



Windows 95/98/NT (Introduction)

27.1 Introduction

DOS has long been the Achilles heel of the PC and has limited its development. It has also been its strength in that it provides a common platform for all packages. DOS and Windows 3.x operated in a 16-bit mode and had limited memory accessing. Windows 3.0 provided a great leap in PC systems as it provided an excellent graphical user interface to DOS. It suffered from the fact that it still used DOS as the core operating system. Windows 95/98 and Windows NT have finally moved away from DOS and operate as full 32-bit protected-mode operating systems. Their main features are:

- Run both 16-bit and 32-bit application programs.
- Allow access to a large virtual memory (up to 4 GB).
- Support for pre-emptive multitasking and multithreading of Windows-based and MS-DOS-based applications.
- Support for multiple file systems, including 32-bit installable file systems such as VFAT, CDFS (CD-ROM) and network redirectors. These allow better performance, use of long file names, and are an open architecture to support future growth.
- Support for 32-bit device drivers which give improved performance and intelligent memory usage.
- A 32-bit kernel which includes memory management, process scheduling and process management.
- Enhanced robustness and cleanup when an application ends or crashes.
- Enhanced dynamic environment configuration.

The three most widely used operating systems are MS-DOS, Microsoft Windows and UNIX. Microsoft Windows comes in many flavors; the main versions are outlined below and Table 27.1 lists some of their attributes.

- Microsoft Windows 3.x – 16-bit PC-based operating system with limited multitasking. It runs from MS-DOS and thus still uses MS-DOS functionality and file system structure.
- Microsoft Windows 95/98 – robust 32-bit multitasking operating system (although there are some 16-bit parts in it) which can run MS-DOS applications, Microsoft Windows 3.x applications and 32-bit applications.
- Microsoft Windows NT – robust 32-bit multitasking operating system with integrated networking. Networks are around NT servers and clients. As with Microsoft Windows 95/98 it can run MS-DOS, Microsoft Windows 3.x applications and 32-bit applications.

Windows NT and 95/98 provide excellent network support as they can communicate directly with many different types of networks, protocols and computer architectures.

They can networks to make peer-to-peer connections and also connections to servers for access to file systems and print servers.

Windows NT has more security in running programs than Windows 95/98 as programs and data are insulated from the operation of other programs. The operating system parts of Windows NT and Windows 95/98 run at the most trusted level of privilege of the Intel processor, which is ring zero. Application programs run at least trusted level of privilege, which is ring three. These programs can use either a 32-bit flat mode or any of the memory models, such as large, medium, compact or small.

Table 27.1 Windows comparisons.

	<i>Windows 3.1</i>	<i>Windows 95/98</i>	<i>Windows NT</i>
Pre-emptive multitasking		✓	✓
32-bit operating system		✓	✓
Long file names		✓	✓
TCP/IP	✓	✓	✓
32-bit applications		✓	✓
Flat memory model		✓	✓
32-bit disk access	✓	✓	✓
32-bit file access	✓	✓	✓
Centralized configuration storage		✓	✓
OpenGL 3D graphics			✓

27.2 Architecture

There was a great leap in performance between the 16-bit Windows 3.x operating system (which was built on DOS) to Windows 95/98 and Window NT. Apart from running in a dual 16-bit and 32-bit mode they also allow for application robustness. Figure 27.1 outlines the internal architecture of Windows 95/98.

27.3 Windows registry

On DOS-based systems, the main configuration files were AUTOEXEC.BAT, CONFIG.SYS and INI files. INI files were a major problem in that each application program and device driver configuration required one or more of these file to store default settings (such as IRQ, I/O addresses, default directories, and so on). Several important INI files are:

- WIN.INI. Information about the appearance of the Windows environment.
- SYSTEM.INI. System-specific information on the hardware and device driver configuration of the system.

In Windows 95/98/NT use a central database called the Registry which stores user-specific and configuration-specific information at a single location. This location could be on the local computer or stored on a networked computer. It thus allows network managers to standardize the configuration of networked PCs.

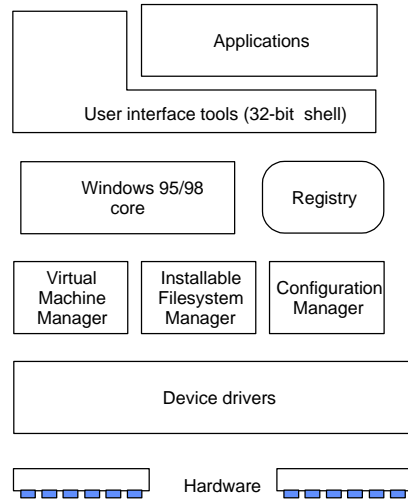


Figure 27.1 Windows 95/98 architecture.

