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 other such as UNIX, OS/2, and NetWare.

- **Performance** Within the constraints of the other design goals, the system should be as fast a 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 (DEC 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 corre 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, ar 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 contem 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 impractica 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 w 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 manage 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 comp
that comprise its architecture. A simplified version of this architecture is shown in Figure 2-3. Keep
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 bel
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 acces
system space). Thus, system processes, server processes (services), the environment subsystems, an
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 Window user applications do not call the native Windows NT operating system services directly; rather, they 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 ca translation might or might not involve sending a message to the environment subsystem process tha 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 communicatio
- The Windows NT *kernel* performs low-level operating system functions, such as thread sche 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 o 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 market 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 also the Merced chip, the first implementation of the new 64-bit architecture family being jointly develo Intel and Hewlett-Packard, called IA64 (for Intel Architecture 64). As Microsoft has stated publicly 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 ways

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.