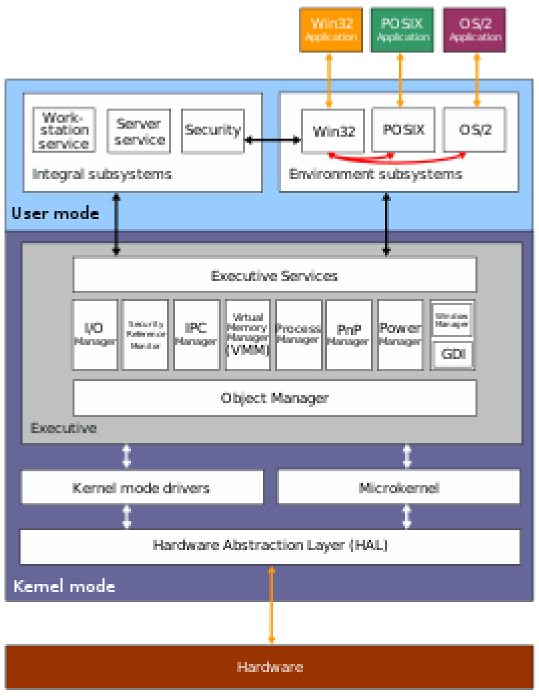
WIKIPEDIA

Architecture of Windows NT

The **architecture of Windows NT**, a line of operating systems produced and sold by Microsoft, is a layered design that consists of two main components, user mode and kernel mode. It is a preemptive, reentrant multitasking operating system, which has been designed to work with uniprocessor and symmetrical multiprocessor (SMP)-based computers. To process input/output (I/O) requests, they use packet-driven I/O, which utilizes I/O request packets (IRPs) and asynchronous I/O. Starting with Windows XP, Microsoft began making 64-bit versions of Windows available; before this, there were only 32-bit versions of these operating systems.



The Windows NT operating system family's architecture consists of two layers (user mode and kernel mode), with many different modules within both of these layers.

Programs and subsystems in user mode are limited in terms of what system resources they have access to, while the kernel mode has unrestricted access to the system memory and external devices. Kernel mode in Windows NT has full access to the hardware and system resources of the computer. The Windows NT kernel is a hybrid kernel; the architecture comprises a simple kernel, hardware abstraction layer (HAL), drivers, and a range of services (collectively named Executive), which all exist in kernel mode.[1]

User mode in Windows NT is made of subsystems capable of passing I/O requests to the appropriate kernel mode device drivers by using the I/O manager. The user mode layer of Windows NT is made up of the "Environment subsystems", which run applications written for many different types of operating systems, and the "Integral subsystem", which operates system-specific functions on behalf of environment subsystems. The kernel mode stops user mode services and applications from accessing critical areas of the operating system that they should not have access to.

The Executive interfaces, with all the user mode subsystems, deal with I/O, object management, security and process management. The kernel sits between the hardware abstraction layer and the Executive to provide *multiprocessor synchronization*, thread and interrupt scheduling and dispatching, and trap handling and exception dispatching. The kernel is also responsible for initializing device drivers at bootup. Kernel mode drivers exist in three levels: highest level drivers, intermediate drivers and low-level drivers. Windows Driver Model (WDM) exists in the intermediate layer and was mainly designed to be binary and source compatible between Windows 98 and Windows 2000. The lowest level drivers are either legacy Windows NT device drivers that control a device directly or can be a plug and play (PnP) hardware bus.

Contents

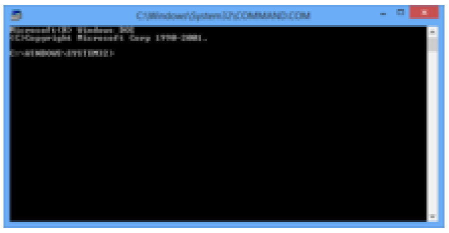
User mode

User mode is made up of various system-defined processes and DLLs.

