11

CASE STUDY 2: WINDOWS 2000

Item	Windows 95/98	Windows NT
Full 32-bit system?	No	Yes
Security?	No	Yes
Protected file mappings?	No	Yes
Private addr space for each MS-DOS prog?	No	Yes
Unicode?	No	Yes
Runs on	Intel 80x86	80x86, Alpha, MIPS, ...
Multiprocessor support?	No	Yes
Re-entrant code inside OS?	No	Yes
Plug and play?	Yes	No
Power management?	Yes	No
FAT-32 file system?	Yes	Optional
NTFS file system	No	Yes
Win32 API?	Yes	Yes
Run all old MS-DOS programs?	Yes	No
Some critical OS data writable by user?	Yes	No

Fig. 11-1. Some differences between Windows 98 and Windows NT.

Version	Max RAM	CPUs	Max clients	Cluster size	Optimized for
Professional	4 GB	2	10	0	Response time
Server	4 GB	4	Unlimited	0	Throughput
Advanced server	8 GB	8	Unlimited	2	Throughput
Datacenter server	64 GB	32	Unlimited	4	Throughput

Fig. 11-2. The different versions of Windows 2000.

Year	AT&T	BSD	MINIX	Linux	Solaris	Win NT
1976	V6 9K					
1979	V7 21K					
1980		4.1 38K				
1982	Sys III 58K					
1984		4.2 98K				
1986		4.3 179K				
1987	SVR3 92K		1.0 13K			
1989	SVR4 280K					
1991				0.01 10K		
1993		Free 1.0 235K			5.3 850K	3.1 6M
1994		4.4 Lite 743K		1.0 165K		3.5 10M
1996				2.0 470K		4.0 16M
1997			2.0 62K		5.6 1.4M	
1999				2.2 1M		
2000		Free 4.0 1.4M			5.8 2.0M	2000 29M

Fig. 11-3. A comparison of some operating system sizes. The first string in each box is the version; the second is the size measured in lines of source code, where K = 1000 and M = 1,000,000. Comparisons within a column have real meaning; comparisons across columns do not, as discussed in the text.

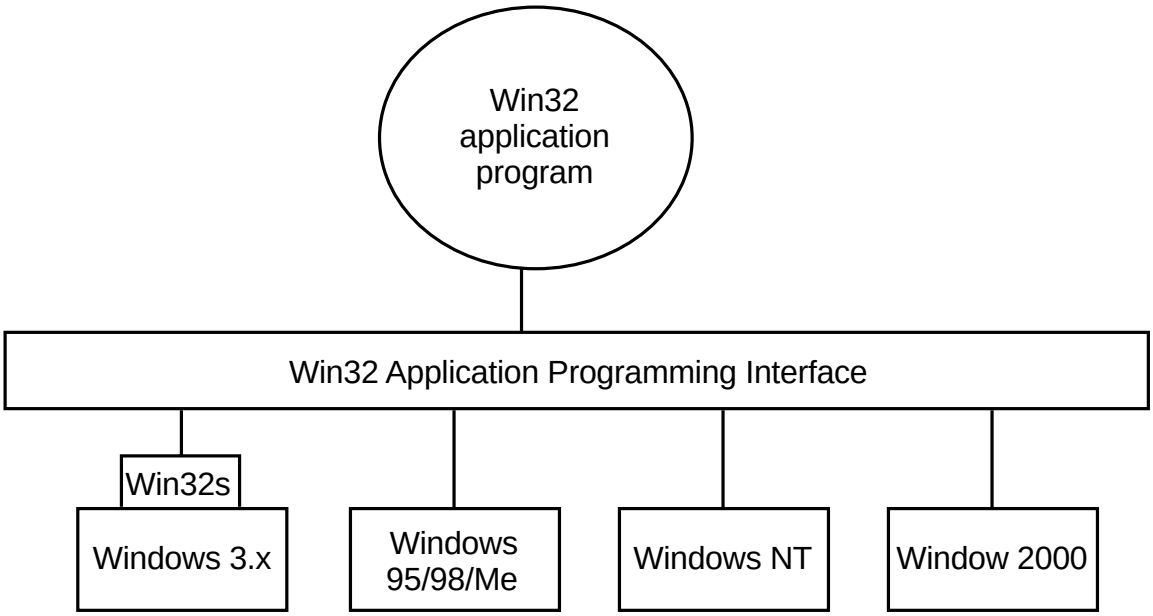


Fig. 11-4. The Win32 API allows programs to run on almost all versions of Windows.

Key	Description
HKEY_LOCAL_MACHINE	Properties of the hardware and software
HARDWARE	Hardware description and mapping of hardware to drivers
SAM	Security and account information for users
SECURITY	System-wide security policies
SOFTWARE	Generic information about installed application programs
SYSTEM	Information for booting the system
HKEY_USERS	Information about the users; one subkey per user
USER-AST-ID	User AST's profile
AppEvents	Which sound to make when (incoming email/fax, error, etc.)
Console	Command prompt settings (colors, fonts, history, etc.)
Control Panel	Desktop appearance, screensaver, mouse sensitivity, etc.
Environment	Environment variables
Keyboard Layout	Which keyboard: 102-key US, AZERTY, Dvorak, etc.
Printers	Information about installed printers
Software	User preferences for Microsoft and third party software
HKEY_PERFORMANCE_DATA	Hundreds of counters monitoring system performance
HKEY_CLASSES_ROOT	Link to HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES
HKEY_CURRENT_CONFIG	Link to the current hardware profile
HKEY_CURRENT_USER	Link to the current user profile

Fig. 11-5. The root keys registry keys and selected subkeys. The capitalization has no meaning but follows the Microsoft practice here.

Win32 API function	Description
RegCreateKeyEx	Create a new registry key
RegDeleteKey	Delete a registry key
RegOpenKeyEx	Open a key to get a handle to it
RegEnumKeyEx	Enumerate the subkeys subordinate to the key of the handle
RegQueryValueEx	Look up the data for a value within a key

Fig. 11-6. Some of the Win32 API calls for using the registry

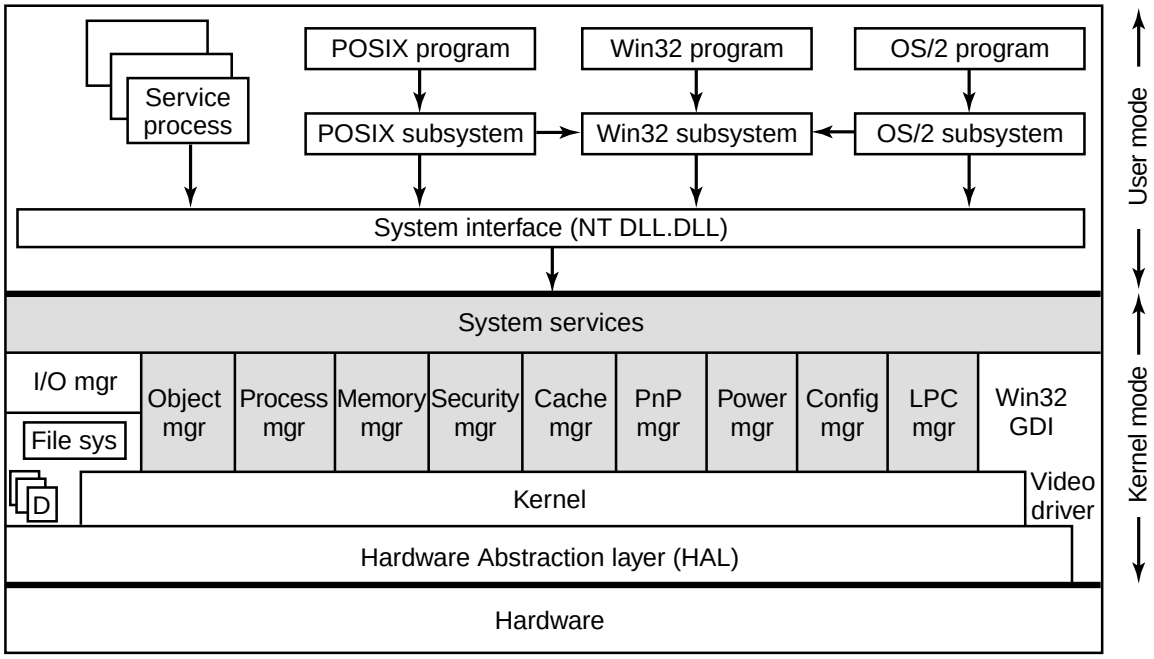


Fig. 11-7. The structure of Windows 2000 (slightly simplified). The shaded area is the executive. The boxes indicated by D are device drivers. The service processes are system daemons.