When a computer initially is upgraded from Windows 3.x to Windows 95/98 the upgrade program reads the `SYSTEM.INI` file and system-specific information which it then puts into the Registry. Many INI files are still retained on the system as many Win16-based applications use them. For example, Microsoft Word Version 6 uses the `WINWORD6.INI` to store package information, such as: location of filters, location of spell checker, location of grammar checker, and so on. An example is:

```
[Microsoft Word]
WPHelp=0
Hyphenate 1033,0=C:\MSOFFICE\WINWORD\HYPH.DLL,C:\MSOFFICE\WINWORD\HY_EN.
NoLongNetNames=Yes
USER-DOT-PATH=C:\MSOFFICE\WINWORD\TEMPLATE
PICTURE-PATH=C:\MSOFFICE\WINWORD
PROGRAMDIR=C:\MSOFFICE\WINWORD
TOOLS-PATH=C:\MSOFFICE\WINWORD
STARTUP-PATH=C:\DOCS\nOTES\
INI-PATH=C:\MSOFFICE\WINWORD
DOC-PATH=C:\DOCS\nOTES\
Hyphenate 2057,0=C:\MSOFFICE\WINWORD\HYPH.DLL,C:\MSOFFICE\WINWORD\HY_EN.
```

An important role for the Registry is to store hardware-specific information which can be used by hardware detection and Plug-and-Play programs. The Configuration Manager determines the configuration of installed hardware (such as, IRQs, I/O addresses, and so on) and it uses this information to update the Registry. This allows new devices to be installed and checked to see if they conflict with existing devices. If they are Plug-and-Play devices then the system assigns hardware parameters which do not conflict with existing devices.

The advantages of the Registry over INI files include:

- **No limit to size and data type.** The Registry has no size restriction and can include binary and text values (INI files are text based and are limited to 64 KB in size).
- **Hierarchical information.** The Registry is hierarchically arranged, whereas INI files are non-hierarchical and support only two levels of information.

- **Standardized setup.** The Registry provides a standardized method of setting up programs, whereas many INI files contain a whole host of switches and entries, and are complicated to configure.
- **Support for user-specific information.** The Registry allows the storage of user-specific information, using the `Hkey_Users` key. This allows each user of a specific computer (or a networked computer) to have their own user-specific information. INI files do not support this.
- **Remote administration and system policies.** The Registry can be used to remotely administer and set system policies (which are stored as Registry values). These can be downloaded from a central server each time a new user logs on.

The Registry will be discussed in more detail in the next chapter. Figure 27.2 shows an example of the Registry in Windows 95/98.

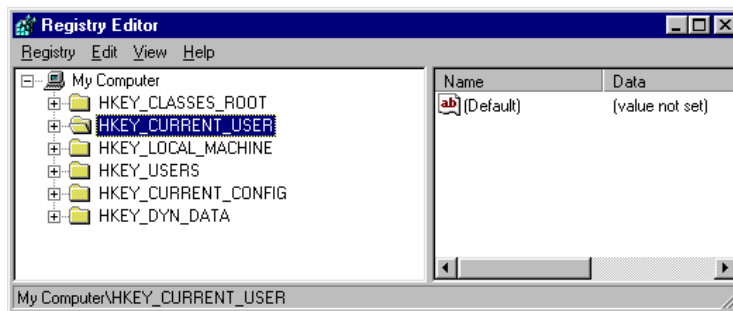


Figure 27.2 Example registry.

27.4 Device drivers

In Windows 3.x, device drivers were complex entities and were, in part, static and unchanging. Windows 95/98/NT now provide enhanced support for hardware devices and peripherals including disk devices. Windows NT will be discussed in Section 27.11. Windows 95/98 uses a universal driver/mini-driver architecture that makes writing device-specific code much easier.

The universal driver provides for most of the code for a specific class of device (such as for printers or mice) and the mini-driver is a relatively small and simple driver that provides for the additional information for the hardware.

The actual system interface to the hardware (or some software parts) is through a virtual device driver (VxD), which is a 32-bit, protected-mode driver. These keep track of the state of the device for each application and ensure that the device is in the correct state whenever an application continues. This allows for multitasking programming and also for multi-access for a single device. VxD files also support hardware emulation, such as in the case of the MS-DOS device driver, where any calls to the PC hardware can be handled by the device driver and not by the physical hardware. Typical VxD drivers are:

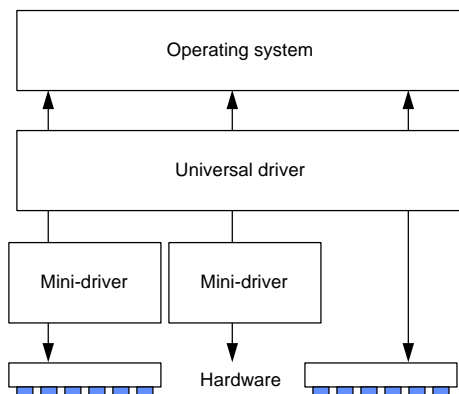


Figure 27.3 Device drivers.

EISA.VXD	EISA bus driver	ISAPNP.VXD	ISA Plug and Play
SERIAL.VXD	Serial port	LPTENUM.VXD	Parallel port
MSMOUSE.VXD	MS Mouse	PARALINK.VXD	Parallel port
PCI.VXD	PCI	QC117.VXD	Tape backup
IRCOMM.VXD	Infra-red comms.	UNIMODEM.VXD	Modem
WSOCK.VXD	WinSock	LPT.VXD	LPT
VMM32.VXD	Memory management	JAVASUP.VXD	JavaScript
PPPMAC.VXD	PPP connection	NDIS.VXD	NDIS
NDIS2SUP.VXD	NDIS 2.0	NETBEUI.VXD	Net BEUI
NWREDIR.VXD	NetWare Redirect	VNETBIOS.VXD	Net BIOS
WSIPX.VXD	IPX	WSHTCP.VXD	TCP

In Windows 95/98, VxD files are loaded dynamically and are thus only loaded when they are required, whereas in Windows 3.x they were loaded statically (and thus took up a lot of memory). In Windows 3.x these virtual device drivers have a 386 file extension.

