CHAPTER 4

# Base System Architecture

Ease on the surface requires power and speed at the core, and the modern, 32-bit architecture of Windows 95 meets these requirements. Freed from the limitations of MS-DOS, Windows 95 preemptively multitasks for better PC responsiveness—for example, users no longer have to wait while the system copies files—and delivers increased robustness and protection for applications. Windows 95 also provides the foundation for a new generation of easier, more powerful multithreaded 32-bit applications. And most important, Windows 95 delivers this power and robustness on today's average PC platform while scaling itself to take advantage of additional memory and CPU cycles.

The mission of Windows 95 is to deliver a complete, integrated operating system that offers modern, 32-bit operating system technology and includes built-in connectivity support. In addition to the high-level mission of Windows 95, market requirements dictate delivery of a high performance, robust, and completely backward-compatible operating system that provides a platform for a new generation of applications.

This chapter discusses the base system architecture used by Windows 95. The base architecture covers low-level system services for managing memory, accessing disk devices, and providing robust support for running applications.

## Summary of Improvements over Windows 3.1

Improvements made to the base architecture of Windows 95 result in many benefits to users. Following is a summary of some of the key improvements:

- **A fully integrated 32-bit protected-mode operating system.** The need for a separate copy of MS-DOS has been eliminated.
- **Preemptive multitasking and multithreading support.** System responsiveness and smooth background processing have been improved.
- **32-bit installable file systems.** Systems such as VFAT, CDFS, and network redirectors provide better performance, use of long filenames, and an open architecture that supports future growth.
- **32-bit device drivers.** Available throughout the system, these drivers deliver improved performance and intelligent memory use.
- **A complete 32-bit kernel.** Included are memory management, scheduler, and process management.

- **Improved system-wide robustness and cleanup.** This more stable and reliable operating environment also cleans up after an application ends or crashes.
- **More dynamic environment configuration.** The need for users to tweak their systems is reduced.
- **Improved system capacity.** Included are better system resource limits to address the problems Windows 3.1 users encountered when running multiple applications.

# A Fully Integrated Operating System

The first thing that users of Windows 3.1 and MS-DOS will notice when they turn on their computers is the lack of an MS-DOS command prompt from which they would formerly have invoked Windows. Windows 95 is a tightly integrated operating system that features a preemptive multitasking kernel which boots directly into the graphical UI and also provides full compatibility with the MS-DOS operating system.

Many of the components of Windows 95 overcome limitations inherent in MS-DOS and Windows 3.1. However, these improvements do not come at the cost of compatibility with existing software, hardware, or computing environments.

# A Preemptive Multitasking Operating System

The job of the operating system is to provide services to the applications running on the system and, in a multitasking environment, to provide support that allows more than one application to run concurrently. In Windows 3.1, multiple applications ran concurrently in a *cooperative* multitasking manner. The Windows 3.1 operating system required an application to check the message queue every once in a while to allow the operating system to give control to other running applications. Applications that did not check the message queue on a frequent basis effectively hogged all the CPU time and prevented switching to another running task.

Windows 95 uses a *preemptive* multitasking mechanism for running Win32–based applications, and the operating system takes control away from or gives control to another running task depending on the needs of the system. Unlike Win16–based applications, Win32–based applications do not need to *yield* to other running tasks in order to multitask in a friendly manner. (Win16–based applications are still cooperatively multitasked for compatibility reasons.) Windows 95 provides a mechanism called *multithreading* that allows Win32–based applications to take advantage of the preemptive multitasking nature of the operating system and that facilitates concurrent application design. In operating-system terms, a running Win32–based application is called a *process*. Each process consists of at least a single *thread*. A thread is a unit of code that can get a time slice from the operating system to run concurrently with other units of code. It must be associated with a process, and it identifies the code path flow as the process is run by the operating system. A Win32-based application can *spawn* (or initiate) multiple threads for a given process. Multiple threads enhance the application for the user by improving throughput and responsiveness and aiding background processing.

Because of the preemptive multitasking nature of Windows 95, threads of execution allow background code to be processed in a smooth manner. For example, a word processing application (process) can implement multiple threads to enhance operation and simplify interaction with the user. The application might have one thread of code responding to the keys pressed on the keyboard by the user to enter characters in a document, while another

thread is performing background operations such as spell-checking or pagination, and yet another thread is spooling a document to the printer in the background.

Some available Windows 3.1 applications provide functionality similar to that just described. However, because Windows 3.1 does not provide a mechanism for supporting multithreaded applications, the application developer has to implement a threading scheme. The use of threads in Windows 95 facilitates the adding of asynchronous processing of information to applications by their developers.

Applications that use multithreading techniques can also take advantage of improved processing performance available from a symmetric multiprocessing (SMP) system running Windows NT, which allows different portions of the application code to run on different processors simultaneously. (Windows NT uses a thread as the unit of code to schedule symmetrically among multiple processors.)

For information about how Windows 95 runs MS-DOS–based applications in a preemptive manner (as Windows 3.1 does today), Win16–based applications in a cooperative manner (as Windows 3.1 does today), and Win32–based applications in a preemptive manner (as Windows NT does today) see later sections in this chapter.

## No CONFIG.SYS or AUTOEXEC.BAT?

Windows 95 doesn't need the separate CONFIG.SYS or AUTOEXEC.BAT file required by MS-DOS and Windows 3.1. Instead, Windows 95 is intelligent about the drivers and settings it requires and automatically loads the appropriate driver files or makes the appropriate configuration settings during its boot process. If a CONFIG.SYS or AUTOEXEC.BAT file is present, the settings in these files are used to set the global environment. For example, the default search path or the default appearance of the command prompt can be defined by using the appropriate entries in the AUTOEXEC.BAT file. While Windows 95 itself does not need a CONFIG.SYS or AUTOEXEC.BAT file, compatibility is maintained with existing software or environments that may require one or both of these files.

## No MS-DOS?

Unlike Windows 3.1, Windows 95 is not dependent on real-mode operating system components for its interaction with the file system. However, the Windows 95 boot sequence does begin by loading real-mode operating system components that are compatible with MS-DOS. During the boot sequence, support for loading any real-mode drivers and TSRs that are identified in a CONFIG.SYS or AUTOEXEC.BAT file is processed. Because these drivers explicitly look for or use MS-DOS application support, the real-mode operating system components of Windows 95 help maintain compatibility with software that users already have on their system. After the real-mode drivers are loaded, Windows 95 begins loading the protect-mode operating system components. In some cases where a protect-mode Windows–based driver is provided, Windows 95 actually removes real-mode drivers from memory. More information about this subject is given later.

# 32-Bit Versus 16-Bit Components

To provide a good balance between delivering compatibility with existing applications and drivers, decreasing the size of the operating system working set, and offering

improved system performance over Windows 3.1, Windows 95 uses a combination of 32-bit and 16-bit code. In general, 32-bit code is provided in Windows 95 to maximize the performance of the system, while 16-bit code balances the requirements for reducing the size of the system and maintaining compatibility with existing applications and drivers. System reliability is also improved without a cost in terms of compatibility or increased size.

The design of Windows 95 deploys 32-bit code wherever it significantly improves performance without sacrificing application compatibility. Existing 16-bit code is retained where it is required to maintain compatibility, or where 32-bit code would increase memory requirements without significantly improving performance. All of the I/O subsystems and device drivers in Windows 95, such as networking and file systems, are fully 32-bit, as are all the memory management and scheduling components (the kernel and virtual memory manager). Figure 26 depicts the relative distribution of 32-bit code versus 16-bit code present in Windows 95 for system-level services.