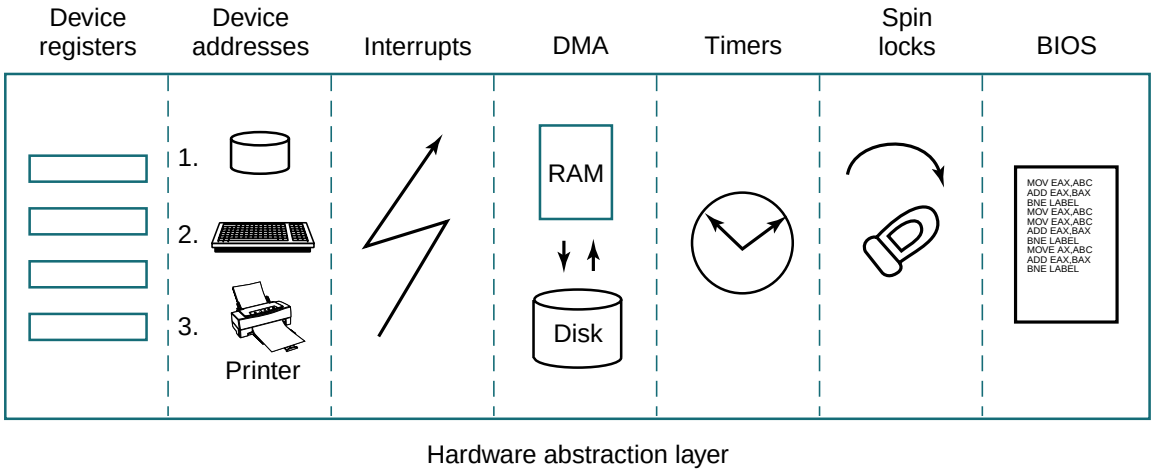


Fig. 11-8. Some of the hardware functions the HAL manages.

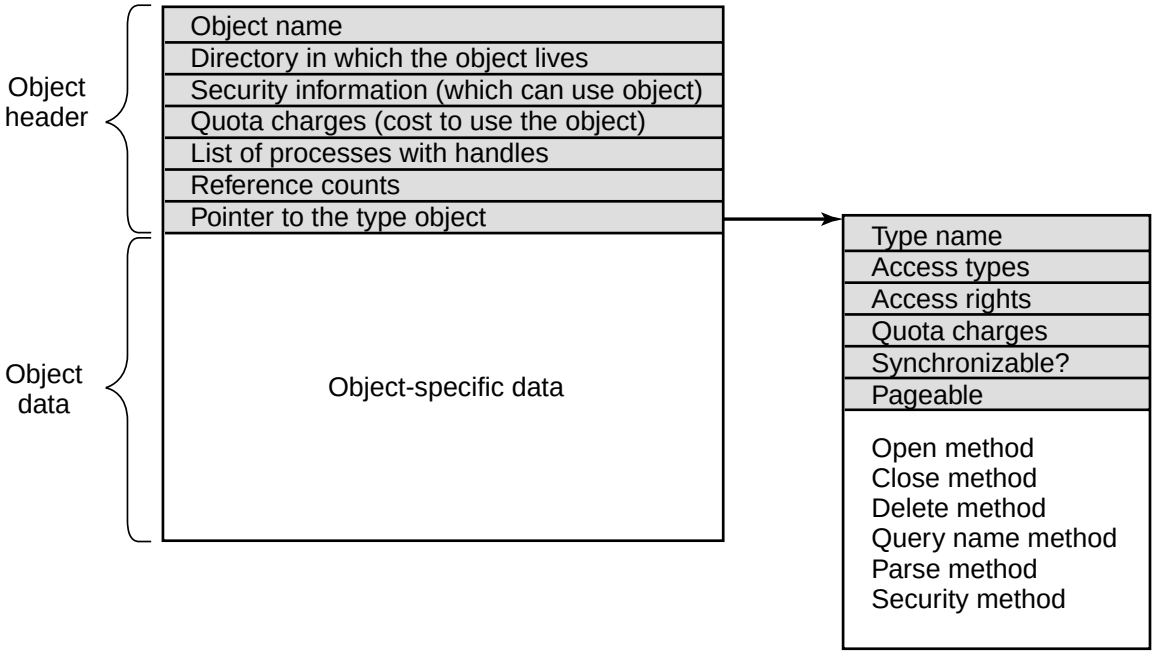


Fig. 11-9. The structure of an object.

Type	Description
Process	User process
Thread	Thread within a process
Semaphore	Counting semaphore used for interprocess synchronization
Mutex	Binary semaphore used to enter a critical region
Event	Synchronization object with persistent state (signaled/not)
Port	Mechanism for interprocess message passing
Timer	Object allowing a thread to sleep for a fixed time interval
Queue	Object used for completion notification on asynchronous I/O
Open file	Object associated with an open file
Access token	Security descriptor for some object
Profile	Data structure used for profiling CPU usage
Section	Structure used for mapping files onto virtual address space
Key	Registry key
Object directory	Directory for grouping objects within the object manager
Symbolic link	Pointer to another object by name
Device	I/O device object
Device driver	Each loaded device driver has its own object

Fig. 11-10. Some common executive object types managed by object manager.

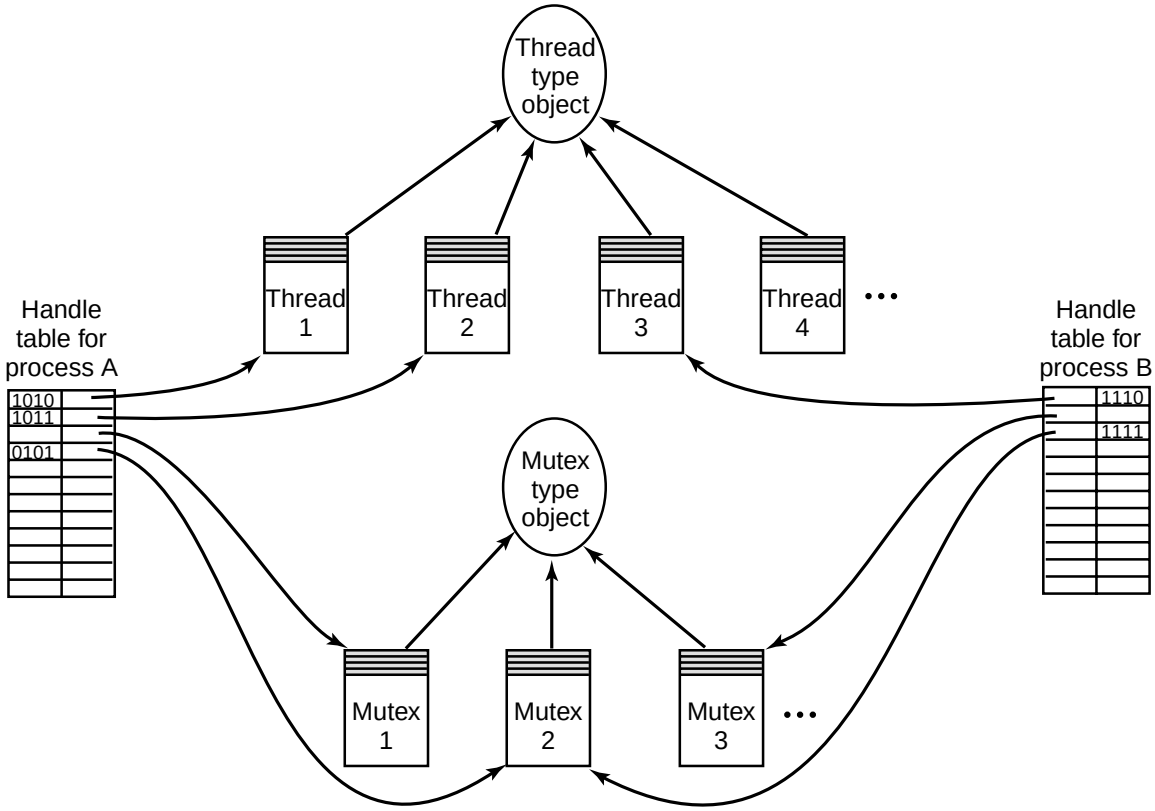


Fig. 11-11. The relationship between handle tables, objects, and type objects.

Directory	Contents
??	Starting place for looking up MS-DOS devices like C:
Device	All discovered I/O devices
Driver	Objects corresponding to each loaded device driver
ObjectTypes	The type objects shown in Fig. 11-11
Windows	Objects for sending messages to all the windows
BaseNamedObjs	User-created objects such as semaphores, mutexes, etc.
Arcname	Partition names discovered by the boot loader
NLS	National language support objects
FileSystem	File system driver objects and file system recognizer objects
Security	Objects belonging to the security system
KnownDLLs	Key shared libraries that are opened early and held open

Fig. 11-12. Some typical directories in the object name space.

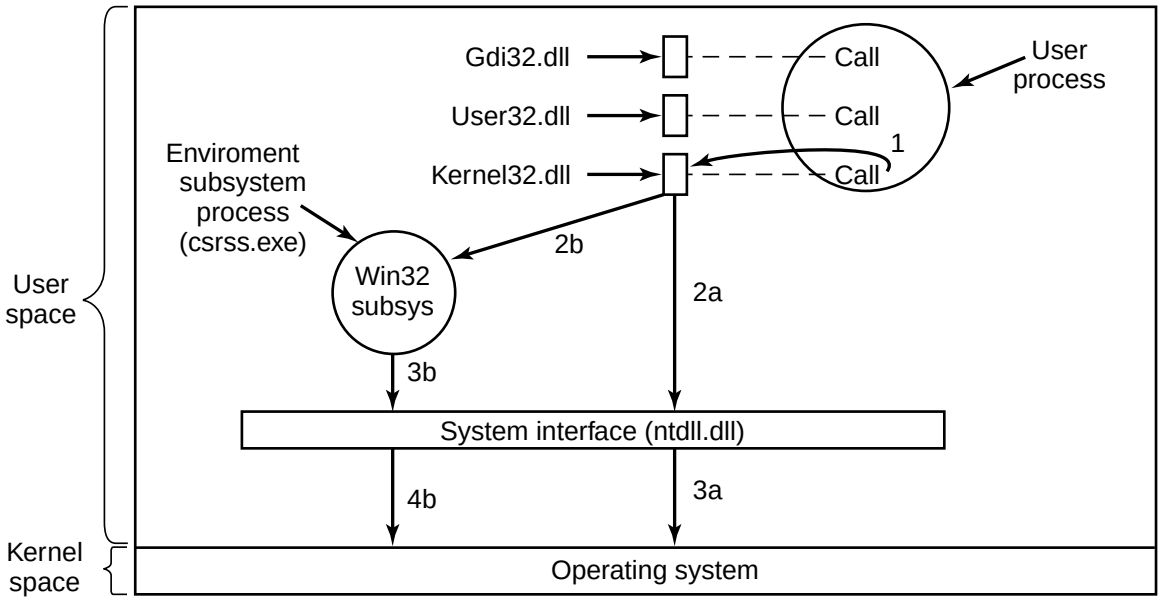


Fig. 11-13. Various routes taken to implement Win32 API function calls.