The interface between user mode applications and operating system kernel functions is called an "environment subsystem." Windows NT can have more than one of these, each implementing a different API set. This mechanism was designed to support applications written for many different types of operating systems. None of the environment subsystems can directly access hardware; access to hardware functions is done by calling into kernel mode routines.

There are three main environment subsystems: the Win32 subsystem, an OS/2 subsystem and a POSIX subsystem.[2]

Win32 environment subsystem



```
C:\Windows\System32\COMMAND.COM
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\WINDOWS\SYSTEM32>
```

COMMAND.COM running in the NTVDM

The Win32 environment subsystem can run 32-bit Windows applications. It contains the console as well as text window support, shutdown and hard-error handling for all other environment subsystems. It also supports Virtual DOS Machines (VDMs), which allow MS-DOS and 16-bit Windows (Win16) applications to run on Windows NT. There is a specific MS-DOS VDM that runs in its own address space and which emulates an Intel 80486 running MS-DOS 5.0. Win16 programs, however, run in a Win16 VDM. Each program, by default, runs in the same process, thus using the same address space, and the Win16 VDM gives each program its own thread on which to run. However, Windows NT does allow users to run a Win16 program in a separate Win16 VDM, which allows the program to be preemptively multitasked, as Windows NT will pre-empt the whole VDM process, which only contains one running application. The Win32 environment subsystem process (csrss.exe) also includes the window management functionality, sometimes called a "window manager". It handles input events (such as from the keyboard and mouse), then passes messages to the applications that need to receive this input. Each application is responsible for drawing or refreshing its own windows and menus, in response to these messages.

OS/2 environment subsystem

The OS/2 environment subsystem supports 16-bit character-based OS/2 applications and emulates OS/2 1.x, but not 32-bit or graphical OS/2 applications as used with OS/2 2.x or later, on x86 machines only.[3] To run graphical OS/2 1.x programs, the Windows NT Add-On Subsystem for Presentation Manager must be installed.[3] The last version of Windows NT to have an OS/2 subsystem was Windows 2000; it was removed as of Windows XP.[4][5]

POSIX environment subsystem

The POSIX environment subsystem supports applications that are strictly written to either the POSIX.1 standard or the related ISO/IEC standards. This subsystem has been replaced by Interix, which is a part of Windows Services for UNIX.[4] This was in turn replaced by the Windows Subsystem for Linux.

Security subsystem

The security subsystem deals with security tokens, grants or denies access to user accounts based on resource permissions, handles login requests and initiates login authentication, and determines which system resources need to be audited by Windows NT. It also looks after Active Directory. The workstation service implements the network redirector, which is the client side of Windows file and print sharing; it implements local requests to remote files and printers by "redirecting" them to the appropriate servers on the network.[6] Conversely, the server service allows other computers on the network to access file shares and shared printers offered by the local system.[7]

Kernel mode

Windows NT kernel mode has full access to the hardware and system resources of the computer and runs code in a protected memory area.[8] It controls access to scheduling, thread prioritization, memory management and the interaction with hardware. The kernel mode stops user mode services and applications from accessing critical areas of the operating system that they should not have access to; user mode processes must ask the kernel mode to perform such operations on their behalf.

While the x86 architecture supports four different privilege levels (numbered 0 to 3), only the two extreme privilege levels are used. Usermode programs are run with CPL 3, and the kernel runs with CPL 0. These two levels are often referred to as "ring 3" and "ring 0", respectively. Such a design decision had been done to achieve code portability to RISC platforms that only support two privilege levels,[9] though this breaks compatibility with OS/2 applications that contain I/O privilege segments that attempt to directly access hardware.[10]

Code running in kernel mode includes: the executive, which is itself made up of many modules that do specific tasks; the kernel, which provides low-level services used by the Executive; the Hardware Abstraction Layer (HAL); and *kernel drivers*.[8][11]