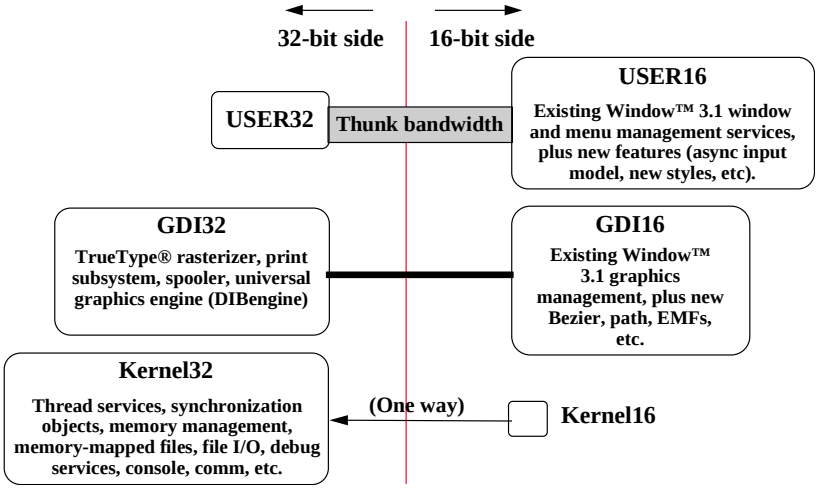


**Figure 26. The relative code distribution in Windows 95**

As shown in the figure, the lowest-level services provided by the operating system kernel are provided as 32-bit code. Most of the remaining 16-bit code consists of hand-tuned assembly language, delivering performance that rivals some 32-bit code used by other operating systems available on the market today. Many functions provided by the Graphics Device Interface (GDI) have been moved to 32-bit code, including the spooler and printing subsystem, the font rasterizer, and the drawing operations performed by the graphics DIB engine. Much of the window management code (User) remains 16-bit to retain application compatibility.

In addition, Windows 95 improves upon the MS-DOS and Windows 3.1 environments by implementing many device drivers as 32-bit protected-mode code. Virtual device drivers in Windows 95 assume the functionality provided by many real-mode MS-DOS–based device drivers, eliminating the need to load them in MS-DOS. This technique results in a minimal conventional-memory footprint, improved performance, and improved reliability and stability of the system over MS-DOS–based device drivers.

# Virtual Device Drivers

A virtual device driver is a 32-bit, protected-mode driver that manages a system resource, such as a hardware device or installed software, so that more than one application can use the resource at the same time. To understand the improvements available in Windows 95 over the combination of MS-DOS and Windows 3.1, it helps to have a basic understanding of what virtual device drivers (VxDs) are and the role they play in the Windows 95 environment.

The term *VxD* refers to a general virtual device driver, with *x* representing the type of device driver. For example, VDD is a virtual device driver for a display device, a VTD is a virtual device driver for a timer device, a VPD is a virtual device driver for a printer device, and so on. Windows uses virtual devices to support multitasking for MS-DOS–based applications, virtualizing the different hardware components on the system to make it appear to each MS-DOS virtual machine (VM) that it is executing on its own computer. Virtual devices work in conjunction with Windows to process interrupts and carry out I/O operations for a given application without disrupting how other applications run.

Virtual device drivers support all hardware devices for a typical computer, including the programmable interrupt controller (PIC), timer, direct-memory-access (DMA) device, disk controller, serial ports, parallel ports, keyboard device, math coprocessor, and display adapter. A virtual device driver can contain the device-specific code needed to carry out operations on the device. A virtual device driver is required for any hardware device that has settable operating modes or retains data over any period of time. In other words, if the state of the hardware device can be disrupted by switching between multiple applications, the device must have a corresponding virtual device. The virtual device keeps track of the state of the device for each application and ensures that the device is in the correct state whenever an application continues.

Although most virtual devices manage hardware, some manage only installed software, such as an MS-DOS device driver or a terminate-and-stay-resident (TSR) program. Such virtual devices often contain code that either emulates the software or ensures that the software uses only data applicable to the currently running application. ROM BIOS, MS-DOS, MS-DOS device drivers, and TSRs provide device-specific routines and operating system functions that applications use to indirectly access the hardware devices. Virtual device drivers are sometimes used to improve the performance of installed software—for example, the 80386 and compatible microprocessors can run the 32-bit protected-mode code of a virtual device more efficiently than the 16-bit real-mode code of an MS-DOS device driver or TSR. In addition, performance is enhanced by eliminating ring transitions that result in executing 32-bit applications that access 16-bit real-mode services, because with virtual device drivers, the system can stay in protected-mode.

Windows 95 benefits from providing more device driver support implemented as a series of VxDs in the Windows environment, instead of using the device drivers previously available as real-mode MS-DOS device drivers. Functionality that was previously supported as MS-DOS device drivers but is now supported as VxDs in Windows 95 includes the following components:

- MS-DOS FAT file system
- SmartDrive
- CD-ROM file system
- Network card drivers and network transport protocols
- Network client redirector and network peer server
- Mouse driver

- MS-DOS SHARE.EXE TSR
- Disk device drivers including support for SCSI devices
- DriveSpace (and DoubleSpace) disk compression

In summary, in Windows 95 VxDs provide the following advantages:

- Improved performance as a result of a 32-bit code path and the elimination or reduction of the need to switch between real and protected mode
- Reduced conventional memory footprint by providing device driver and TSR functionality as protected-mode components that reside in extended memory
- Improved system stability and reliability compared to MS-DOS device driver counterparts

Virtual device drivers in Windows 95 can be identified by .VXD extensions, and virtual device drivers from Windows 3.1 can be identified by .386 extensions.

# The System Architecture Layout in Windows 95

Figure 27 illustrates the layout of the base system architecture for Windows 95. Components of the system are divided between Ring 0 and Ring 3 code, offering different levels of system protection. The Ring 3 code is protected from other running processes by protection services provided by the Intel processor architecture. The Ring 0 code consists of low-level operating system services such as the file system and the virtual machine manager.

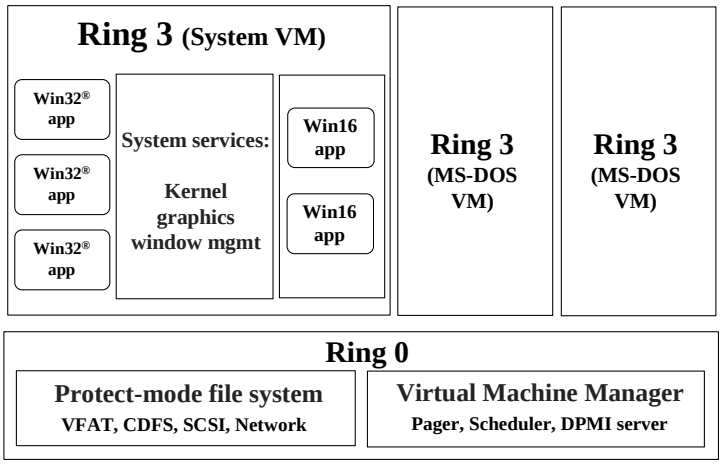


**Figure 27. The integrated architecture of Windows 95, which supports running MS-DOS–based, Win16–based, and Win32–based applications**

Figure 27 also depicts the way that MS-DOS–based, Win16–based, and Win32–based applications run in the system. The following section discusses the provisions that the system makes for running these applications.

# Support for Win16–Based Applications

Win16–based (16-bit) applications run together within a unified address space and run in a cooperatively multitasking manner, as they do under Windows 3.1. Win16–based applications benefit from the preemptive multitasking of other system components, including the 32-bit print and communications subsystem and the improvements made in system robustness and protection from the system kernel in Windows 95.

Based on customer needs, resource needs, and market needs, three goals drove the architectural design of Win16–based application support: compatibility, size, and performance. Functionality adjustments, such as preemptively running Win16–based applications together in the Win16 subsystem or running Win16–based applications in separate VMs, were considered, but each of the options considered failed to meet the three design goals. The following discussion provides some insight into the architecture of Windows 95 as far as running Win16–based applications in a fast, stable, and reliable way is concerned.

