that one part of an application is processed on a server machine while another is
processed on the client.

In reality, client/server systems fall within a spectrum, some doing very little work in kernel mode
others doing more. For example, the Carnegie Mellon University Mach operating system, a conten
example of the client/server microkernel architecture, implements a minimal kernel that comprises
scheduling, message passing, virtual memory, and device drivers. Everything else, including variou
file systems, and networking, runs in user mode. However, commercial implementations of the Ma
microkernel operating system typically run at least all file system, networking, and memory manag
code in kernel mode. The reason is simple: the pure microkernel design is commercially impractic
because it is too computationally expensive--that is, it's too slow.

Click to view graphic (8 KB)

Figure 2-2
Client/server operating system

So what model does Windows NT embody? It merges the attributes of a layered operating system
those of a client/server or microkernel operating system. Performance-sensitive operating system
components run in kernel mode, where they can interact with the hardware and with each other wit
incurring the overhead of context switches and mode transitions. For example, the memory manag
manager, object and security managers, network protocols, file systems (including network servers

redirectors), and all thread and process management run in kernel mode.

Of course, all of these components are fully protected from errant applications, because application
have direct access to the code and data of the privileged part of the operating system (though they c
quickly call other kernel services). This protection is one of the reasons that Windows NT has the r
for being both robust and stable as an application server and a workstation platform yet fast and nir
from the perspective of core operating system services, such as virtual memory management, file I/
networking, and file and print sharing.

Does the fact that so much of Windows NT runs in kernel mode mean it is more susceptible to cras
a true microkernel operating system? Not really. Consider the following scenario: suppose the file s
code of an operating system has a bug that causes it to crash from time to time. In a traditional oper
system or a modified microkernel operating system, a bug in kernel-mode code such as the memory
manager or the file system would likely crash the entire operating system. In a pure microkernel op
system, such components run in user mode, so theoretically a bug would simply mean that the com
process exits. But in practical terms, the failure of such a critical process would result in a system c
since recovery from the failure of such a component would likely be impossible.

The kernel-mode components of Windows NT also embody basic object-oriented design principles
example, they don't reach into one another's data structures to access information maintained by in
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, however, Windows NT is
object-oriented system in the strict sense. Most of the operating system code is written in C for por
and because development tools are widely available. C does not directly support object-oriented co
such as dynamic binding of data types, polymorphic functions, or class inheritance. Therefore, the
implementation of objects in Windows NT borrows from, but does not depend on, esoteric features
particular object-oriented languages.

Architecture Overview

Now that you understand the basic model of Windows NT, let's take a look at the key system com
that comprise its architecture. A simplified version of this architecture is shown in Figure 2-3. Kee
that this diagram is basic--it doesn't show everything. The various components of Windows NT are
in detail later in the chapter.

In Figure 2-3, first notice the line dividing the user-mode and kernel-mode parts of the Windows N
operating system. The boxes above the line represent user-mode processes, and the components be
line are kernel-mode operating system services. As mentioned in Chapter 1, user-mode threads exe
protected process address space (although while they are executing in kernel mode, they have acce
system space). Thus, system processes, server processes (services), the environment subsystems, a
applications each have their own private process address space.

Click to view graphic (8 KB)

Figure 2-3
Simplified Windows NT architecture

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, that a
  Windows NT services (that is, not started by the service controller).

- *Server processes* that are Windows NT services, such as the Event Log and Schedule servic

add-on server applications, such as Microsoft SQL Server and Microsoft Exchange Server, a
include components that run as Windows NT services.

- *Environment subsystems,* which expose the native operating system services to user applicat
thus providing an operating system *environment*, or personality. Windows NT ships with thr
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,
1.2.

In Figure 2-3, notice the "Subsystem DLLs" box below the "User applications" one. Under Windov
user applications do not call the native Windows NT operating system services directly; rather, the
through one or more *subsystem dynamic-link libraries* (*DLLs*). The role of the subsystem DLLs is t
translate a documented function into the appropriate undocumented Windows NT system service c
translation might or might not involve sending a message to the environment subsystem process th
serving the user application.