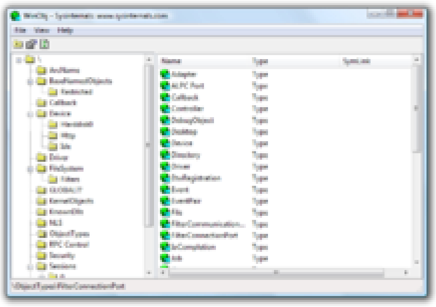
Executive

The Windows Executive services make up the low-level kernel-mode portion, and are contained in the file NTOSKRNL.EXE.[8] It deals with I/O, object management, security and process management. These are divided into several *subsystems*, among which are *Cache Manager*, *Configuration Manager*, *I/O Manager*, *Local Procedure Call* (*LPC*), *Memory Manager*, *Object Manager*, *Process Structure* and *Security*

Reference Monitor (SRM). Grouped together, the components can be called *Executive services* (internal name *Ex*). *System Services* (internal name *Nt*), i.e., system calls, are implemented at this level, too, except very few that call directly into the kernel layer for better performance.

The term "service" in this context generally refers to a callable routine, or set of callable routines. This is distinct from the concept of a "service process", which is a user mode component somewhat analogous to a daemon in Unix-like operating systems.

Object Manager


Each object in Windows NT exists in a global namespace. This is a screenshot from Sysinternals WinObj (https://docs.microsoft.com/en-us/sysinternals/downloads/winobj).

The *Object Manager* (internal name *Ob*) is an executive subsystem that all other executive subsystems, especially system calls, must pass through to gain access to Windows NT resources—essentially making it a resource management infrastructure service.[12] The object manager is used to reduce the duplication of object resource management functionality in other executive subsystems, which could potentially lead to bugs and make development of Windows NT harder.[13] To the object manager, each resource is an object, whether that resource is a physical resource (such as a file system or peripheral) or a logical resource (such as a file). Each object has a structure or *object type* that the object manager must know about.

Object creation is a process in two phases, *creation* and *insertion*. *Creation* causes the allocation of an empty object and the reservation of any resources required by the object manager, such as an (optional) name in the namespace. If creation was successful, the subsystem responsible for the creation fills in the empty object.[14] Finally, if the subsystem deems the initialization successful, it instructs the object manager to *insert* the object, which makes it accessible through its (optional) name or a cookie called a *handle*.[15] From then on, the lifetime of the object is handled by the object manager, and it's up to the subsystem to keep the object in a working condition until being signaled by the object manager to dispose of it.[16]

Handles are identifiers that represent a reference to a kernel resource through an opaque value.[17] Similarly, opening an object through its name is subject to security checks, but acting through an existing, open handle is only limited to the level of access requested when the object was opened or created.

Object types define the object procedures and any data specific to the object. In this way, the object manager allows Windows NT to be an object-oriented operating system, as object types can be thought of as polymorphic classes that define objects. Most subsystems, though, with a notable exception in the I/O Manager, rely on the default implementation for all object type procedures.

Each instance of an object that is created stores its name, parameters that are passed to the object creation function, security attributes and a pointer to its object type. The object also contains an object close procedure and a reference count to tell the object manager how many other objects in the system reference that object and thereby determines whether the object can be destroyed when a close request is sent to it.[18] Every named object exists in a hierarchical object namespace.

Cache Controller

Closely coordinates with the Memory Manager, I/O Manager and I/O drivers to provide a common cache for regular file I/O. The Windows Cache Manager operates on file blocks (rather than device blocks), for consistent operation between local and remote files, and ensures a certain degree of coherency with memory-mapped views of files, since cache blocks are a special case of memory-mapped views and cache misses a special case of page faults.

Configuration Manager

Implements the system calls needed by Windows Registry.

I/O Manager

Allows devices to communicate with user-mode subsystems. It translates user-mode read and write commands into read or write IRPs which it passes to device drivers. It accepts file system I/O requests and translates them into device specific calls, and can incorporate low-level device drivers that directly manipulate hardware to either read input or write output. It also includes a cache manager to improve disk performance by caching read requests and write to the disk in the background.