File	Mode	Fcns	Contents
hal.dll	Kernel	95	Low-level hardware management, e.g., port I/O
ntoskrnl.exe	Kernel	1209	Windows 2000 operating system (kernel + executive)
win32k.sys	Kernel	-	Many system calls including most of the graphics
ntdll.dll	User	1179	Dispatcher from user mode to kernel mode
csrss.exe	User	0	Win32 environment subsystem process
kernel32.dll	User	823	Most of the core (nongraphics) system calls
gdi32.dll	User	543	Font, text, color, brush, pen, bitmap, palette, drawing, etc. calls
user32.dll	User	695	Window, icon, menu, cursor, dialog, clipboard, etc. calls
advapi32.dll	User	557	Security, cryptography, registry, management calls

Fig. 11-14. Some key Windows 2000 files, the mode they run in, the number of exported function calls, and the main contents of each file. The calls in *win32k.sys* are not formally exported since *win32k.sys* is not called directly.

Name	Description
Job	Collection of processes that share quotas and limits
Process	Container for holding resources
Thread	Entity scheduled by the kernel
Fiber	Lightweight thread managed entirely in user space

Fig. 11-15. Basic concepts used for CPU and resource management.

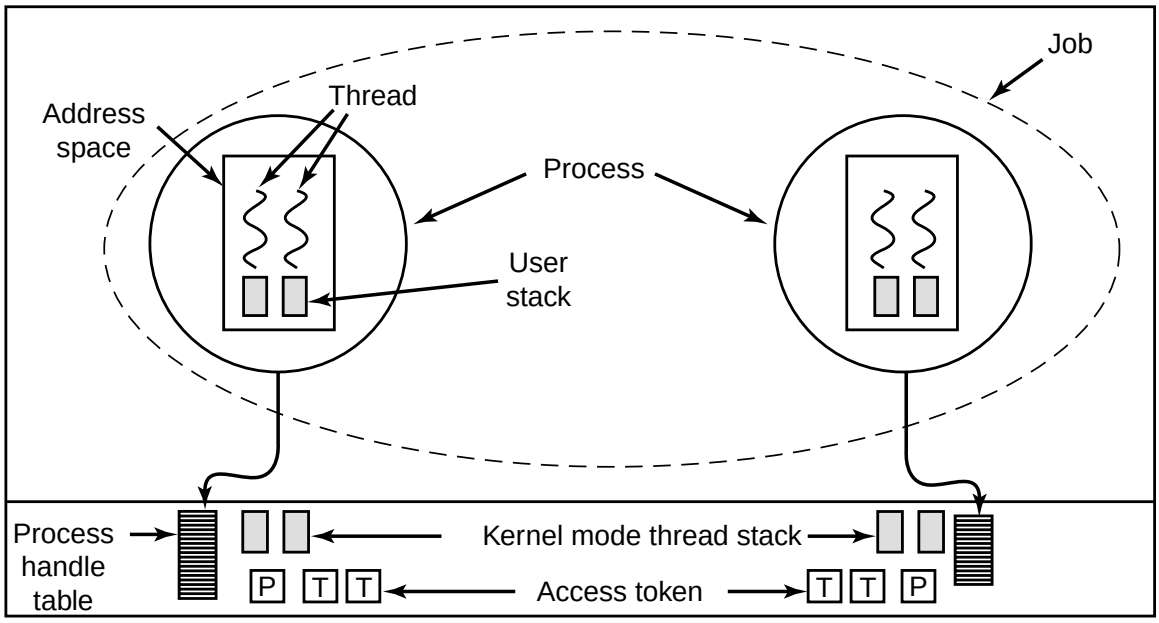


Fig. 11-16. The relationship between jobs, processes, and threads. Several fibers can also be multiplexed on one thread (not shown).

Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semapahore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Fig. 11-17. Some of the Win32 calls for managing processes, threads, and fibers.

		Win32 process class priorities					
		Realtime	**High**	**Above Normal**	**Normal**	**Below Normal**	**Idle**
Win32 thread priorities	**Time critical**	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Fig. 11-18. Mapping of Win32 priorities to Windows 2000 priorities.

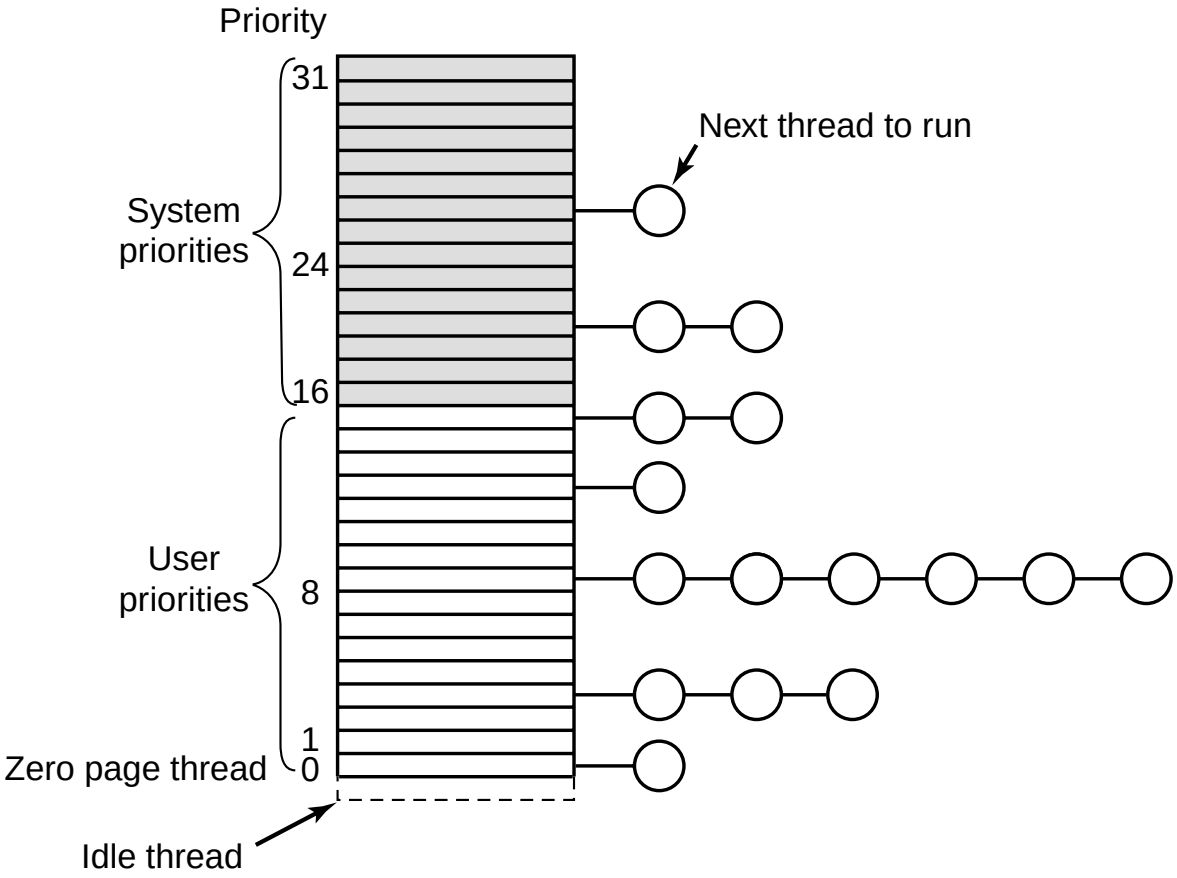


Fig. 11-19. Windows 2000 supports 32 priorities for threads.