27.5 Configuration Manager

A major drawback with Windows 3.x and DOS is that they did not automate PC configuration. For this purpose, Windows 95/98 has a Configuration Manager. The left-hand side of Figure 27.4 shows how it integrates into the system and the right side of Figure 27.4 shows an example device connection of a PC. Its aim is to:

- Determine, with the aid of several subcomponents, each bus and each device on the system, and their configuration settings. This is used to ensure that each device has unique IRQs and I/O port addresses and that there are no conflicts with other devices. With Plug-and-Play, devices can be configured so that they do not conflict with other devices.
- Monitor the PC for any changes to the number of devices connected and also the device types. If it detects any changes then it manages the reconfiguration of the devices.

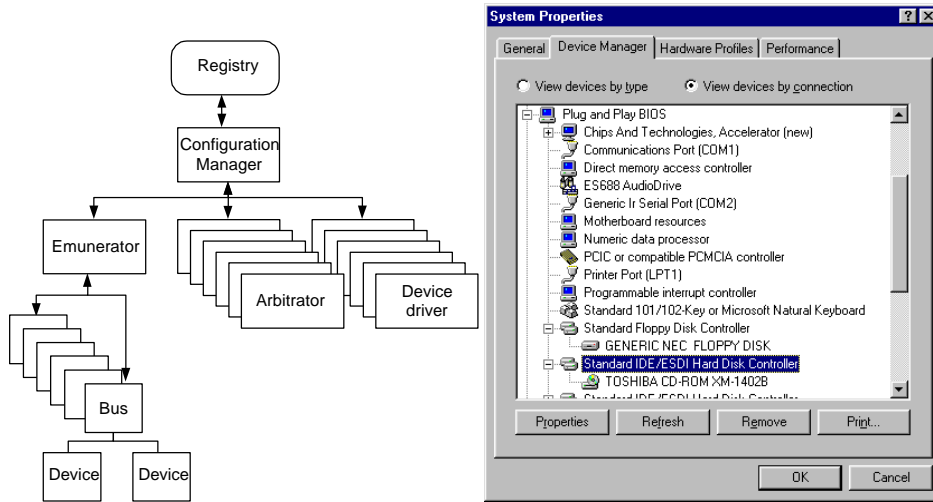


Figure 27.4 Configuration Manager and example connection of devices.

The operation is as follows:

1. The Configuration Manager communicates with each of the bus enumerators and asks them to identify all the devices on the buses and their respective resource requirements. A bus enumerator is a driver that is responsible for creating a hardware tree, which is a hierarchical representation of all the buses and devices on a computer. Figure 27.5 shows an example tree.
2. The bus enumerator locates and gathers information from either the device drivers or the BIOS services for that particular device type. For example, the CD-ROM bus enumerator calls the CD-ROM drivers to gather information.
3. Each of the drivers is then loaded and wait for the Configuration Manager to assign their specific resources (such as IRQs, I/O addresses, and so on).

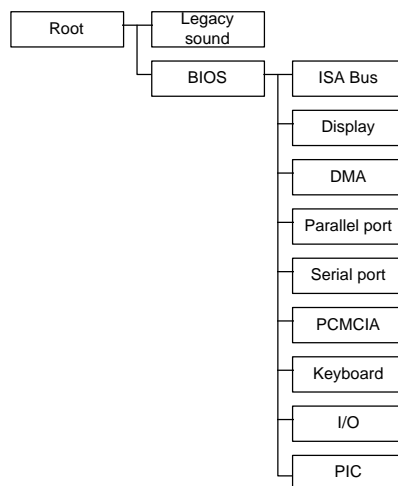


Figure 27.5 Hierarchical representation of the system.

4. Configuration Manager calls on resource arbitrators to allocate resources for each device.
5. Resource arbitrators identify any devices which are conflicting and tries to resolve them.
6. The Configuration Manager informs all device drivers of their device configuration. This process is repeated when the BIOS or one of the other bus enumerators informs Configuration Manager about a system configuration change.

27.6 Virtual Machine Manager (VMM)

The perfect environment for a program is to run on a stand-alone, dedicated computer, which does not have any interference from any other programs and can have access to any device when it wants. This is the concept of the Virtual Machine. In Windows 95/98 the Virtual Machine Manager (VMM) provides each application with the system resources when it needs them. It creates and maintains the virtual machine environments in which applications and system processes run (in Windows 3.x the VMM was called WIN386.EXE).

The VMM is responsible for three areas:

- **Process scheduling.** This is responsible for scheduling processes. It allows for multiple applications to run concurrently and also for providing system resources to the applications and other processes that run. This allows multiple applications and other processes to run concurrently, using either cooperative multitasking or pre-emptive multitasking.
- **Memory paging.** Windows 95/98/NT uses a demand-paged virtual memory system, which is based on a flat, linear address space accessed using 32-bit addresses. The system allocates each process a unique virtual address space of 4 GB. The upper 2 GB is shared, while the lower 2 GB is private to the application. This virtual address space is divided into equal blocks (or pages).
- **MS-DOS Mode support.** Provides support for MS-DOS-based applications which must have exclusive access to the hardware. When an MS-DOS-based application runs in this mode then no other applications or processes are allowed to compete for system resources. The application thus has sole access to the resources.