The kernel mode of the operating system includes these components:

- The Windows NT *executive* contains the base operating system services, such as memory
management, process and thread management, security, I/O, and interprocess communicatic

- The Windows NT *kernel* performs low-level operating system functions, such as thread sch
interrupt and exception dispatching, and multiprocessor synchronization. It also provides a s
routines and basic objects that the rest of the executive uses to implement higher-level const

- The *hardware abstraction layer* (*HAL*) is a layer of code that isolates the kernel, device driv
the rest of the Windows NT executive from platform-specific hardware differences.

- *Device drivers* include both file system and hardware device drivers that translate user I/O f
calls into specific hardware device I/O requests.

- The *windowing and graphics system* implements the graphical user interface (GUI) function
known as the Win32 USER and GDI functions), such as dealing with windows, controls, an
drawing.

Each of these components is covered in greater detail both later in this chapter and in the chapters t
follow.

Before we dig into the details of these system components, though, let’s review two key attributes
Windows NT architecture--portability and multiprocessing--and also examine the differences betw
Windows NT Workstation and Windows NT Server.

Portability

Windows NT was designed to run on a variety of hardware architectures, including Intel-based CIS
systems as well as RISC systems. The initial release of Windows NT supported the *x*86 and MIPS
architecture. Support for the DEC Alpha AXP was added shortly thereafter. Support for a fourth pr
architecture, the Motorola PowerPC, was added in Windows NT 3.51. Because of changing marke
demands, however, support for both the MIPS and PowerPC was dropped after the release of Wind
4.0. Windows NT 5.0 will run only on *x*86 and Alpha machines. Eventually, Windows NT will als
the Merced chip, the first implementation of the new 64-bit architecture family being jointly devel
Intel and Hewlett-Packard, called IA64 (for Intel Architecture 64). As Microsoft has stated publicl
Windows NT will be enhanced to support a true 64-bit programming interface on both IA64 and A
systems.

Windows NT achieves portability across hardware architectures and platforms in two primary way

- Windows NT has a layered design, with low-level portions of the system that are processor-architecture-specific or platform-specific isolated into separate modules so that up layers of the system can be shielded from the differences among hardware platforms. The tv components that provide operating system portability are the HAL and the kernel. Functions architecture-specific (such as thread context switching) are implemented in the kernel. Func can differ from machine to machine within the same architecture are implemented in the HA

- The majority of Windows NT is written in a portable language--the operating system execut utilities, and device drivers are written in C, and portions of the graphics subsystem and use interface are written in C++. Assembly language is used only for those parts of the operating that must communicate directly with system hardware (such as the interrupt trap handler) or extremely performance-sensitive (such as context switching). Assembly language code exis only in the kernel and the HAL but also in a few places within the executive (such as the exe routines that implement interlocked instructions as well as one module in the local procedur facility), in the kernel-mode part of the Win32 subsystem, and even in some user-mode libra such as the process startup code in NTDLL.DLL (explained later in this chapter).

Symmetric Multiprocessing

Multitasking is the operating system technique for sharing a single processor among multiple threa execution. When a computer has more than one processor, however, it can execute two threads simultaneously. Thus, whereas a multitasking operating system only appears to execute multiple th the same time, a multiprocessing operating system actually does it, executing one thread on each o processors.

As mentioned at the beginning of the chapter, a key Windows NT design goal from the start of the was to run well on multiprocessor computer systems. Windows NT supports *symmetric multiproce* (*SMP*). There is no master processor--the operating system as well as user threads can be schedulec on any processor. Also, all the processors share just one memory space. This model contrasts with *asymmetric multiprocessing* (*ASMP*), in which the operating system typically selects one processor execute operating system code while other processors run only user code. The differences in the tw multiprocessing models are illustrated in Figure 2-4.