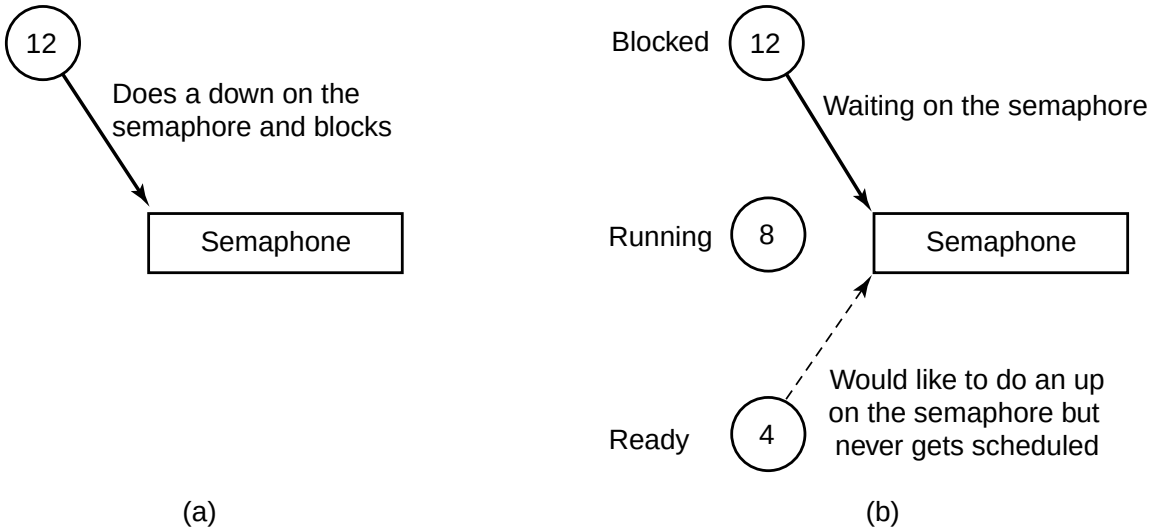


Fig. 11-20. An example of priority inversion.

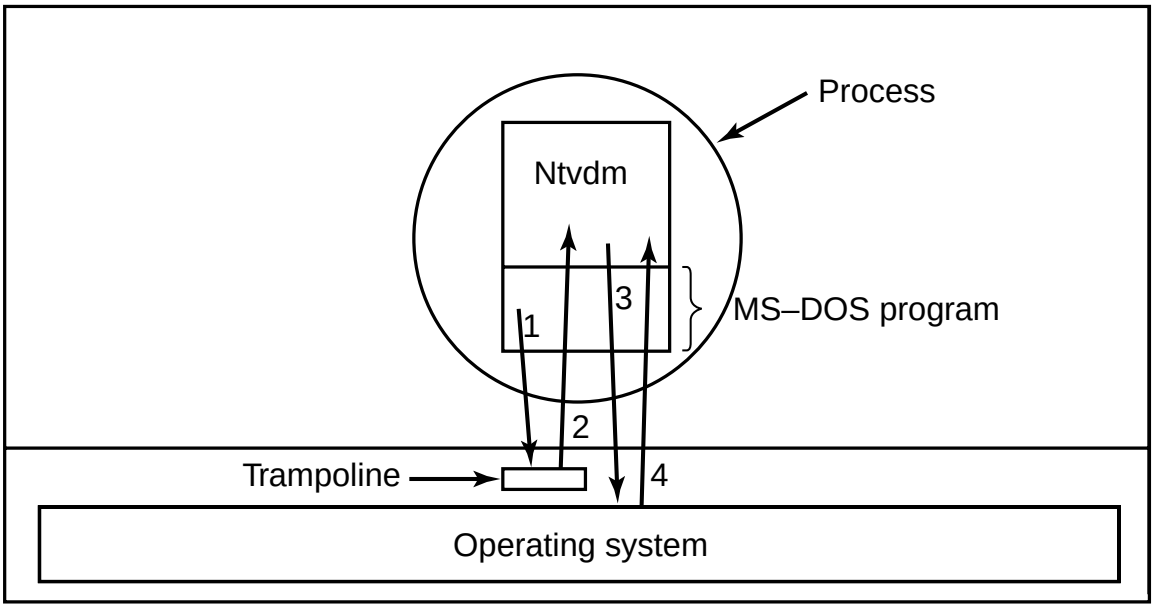


Fig. 11-21. How old MS-DOS programs are run under Windows 2000.

Process	Description
idle	Not really a process, but home to the idle thread
system	Creates smss.exe & paging files; reads registry; opens DLLs
smss.exe	First real proc; much initialization; creates csrss & winlogon
csrss.exe	Win32 subsystem process
winlogon.exe	Login daemon
lsass.exe	Authentication manager
services.exe	Looks in registry and starts services
Printer server	Allows remote jobs to use the printer
File server	Serves requests for local files
Telnet daemon	Allows remote logins
Incoming email handler	Accepts and stores inbound email
Incoming fax handler	Accepts and prints inbound faxes
DNS resolver	Internet domain name system server
Event logger	Logs various system events
Plug-and-play manager	Monitors hardware to see what is out there

Fig. 11-22. The processes starting up during the boot phase. The ones above the line are always started. The ones below it are examples of services that could be started.

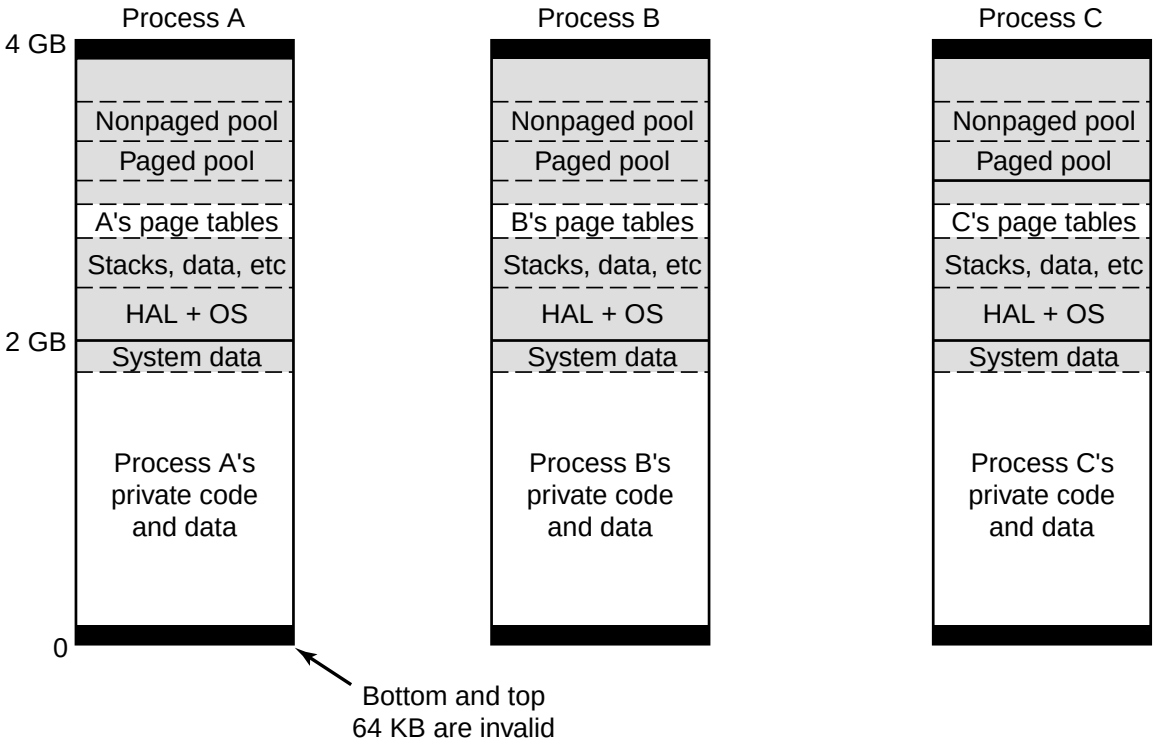


Fig. 11-23. Virtual address space layout for three user processes. The white areas are private per process. The shaded areas are shared among all processes.

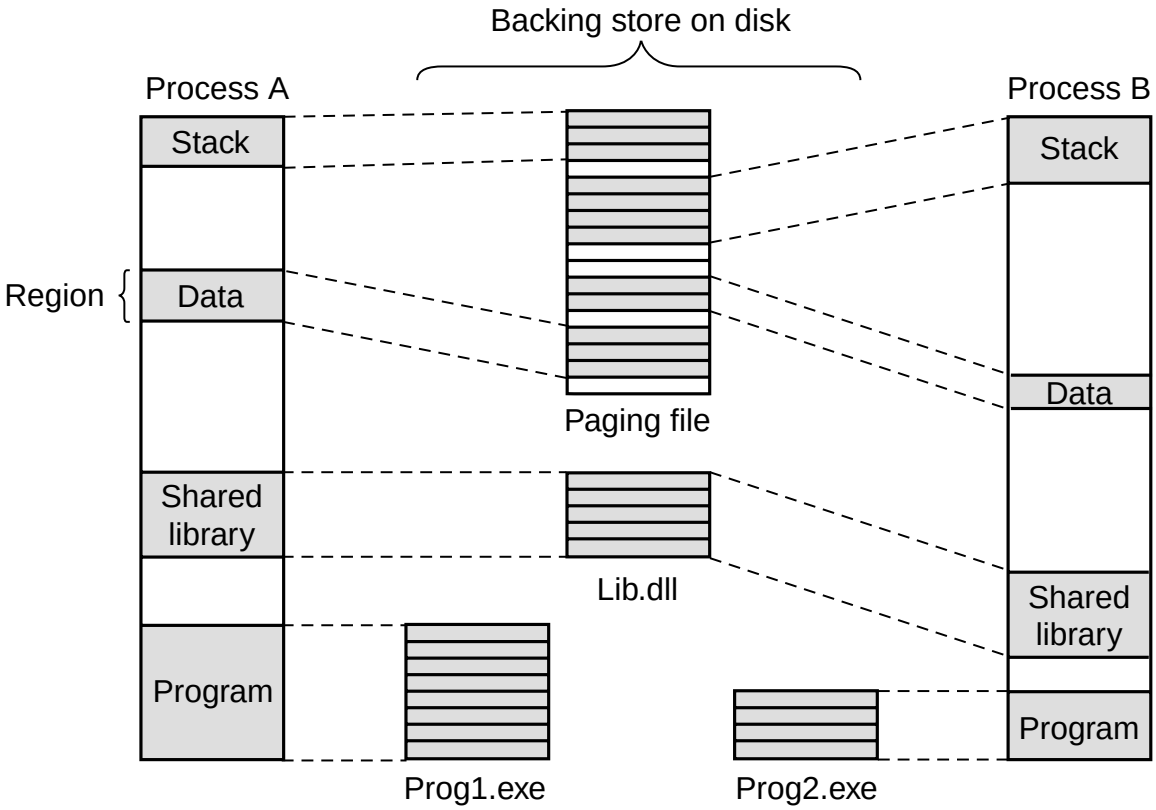
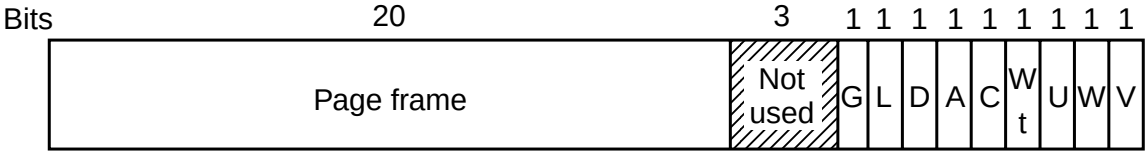