Windows 95/98 has a single VMM (named System VMM) in which all system processes run. Win32-based and Win16-based applications run within this VMM. Each MS-DOS-based application runs in its own VM.

27.6.1 Process scheduling and multitasking

This allows multiple applications and other processes to run concurrently, using either cooperative multitasking and pre-emptive multitasking. In Windows 3.x, applications ran using cooperative multitasking. This method requires that applications check the message queue periodically and to give-up control of the system to other applications. Unfortunately, applications that do not check the message queue at frequent intervals can effectively 'hog' the processor and prevent other applications from running. As this does not provide effective multi-processing, Windows 95/98/NT uses pre-emptive multitask-

ing for Win32-based applications (but also supports cooperative multitasking for computability reasons). Thus, the operating system takes direct control away from the application tasks.

Win16 programs need to yield to other tasks in order to multitask properly, whereas Win32-based programs do not need to yield to share resources. This is because Win32-based applications (called processes) use multithreading, which provides for multiprocessing. A thread in a program is a unit of code that can get a time slice from the operating system to run concurrently with other code units. Each process consists of one or more execution threads that identify the code path flow as it is run on the operating system. A Win32-based application can have multiple threads for a given process. This enhances the running of an application by improving throughput and responsiveness. It allows processes for smooth background processing.

27.6.2 Memory paging

Windows 95/98/NT use a demand-paged virtual memory system, which is based on a flat, linear address space using 32-bit addresses. The system allocates each process a unique virtual address space of 4 GB (which should be enough for most applications). The upper 2 GB is shared, while the lower 2 GB is private to the application. This virtual address space divides into equal blocks (or pages), as illustrated in Figure 27.6.

Demand paging is a method by which code and data are moved in pages from physical memory to a temporary paging file on disk. When required, information is then paged back into physical memory.

The functions of the Memory Pager are:

- To map virtual addresses from the process's address space to physical pages in memory. This then hides the physical organization of memory from the process's threads and ensures that the thread can access the required memory when required. It also stops other processes from writing to another memory location.

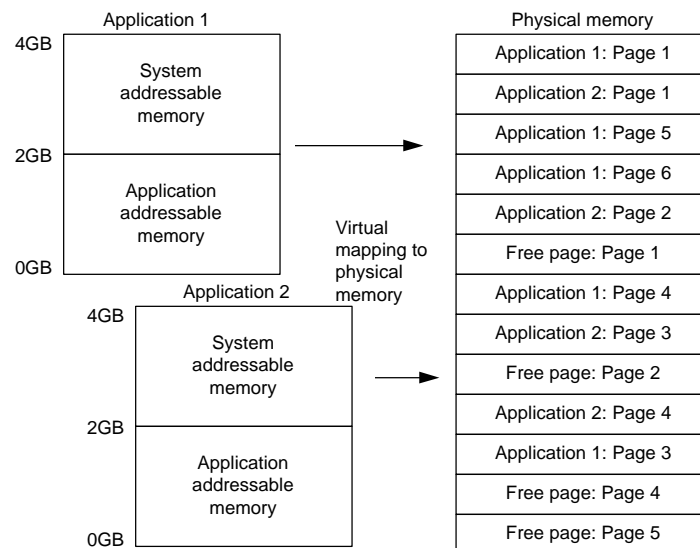


Figure 27.6 Memory paging.

- To support a 16-bit segmented memory model for Windows 3.x and MS-DOS applications. In this addressing scheme the addresses are made from a 16-bit segment address and a 16-bit offset address (see Section 1.3).

Windows 95/98/NT use the full addressing capabilities of the 80x86/Pentium processors by supporting a flat, linear memory model for 32-bit operating system functionality and Win32-based applications. This linear addressing model simplifies the development process for application vendors, and removes the performance penalties of a segmented memory architecture.

27.7 Multiple file systems

Windows 95/98/NT supports a layered file system architecture that directly supports multiple file systems (such as, FAT and CDFS). Windows 95/98/NT have great performance improvements over Windows 3.x, for example:

- Support for 32-bit protected-mode code when reading and writing information to and from a file system.
- Support for 32-bit dynamically allocated cache size.
- Support for an open file system architecture to enhance future system support.

Figure 27.7 shows the file system architecture used by Windows 95/98. It has the following components:

- IFS (Installable File System) Manager. This is the arbiter for the access to different file system components. On MS-DOS and Windows 3.x it was provided by interrupt 21h. Unfortunately, some add-on components did not run correctly and interfered with other installed drivers. It also did not directly support multiple network redirections (the IFS Manager can have an unlimited number of 32-bit redirectors).
- File system drivers. These provide support file systems, such as FAT-based disk devices, CD-ROM file systems and redirected network devices. They are ring 0 components, whereas with Windows 3.x supported them through MS-DOS. The two enhanced file systems are:
 - 32-bit VFAT. The 'legacy' 16-bit FAT file system suffers from many problems, such as the 8.3 file format. The 32-bit VFAT format is an enhanced form which works directly in the protected mode, and thus provides smooth multitasking as it is reentrant and multithreaded (a non reentrant system does not allow an interrupt within an interrupt). It uses the VFAT.VXD driver and uses 32-bit code for all file accesses. Another advantage is that it provides for real-mode disk caching (VCACHE), where cache memory is automatically allocated or deallocated when it is required (In Windows 3.x this was provided by the SMARTDRV.EXE program).
 - 32-bit CDFS. The 32-bit, protected-mode CDFS format (as defined in the ISO 9660 standard) gives improved CD-ROM access and support for a dynamic cache (in Windows 3.x the MSCDEX driver provided to access CD-ROMs).