Local Procedure Call (LPC)

Provides inter-process communication ports with connection semantics. LPC ports are used by user-mode subsystems to communicate with their clients, by Executive subsystems to communicate with user-mode subsystems, and as the basis for the local transport for Microsoft RPC.

Memory Manager

Manages virtual memory, controlling memory protection and the paging of memory in and out of physical memory to secondary storage, and implements a general-purpose allocator of physical memory. It also implements a parser of PE executables that lets an executable be mapped or unmapped in a single, atomic step.

Starting from Windows NT Server 4.0, Terminal Server Edition, the memory manager implements a so-called *session space*, a range of kernel-mode memory that is subject to context switching just like user-mode memory. This lets multiple instances of the kernel-mode Win32 subsystem and GDI drivers run side-by-side, despite shortcomings in their initial design. Each session space is shared by several processes, collectively referred to as a "session".

To ensure a degree of isolation between sessions without introducing a new object type, the association between processes and sessions is handled by the Security Reference Monitor, as an attribute of a security subject (token), and it can only be changed while holding special privileges.

The relatively unsophisticated and ad hoc nature of sessions is due to the fact they weren't part of the initial design, and had to be developed, with minimal disruption to the main line, by a third party (Citrix Systems) as a prerequisite for their terminal server product for Windows NT, called WinFrame. Starting with Windows Vista, though, sessions finally became a proper aspect of the Windows architecture. No longer a memory manager construct that creeps into user mode indirectly through Win32, they were expanded into a pervasive abstraction affecting most Executive subsystems. As a matter of fact, regular use of Windows Vista always results in a multi-session environment.[19]

Process Structure

Handles process and thread creation and termination, and it implements the concept of *Job*, a group of processes that can be terminated as a whole, or be placed under shared restrictions (such a total maximum of allocated memory, or CPU time). Job objects were introduced in Windows 2000.

PnP Manager

Handles plug and play and supports device detection and installation at boot time. It also has the responsibility to stop and start devices on demand—this can happen when a bus (such as USB or IEEE 1394 FireWire) gains a new device and needs to have a device driver loaded to support it. Its bulk is actually implemented in user mode, in the *Plug and Play Service*, which handles the often complex tasks of installing the appropriate drivers, notifying services and applications of the arrival of new devices, and displaying GUI to the user.

Power Manager

Deals with power events (power-off, stand-by, hibernate, etc.) and notifies affected drivers with special IRPs (*Power IRPs*).

Security Reference Monitor (SRM)

The primary authority for enforcing the security rules of the security integral subsystem.[20] It determines whether an object or resource can be accessed, via the use of access control lists (ACLs), which are themselves made up of access control entries (ACEs). ACEs contain a Security Identifier (SID) and a list of operations that the ACE gives a select group of trustees—a user account, group account, or login session[21]—permission (allow, deny, or audit) to that resource.[22][23]

GDI

The Graphics Device Interface is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. The Windows NT 3.x series of releases had placed the GDI component in the user-mode Client/Server Runtime Subsystem, but this was moved into kernel mode with Windows NT 4.0 to improve graphics performance.[24]

Kernel

The kernel sits between the HAL and the Executive and provides multiprocessor synchronization, thread and interrupt scheduling and dispatching, and trap handling and exception dispatching; it is also responsible for initializing device drivers at bootup that are necessary to get the operating system up and running. That is, the kernel performs almost all the tasks of a traditional microkernel; the strict distinction between Executive and Kernel is the most prominent remnant of the original microkernel design, and historical design documentation consistently refers to the kernel component as "the microkernel".

The kernel often interfaces with the process manager.[25] The level of abstraction is such that the kernel never calls into the process manager, only the other way around (save for a handful of corner cases, still never to the point of a functional dependence).