Fig. 11-24. Mapped regions with their shadow pages on disk. The *lib.dll* file is mapped into two address spaces at the same time.

Win32 API function	Description
VirtualAlloc	Reserve or commit a region
VirtualFree	Release or decommit a region
VirtualProtect	Change the read/write/execute protection on a region
VirtualQuery	Inquire about the status of a region
VirtualLock	Make a region memory resident (i.e., disable paging for it)
VirtualUnlock	Make a region pageable in the usual way
CreateFileMapping	Create a file mapping object and (optionally) assign it a name
MapViewOfFile	Map (part of) a file into the address space
UnmapViewOfFile	Remove a mapped file from the address space
OpenFileMapping	Open a previously created file mapping object

Fig. 11-25. The principal Win32 API functions for managing virtual memory in Windows 2000.



G: Page is global to all processes
L: Large (4-MB) page
D: Page is dirty
A: Page has been accessed
C: Caching enabled/disabled
Wt: Write through (no caching)
U: Page is accessible in user mode
W: Writing to the page permitted
V: Valid page table entry

Fig. 11-26. A page table entry for a mapped page on the Pentium.

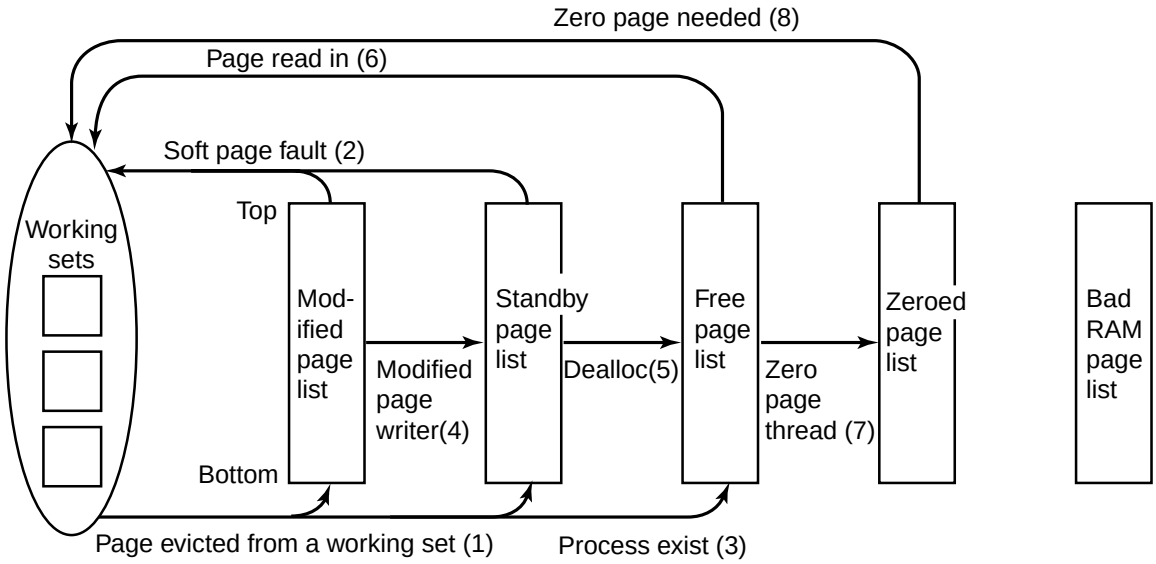


Fig. 11-27. The various page lists and the transitions between them.

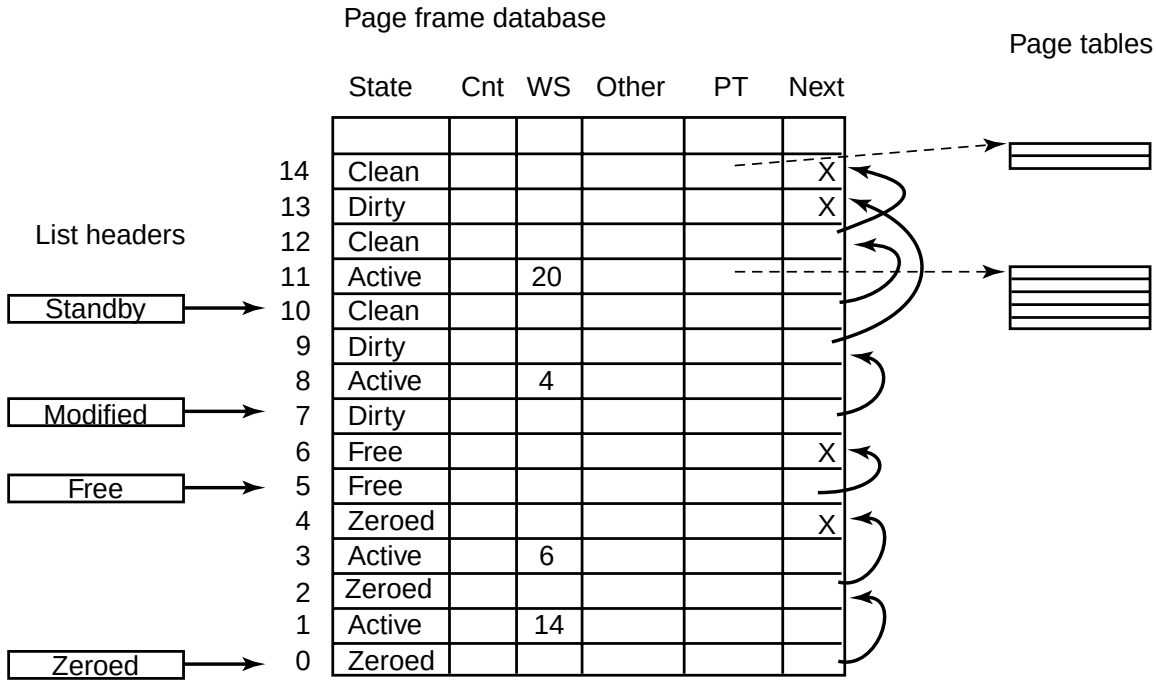


Fig. 11-28. Some of the major fields in the page frame database for a valid page.

API group	Description
Window management	Create, destroy, and manage windows,
Menus	Create, destroy, and append to menus and menu bars
Dialog boxes	Pop up a dialog box and collect information
Painting and drawing	Display points, lines, and geometric figures
Text	Display text in some font, size, and color
Bitmaps and icons	Placement of bitmaps and icons on the screen
Colors and palettes	Manage the set of colors available
The clipboard	Pass information from one application to another
Input	Get information from the mouse and keyboard

Fig. 11-29. Some categories of Win32 API calls.