Click to view graphic (15 KB)

Figure 2-4
Symmetric vs. asymmetric multiprocessing

Windows NT was architecturally designed to run on up to 32 processors. The number of licensed processors is stored in the registry at HKLM\System\CurrentControlSet\Control\Session Manager\LicensedProcessors. (Tampering with that data is a violation of the software license; and modifying Windows NT to use more processors is more complicated than just changing this value. default value depends on the edition of Windows NT, as you can see in Table 2-1.

Table 2-1. Number of Licensed Processors for Various Editions of Windows NT

Edition	Number of Licensed Processors
Windows NT Server, Enterprise Edition	8
Windows NT Server	4
Windows NT Workstation	2

System manufacturers that sell Windows NT Server systems that support more than eight processo

ship their own remastered Windows NT CD-ROM with a registry set to enable a higher number of
processors. They might also need to provide their own HAL.

One of the key issues with multiprocessor systems is scalability. To run correctly on an SMP syste
operating system code must adhere to strict guidelines and rules to ensure correct operation. Resou
contention and other performance issues are more complicated in multiprocessing systems than in
operating systems and must be accounted for in the system's design. Windows NT incorporates se
features that are crucial to its success as a multiprocessing operating system:

- The ability to run operating system code on any available processor and on multiple process
  same time. With the exception of its kernel component, which handles thread scheduling an
  interrupts, all operating system code can be preempted (forced to give up a processor) when
  higher-priority thread needs attention.

- Multiple threads of execution within a single process, each of which can potentially execute
  simultaneously on different processors.

- Fine-grained synchronization within the kernel as well as within device drivers and server p
  allow more components to run concurrently on multiple processors.

- Server processes that use multiple threads to process requests from more than one client
  simultaneously.

- Convenient mechanisms for sharing objects among processes and flexible interprocess
  communication capabilities, including shared memory and an optimized message-passing fa

Chapter 4 describes how threads are scheduled in a multiprocessor system.

Are there two versions of Windows NT--one for uniprocessor systems and one for multiprocessor
Not really. Besides the HAL, which by its very nature is different for a uniprocessor system than fo
multiprocessor system, of the more than 2000 files on the Windows NT CD-ROM, only *one file* is
in different uniprocessor and multiprocessor versions: the core operating system image that contai
executive and kernel, NTOSKRNL.EXE. The rest of the binary files that comprise Windows NT (i
all utilities, libraries, and device drivers) are built to run properly on both uniprocessor and multipr
systems. For example, they handle multiprocessor synchronization issues correctly. You should use
approach on any software you build, whether it be a Win32 application or a device driver--build yo
assuming it might run on a multiprocessor system so that if it does, it won't break.

The Windows NT CD-ROM includes two versions of NTOSKRNL:

- NTOSKRNL.EXE is the executive and kernel for uniprocessor systems.

- NTKRNLMP.EXE is the executive and kernel for multiprocessor systems.

These two images are built from the same source files. They are built using compile-time conditio
so that multiprocessor-specific support is not included in the uniprocessor version of NTOSKRNL
versa. Because of this, single processor systems don't have to pay for the overhead of multiprocess
synchronization at the operating system level.

At installation time, the appropriate file is selected and copied to the local \winnt\system32 directo
either case, however, the file is named NTOSKRNL.EXE on the local hard drive.

You'll notice that on the checked build CD-ROM (the special debug version of Windows NT, whic
explained on page 22 in Chapter 1), both NTOSKRNL.EXE and NTKRNLMP.EXE are identical--
both built for multiprocessor systems. In other words, there is no uniprocessor version of the check
version of NTOSKRNL.

EXPERIMENT: Checking Which Version of NTOSKRNL You're Running

You can tell which version of NTOSKRNL you're running by running WINMSD.EXE. (From the Start menu, choose Programs, and then select Administrative Tools, Windows N Diagnostics.) If you click the Version tab, you'll see something like the following:

Click to view graphic (9 KB)

As you can see, the system is running the multiprocessor free build for *x*86 systems. (This screen shot was taken from the dual processor Pentium Pro workstation that Compaq so graciously loaned me for this book project.)