## Compatibility

First and foremost, Windows 95 needs to run existing Win16–based applications without modification. This factor is extremely important to existing users who want to take advantage of the new functionality offered in Windows 95, such as 32-bit networking, but don't want to have to wait until new Windows 95–enabled applications are available on the market.

Windows 95 builds upon the Windows 3.1 platform to provide support for running existing Win16–based applications and using existing Windows–based device drivers, while providing support for the next generation of 32-bit applications and components. Windows 95 extends the Windows 3.1 architecture in areas that have little or no impact on compatibility, as well as enhances the architecture to deliver a faster, more powerful 32-bit operating system.

## Size

While many newer computer purchases are Intel 80486–based computers with 4 MB or 8 MB (or more) of memory, a high percentage of 80386DX–based computers with 4 MB of memory running Windows 3.1 are still in use. To support the needs of the market, Windows 95 must run on a base platform of an Intel 80386DX–based computer with 4 MB of RAM and provide access to its new features and functionality without requiring an upgrade of existing hardware or the addition of more RAM.

To meet its goals, Windows 95 is designed to occupy a working set of components no larger than Windows 3.1, thereby ensuring that any Win16–based application running at a perceived speed on a 4 MB or 8 MB (or greater) computer runs at the same (or higher) speed under Windows 95 without suffering any performance degradation. To meet the size goals of Windows 95, Win16–based applications run within a unified address space, resulting in little overhead beyond that required by Windows 3.1 to support the running of Windows–based applications. Running in a unified address space allows Windows 95 not only to fit on a 4 MB computer, but also to perform well. The architecture of Windows 95 includes innovative design features, such as dynamically loadable VxDs, to decrease the working set of components and memory requirements used by the operating system.

Meeting the size design goal (as well as meeting the compatibility goal) precluded the strategies of running Win16–based applications in a separate VM (by running a separate copy of Windows 3.1 on top of the operating system, which would involve paying a several megabyte "memory tax" for each application) as OS/2 does, or of emulating Windows 3.1 on top of the Win32 subsystem (which would also involve paying a "memory tax" for running Win16–based applications) as Windows NT does.

Running Win16–based applications in separate VMs is very expensive memory-wise. This strategy would require separate GDI, USER, and KERNEL code in each VM that is created, increasing the working set by as much as 2 MB for each Win16–based application that is running (as is the case with OS/2 for Windows). On a computer with 16 MB or more, this increase may not appear significant. However, bearing in mind the existing installed base of computers, running Win16–based applications in their own separate VMs in 4 MB of memory is impossible, and running them in 8 MB with the level of performance observed and expected under Windows 3.1 is very difficult.

# Performance

Users expect their existing Win16–based applications to run as fast as or faster than they do under Windows 3.1. Both Win16–based applications and MS-DOS–based applications benefit from the 32-bit architecture of Windows 95, including the increased use of 32-bit device driver components and 32-bit subsystems.

Win16–based applications run within a unified address space and interact with the system much as they do under Windows 3.1. Running Win16–based applications in separate VMs requires either mapping Win16 system components in each address space, as Windows NT does, or providing a separate copy of each system component in each address space, as OS/2 for Windows does. The additional memory overhead required for Win16 system components in each VM to run a Win16–based application has a negative impact on system performance.

Windows 95 balances the issue of system protection and robustness with the desire for better system performance and improves on the system robustness of Windows 3.1. The improvements in this area are briefly discussed in the next section and are described in greater detail in Chapter 5, "Robustness."

# Protection

The support for running Win16–based applications provides protection of the system from other running MS-DOS–based applications or Win32–based applications. Unlike Windows 3.1, an errant Win16–based application cannot easily bring down the system or other running processes on the system. While Win32–based applications benefit the most from system memory protection, the robustness improvements in Windows 95 result in a more stable and reliable operating environment than Windows 3.1.

Win16–based applications run within a unified address space and cooperatively multitask as they do under Windows 3.1. The improvements made to overall system-wide robustness greatly enhance the system's ability to recover from an errant application, and improved cleanup of the system lessens the likelihood of application errors. General protection faults (GPFs) under Windows 3.1 are most commonly caused by an application overwriting its own memory segments, rather than by an application overwriting memory belonging to another application. Windows 3.1 did not recover gracefully when a Windows–based application crashed or hung. When a GPF caused the system to halt an

application, the system commonly left allocated resources in memory, causing the system to degenerate.

Because of improved protection in Windows 95, an errant Win16–based application cannot easily bring down either the system as a whole or other running MS-DOS or Win32–based applications. At most, it can impact other running Win16–based applications.

Other protection improvements include the use of separate message queues for each running Win32–based application. The use of a separate message queue for the Win16 address space and for each running Win32–based application provides better recovery of the system and doesn't halt the system if a Win16–based application hangs.

## Robustness Improvements

System robustness when running Win16–based applications under Windows 95 is greatly improved over Windows 3.1. Windows 95 now tracks resources allocated by Win16–based applications and uses the information to clean up the system after an application exits or ends abnormally, thus freeing up unused resources for use by the rest of the system.

Robustness improvements are discussed in Chapter 5, "Robustness."

# Support for MS-DOS–Based Applications

Windows 95 includes many improvements over Windows 3.1 for running MS-DOS–based applications. As with Windows 3.1, each MS-DOS–based application runs in its own VM. A VM takes advantage of the Intel 80386 (and higher) architecture, which allows multiple 8086-compatible sessions to run on the CPU and thereby allows existing MS-DOS applications to run preemptively with the rest of the system. As with Windows 3.1, the use of virtual device drivers provides common regulated access to hardware resources, causing each application running in a VM to think that it is running on its own individual computer and allowing applications not designed to multitask to run concurrently with other applications.

Windows 95 provides a flexible environment for running MS-DOS–based applications. In Windows 3.1, users sometimes needed to exit Windows to run MS DOS–based applications that were either ill-behaved or required direct access to system resources. MS-DOS–based application compatibility is improved in Windows 95 to the point that almost all MS-DOS–based applications should run under Windows 95.

A detailed discussion of the improvements made to the support for running MS-DOS–based applications within the Windows environment is provided in Chapter 6, "Support for Running MS-DOS–based Applications."

## Protection

In Windows, VMs are fully protected from one another, as well as from other applications running on the system. This protection prevents errant MS-DOS–based applications from overwriting memory occupied or used by system components or other applications. If an MS-DOS–based application attempts to access memory outside of its address space, the system notifies the user and terminates the MS-DOS–based application.

## Robustness Improvements

System robustness is greatly improved when running MS-DOS–based applications in Windows 95. Robustness is discussed in Chapter 5, "Robustness."

# Support for Win32–Based Applications

Win32–based applications can fully exploit and benefit significantly from the design of the Windows 95 architecture. In addition, each Win32–based application runs in its own fully protected, private address space. This strategy prevents Win32–based applications from crashing each other, from crashing running MS-DOS–based applications, from crashing running Win16–based applications, or from crashing the Windows 95 system as a whole.

Win32–based applications feature the following benefits over Win16–based applications in Windows 95 and over Windows 3.1:

- Preemptive multitasking
- Separate message queues
- Flat address space
- Compatibility with Windows NT
- Long filename support
- Memory protection
- Robustness improvements

## Preemptive Multitasking

Unlike the cooperative multitasking used by Win16–based applications under Windows 3.1, 32-bit Win32–based applications are preemptively multitasked in Windows 95. The operating system kernel is responsible for scheduling the time allotted for running applications in the system, and support for preemptive multitasking results in smoother concurrent processing and prevents any one application from utilizing all system resources without permitting other tasks to run.

Win32–based applications can optionally implement threads to improve the granularity at which they multitask within the system. The use of threads by an application improves the interaction with the user and results in smoother multitasking operation.