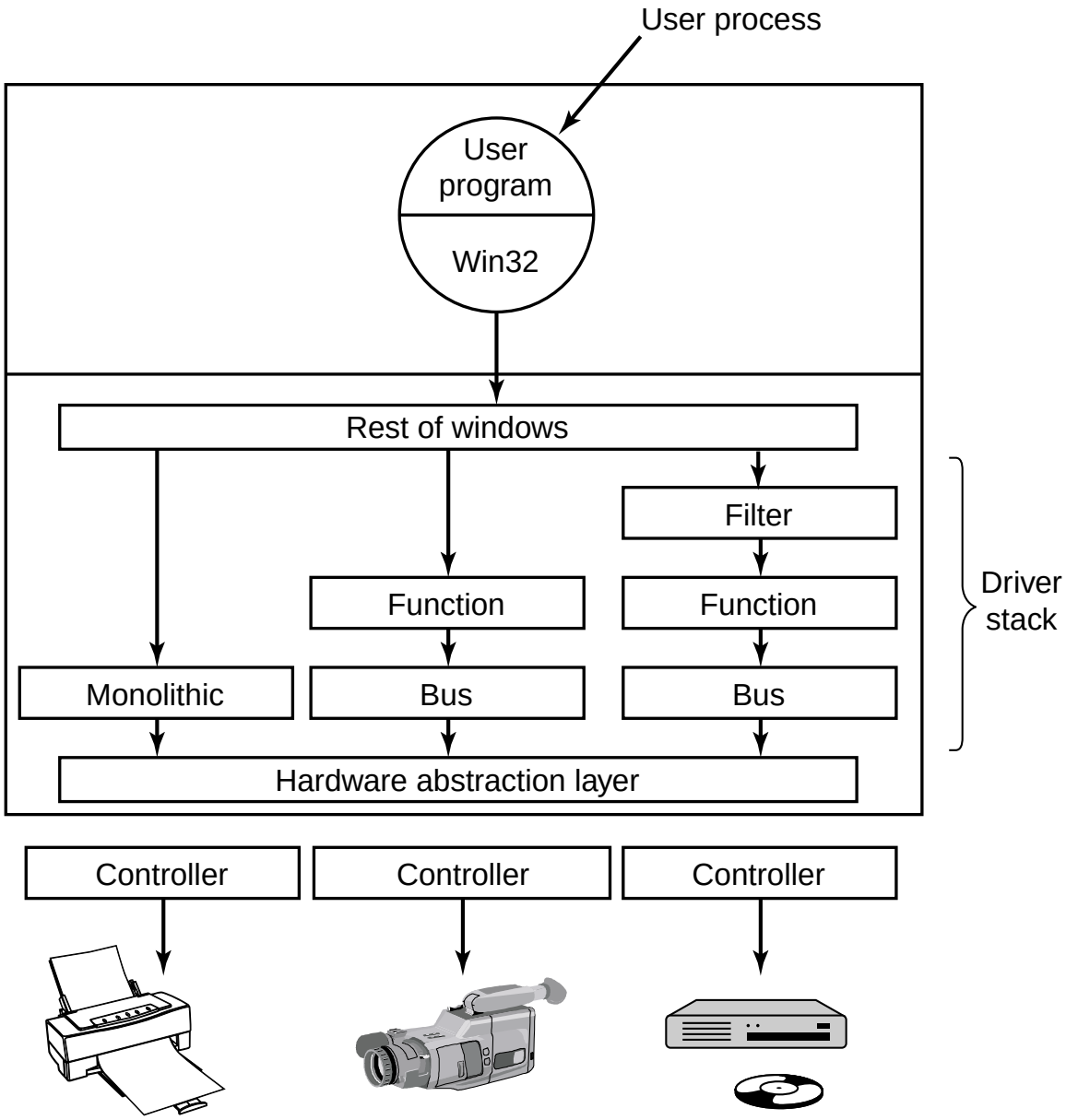


Fig. 11-30. Windows 2000 allows drivers to be stacked.

Win32 API function	UNIX	Description
CreateFile	open	Create a file or open an existing file; return a handle
DeleteFile	unlink	Destroy an existing file
CloseHandle	close	Close a file
ReadFile	read	Read data from a file
WriteFile	write	Write data to a file
SetFilePointer	lseek	Set the file pointer to a specific place in the file
GetFileAttributes	stat	Return the file properties
LockFile	fcntl	Lock a region of the file to provide mutual exclusion
UnlockFile	fcntl	Unlock a previously locked region of the file

Fig. 11-31. The principal Win32 API functions for file I/O. The second column gives the nearest UNIX equivalent.

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
   FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
   s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
   if (s && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
} while (s > 0 && count > 0);