Hybrid kernel design

The Windows NT design includes many of the same objectives as Mach, the archetypal microkernel system, one of the most important being its structure as a collection of modules that communicate via well-known interfaces, with a small microkernel limited to core functions such as first-level interrupt handling, thread scheduling and synchronization primitives. This allows for the possibility of using either direct procedure calls or interprocess communication (IPC) to communicate between modules, and hence for the potential location of modules in different address spaces (for example in either kernel space or server processes). Other design goals shared with Mach included support for diverse architectures, a kernel with abstractions general enough to allow multiple operating system personalities to be implemented on top of it and an object-oriented organisation.[26][27]

The primary operating system personality on Windows is the Windows API, which is always present. The emulation subsystem which implements the Windows personality is called the Client/Server Runtime Subsystem (csrss.exe). On versions of NT prior to 4.0, this subsystem process also contained the window manager, graphics device interface and graphics device drivers. For performance reasons, however, in version 4.0 and later, these modules (which are often implemented in user mode even on monolithic systems, especially those designed without internal graphics support) run as a kernel-mode subsystem.[26]

Applications that run on NT are written to one of the OS personalities (usually the Windows API), and not to the native NT API for which documentation is not publicly available (with the exception of routines used in device driver development). An OS personality is implemented via a set of user-mode DLLs (see Dynamic-link library), which are mapped into application processes' address spaces as required, together with an emulation subsystem server process (as described previously). Applications access system services by calling into the OS personality DLLs mapped into their address spaces, which in turn call into the NT run-time library (ntdll.dll), also mapped into the process address space. The NT run-time library services these requests by trapping into kernel mode to either call kernel-mode Executive routines or make Local Procedure Calls (LPCs) to the appropriate user-mode subsystem server processes, which in turn use the NT API to communicate with application processes, the kernel-mode subsystems and each other.[28]

Kernel-mode drivers

Windows NT uses kernel-mode device drivers to enable it to interact with hardware devices. Each of the drivers has well defined system routines and internal routines that it exports to the rest of the operating system. All devices are seen by user mode code as a file object in the I/O manager, though to the I/O manager itself the devices are seen as device objects, which it defines as either file, device or driver objects. Kernel mode drivers exist in three levels: highest level drivers, intermediate drivers and low level drivers. The highest level drivers, such as file system drivers for FAT and NTFS, rely on intermediate drivers. Intermediate drivers consist of function drivers—or main driver for a device—that are optionally sandwiched between lower and higher level filter drivers. The function driver then relies on a bus driver—or a driver that services a bus controller, adapter, or bridge—which can have an optional bus filter driver that sits between itself and the function driver. Intermediate drivers rely on the lowest level drivers to function. The Windows Driver Model (WDM) exists in the intermediate layer. The lowest level drivers are either legacy Windows NT device drivers that control a device directly or can be a PnP hardware bus. These lower level drivers directly control hardware and do not rely on any other drivers.

Hardware abstraction layer

The Windows NT hardware abstraction layer, or HAL, is a layer between the physical hardware of the computer and the rest of the operating system. It was designed to hide differences in hardware and provide a consistent platform on which the kernel is run. The HAL includes hardware-specific code that controls I/O interfaces, interrupt controllers and multiple processors.

However, despite its purpose and designated place within the architecture, the HAL isn't a layer that sits entirely below the kernel, the way the kernel sits below the Executive: All known HAL implementations depend in some measure on the kernel, or even the Executive. In practice, this means that kernel and HAL variants come in matching sets that are specifically constructed to work together.

In particular hardware abstraction does *not* involve abstracting the instruction set, which generally falls under the wider concept of portability. Abstracting the instruction set, when necessary (such as for handling the several revisions to the x86 instruction set, or emulating a missing math coprocessor), is performed by the kernel, or via hardware virtualization.