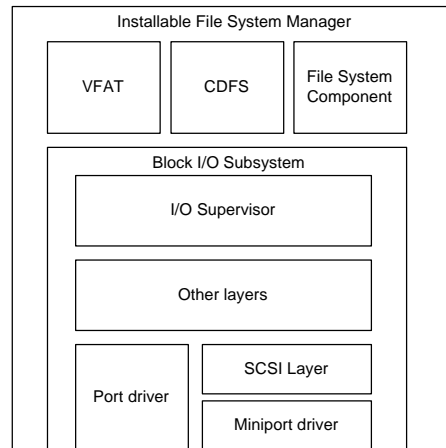


Figure 27.7 File system architecture.

- **Block I/O subsystem.** This is responsible for the actual physical access to the disk drive. Its components are:
 - **Input/Output Supervisor (IOS).** This component provides for an interface between the file systems and drivers. It is responsible for the queuing of file service requests and for routing the requests to the appropriate file system driver.
 - **Port driver.** This component is a 32-bit, protected-mode driver that communicates with a specific IDE disk device. It implements the functionality of the SCSI manager and miniport driver.
 - **SCSI layer.** This component is a 32-bit, protected-mode, universal driver model architecture for communicating with SCSI devices. It provides all the high-level SCSI functionality, and then uses a miniport driver to handle device-specific I/O calls.
 - **Miniport driver.** In Windows 95/98 these miniport driver models are used to write device-specific code. The Windows 95/98 miniport driver is a 32-bit protected-mode code, and is binary-compatible with Windows NT miniport drivers.

In Windows 95/98, the I/O Supervisor (IOS) is a VxD that controls and manages all protected-mode file system and block device drivers. It loads and initializes protected-mode device drivers and provides services needed for I/O operations. In Windows 3.x the I/O Supervisor was *BLOCKDEV. Other responsibilities of the IOS include:

- Registering drivers.
- Routing and queuing I/O requests, and sending asynchronous notifications to drivers as needed.
- Providing services that drivers can use to allocate memory and complete I/O requests.

On Windows 95/98, the IOS stores port drivers, miniport and VxD drivers in the SYSTEM\IOSUBSYS directory. The PDR file extension identifies the port drivers, MPD identifies miniport drivers and VxD (or 386) identifies the VxD drivers. Other

clients or virtual device drivers should be stored in other directories and explicitly loaded using device= entries in SYSTEM.INI. A sample listing of the IOSUBSYS directory is given next:

```
Directory of C:\WINDOWS\SYSTEM\IOSUBSYS
AIC78XX.MPD      AMSINT.MPD      APIX.VXD        ATAPCHNG.VXD
BIGMEM.DRV      CDFS.VXD        CDTSD.VXD      CDVSD.VXD
DISKTS.D.VXD    DISKVSD.VXD     DRVSPACX.VXD   ESDI_506.PDR
HSFLOP.PDR      NCRC710.MPD     NCRC810.MPD    NECATAPI.VXD
RMM.PDR         SCSI1HLP.VXD    SCSI1PORT.PDR  TORISAN3.VXD
VOLTRACK.VXD
```

27.8 Core system components

The core of Windows 95/98 has three components: User, Kernel, and GDI (graphical device interface), each of which has a pair of DLLs (one for 32-bit accesses the other for 16-bit accesses). The 16-bit DLLs allow for Win16 and MS-DOS computability.

Figure 27.8 shows that the lowest-level services provided by the Windows 95/98 Kernel are implemented as 32-bit code. In Windows 95/98 the names of the files are GDI32.DLL, KERNAL32.DLL and USER32.DLL; these are contained in the \WINDOWS\SYSTEM directory.

27.8.1 User

The User component provides input and output to and from the user interface. Input is from the keyboard, mouse, and any other input device and the output is to the user interface. It also manages interaction with the sound driver, timer, and communications ports.

Win32 applications and Windows 95/98 use an asynchronous input model for system input. With this devices have an associated interrupt handler (for example, the keyboard interrupts with IRQ1) which converts the interrupt into a message. This message is then sent to a raw input thread area, which then passes the message to the appropriate message queue. Each Win32 application can have its own message queue, whereas all Win16 applications share a common message queue. Win32 messages will be discussed in Chapter 31.

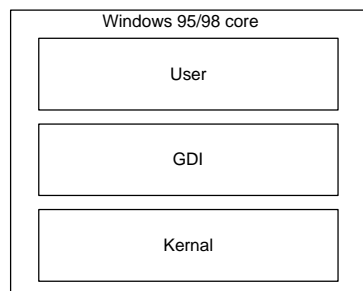


Figure 27.8 Core components.

27.8.2 Kernel

The Kernel provides for core operating system components including file I/O services, virtual memory management, task scheduling and exception handling, such as:

- File I/O services.
- Exceptions. These are events that occur as a program runs and call additional software which is outside of the normal flow of control. For example, if an application generates an exception, the Kernel is able to communicate that exception to the application to perform the necessary functions to resolve the problem. A typical exception is caused by a divide-by-zero error in a mathematical calculation, an exception routine can be designed so that it handles the error and does not crash the program.
- Virtual memory management. This resolves import references and supports demand paging for the application.
- Task scheduling. The Kernel schedules and runs threads of each process associated with an application.
- Provides services to both 16-bit and 32-bit applications by using a thunking process which is the translation process between 16-bit and 32-bit formats. It is typically used by a Win16 program to communicate with the 32-bit operating system core.