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(outhandle);
```

Fig. 11-32. A program fragment for copying a file using the Windows 2000 API functions.

Win32 API function	UNIX	Description
CreateDirectory	mkdir	Create a new directory
RemoveDirectory	rmdir	Remove an empty directory
FindFirstFile	opendir	Initialize to start reading the entries in a directory
FindNextFile	readdir	Read the next directory entry
MoveFile	rename	Move a file from one directory to another
SetCurrentDirectory	chdir	Change the current working directory

Fig. 11-33. The principal Win32 API functions for directory management. The second column gives the nearest UNIX equivalent, when one exists.

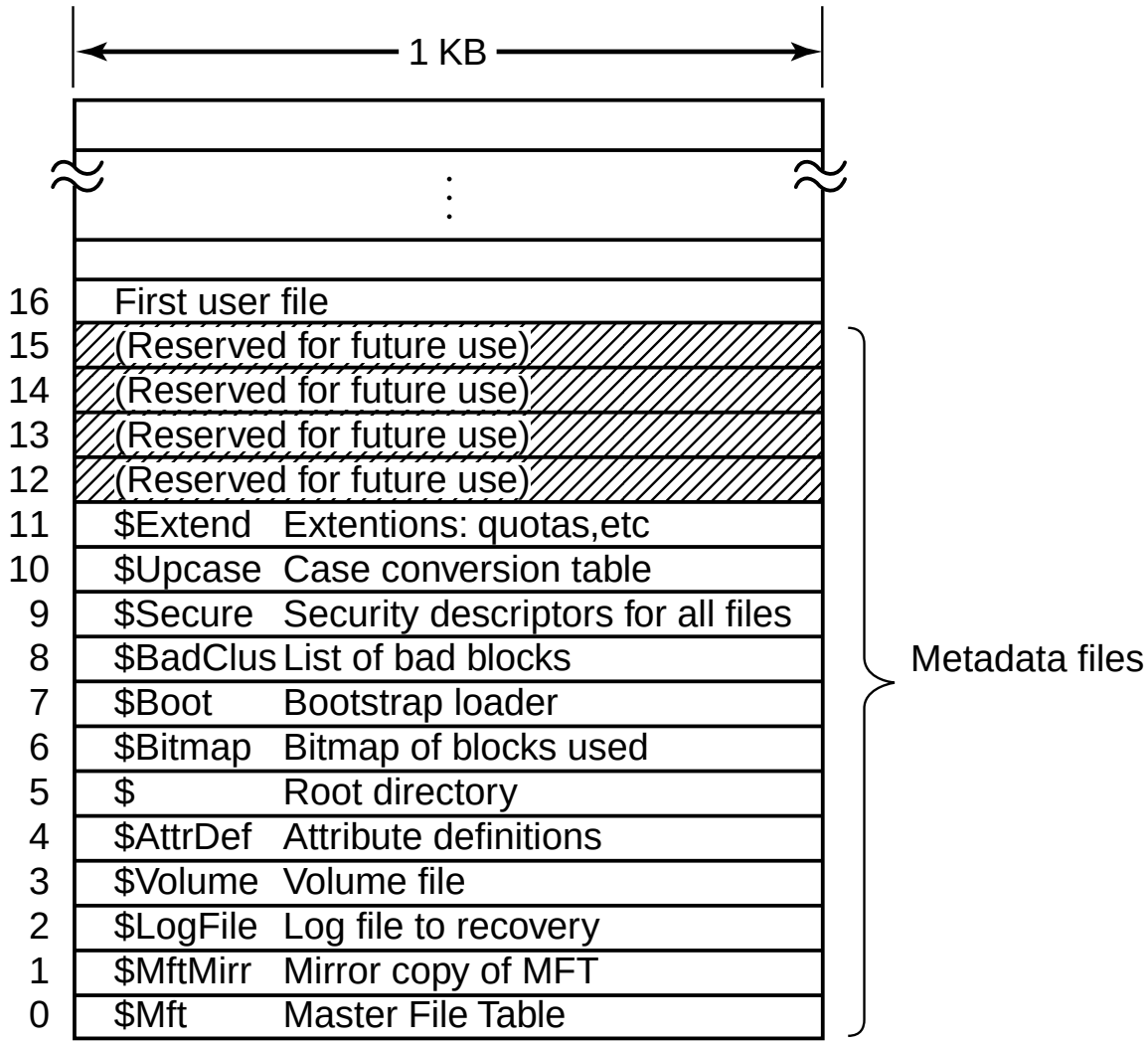


Fig. 11-34. The NTFS master file table.

Attribute	Description
Standard information	Flag bits, timestamps, etc.
File name	File name in Unicode; may be repeated for MS-DOS name
Security descriptor	Obsolete. Security information is now in $Extend$Secure
Attribute list	Location of additional MFT records, if needed
Object ID	64-bit file identifier unique to this volume
Reparse point	Used for mounting and symbolic links
Volume name	Name of this volume (used only in $Volume)
Volume information	Volume version (used only in $Volume)
Index root	Used for directories
Index allocation	Used for very large directories
Bitmap	Used for very large directories
Logged utility stream	Controls logging to $LogFile
Data	Stream data; may be repeated

Fig. 11-35. The attributes used in MFT records.

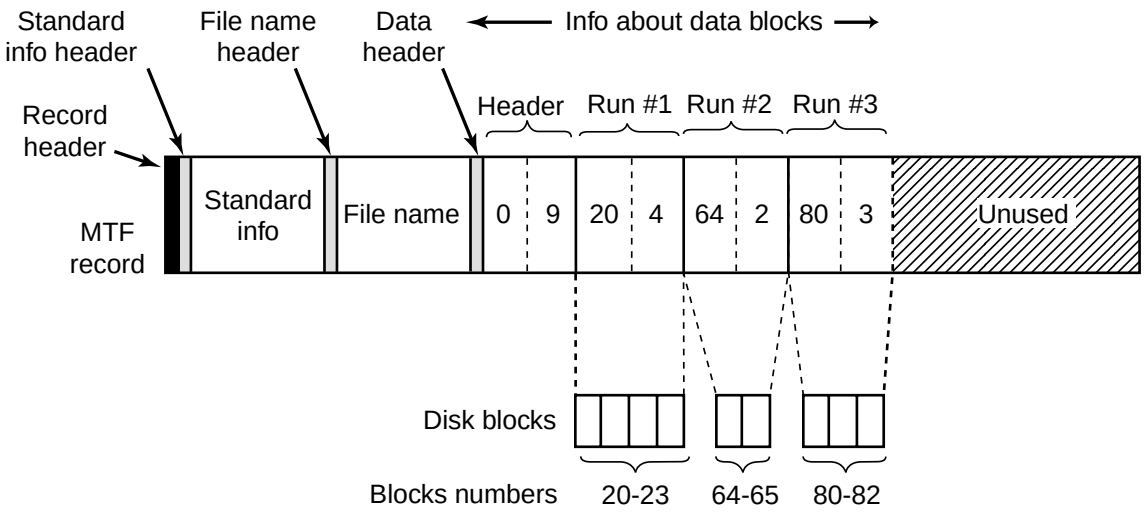