See also

- Microsoft Windows library files
- MinWin
- Unix architecture
- Comparison of operating system kernels
- User-Mode Driver Framework
- Kernel-Mode Driver Framework
- Hybrid Kernel

Further reading

- Martignetti, E.; *What Makes It Page?: The Windows 7 (x64) Virtual Memory Manager* (ISBN 978-1479114290)
- Russinovich, Mark E.; Solomon, David A.; Ionescu, A.; *Windows Internals, Part1: Covering Windows Server 2008 R2 and Windows 7* (ISBN 978-0735648739)
- Russinovich, Mark E.; Solomon, David A.; Ionescu, A.; *Windows Internals, Part2: Covering Windows Server 2008 R2 and Windows 7* (ISBN 978-0735665873)

Notes and references

Notes

1. Finnel 2000, Chapter 1: Introduction to Microsoft Windows 2000, pp. 7–18.
2. "Appendix D - Running Nonnative Applications in Windows 2000 Professional" (https://technet.microsoft.com/en-us/library/cc939090.aspx). *Microsoft Windows 2000 Professional Resource Kit*. Microsoft.
3. "Windows NT Workstation Resource Kit Chapter 28 - OS/2 Compatibility" (http://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/os2comp.mspx?mfr=true). Microsoft.
4. "POSIX and OS/2 are not supported in Windows XP or in Windows Server 2003" (http://support.microsoft.com/kb/308259). Microsoft.
5. Reiter, Brian (August 24, 2010). "The Sad History of the Microsoft POSIX Subsystem" (http://brianreiter.org/2010/08/24/the-sad-history-of-the-microsoft-posix-subsystem/).
6. "Basic Architecture of a Network Redirector" (https://msdn.microsoft.com/en-us/windows/hardware/drivers/ifs/basic-architecture-of-a-network-redirector). Microsoft. Retrieved 2016-11-18.
7. "Windows NT Networking Architecture" (https://www.microsoft.com/resources/documentation/windowsnt/4/server/reskit/en-us/net/chptr1.mspx?mfr=true). Microsoft. Retrieved 2016-11-18.
8. Roman, Steven (1999). "Windows Architecture" (https://technet.microsoft.com/en-us/library/cc768129.aspx). *Win32 API Programming with Visual Basic*. O'Reilly and Associates, Inc. ISBN 1-56592-631-5.
9. "MS Windows NT Kernel-mode User and GDI White Paper" (http://www.microsoft.com/technet/archive/ntwrkstn/evaluate/featfunc/kernelwp.mspx). *Windows NT Workstation documentation*. Microsoft TechNet. Archived (https://web.archive.org/web/20071215042008/http://www.microsoft.com/technet/archive/ntwrkstn/evaluate/featfunc/kernelwp.mspx) from the original on 15 December 2007. Retrieved 2007-12-09.
10. "Chapter 28 - OS/2 Compatibility" (http://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/os2comp.mspx?mfr=true). *Windows NT Workstation Resource Kit*. Microsoft. Archived (https://web.archive.org/web/20090210125723/http://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/os2comp.mspx?mfr=true) from the original on 10 February 2009. Retrieved 2009-01-18.
11. Mark E. Russinovich; David A. Solomon; Alex Ionescu. *Windows Internals, Fifth Edition*. Microsoft Press. pp. 228–255.
12. Russinovich & Solomon 2005, pp. 124-125.
13. Russinovich 1997, Introduction.
14. Russinovich 1997, "Object Types".
15. Russinovich & Solomon 2005, pp. 135-140.
16. Russinovich & Solomon 2005, pp. 141-143.
17. "Handles and Objects" (http://msdn.microsoft.com/en-us/library/ms724457(VS.85).aspx). *MSDN - Win32 and COM Development*. Microsoft. Retrieved 2009-01-17.