Virtual memory allows processes to allocate more memory than can be physically allocated. The operating system allocates each process a unique virtual address space, which is a set of addresses available for the process's threads. This virtual address space appears to be 4 GB in size, where 2 GB are reserved for program storage and 2 GB for system storage.

Figure 27.9 illustrates where the system components and applications reside in virtual memory. Its contents are:

- 3 GB–4 GB. All Ring 0 components.
- 2 GB–3 GB. Operating system core components and shared DLLs. These are available to all applications.
- 4 MB–2 GB. Win32-based applications, where each has its own address space. This memory is protected so that other programs cannot corrupt or otherwise hinder the application.
- 0–640 KB. Real-mode device drivers and TSRs.

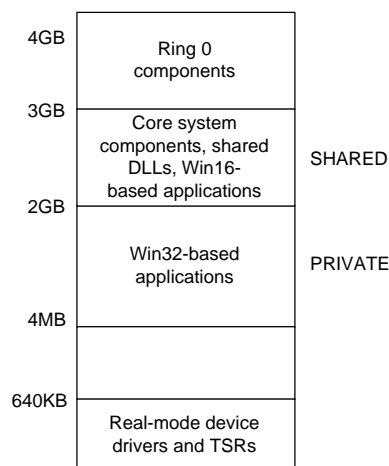


Figure 27.9 System memory usage.

27.8.3 GDI

The Graphical Device Interface (GDI) is the graphical system that:

- Manages information that appears on the screen.
- Draws graphic primitives and manipulates bitmaps.
- Interacts with device-independent graphics drivers, such as display and printer drivers.

The graphics subsystem provides input and output graphics support. Windows uses a 32-bit graphics engine (known as DIB, Device-independent Bitmaps) which:

- Directly controls the graphics output on the screen.
- Provides a set of optimized generic drawing functions for monochrome, 16-color, 16-bit high color, 256-color, and 24-bit true color graphic devices. It also supports Bézier curves and paths.
- Support for Image Color Matching for better color matching between display and color output devices.

The Windows graphics subsystem is included as a universal driver with a 32-bit mini-driver. The mini-driver provides only for the hardware-specific instructions.

The 32-bit Windows 95/98 printing subsystem has several enhancements over Windows 3.x; these include:

- They use a background thread processing to allow for smooth background printing.
- Smooth printing where the operating system only passes data to the printer when it is ready to receive more information.
- They send enhanced metafile (EMF) format files, rather than raw printer data. This EMF information is interpreted in the background and the results are then sent to the printer.
- Support for deferred printing, where a print job can be sent to a printer and then stored until the printer becomes available.
- Support for bi-directional communication protocols for printers using the Extended Communication Port (ECP) printer communication standard. ECP mode allows printers to send messages to the user or to application programs. Typical messages are: 'Paper Jam', 'Out-of-paper', 'Out-of-Memory', 'Toner Low', and so on.
- Plug-and-play.

27.9 Multitasking and threading

Multitasking involves running several tasks at the same time. It normally involves running a process for a given amount of time, before it is released and allowing another process a given amount of time. There are two forms of multitasking; these are:

- Pre-emptive multitasking. This type of multitasking involves the operating system controlling how long a process stays on the processor. This allows for smooth multitasking and is used in Windows NT/95/98 32-bit programs.

- Co-operative multitasking. This type of multitasking relies on a process giving up the processor. It is used with Windows 3.x programs and suffers from processor hogging, where a process can stay on a processor and the operating system cannot kick it off.

The logical extension to multitasking programs is to split a program into a number of parts (threads) and run each of these on the multitasking system (multithreading). A program which is running more than one thread at a time is known as a multithreaded program. Multithreaded programs have many advantages over non-multithreaded programs, including:

- They make better use of the processor, where different threads can be run when one or more threads are waiting for data. For example, a thread could be waiting for keyboard input, while another thread could be reading data from the disk.
- They are easier to test, where each thread can be tested independently of other threads.
- They can use standard threads, which are optimized for a given hardware.

They also have disadvantages, including:

- The program has to be planned properly so that threads must know on which threads they depend.
- A thread may wait indefinitely for another thread which has crashed or terminated.

The main difference between multiple processes and multiple threads is that each process has independent variables and data, while multiple threads share data from the main program.

27.9.1 Scheduling

Scheduling involves determining which thread should be run on the process at a given time. This element of time is named a time slice, and its actual value depends on the system configuration.

Each thread currently running has a base priority. The programmer who created the program sets this base priority level of the thread. This value defines how the thread is executed in relation to other system threads. The thread with the highest priority gets use of the processor.

NT and 95/98 have 32 priority levels. The lowest priority is 0 and the highest is 31. A scheduler can change a thread's base priority by increasing or decreasing it by two levels. This changes the thread's priority.

The scheduler is made up from two main parts:

- **Primary scheduler.** This scheduler determines the priority numbers of the threads which are currently running. It then compares their priority and assigns resources to them depending on their priority. Threads with the highest priority are executed for the current time slice. When two or more threads have the same priority then the threads are put on a stack. One thread is run and then put to the bottom of the stack, then the next is run and it is put to the bottom, and so on. This continues until all threads with the same priority have been run for a given time slice.