Windows NT Workstation vs. Windows NT Server

Many people wonder what exactly the differences are between Windows NT Workstation, Windov Server, and Windows NT Server, Enterprise Edition. First, Windows NT Server behaves differentl Windows NT Workstation does--Windows NT Server is optimized to be a high-performance netwo server platform, whereas Windows NT Workstation, although it has server capabilities, is optimize interactive desktop use.

Second, Windows NT Server, Enterprise Edition, is a superset of Windows NT Server, which in tu superset of Windows NT Workstation. For example, the following optionally installable networkin server components come with Windows NT Server but are not available for Windows NT Worksta

- Enterprise network management and directory services through the formation of domains (g Windows NT systems treated as a single security perimeter)
- Disk fault-tolerance features (striping with parity and mirroring)
- Services for Macintosh: file and printer sharing, user administration
- Gateway Service for NetWare, which permits a number of Windows NT clients to access a server using the Windows NT Server as a gateway
- TCP/IP server addressing management, such as a complete Domain Name System (DNS) a Dynamic Host Configuration Protocol (DHCP)
- Remote boot server for diskless MS-DOS, Windows 3.1, and Windows 95 PCs

Windows NT Server, Enterprise Edition, contains additional components and features beyond thos Windows NT Server, such as Microsoft Cluster Server, Microsoft Message Queue Server, and Mic Transaction Server. (The Windows NT 4.0 Option Pack, which installs on both Windows NT Serv Windows NT Server, Enterprise Edition, includes the latter two components in addition to Microsc Internet Information Server 4.0 and Internet Connection Services for Microsoft RAS.) Also, on *x*8 systems, Windows NT Server, Enterprise Edition, can allow certain applications to have a 3-GB us address space (as opposed to 2 GB on the other editions). This capability is explained in further de Chapter 5.

There are also licensing differences between Windows NT Workstation and Windows NT Server:

- The Windows NT Workstation license permits only 10 unique IP connections in a 10-minut (though the code doesn't enforce this connection limit). Windows NT Server has no such re

- Windows NT Server supports an unlimited number of clients (assuming that you have licens of them) accessing the built-in file and print-sharing services, whereas Windows NT Works permits only up to 10 simultaneous inbound connections to shared files or printers.

- Windows NT Server, Enterprise Edition, supports eight processors, Windows NT Server fou Windows NT Workstation only two.

Although Windows NT Server and Windows NT Server, Enterprise Edition, contain significant add functionality over Windows NT Workstation, the majority of the files in all three products are iden including such core components as the executive, kernel, device drivers, utilities, and libraries. Hov number of these components operate differently depending on which edition is running.

How does Windows NT know which product is running? At boot time, the registry is queried and t is stored in the system global variable *MmProductType*. One element of this information is in the re key HKLM\System\CurrentControlSet\Control\ProductOptions. Changing this information is a vio the software license. Table 2-2 shows the values for this key as they correspond to the different edi Windows NT.

Table 2-2. Product Type Registry Values

Edition of Windows NT	**Value of Product Options***
Windows NT Workstation	WinNT
Windows NT Server (domain controller)	LanmanNT
Windows NT Server (server only)	ServerNT

* A different key, ProductSuite, distinguishes Windows NT Server, Enterprise Edition.

If user programs need to determine which Windows NT product is running, they can query for this information. (For sample code to do this, see the article Q124305 "Which Windows NT (Server or Workstation) Is Running?" in the MSDN Knowledge Base.) Device drivers running in kernel mode the internal executive routine used by Windows NT itself, *MmIsThisAnNtasSystem*, documented in Windows NT Device Driver Kit (DDK).

