

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.

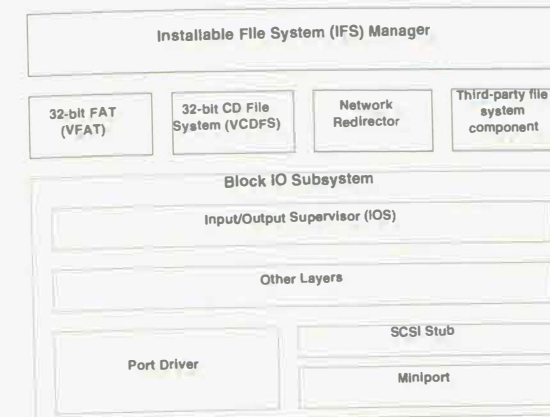


Figure 28. The file system architecture

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.

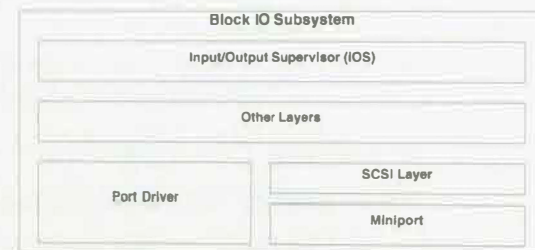


Figure 29. The architecture of the block I/O subsystem

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 layer 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 convention 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 diskettes as well as hard disks. If a file on a diskette 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 and their corresponding 8.3 filename mappings on a computer running Windows 95.

```
Volume in drive C is MY HARDDISK
Volume Serial Number is 1B47-7161
Directory of C:\LONGFILE

. <DIR> 05-11-94 10:34a .
.. <DIR> 05-11-94 10:34a ..
4THQUART XLS 147 05-11-94 12:25a 4th Quarter Analysis.xls
TEXTFILE TXT 147 05-11-94 12:25a TEXTFILE.TXT
```

(continued)