- **Secondary scheduler.** The primary scheduler runs threads with the highest priority, whereas the secondary scheduler is responsible for increasing the priority of non-executing threads (which are all other threads apart from the currently executed thread). It is thus important for giving low priority threads a chance to run on the operating system. Threads which are given a higher or lower priority are:
 - A thread which is waiting for user input has its priority increased.
 - A thread that has completed a voluntary wait also has its priority increased.
 - Threads with a computation-bound thread get their priorities reduced. This prevents the blocking of I/O operations.

Apart from these, all threads get a periodic increase. This prevents lower-priority threads hogging shared resources that are required by higher-priority threads.

27.9.2 Priority inheritance boosting

One problem that can occur is when a low priority thread access resources which are required by a higher priority thread. For example, an RS-232 program could be loading data into memory while another program requires to access the memory. One method which can be used to overcome this is Priority Inheritance Boosting. In this case, low priority threads gets a boost so that they can quickly release resources. For example, if a system has three threads: Thread A, Thread B and Thread C. If Thread A has the highest priority and it requires a resource from Thread C then Thread C gets a boost in its priority. Thread A remains blocked until Thread C releases the required resource. When it does release it then Thread C goes back to its normal priority and Thread A then gets access to the resource.

27.10 Plug-and-play process

Plug-and-play allows the operating system to configure hardware as required. On system startup, the configuration manager scans the system hardware. When it finds a new plug-and-play device it does the following:

- **Sets the device into configuration mode.** This is achieved by using 3 I/O ports. Some data (the initiation key) is written to one of the ports and enables the Plug-and-Play logic.
- **Isolate and identify each device.** Each device is isolated, one at a time. The method used is to assign each device a unique number, which is a unique handle for the device. This number is made from a device ID and a serial number.
- **Determine device specifications.** Each device sends its functionality to the operating system, such as how many joysticks it supports, its audio functions, its networking modes, and so on.
- **Allocate resources.** The operating system then allocates resources to the device depending on its functionality and the Plug-and-Play device is informed of the allocated resources (such as IRQs, I/O addresses, DMA channels, and so on). It also checks for conflicts on these resources.
- **Activate device.** When the above have been completed the device is enabled. Only the initiation key can re-initialize the device.

27.11 Windows NT architecture

Windows NT uses two modes:

- User mode. This is a lower privileged mode than kernel mode. It has no direct access to the hardware or to memory. It interfaces to the operating system through well-defined API (Application Program Interface) calls.
- Kernel mode. This is a privileged mode of operation and allows all code direct access to the hardware and memory, including memory allocated to user mode processes. Kernel mode processes also have a higher priority over user mode processes.

Figure 27.10 shows an outline of the architecture of NT. It can be seen that only the kernel mode has access to the hardware. This kernel includes an executive services which include managers (for I/O, interprocess communications, and so on) and device drivers (which control the hardware). Its parts include:

- Microkernel. Controls basic operating system services, such as interrupt handling and scheduling.
- HAL. This a library of hardware-specific programs which give a standard interface between the hardware and software. This can either be Microsoft written or manufacturer provided. They have the advantage of allowing for transportability of programs across different hardware platforms.
- Win32 Window Manager. Supports Win32, MS-DOS and Windows 3.x applications.

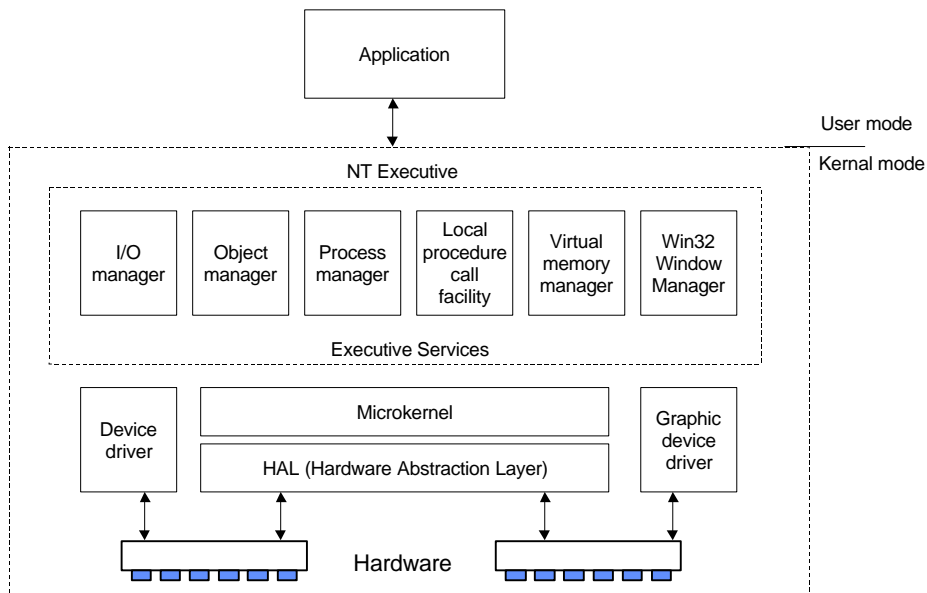


Figure 27.10 NT architecture.