## Separate Message Queues

Under Windows 3.1, the system uses the point when an application checks the system message queue as the mechanism to pass control to another task, allowing that task to run in a cooperative manner. If an application doesn't check the message queue on a regular basis, or if the application hangs and thus prevents other applications from checking the message queue, the system keeps the other tasks in the system suspended until the errant application ends.

Each Win32–based application has its own message queue and is thus not affected by the behavior of other running tasks on their own message queues. If a Win16–based application hangs, or if another running Win32–based application crashes, a Win32–based application continues to run preemptively and can still receive incoming messages or event notifications.

Message queues are discussed in more detail in Chapter 5, “Robustness.”

# Flat Address Space

Win32–based applications benefit from improved performance and simpler construct because they can access memory in a linear fashion, rather than being limited to the segmented memory architecture used by MS-DOS and Windows 3.1. To provide a means of accessing high amounts of memory using a 16-bit addressing model, the Intel CPU architecture provides support for accessing 64K chunks of memory, called *segments*, at a time. Applications and the operating system suffer a performance penalty under this architecture because of the manipulations required by the processor for mapping memory references from the segment/offset combination to the physical memory structure.

The use of a flat address space by the 32-bit components in Windows 95 and by Win32–based applications allows application and device driver developers to write software without the limitations or design issues inherent in the segmented memory architecture used by MS-DOS and Windows 3.1.

# Compatibility with Windows NT

Win32–based applications that exploit Win32 APIs common to Windows 95 and Windows NT can run without modification on either platform on Intel–based computers. The commonality of the Win32 API provides a consistent programmatic interface and allows application developers to leverage a single development effort to deliver software that runs on multiple platforms. It also provides scaleability of applications and broadens the base of platforms available for running ISV or custom applications with minimal additional effort.

Application developers are encouraged to develop applications either under Windows 95 or under Windows NT and to test compatibility on both platforms.

# Long Filename Support

Win32–based applications that call the file I/O functions supported by the Win32 API benefit from the ability to support and manipulate filenames of up to 255 characters with no additional development effort. To ease the burden of the application developer, the Win32 APIs and common dialog support handle the work of manipulating long filenames, and the file system provides compatibility with MS-DOS and other systems by automatically maintaining the traditional 8.3 filename.

# Memory Protection

Each Win32–based application runs in its own private address and is protected by the system from other applications or processes that are running in the system. Unlike errant Win16–based applications under Windows 3.1, errant Win32–based applications under Windows 95 end only themselves, instead of bringing down the entire system if they attempt to access memory belonging to another application.

The use of separate message queues for Win32–based applications also ensures that the system continues to run if an application hangs or stops responding to messages or events.

# Robustness Improvements

Win32–based applications benefit from the highest level of system robustness supported under Windows 95. Resources allocated for each Win32–based application are tracked on a per-thread basis and are automatically freed when the application ends. If an application hangs, users can perform a *local reboot* operation to end the hung application without affecting other running tasks, and the system then cleans up properly.

Detailed information about robustness enhancements is given in Chapter 5, "Robustness."

# 32-Bit File System Architecture

The file system in Windows 95 has been redesigned to support the characteristics and needs of the multitasking nature of its kernel. The changes present in Windows 95 provide many benefits to users and have the following results:

- **Improved ease of use.** Ease of use is improved by the support of long filenames because users no longer need to reference files by the MS-DOS 8.3 filename structure. Instead they can use up to 255 characters to identify their documents. Ease of use is also improved by hiding the filename extensions.
- **Improved performance.** As in Windows for Workgroups 3.11, file I/O performance is improved dramatically over Windows 3.1 by featuring 32-bit protected-mode code for reading information from and writing information to the file system, reading from and writing to the disk device, and intelligent 32-bit caching mechanisms (a full 32-bit code path is available from the file system to the disk device).
- **Improved system stability and reliability.** File system components implemented as 32-bit protected-mode device drivers offer improved system stability and reliability over their MS-DOS device driver counterparts because they can remain in protected mode for code execution and because they leverage existing driver technology first implemented in Windows NT and also available in Windows for Workgroups 3.11.

# Architecture Overview

Windows 95 features a layered file system architecture that supports multiple file systems and provides a protected-mode path from the application to the media device, resulting in improved file and disk I/O performance over Windows 3.1. The following features are included in the new file system architecture:

- Win32 API support
- Long filename support
- 32-bit FAT file system
- 32-bit CD-ROM file system
- Dynamic system cache for file and network I/O
- Open architecture for future system support
- Disk device driver compatibility with Windows NT

Figure 28 depicts the file system architecture used by Windows 95.

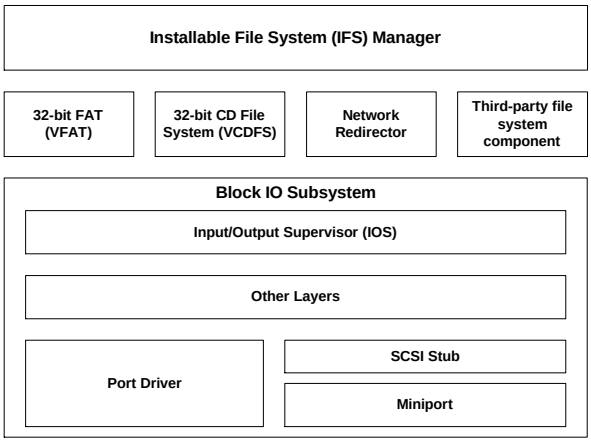


**Figure 28. The file system architecture**

The file system architecture in Windows 95 is made up of the following components:

- **Installable File System (IFS) Manager.** The IFS Manager is responsible for arbitrating access to different file system components.
- **File system drivers.** The file system drivers layer includes access to file allocation table (FAT)–based disk devices, CD-ROM file systems, and redirected network device support.
- **Block I/O subsystem.** The block I/O subsystem is responsible for interacting with the physical disk device.

Components of each of these layers are examined in the next three sections.

# The Installable File System Manager

Under MS-DOS and Windows 3.1, the MS-DOS Int 21h interrupt is responsible for providing access to the file system to manipulate file information on a disk device. To support redirected disk devices, such as a network drive or a CD-ROM drive, other system components, such as the network redirector, would hook the Int 21h function so that it could examine a file system request to determine whether it should handle the request or the base file system should. Although this mechanism provided the ability to add additional device drivers, some add-on components were ill-behaved and interfered with other installed drivers.

Another problem with the MS-DOS–based file system was the difficulty in supporting the loading of multiple network redirectors to provide concurrent access to different network types. Windows for Workgroups provided support for running the Microsoft network redirector at the same time as an additional network redirector, such as Novell NetWare, Banyan VINES, or SUN PC-NFS. However, support for running more than two network redirectors at the same time was not provided.

The key to friendly access to disk and redirected devices in Windows 95 is the Installable File System (IFS) Manager. The IFS Manager is responsible for arbitrating access to file system devices, as well as other file system device components.

# File System Drivers

Windows 95 includes support for the following file systems:

- 32-bit file allocation table (VFAT) driver
- 32-bit CD-ROM file system (CDFS) driver
- 32-bit network redirector for connectivity to Microsoft network servers, such as Windows NT Server, along with a 32-bit network redirector to connect to Novell NetWare servers

In addition, third parties will use the IFS Manager APIs to provide a clean way of concurrently supporting multiple device types and adding additional disk device support and network redirector support.

## The 32-Bit Protected-Mode FAT File System

The 32-bit VFAT driver provides a 32-bit protected-mode code path for manipulating the file system stored on a disk. It is also re-entrant and multithreaded, providing smoother multitasking performance. The 32-bit file access driver is improved over that provided originally with Windows for Workgroups 3.11 and is compatible with more MS-DOS-device drivers and hard disk controllers.