Based on the product type, several resource allocation decisions are made differently at system boo such as the size and number of operating system heaps (or pools), the number of internal system wo threads, and the size of the system data cache. Also, run-time policy decisions, such as the way the manager trades off system and process memory demands, differ between Windows NT Server and Windows NT Workstation. Even some thread-scheduling details are handled differently in the two Where there are significant operational differences in the two products, these are highlighted in the pertinent chapters throughout the rest of the book. Thus, unless otherwise noted, everything in this applies to both Windows NT Server and Windows NT Workstation.

Windows NT vs. Windows 95 and Windows 98

Windows NT and Windows 95 (and its follow-on release, Windows 98) are part of the "Windows family of operating systems," sharing a common subset API (Win32 and COM), device driver model (WDM), and in some cases shared operating system code. Although Windows NT 4.0 doesn’t have some of the features that Windows 95 has today, Microsoft has always made it clear that Windows NT was to be the strategic operating system platforn for the future--not just for servers and business desktops but eventually for consumers as we Following are some of the architectural differences and advantages that Windows NT has over Windows 95. (These comparisons also apply to Windows 98.)

- Windows NT supports multiprocessor systems--Windows 95 doesn't.
- Windows NT runs on a variety of machine architectures--Windows 95 is limited to *x* systems.
- Windows 95 doesn't have a file system that supports security (such as discretionary access control).
- Windows NT is a fully 32-bit operating system--it contains no 16-bit code. Windows 95 contains a large amount of old 16-bit code from its predecessors, Windows 3.1 an MS-DOS.
- Windows NT is fully reentrant--significant parts of Windows 95 are nonreentrant (mainly the older 16-bit code taken from Windows 3.1). This nonreentrant code includes the majority of the graphics and window management functions (USER and GDI). When a 32-bit application on Windows 95 attempts to call a system service implemented in nonreentrant 16-bit code, it must first obtain a systemwide lock (or mutex) to block other threads from entering the nonreentrant code base. And even worse, a 16-bit application holds this lock *while running*. Thus, although the core of Windows 95 contains a preemptive 32-bit multithreaded scheduler, because so much of the system is still implemented in nonreentrant code, applications many times run single threaded.
- Windows NT provides an option to run 16-bit Windows applications in their own address space--Windows 95 always runs 16-bit Windows applications in a shared address space, in which they can corrupt (and hang) each other.
- Shared memory on Windows NT is visible only to the processes that have the same shared memory section (called *file mapping objects* in the Win32 API) open. On Windows 95, all shared memory is visible and writable from all processes. Thus, any process can write to any file mapping object.
- Windows 95 has some critical operating system pages that are writable from user mode, thus allowing a user application to crash the system.

What does Windows 95 have that Windows NT 4.0 doesn't? Full Plug and Play, power management, infrared support, and support for the FAT32 file system. However, all of these features will be a part of Windows NT 5.0, making it the first release of Windows NT to be true superset of the Windows platform.

The one thing both Windows 95 and Windows 98 can do that Windows NT will never do is run *all* older MS-DOS and Windows 3.1 applications (notably ones that require direct hardware access) as well as 16-bit MS-DOS device drivers. Whereas 100 percent compatibility with MS-DOS and Windows 3.1 was a mandatory goal for Windows 95, the goal for Windows NT was to run *most* existing 16-bit applications.

TOP | NEXT PAGE

Inside Windows NT Disk Defragmenting. Copyright Â© 1997 Mark Russinovich Note: The information presented here is the result of my own study. No source code was used. Introduction. The first defragmentation support for Windows NT was introduced by Executive Software. Their Diskeeper product was initially released for Windows This continued all the way until Windows NT 3.51. We now knew that the AND operation was implemented at some point during Windows NT 4.0â€™s development. Thanks to friends that had beta builds of NT 4 saved locally, we were quickly able to find the very first publicly available build with this â€œfeatureâ€ â€" build 1264. To test our theory, we modified the kernel of build 1264 to identify itself as build 16384. This showed us the two expected faces of the build / the code â€" the raw data and the handled version info.