18. Russinovich 1997, "Objects".
19. "Impact of Session 0 Isolation on Services and Drivers in Windows Vista" (http://www.microsoft.com/whdc/system/vista/services.mspx). Microsoft.
20. "Active Directory Data Storage" (http://www.microsoft.com/resources/documentation/Windows/2000/server/reskit/en-us/Default.asp?url=/resources/documentation/Windows/2000/server/reskit/en-us/distrib/dsbg_dat_brlr.asp). Microsoft.
21. "Trustee definition" (http://msdn.microsoft.com/library/en-us/secgloss/security/t_gly.asp?FRAME=true#_security_trustee_gly). MSDN.
22. Siyan 2000.
23. "1.2 Glossary" (https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-azod/9baf9127-2ffb-4dda-aa45-0efcf409cee5). *[MS-AZOD]: Authorization Protocols Overview*. access control entry (ACE).
24. "The Windows NT 4.0 Kernel mode change" (https://technet.microsoft.com/en-us/library/cc750820.aspx#XSLTsection124121120120). *MS Windows NT Kernel-mode User and GDI White Paper*. Microsoft. Archived (https://web.archive.org/web/20090113021015/http://technet.microsoft.com/en-us/library/cc750820.aspx) from the original on 13 January 2009. Retrieved 2009-01-19.
25. Solomon & Russinovich 2000, pp. 543–551.
26. "MS Windows NT Kernel-mode User and GDI White Paper" (http://www.microsoft.com/technet/archive/ntwrkstn/evaluate/featfunc/kernelwp.mspx?mfr=true). Microsoft Corporation. 2007. Retrieved 2007-03-01.
27. Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg (2005). *Operating System Concepts; 7th Edition* (http://higheredbcs.wiley.com/legacy/college/silberschatz/0471694665/appendices/appb.pdf) (PDF). Hoboken, New Jersey: John Wiley & Sons Inc. ISBN 978-0-471-69466-3.
28. Probert, Dave (2005). "Overview of Windows Architecture" (http://research.microsoft.com/ur/asia/curriculum/download/BeijingPresentation.ppt). *Using Projects Based on Internal NT APIs to Teach OS Principles*. Microsoft Research/Asia - Beijing. Retrieved 2007-03-01.

References

- Finnel, Lynn (2000). *MCSE Exam 70-215, Microsoft Windows 2000 Server*. Microsoft Press. ISBN 1-57231-903-8.
- Russinovich, Mark (October 1997). "Inside NT's Object Manager" (http://www.windowsitpro.com/Articles/Index.cfm?ArticleID=299). Windows IT Pro.
- "Active Directory Data Storage" (http://www.microsoft.com/resources/documentation/Windows/2000/server/reskit/en-us/Default.asp?url=/resources/documentation/Windows/2000/server/reskit/en-us/distrib/dsbg_dat_brlr.asp). Microsoft. Retrieved 2005-05-09.
- Solomon, David; Russinovich, Mark E. (2000). *Inside Microsoft Windows 2000* (https://web.archive.org/web/20050323090649/http://mipagina.cantv.net/jjaguilerap/w2k_arq.html) (Third ed.). Microsoft Press. ISBN 0-7356-1021-5. Archived from the original (http://mipagina.cantv.net/jjaguilerap/w2k_arq.html) on 2005-03-23.
- Russinovich, Mark; Solomon, David (2005). *Microsoft Windows Internals* (4th ed.). Microsoft Press. ISBN 0-7356-1917-4.
- Schreiber, Sven B. (2001). *Undocumented Windows 2000 Secrets*. Addison-Wesley Longman. ISBN 978-0201721874.
- Siyan, Kanajit S. (2000). *Windows 2000 Professional Reference*. New Riders. ISBN 0-7357-0952-1.

External links

- "Microsoft's official Windows 2000 site" (https://web.archive.org/web/20000229142634/http://www.microsoft.com/windows2000/default.asp). Archived from the original (http://www.microsoft.com/windows2000/) on February 29, 2000.
- "Microsoft Windows 2000 Plug and Play" (https://web.archive.org/web/20040808162827/http://www.microsoft.com/technet/prodtechnol/windows2000pro/evaluate/featfunc/plugplay.mspx). Archived from the original (http://www.microsoft.com/technet/prodtechnol/windows2000pro/evaluate/featfunc/plugplay.mspx) on August 8, 2004.
- Memory management in the Windows XP kernel (http://www.reactos.org/wiki/Techwiki:Memory_management_in_the_Windows_XP_kernel)