Benefits of the 32-bit file access driver over MS-DOS–based driver solutions include the following:

- Dramatically improved performance and real-mode disk caching
- No conventional memory used (replacement for real-mode SmartDrive)
- Better multitasking when accessing information on disk with no blocking
- Dynamic cache support

Under MS-DOS and Windows 3.1, manipulation of the FAT and writing to or reading from the disk is handled by the Int 21h MS-DOS function and is 16-bit real-mode code. Being able to manipulate the disk file system from protected mode removes or reduces the need to transition to real mode in order to write information to the disk through MS-DOS, which results in a performance gain for file I/O access.

The 32-bit VFAT driver interacts with the block I/O subsystem to provide 32-bit disk access to more device types than are supported by Windows 3.1. Support is also provided for mapping to existing real-mode disk drivers that may be in use on a user's system. The combination of the 32-bit file access and 32-bit disk access drivers results in significantly improved disk and file I/O performance.

### The 32-Bit Cache

The 32-bit VFAT works in conjunction with a 32-bit protected-mode cache (VCACHE) driver and replaces and improves on the 16-bit real-mode SmartDrive disk cache software provided with MS-DOS and Windows 3.1. The VCACHE driver features a more intelligent algorithm for caching information read from or written to a disk drive than SmartDrive, and results in improved performance when reading information from cache. The VCACHE driver is also responsible for managing the cache pool for the CD-ROM File System (CDFS) and the provided 32-bit network redirectors.

Another big improvement VCACHE provides over SmartDrive is that the memory pool used for the cache is dynamic and is based on the amount of available free system

memory. Users no longer need to statically allocate a block of memory to set aside as a disk cache because the system automatically allocates or deallocates memory used for the cache based on system use. Because of intelligent cache use, the performance of the system also scales better than with Windows 3.1 or Windows for Workgroups 3.11.

## The 32-Bit Protected-Mode CD-ROM File System

The 32-bit protected-mode CD-ROM file system (CDFS) implemented in Windows 95 provides improved CD-ROM access performance over the real-mode MSCDEX driver in Windows 3.1 and is a full 32-bit ISO 9660 CD file system. The CDFS driver replaces the 16-bit real-mode MSCDEX driver and features 32-bit protected-mode caching of CD-ROM data. The CDFS driver cache is dynamic and shares the cache memory pool with the 32-bit VFAT driver, requiring no configuration or static allocation on the part of the user.

Benefits of the new 32-bit CDFS driver include the following:

- No conventional memory used (replaces real-mode MSCDEX)
- Improved performance over MS-DOS–based MSCDEX and real-mode cache
- Better multitasking when accessing CD-ROM information, with no blocking
- Dynamic cache support to provide a better balance between providing memory to run applications versus memory to serve as a disk cache

If MSCDEX is specified in the AUTOEXEC.BAT, the 32-bit CDFS driver takes over the role played by the MSCDEX driver and communicates with the CD-ROM device. The use of MSCDEX is no longer necessary under Windows 95.

Users of CD-ROM multimedia applications benefit greatly from the new 32-bit CDFS. Their multimedia applications run smoother and information is read from the CD-ROM quicker, providing improved performance.

# The Block I/O Subsystem

The block I/O subsystem in Windows 95 improves upon the 32-bit disk access fast-disk device architecture in Windows 3.1 and therefore improves performance for the entire file system and a broader array of device support.

As shown in Figure 29, the components of the block I/O subsystem include the high-level I/O Supervisor (IOS) layer, which provides an interface to the block I/O subsystem for the higher layer components; the port driver, which represents a monolithic disk device driver; the SCSI layer, which provides a standard interface and driver layer to provide device-independent control code for SCSI devices; and the SCSI mini-port driver, which contains the device-dependent control code responsible for interacting with individual SCSI controllers.

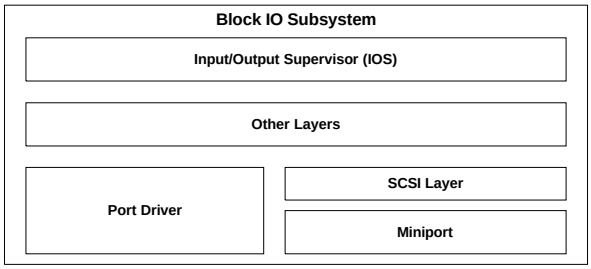


**Figure 29. The architecture of the block I/O subsystem**

The block I/O subsystem provides the following support in Windows 95:

- A fully Plug and Play–enabled architecture
- Support for mini-port drivers that are binary compatible with Windows NT
- Support for Windows 3.1 fast disk drivers for backward compatibility
- Protected-mode drivers that take over real-mode MS-DOS device drivers when safe to do so
- The ability to support existing MS-DOS real-mode disk device drivers for compatibility

The following sections examine the different areas that make up the block I/O subsystem. The explanations are provided to facilitate an understanding of the components, bearing in mind that the configuration of the disk device driver layers is isolated from the user.

## The I/O Supervisor

The I/O Supervisor (IOS) provides services to file systems and drivers. The IOS is responsible for the queuing of file service requests and for routing the requests to the appropriate file system driver. The IOS also provides asynchronous notification of file system events to installed drivers.

## The Port Driver

The port driver is a monolithic 32-bit protected-mode driver that communicates with a specific disk device, such as a hard disk controller. This driver is specifically for use with Windows 95 and resembles the 32-bit disk access (fast disk) driver used in Windows 3.1, such as the WDCTRL driver used for Western Digital compatible hard disk controllers. In Windows 95, the driver that communicates with IDE/ESDI hard disk controllers and floppy disk controllers is implemented as a port driver. A port driver provides the same functionality as the combination of the SCSI manager and the mini-port driver.

## The SCSI Layer

The SCSI layer applies a 32-bit protected-mode universal driver model architecture to communication with SCSI devices. The SCSI layer provides all the high-level functionality that is common to SCSI-like devices and then uses a mini-port driver to handle device-specific I/O calls. The SCSI Manager is part of this system and provides compatibility support for using Windows NT mini-port drivers.

## The Mini-Port Driver

The mini-port driver model used in Windows 95 simplifies the task of writing device drivers for disk device hardware vendors. Because the SCSI Stub provides the high-level functionality for communicating with SCSI devices, disk device hardware vendors need to create only a mini-port driver that is tailored to their own disk device. The mini-port driver for Windows 95 is 32-bit protected-mode code and is binary compatible with Windows NT mini-port drivers, another factor that simplifies the task of writing device drivers. Binary compatibility with NT also results in a more stable and reliable device driver because hardware vendors need to maintain only one code base for device support. Users of Windows 95 also benefit because many mini-port drivers are already available for Windows NT.

## Support for IDE, ESDI, and SCSI Controllers

Through the use of either a port driver or a mini-port driver, support for a broad array of disk devices will be available for Windows 95, including popular IDE, ESDI, and SCSI disk controllers. Users won't have to decide whether to use a port driver or a mini-port driver because the driver is provided by the hardware vendor and configuration of the driver is handled by the Windows 95 system.

## The Real-Mode Mapper

To provide binary compatibility with real-mode MS-DOS–based disk device drivers for which a protected-mode counterpart does not exist in Windows 95, the block I/O subsystem provides a mapping layer to allow the protected-mode file system to communicate with a real-mode driver as if it were a protected-mode component. The layers above and including this real-mode mapper (RMM) are protected-mode code, and the real-mode mapper translates file I/O requests from protected mode to real mode so that the MS-DOS device driver can perform the desired read or write operation from or to the disk device. An example of when the real-mode mapper would come into play is when real-mode disk-compression software is running and a protected-mode disk-compression driver is not available.

# Long Filename Support

The use of long filenames of up to 255 characters in Windows 95 overcomes the sometimes cryptic 8.3 MS-DOS filename convention and allows more user-friendly filenames. MS-DOS 8.3 filenames are maintained and tracked by the system to provide compatibility with existing Win16–based and MS-DOS–based applications that manipulate only 8.3 filenames, but as users migrate to Win32–based applications, the use of 8.3 filename conventions is hidden from the user.