Figure 30. (continued)

```
THISISMY DOC 147 05-11-94 12:25a this is my long filename.doc
1994FINA DOC 147 05-11-94 10:35a 1994 Financial Projections.doc
4 file(s) 588 bytes
2 dir(s) 48,009,216 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 on the following page.

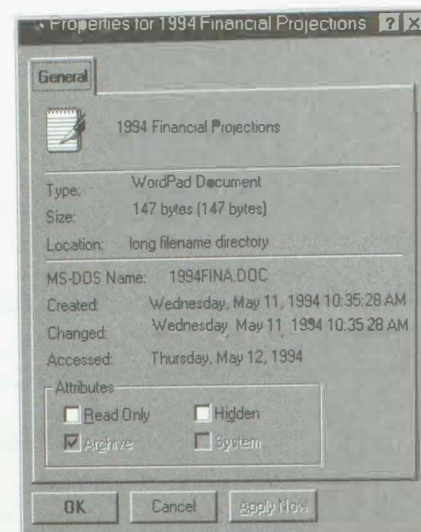


Figure 31. The properties for a file, showing the new file date/time attributes

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 the 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 on the following page, graphically depicts the amount of free space available before compression and the estimated space available after compression.

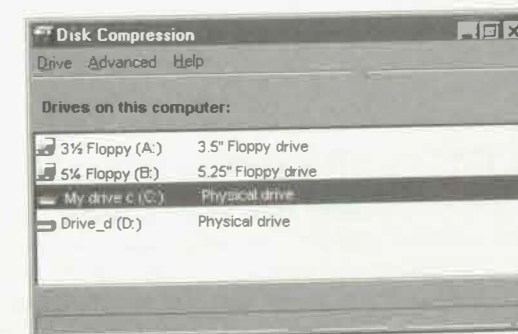


Figure 32. The DriveSpace disk compression tool

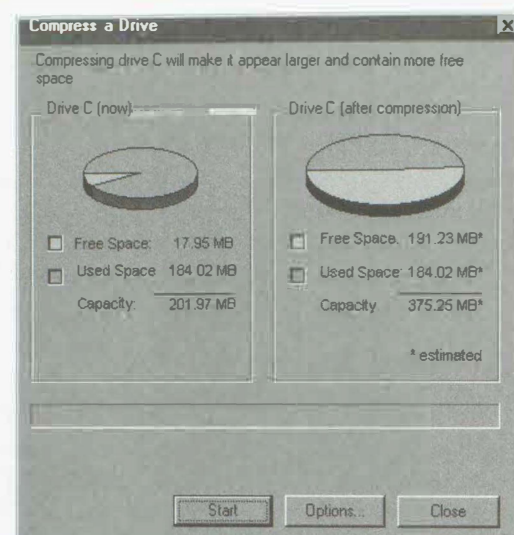


Figure 33. The Compress a Drive dialog box, which graphically displays free space

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 heap^s 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.

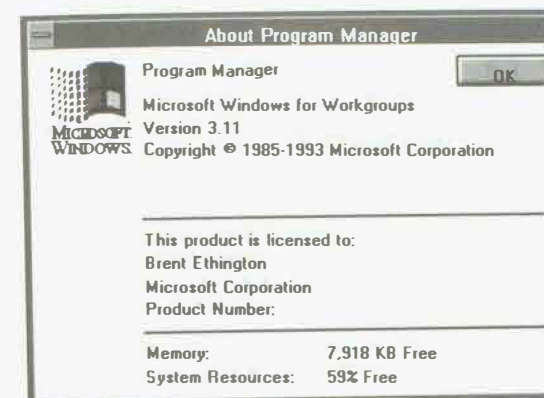


Figure 34. The About dialog box in Program Manager in Windows 3.1, showing free system resources

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 resource^s 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 the^e heaps and stored in 32-bit heap^s, providing more room for the remaining data element^s to be created. As a result, system resource^s decrease less rapidly in Windows 95 than they did in Windows 3.1.

For compatibility, not all object^s 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 publi^hed API mechanisms for doing so, because their developers think direct manipulation increase^s performance. However, because these applications bypass the Windows API mechanisms, moving their data from the existing heap^s ructures and placing them in 32-bit heap^s would cau^e these applications to fail because of memory acces^s violations.

Win16-based and Win32-based applications u^e the^e same GDI and USER heaps. The impact of removing selected items from the heaps wa^s clo^ely examined and objects were selected ba^{ed} 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 addressability 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.

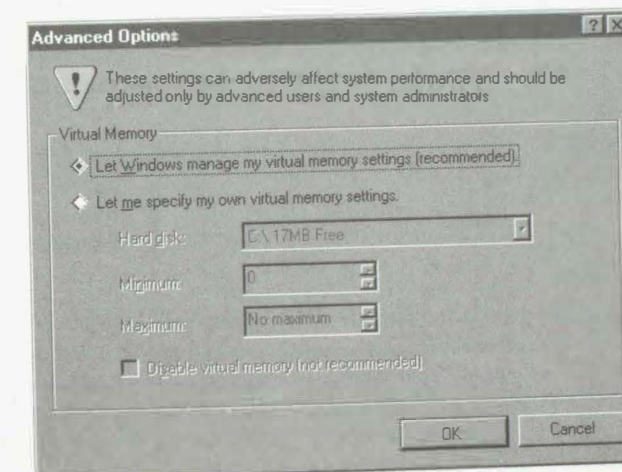


Figure 35. The simplified virtual memory settings

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.

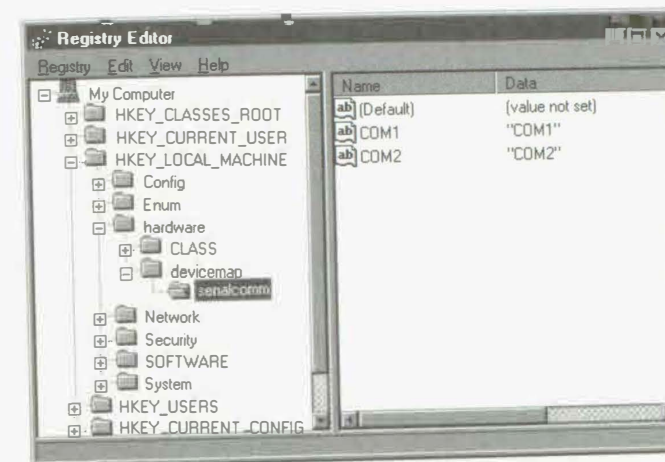


Figure 36. The hierarchy of the Registry, as displayed by the Registry Editor

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.

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. It also provides capabilities such as font smoothing for fonts that have not previously been offered as part of a mainstream operating system.

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.

CHAPTER 5

Robustness

Windows 95 improves on the robustness of Windows 3.1 to provide great support for running MS-DOS-based, Win16-based, and Win32-based applications, and to provide a high level of system protection from errant applications.

Windows 3.1 provided a number of mechanisms to support a more robust and stable environment over Windows 3.0, including the following:

- **Better resource clean-up.** When an MS-DOS-based or Windows-based application crashed, users could continue running in a way that allowed them to save their work.
- **Local reboot.** Users could shut down an application that hung.
- **Parameter validation for API calls.** The system could catch many common application errors and fail the API call, rather than allow bad data to be passed to an API.

Just as the improvements in Windows 3.1 provided a more robust and stable environment than Windows 3.0, the improvements in Windows 95 provide an even better environment.

System-Wide Robustness Improvements

System-wide improvements resulting in a more robust operating system environment than Windows 3.1 include:

- Better local reboot
- Virtual device driver thread clean-up when a process ends
- Per-thread state tracking
- Virtual device driver parameter validation

Better Local Reboot