Fig. 11-36. An MFT record for a three-run, nine-block file.

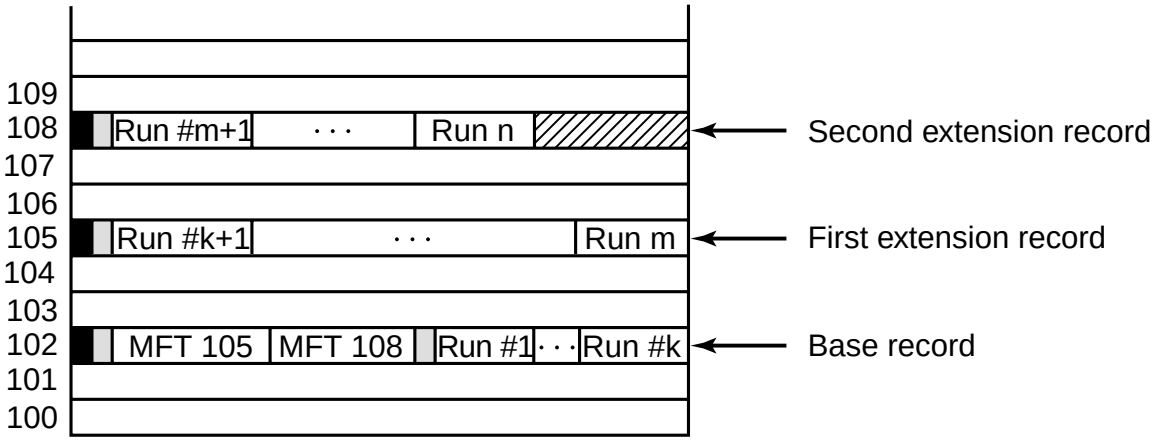


Fig. 11-37. A file that requires three MFT records to store all its runs.

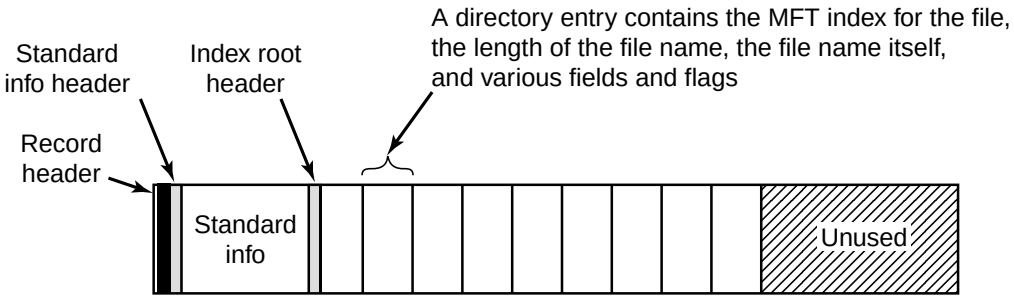


Fig. 11-38. The MFT record for a small directory.

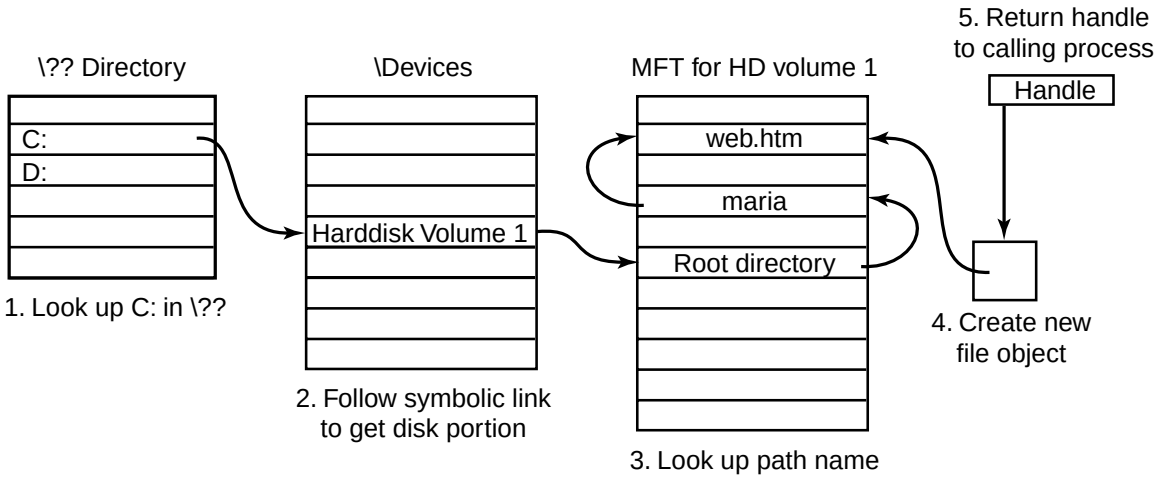


Fig. 11-39. Steps in looking up the file *C:\maria\web.htm*.

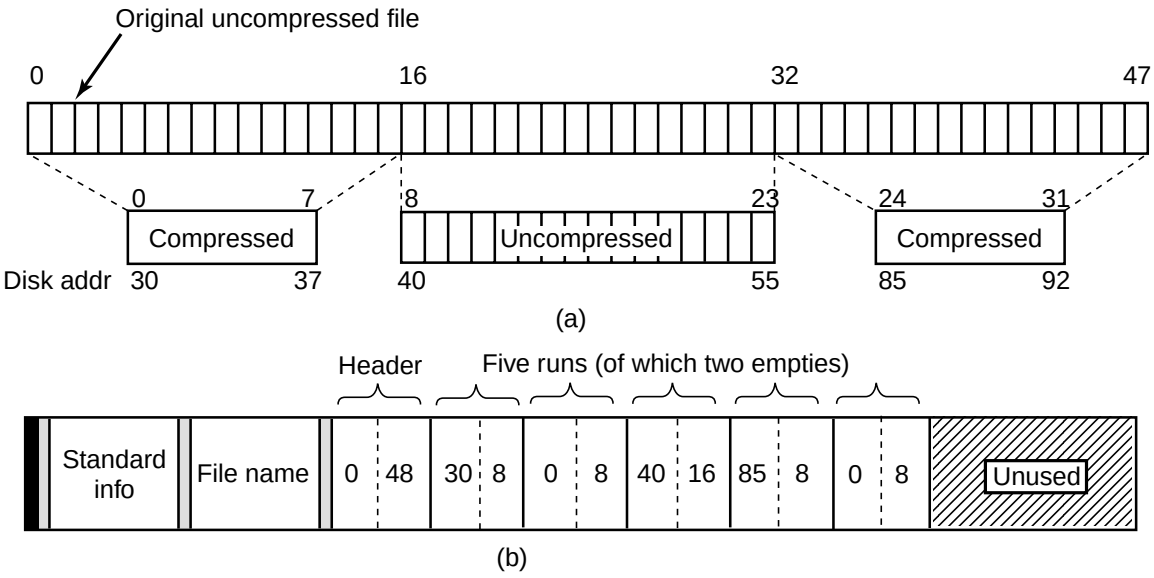


Fig. 11-40. (a) An example of a 48-block file being compressed to 32 blocks. (b) The MFT record for the file after compression.

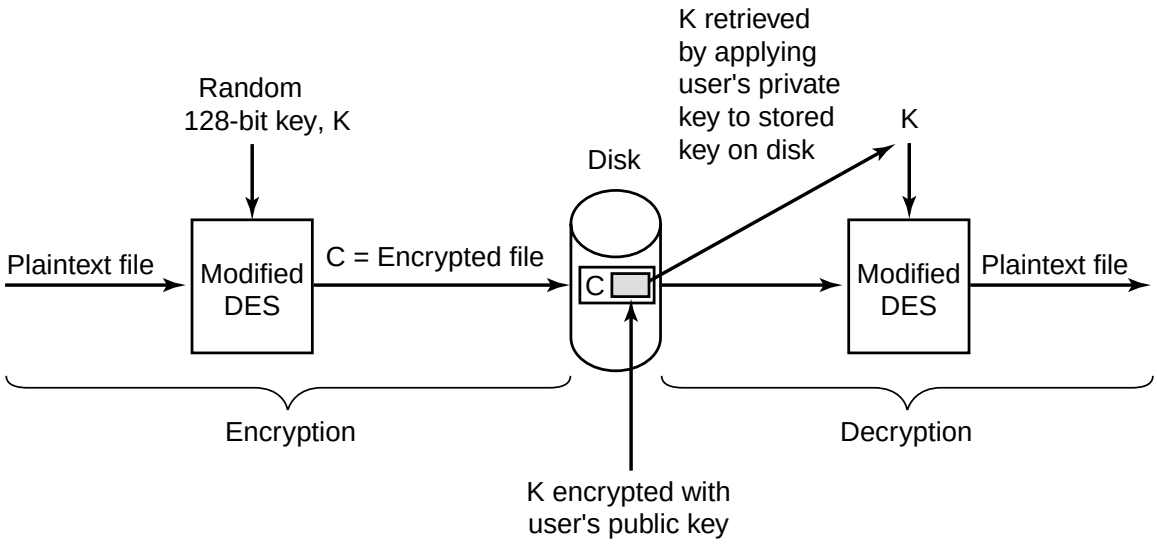


Fig. 11-41. Operating of the encrypting file system.

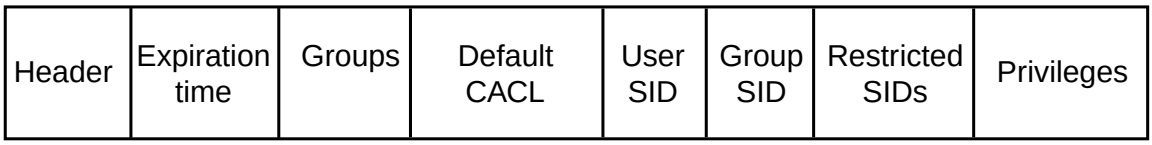


Fig. 11-42. Structure of an access token .

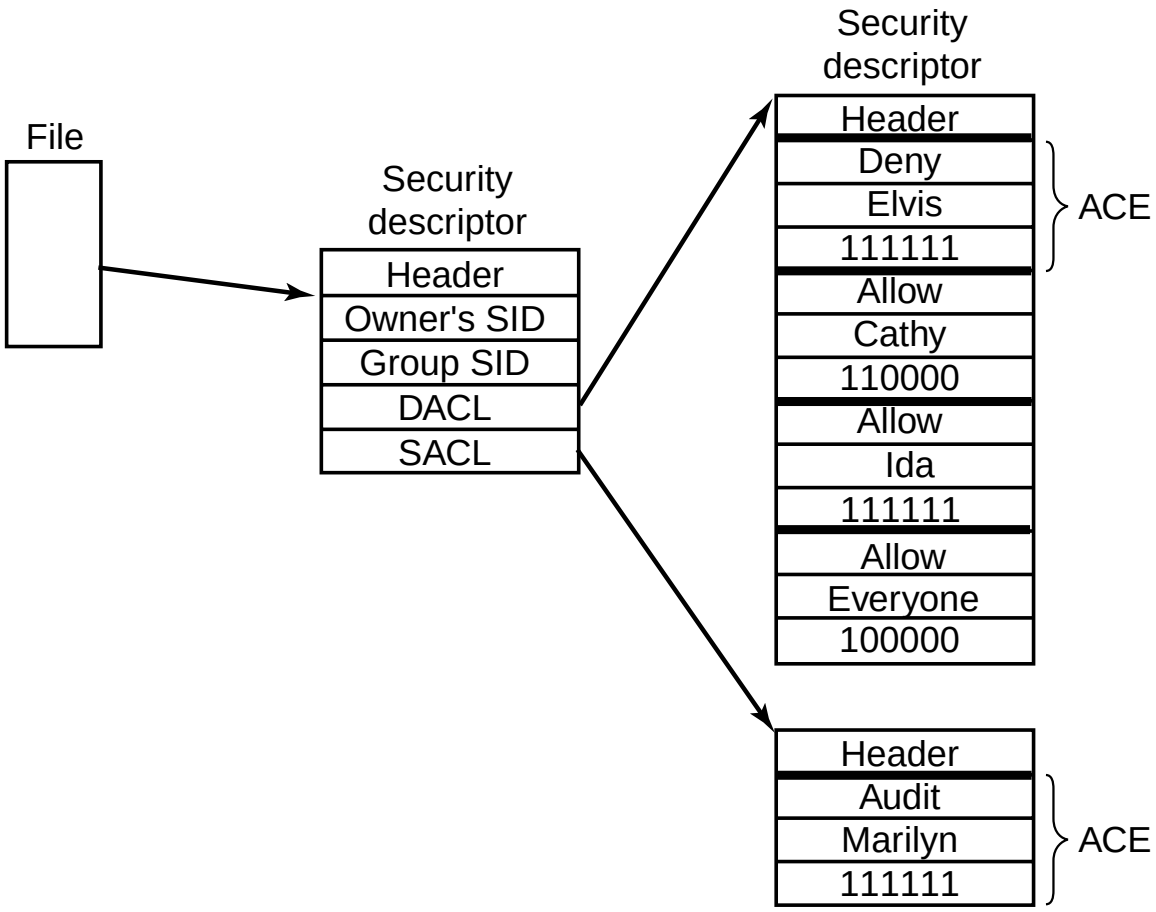


Fig. 11-43. An example security descriptor for a file.

Win32 API function	Description
InitializeSecurityDescriptor	Prepare a new security descriptor for use
LookupAccountSid	Look up the SID for a given user name
SetSecurityDescriptorOwner	Enter the owner SID in the security descriptor
SetSecurityDescriptorGroup	Enter a group SID in the security descriptor
InitializeAcl	Initialize a DACL or SACL
AddAccessAllowedAce	Add a new ACE to a DACL or SACL allowing access
AddAccessDeniedAce	Add a new ACE to a DACL or SACL denying access
DeleteAce	Remove an ACE from a DACL or SACL
SetSecurityDescriptorDacl	Attach a DACL to a security descriptor

Fig. 11-44. The principal Win32 API functions for security.

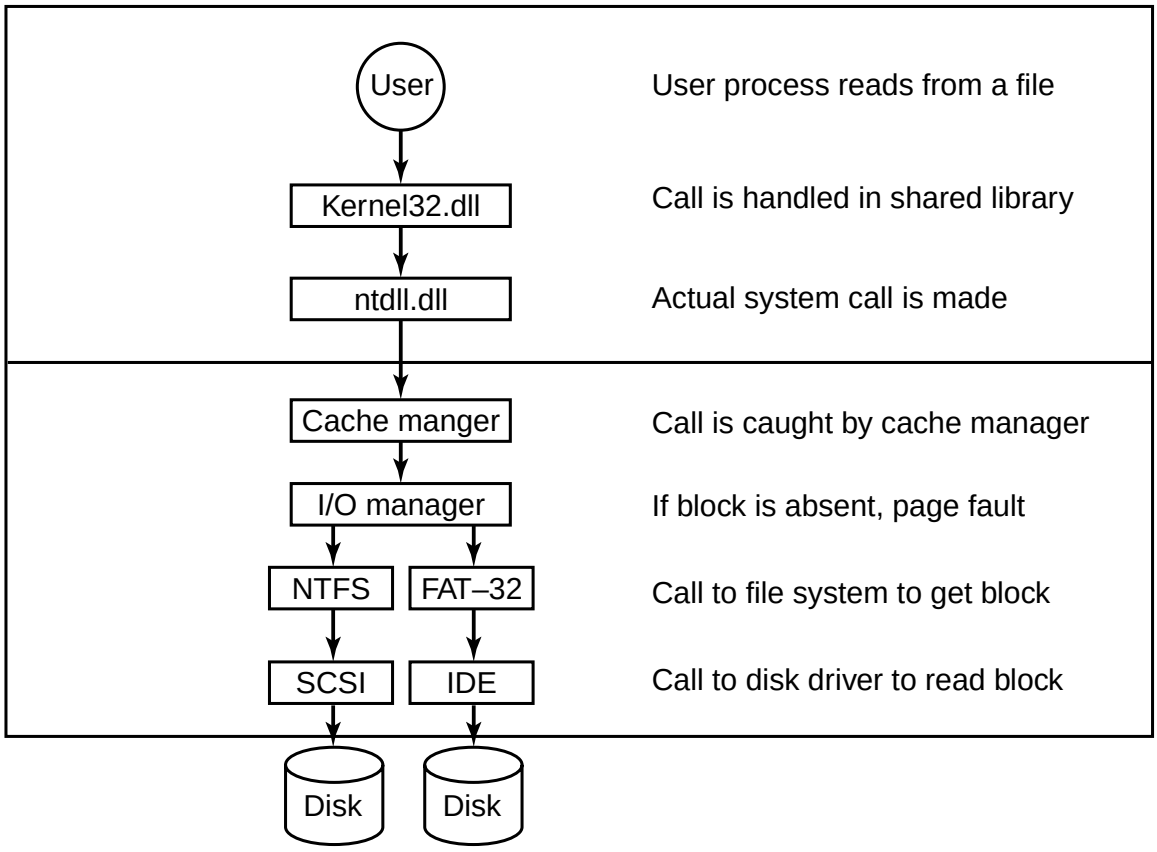


Fig. 11-45. The path through the cache to the hardware.