Long filenames are supported by extending the MS-DOS FAT file system and using bits and fields that were previously reserved by the operating system to add special directory entries that maintain long filename information. Extending the MS-DOS FAT layout, rather than creating a new format, allows users to install and use Windows 95 on existing disk formats without having to change their disk structure or reformat their drives. This implementation provides ease of use and allows future growth while maintaining backward compatibility with existing applications.

Because Windows 95 simply extends the FAT structure, long filenames are supported on disks as well as hard disks. If a file on a disk that has a long filename is viewed on a computer that is not running Windows 95, only the 8.3 filename representation is seen.

Figure 30 shows a disk directory with long filenames (shown graphically in Figure 14) and their corresponding 8.3 filename mappings on a computer running Windows 95.

```
 Volume in drive C is MY HARDDISK
 Volume Serial Number is 1E30-830F
 Directory of C:\Long Filename Directory

.              <DIR>        07-11-94 10:02a .
..             <DIR>        07-11-94 10:02a ..
4THQUART XLS            147  05-11-94 12:25a 4th Quarter Analysis.xls
BOSS'SBI TXT            147  05-11-94 12:25a Boss's birthday card.txt
1994FINA DOC            147  05-11-94 10:35a 1994 Financial Projections.doc
FISCALYE       <DIR>        07-11-94 10:02a Fiscal Year Information
COMPANYL BMP            478  03-27-94 12:00a Company Logo.bmp
SHORTC~2 PIF            967  02-16-95  4:55p Shortcut to MS-DOS Application.pif
NEWWAVES WAV              0  06-14-94  1:14p New Wave Sound.wav
NEWVID~1 AVI              0  06-14-94  1:15p New video.avi
DIRECTIO DOC            147  05-11-94 12:25a Directions to company picnic.doc
         8 file(s)          2,033 bytes
         3 dir(s)     134,643,712 bytes free
```

**Figure 30. A directory listed from the command prompt, showing both 8.3 and long filenames**

## Support for Existing Disk Management Utilities

For existing disk management utilities to recognize and preserve long filenames, utility vendors need to revise their software products. Microsoft is working closely with utilities vendors and is documenting long filename support and its implementation as an extension to the FAT format as part of the Windows 95 Software Development Kit (SDK).

Existing MS-DOS–based disk management utilities that manipulate the FAT format, including disk defragmenters, disk bit editors, and some tape backup software, may not recognize long filenames as used by Windows 95 and may destroy long filename entries in the FAT format. However, no data is lost if the long filename entry is destroyed because the corresponding system-defined 8.3 filename is preserved.

# Hidden File Extensions

Like Windows 3.1, Windows 95 uses file extensions to associate a given file type with an application. However, to make it easier to manipulate files, file extensions are hidden from users in the Windows 95 shell and in the Windows Explorer, and instead, icons are used in the UI in Windows 95 to differentiate the documents associated with different applications. Information about file type associations is stored in the Registry, and the associations are used to map a given file with the icon that represents the document type. (For compatibility reasons, Windows 95 must track filename extensions for use with existing MS-DOS and Win16–based applications.)

In addition to hiding filename extensions in the Windows 95 shell and the Windows Explorer, application developers can hide filenames from users in their applications. Mechanisms for hiding filenames are documented in the Windows 95 SDK. A good Windows 95 application makes use of these mechanisms for handling files to be consistent with the rest of the Windows 95 environment.

# Additional File Date/Time Attributes

To further enhance the file system, Windows 95 maintains additional date/time attributes for files that MS-DOS does not track. Windows 95 tracks the date/time when a new file was created, the date/time when a file was modified, and the date when a file was last opened. These file attributes are displayed in the file's property sheet, as shown in Figure 31.

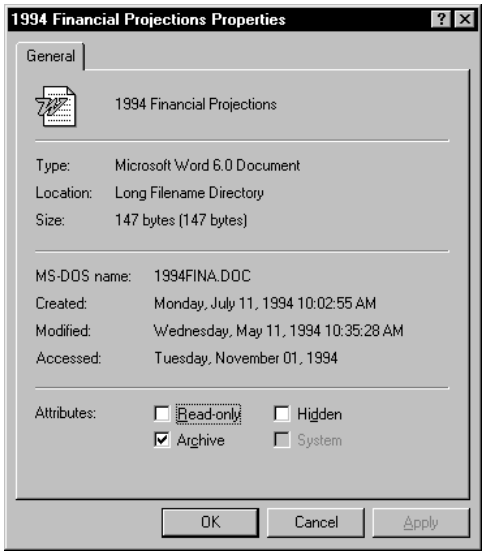


**Figure 31. The properties for a file, showing the new file date/time attributes**

Utilities vendors can take advantage of this additional date/time information to provide enhanced backup utilities—for example, to use a better mechanism when determining whether a given file has changed.

# Coordinated Universal Time Format

MS-DOS has traditionally used the local time of the computer as the time stamp for the directory entry of a file, and continues to use local time for files stored on the local system. However, Windows 95 supports the use of the coordinated universal time (UTC) format for accessing or creating information on network file servers. This format provides the superior, more universal tracking of time information required by networks that operate across time zones.

# Exclusive Access for Disk Management Tools

Disk management utilities, such as disk defragmenters, sector editors, and disk-compression utilities, don't get along well with Windows 3.1. File system programs, such as CHKDSK and DEFRAG, require exclusive access to the file system to minimize the disk access complexities that are present in a multitasking environment where disk I/O occurs. For example, without exclusive access to the disk, data corruption might occur if a user requests that a disk operation move information on the disk at the same time that another task is accessing that information or writing other information to disk. However, Windows 3.1 and MS-DOS do not provide a means of controlling access to the disk, so users have been forced to exit Windows and enter MS-DOS to run disk management utilities.

The file system in Windows 95 has been enhanced to support the use of Windows–based disk management utilities by permitting exclusive access to a disk device. Exclusive disk access is handled as part of the file system through a new API mechanism and can be used by utilities vendors to write Windows–based disk management utilities. Microsoft is encouraging third-party utilities vendors to use this API mechanism to move existing MS-DOS–based utilities to Windows, and is also using it to deliver disk management utilities as part of Windows 95.

For example, this mechanism is used by the Disk Defragmenter (Optimizer) utility delivered as part of Windows 95. Unlike the disk defragment utility used under the combination of MS-DOS and Windows 3.1, the Disk Defragmenter in Windows 95 can be run from the Windows 95 shell and can even be run in the background while users continue to work on their systems.

# DriveSpace Disk Compression

Windows 95 provides built-in support for DriveSpace disk compression. Compatible with DoubleSpace and DriveSpace disk compression provided with MS-DOS, Windows 95 provides base compression in the form of a 32-bit virtual device driver that delivers improved performance over previously available real-mode compression drivers and frees conventional memory for use by MS-DOS–based applications. Users of MS-DOS–based DoubleSpace and DriveSpace don't need to change their existing compressed volume file (CVF) and thus don't need to take any special actions when they install Windows 95.

As shown in Figure 32, the DriveSpace disk compression tool provided with Windows 95 is GUI-based and provides the ability to compress a physical hard drive or removable floppy drive. The Compress a Drive dialog box, shown in Figure 33, graphically depicts the amount of free space available before compression and the estimated space available after compression.