The capability whereby users can end an application or VM that hangs is called a *local reboot*. With Windows 3.1, users could perform a local reboot by pressing the three-key CTRL+ALT+DEL combination. Users could pretty easily end errant VMs with a local reboot request, but a local reboot request for a Windows-based application often didn't end the errant Windows-based process or brought the entire system down.

Windows 95 greatly improves the local reboot support by providing a means to end an MS-DOS-based application running in a VM, a Win16-based application, or a Win32-based application without bringing down the entire system. Moreover, the process of cleaning up the system after a local reboot is now more complete than for Windows 3.1. (This process is described later in this chapter.)

In Windows 3.1, when a user requests a local reboot, the system may identify the active application as the application that has the focus of the local reboot request, or it may report back that there is no application in a hung or inactive state. In Windows 95, the system displays the Close Program dialog box, which identifies the tasks that are running and the state that the system perceives each one to be in, as shown in Figure 37. This level of detail affords the user much more flexibility and control over the local reboot than with Windows 3.1.

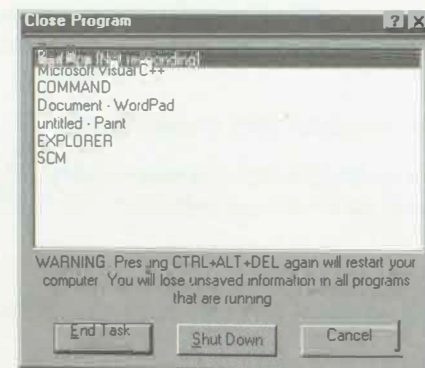


Figure 37. The Close Program dialog box

Applications are identified as “not responding” when they haven’t checked the message queue for a period of time. Although some applications don’t check the message queue while performing computationally intensive operations, well-behaved applications check the message queue frequently. In Windows 95, as in Windows 3.1, Win16-based applications must check the message queue to relinquish control to other running tasks.

Try It!

Perform a Local Reboot

1. With a couple of applications running, press CTRL+ALT+DEL. You are presented with a list of active applications. Applications that are hung are identified as *Not responding*.
2. Terminate one of the tasks by clicking *End Task*.

Virtual Device Driver Thread Clean-Up

Local reboot support is also aided by improved VxD thread clean-up when a given process ends. With Windows 3.1, the system often couldn’t recover either if it was running real-mode code, such as BIOS routines, when an application ended abnormally, or if the user requested a local reboot to end a seemingly-hung application. For example, if an operation (such as a network operation in real-mode, a disk I/O, or an asynchronous application request) ended abnormally because of another application-based error, Windows 3.1 sometimes couldn’t clean up properly to free allocated resources and sometimes couldn’t even return control to the user.

Windows 95 improves system clean-up by providing each system VxD with the ability to track the resources it allocates on a per-thread basis. Because most computer system functionality and support is handled in Windows 95 by VxDs rather than by real-mode code or BIOS routines, Windows 95 can recover from errors or situations that, under Windows 3.1, would require that the computer be rebooted.

When Windows 95 ends a given thread (because the user exited the application, a local reboot was requested, or the application ended abnormally), each VxD receives notification that the thread is ended. This notification allows the VxD to safely cancel any operations it is waiting to finish and frees any resources that the VxD previously allocated for the thread or application. Because the system tracks each VM, Win16-based application, and Win32 thread as a separate per-thread instance, the system can clean up properly at each of these levels, without affecting the integrity of the system.

Per-Thread State Tracking

To aid in system clean-up, resource tracking in Windows 95 is much better than in Windows 3.1. In addition to tracking resources on a per-thread basis by system VxDs, resources such as memory blocks, memory handles, graphics objects, and other system items are allocated and also tracked by system components on a per-thread basis. Tracking these resources on a per-thread basis allows the system to clean up safely when a given thread ends, either normally—at the user’s request—or abnormally. Resources are identified and tracked by both a thread ID and the major Windows version number that is stored in the .EXE header of the application.