27.11.1 MS-DOS support

Windows NT supports MS-DOS-based applications with an NT Virtual DOS Machine (NTVDM), where each MS-DOS application has its own NTVDM. The NTVDM is started by the application Ntvdm.exe and when this has started the application communicates with two system files Ntio.sys (equivalent to IO.SYS) and Ntdos.sys (equivalent to MSDOS.SYS). Note that the AUTOEXEC.BAT and CONFIG.SYS files have also been replaced by Autoexec.nt and Config.nt (which are normally located in \WINNT\System32).

Multiple NTVDMs have the advantage of being reliable because if one NTVDM fails then it does not effect any others. It also allows MS-DOS-based applications to be multitasked. Unfortunately, each NTVDM needs at least 1 MB of physical memory.

Some MS-DOS applications require direct access to the hardware. NT supports this by providing virtual device drivers (VDDs). These detect a call to hardware and communicate with the NT 32-bit device driver.

Windows NT communicates with hardware through device drivers. These drivers have a .sys file extension. An example listing of these is given next:

Directory of C:\WINNT\system32\drivers

afd.sys	atapi.sys	atdisk.sys	beep.sys
cdaudio.sys	cdfs.sys	cdrom.sys	changer.sys
cirrus.sys	disk.sys	diskdump.sys	diskperf.sys
fastfat.sys	floppy.sys	ftdisk.sys	hpscan16.sys
i8042prt.sys	kbdclass.sys	ksecdd.sys	modem.sys
mouclass.sys	msfs.sys	mup.sys	ndis.sys
netdtecl.sys	npfs.sys	ntfs.sys	null.sys
parallel.sys	parport.sys	parvdm.sys	pcmcia.sys
scsiport.sys	scsiprnt.sys	scsiscan.sys	serial.sys
sfloppy.sys	streams.sys	tape.sys	tdi.sys
vga.sys	videoprt.sys		

With this, virtual memory applications can have access to the full available memory but NT then maps this to a private memory range (called a virtual memory space). It maps physical memory to virtual memory in 4 KB blocks (called pages). This was previously illustrated in Figure 27.6. The driver used to perform the page file access is Pagefile.sys (which is normally found in the top-level directory).

Windows NT has 32 levels of priority (0 to 31). Levels 0 to 15 are used for dynamic applications (such as non-critical operations) and 16 to 31 are used for real-time applications (such as Kernel operations). NT provides a virtual memory by paging file(s) onto the hard disk. Priority levels 0 to 15 can be paged, but levels 16 to 31 cannot.

A summary of the system32 directory is shown below. The wowdeb.exe and wowexec.exe files allow Windows 3.x programs to run in a 32-bit environment.

Directory of C:\winnt\system32

ansi.sys	append.exe	at.exe	atsvc.exe
attrib.exe	autoexec.nt	backup.exe	bootok.exe
bootvrfy.exe	cacls.exe	chcp.com	chkdsk.exe
clipsrv.exe	comm.drv	command.com	comp.exe
compact.exe	config.nt	control.exe	convert.exe
country.sys	csrss.exe	dcomcnfg.exe	ddshare.exe
ddhelp.exe	ebug.exe	diskcomp.com	diskcopy.com
diskperf.exe	doskey.exe	dosx.exe	DRIVERS
edit.com	exe2bin.exe	expand.exe	fastopen.exe
fc.exe	find.exe	findstr.exe	finger.exe
fontview.exe	forcedos.exe	format.com	ftp.exe
gdi.exe	graftabl.com	graphics.com	grpconv.exe

help.exe	himem.sys	inetins.exe	internat.exe
kbl6.com	keyb.com	keyboard.drv	keyboard.sys
krnl386.exe	label.exe	lights.exe	lodctr.exe
mem.exe	mode.com	more.com	mpnotify.exe
mscdexnt.exe	nddeagnt.exe	nddeapir.exe	net.exe
nlsfunc.exe	notepad.exe	ntdos.sys	ntio.sys
ntvdm.exe	os2ss.exe	pax.exe	pentnt.exe
ping.exe	portuas.exe	posix.exe	print.exe
psxss.exe	rdisk.exe	recover.exe	redir.exe
replace.exe	restore.exe	rpcss.exe	rundll32.exe
runonce.exe	savedump.exe	setup.exe	setver.exe
share.exe	shmgrate.exe	skeys.exe	smss.exe
sort.exe	SPOOL	sprestrt.exe	subst.exe
syncapp.exe	sysedit.exe	systray.exe	taskman.exe
taskmgr.exe	telnet.exe	tree.com	unlodctr.exe
ups.exe	user.exe	userinit.exe	VIEWERS
win.com	winhlp32.exe	winspool.exe	winver.exe
wowdeb.exe	wowexec.exe		

27.12 Exercises

- 27.12.1** Discuss the architecture of the 32-bit Windows system.
- 27.12.2** Discuss how Windows 95/98 use VxD device drivers to interface to the hardware. How do equipment manufacturers develop drivers which use the VxD drivers. How do device drivers in NT differ from Windows 95/98?
- 27.12.3** Explain how the Configuration Manager is used to determine the devices which are connected to the system.
- 27.12.4** Explain how the operation of the Virtual Machine Manager.
- 27.12.5** Explain the main differences between pre-emptive and co-operative multitasking. Discuss also how multitasking and threading is implemented.
- 27.12.6** Discuss how 95/98 use a priority systems to schedule processes.