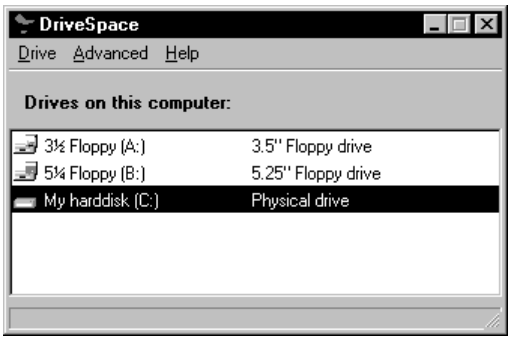


**Figure 32. The DriveSpace disk compression tool**

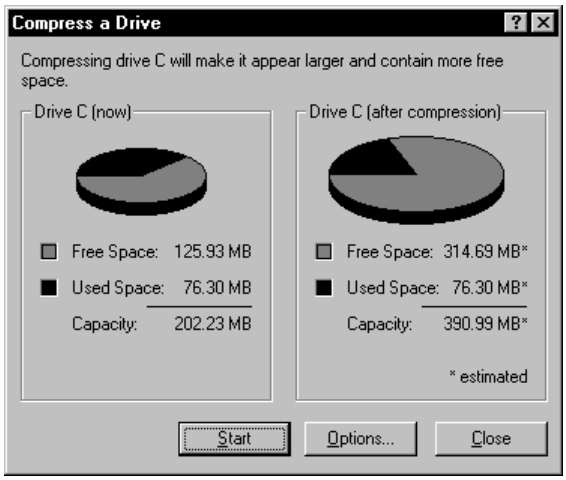


**Figure 33. The Compress a Drive dialog box, which graphically displays free space**

# System Capacity Improvements

Windows 95 provides better system capacity for running MS-DOS and Win16–based applications than Windows 3.1. A number of internal enhancements to the base system prevent internal system resources from being exhausted as quickly as was possible when running multiple Windows–based applications under Windows 3.1.

Many of the artificial limitations present in Windows 3.1 were due to its architecture or internal data structures, which were in turn largely due to the fact that Windows 3.1 had to run on an Intel 80286–based computer. These limitations have for the most part been overcome in Windows 95, to the benefit of users as well as ISVs and other developers.

# System Resource Limitation Improvements

Many users have encountered *Out of Memory* error messages when running multiple Windows–based applications under Windows 3.1, even though the system still reports several megabytes of available free memory. Typically these messages were displayed because the system could not allocate an internal memory resource in a Windows API function call due to lack of available space in a region of memory called a *heap*.

Windows 3.1 maintains heaps for system components called GDI and USER. Each heap is 64 KB in size and is used for storing GDI or memory *object* information allocated when an application calls a Windows API function. The amount of space available in the combination of these two heaps is identified as the percentage of system resources that are free and is displayed in the About dialog box in Program Manager and other Windows applications, as shown in Figure 34.

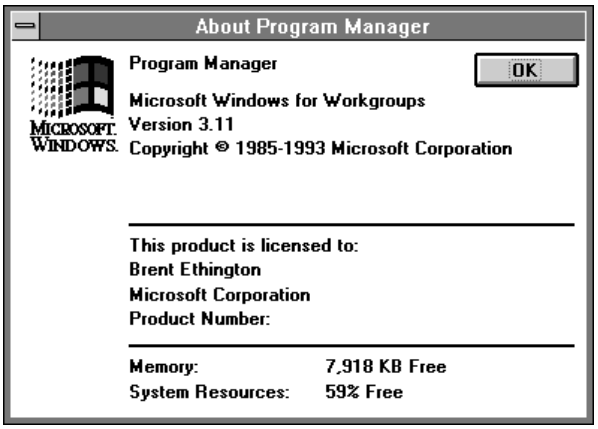


**Figure 34. The About dialog box in Program Manager in Windows 3.1, showing free system resources**

The percentage of free system resources displayed in the About dialog box is calculated using an internal algorithm to represent the aggregate percentage of free memory in the GDI and USER heaps. When the free system resources percentage gets too low, users commonly see an *Out of Memory* error message, even though the amount of free memory shown in the About dialog box is still quite high. This error can result from low memory in either the GDI or USER heap (or both).

To help reduce the system resource limitation, a number of the data structures stored in the 16-bit GDI and USER heaps in Windows 3.1 have been moved out of these heaps and stored in 32-bit heaps, providing more room for the remaining data elements to be created. As a result, system resources decrease less rapidly in Windows 95 than they did in Windows 3.1.

For compatibility, not all objects were removed from the 16-bit GDI or USER heap and placed in a 32-bit heap. For example, some Windows–based applications manipulate the contents of the GDI heap directly, bypassing the published API mechanisms for doing so, because their developers think direct manipulation increases performance. However, because these applications bypass the Windows API mechanisms, moving their data from the existing heap structures and placing them in 32-bit heaps would cause these applications to fail because of memory access violations.

Win16–based and Win32–based applications use the same GDI and USER heaps. The impact of removing selected items from the heaps was closely examined and objects were selected based on the biggest improvement that could be achieved while affecting the fewest number of applications. For example, the GDI heap can quickly become full because of the creation of memory-intensive region objects that are used by applications for creating complex images and by the printing subsystem for generating complex output. Region objects were removed from the 64 KB 16-bit GDI heap and placed in a 32-bit heap, benefiting graphic-intensive applications and providing for the creation of more smaller objects by the system. Windows 95 improves the system capacity for the USER heap by moving menu and window handles to the 32-bit USER heap. Instead of the total limit of 200 for these data structures in Windows 3.1, Windows 95 allows 32,767 menu handles and an additional 32,767 window handles *per process* rather than system-wide.

In addition to moving information from the GDI and USER heaps, robustness improvements in Windows 95 that facilitate system cleanup of unfreed resources also relieve system resource limitations. When Windows 95 determines that the owner and

other ended processes no longer need the resources in memory, Windows 95 cleans up and deallocates leftover data structures. The robustness improvements in Windows 95 are discussed in Chapter 5, “Robustness.”

# Better Memory Management

Windows 95 improves addressibility to provide better access to physical memory, as well as improves upon the swapfile implementation provided in Windows 3.1 to support virtual memory supplementation of physical memory.

## Linear Memory Addressing for Win32–Based Applications

To support a 16-bit operating environment, the Intel processor architecture uses a mechanism, called *segments*, to reference memory by using a 16-bit segment address and a 16-bit offset address within the segment. A segment is 64 KB in size, and applications and the operating system pay a performance penalty when they access information across segments. For 32-bit operating system functionality and Win32–based applications, Windows 95 addresses this issue by using the 32-bit capabilities of the Intel 80386 (and above) processor architecture to support a flat, linear memory mode. A linear addressing model simplifies the development process for application developers, removes the performance penalties imposed by the segmented memory architecture, and provides access to a virtual address space that permits the addressing of up to 4 GB (4 gigabytes, or 4 billion bytes) of memory. Windows 95 uses the flat memory model internally for 32-bit components and virtual device drivers.

## Compatibility with the Windows NT Memory Model

Windows 95 uses the same memory model architecture as Windows NT, providing high-end operating system functionality for the mainstream system. Windows 95 allows full use of the 4 GB of addressable memory space to support even the largest desktop application. The operating system provides a 2 GB memory range for applications and reserves a 2 GB range for itself.

## Virtual Memory Support (Swapfile) Improvements

Windows 95 addresses problems and limitations imposed in Windows 3.1 by its virtual memory swapfile implementation. With Windows 3.1, users were faced with a myriad of choices and configuration options for setting up a swapfile to support virtual memory. They had to decide whether to use a temporary swapfile or a permanent swapfile, how much memory to allocate to the swapfile, and whether to use 32-bit disk access to access the swapfile. A temporary swapfile did not need to be contiguous, and Windows would dynamically allocate hard disk space when it was started and free up the space when it was terminated. A permanent swapfile provided the best performance, but it had to be contiguous, had to be set up on a physical hard disk, and was statically specified by the user and not freed up when the user exited Windows.

The swapfile implementation in Windows 95 simplifies the configuration task for the user and, because of improved virtual memory algorithms and access methods, combines the best of a temporary swapfile and a permanent swapfile. The swapfile in Windows 95 is

dynamic and can shrink or grow based on the operations performed on the system. The swapfile can occupy a fragmented region of the hard disk and it can be located on a compressed disk volume.

Windows 95 uses intelligent system defaults for the configuration of virtual memory, relieving the user of the task of changing virtual memory settings. Figure 35 shows the simplified virtual memory configuration settings.

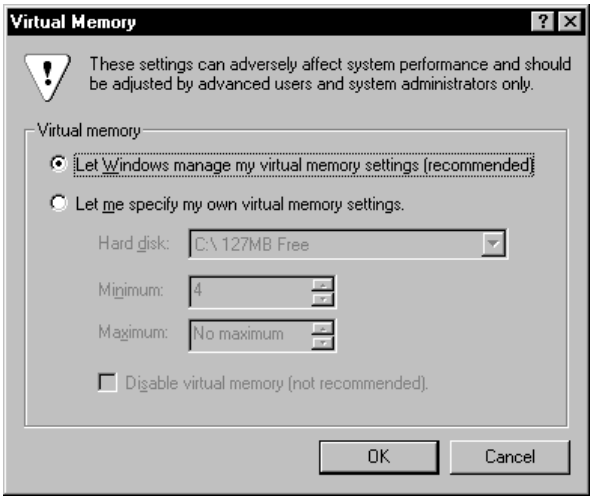


**Figure 35. The simplified virtual memory settings**

# The Registry: A Centralized Configuration Store

Windows 95 uses a mechanism called the *Registry* to serve as the central configuration store for user, application, and computer-specific information. The Registry solves problems associated with the .INI files used in Windows 3.1 and is a hierarchical database that stores system-wide information in a single location, making it easy to manage and support.

## Solutions to .INI Problems

Windows 3.1 uses initialization (.INI) files to store system-specific or application-specific information about the state or configuration of the system. For example, the WIN.INI file stores state information about the appearance or customization of the Windows environment; the SYSTEM.INI file stores system-specific information on the hardware and device-driver configuration of the system; and various .INI files, such as WINFILE.INI, MSMAIL.INI, CLOCK.INI, CONTROL.INI, and PROGMAN.INI, store application-specific information about the default state of an application.

Problems with .INI files under Windows 3.1 for configuration management include the following:

- Information is stored in several different locations, including CONFIG.SYS, AUTOEXEC.BAT, WIN.INI, SYSTEM.INI, PROTOCOL.INI, private .INI files, and private .GRP files.

- .INI files are text-based and limited in total size to 64KB, and APIs allow for get/write operations only.
- Information stored in .INI files is non-hierarchical and supports only two levels of information: key names broken up by section headings.
- Many .INI files contain a myriad of switches and entries that are complicated to configure or are used only by operating system components.
- .INI files provide no mechanism for storing user-specific information, thus making it difficult for multiple users to share a single computer.
- Configuration information in .INI files is local to each system, and because no API mechanisms are available for remotely managing configuration, managing multiple systems is difficult.

To solve these problems, the Registry was designed with the following goals in mind:

- Simplify the support burden.
- Centralize configuration information.
- Provide a means to store user, application, and computer-specific information.
- Provide local and remote access to configuration information.

The Registry is structured as a database of keys in which each key can contain a value or other keys (subkeys). As shown in Figure 36, the Registry uses a hierarchical structure to store text or binary value information and maintains all of the configuration parameters normally stored in the Windows system .INI files such as WIN.INI, SYSTEM.INI, and PROTOCOL.INI. Although similar in some ways to the Registration Database used in Windows 3.1, which served as a central repository for file associations and OLE registration information, the Registry in Windows 95 extends the Registration Database structure to support keys that can have more than one value and also support data of different types.

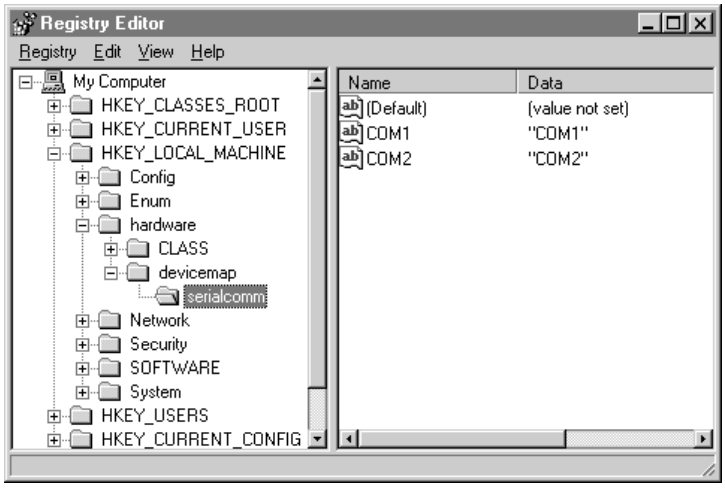


**Figure 36. The hierarchy of the Registry, as displayed by the Registry Editor**

The Registry is made up of several .DAT files that contain system-specific information (SYSTEM.DAT) or user-specific information (USER.DAT). System-specific information, such as the static reference to loading virtual device drivers, is moved as appropriate from the SYSTEM.INI file to the Registry.

### System Switch Simplification

Another improvement in Windows 95 over the Windows 3.1 use of .INI files is related to system switch simplification. Windows 3.1 supports several hundred different configuration switches that can be specified in system .INI files, including WIN.INI and SYSTEM.INI. With intelligent enhancements made to the system and better dynamic configuration properties, Windows 95 has reduced the number of entries normally associated with .INI files. These reductions didn't result from simply moving .INI entries to the Registry but by examining and justifying the presence of each and every one.

## No .INI Files?

Like CONFIG.SYS and AUTOEXEC.BAT, WIN.INI and SYSTEM.INI and application-specific .INI files still exist for compatibility reasons. The Win16 APIs for manipulating .INI files still manipulate .INI files, but developers of Win32–based applications are encouraged to use the Registry APIs to consolidate application-specific information.

Many existing Win16–based applications expect to find and manipulate the WIN.INI and SYSTEM.INI files to add entries or load unique device drivers, so Windows 95 examines .INI files during the boot process. For example, the [386Enh] section of SYSTEM.INI is checked for virtual device drivers during start up.

## Role in Plug and Play

One of the primary roles of the Registry in Windows 95 is to serve as a central repository for hardware-specific information for use by the Plug and Play system components. Windows 95 maintains information about hardware components and devices that have been identified through an enumeration process in the hierarchical structure of the Registry. When new devices are installed, the system checks the existing configuration in the Registry to determine which hardware resources—for example, IRQs, I/O addresses, DMA channels, and so on—are not being used, so that the new device can be properly configured without conflicting with a device already installed in the system.

## Remote Access to Registry Information

Another advantage of the Registry for Win32–based applications is that many of the Win32 Registry APIs use the remote procedure call (RPC) mechanism in Windows 95 to provide remote access to Registry information across a network. As a result, desktop management applications can be written to aid in the management and support of Windows–based computers, and the contents of the Registry on a given PC can be queried over a network. Industry management mechanisms, such as SNMP or DMI, can easily be integrated into Windows 95, simplifying the management and support burden of an MIS organization. For more information about manageability and remote administration, see Chapter 9, "Networking."

# Better Font Support

Font support in Windows 95 has been enhanced to provide better integration with the UI and has been optimized for the 32-bit environment.

# The 32-Bit TrueType Rasterizer

The rasterizer component for rendering and generating TrueType fonts is enhanced in Windows 95. The rasterizer is written as a 32-bit component, and delivers better fidelity from the mathematical representation to the generated bitmap, as well as better performance for rendering TrueType fonts.

In addition to performance enhancements, the new 32-bit rasterizer provides support for generating complicated glyphs—for example, Han—and results in a faster initial boot time in Windows 95 than in Windows 3.1 when many fonts are installed.