History of Vista

Year	MS-DOS	MS-DOS-based Windows	NT-based Windows	Notes
1981	MS-DOS 1.0			Initial release for IBM PC
1983	MS-DOS 2.0			Support for PC/XT
1984	MS-DOS 3.0			Support for PC/AT
1990		Windows 3.0		Ten million copies in 2 years
1991	MS-DOS 5.0			Added memory management
1992		Windows 3.1		Runs only on 286 and later
1993			Windows NT 3.1	
1995	MS-DOS 7.0	Windows 95		MS-DOS embedded in Win 95
1996			Windows NT 4.0	
1998		Windows 98		
2000	MS-DOS 8.0	Windows Me	Windows 2000	Win Me was inferior to Win 98
2001			Windows XP	Replaced Windows 98
2006			Windows Vista	

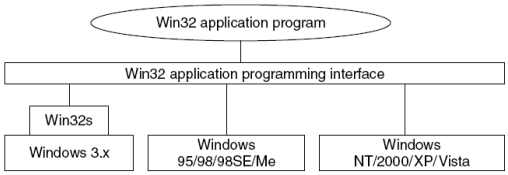
Major releases in the history of Microsoft operating systems for desktop PCs.

2000s: NT-based Windows (1)

Year	DEC operating system	Characteristics
1973	RSX-11M	16-bit, multi-user, real-time, swapping
1978	VAX/VMS	32-bit, virtual memory
1987	VAXELAN	Real-time
1988	PRISM/Mica	Canceled in favor of MIPS/Ultrix

DEC Operating Systems developed by Dave Cutler.

2000s: NT-based Windows (2)



The Win32 API allows programs to run on almost all versions of Windows.

2000s: NT-based Windows (3)

Year	Client version	Year	Server version
1996	Windows NT	1996	Windows NT Server
1999	Windows 2000	1999	Windows 2000 Server
2001	Windows XP	2003	Windows Server 2003
2006	Windows Vista	2007	Windows Server 2008

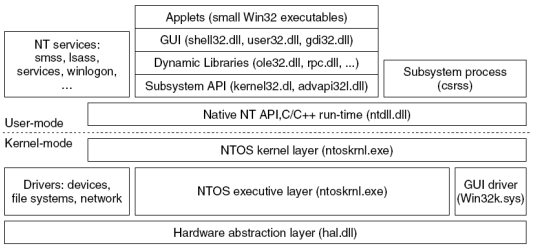
Split client and server releases of Windows.

Windows Vista

Kernel area	Linux	Vista
CPU Scheduler	50,000	75,000
I/O infrastructure	45,000	60,000
Virtual Memory	25,000	175,000

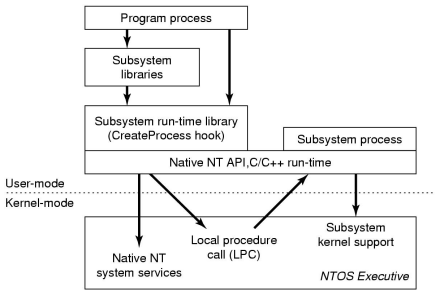
Comparison of lines of code for selected kernel-mode modules in Linux and Windows (from Mark Russinovich, co-author of Microsoft Windows Internals).

Programming Windows Vista



The programming layers in Windows.

Programming Windows Vista (2)



The components used to build NT subsystems.

The Native NT Application Programming Interface (1)

Object category	Examples
Synchronization	Semaphores, mutexes, events, IPC ports, I/O completion queues
I/O	Files, devices, drivers, timers
Program	Jobs, processes, threads, sections, tokens
Win32 GUI	Desktops, application callbacks

Common categories of kernel-mode object types.

The Native NT Application Programming Interface (2)

NtCreateProcess(&ProcHandle, Access, SectionHandle, DebugPortHandle, ExceptPortHandle, ...)
NtCreateThread(&ThreadHandle, ProcHandle, Access, ThreadContext, CreateSuspended, ...)
NtAllocateVirtualMemory(ProcHandle, Addr, Size, Type, Protection, ...)
NtMapViewOfSection(SectHandle, ProcHandle, Addr, Size, Protection, ...)
NtReadVirtualMemory(ProcHandle, Addr, Size, ...)
NtWriteVirtualMemory(ProcHandle, Addr, Size, ...)
NtCreateFile(&FileHandle, FileNameDescriptor, Access, ...)
NtDuplicateObject(srcProcHandle, srcObjHandle, dstProcHandle, dstObjHandle, ...)

Examples of native NT API calls that use handles to manipulate objects across process boundaries.

The Win32 Application Programming Interface

Win32 call	Native NT API call
CreateProcess	NtCreateProcess
CreateThread	NtCreateThread
SuspendThread	NtSuspendThread
CreateSemaphore	NtCreateSemaphore
ReadFile	NtReadFile
DeleteFile	NtSetInformationFile
CreateFileMapping	NtCreateSection
VirtualAlloc	NtAllocateVirtualMemory
MapViewOfFile	NtMapViewOfSection
DuplicateHandle	NtDuplicateObject
CloseHandle	NtClose

Examples of Win32 API calls and the native NT API calls that they wrap.

The Windows Registry (1)

Hive file	Mounted name	Use
SYSTEM	HKLM TEM	OS configuration information, used by kernel
HARDWARE	HKLM DWARE	In-memory hive recording hardware detected
BCD	HKLM BCD*	Boot Configuration Database
SAM	HKLM	Local user account information
SECURITY	HKLM URITY	lsass' account and other security information
DEFAULT	HKEY_USERS .DEFAULT	Default hive for new users
NTUSER.DAT	HKEY_USERS <user id>	User-specific hive, kept in home directory
SOFTWARE	HKLM TWARE	Application classes registered by COM
COMPONENTS	HKLM NENTS	Manifests and dependencies for sys. components

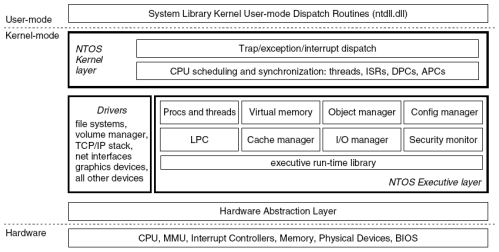
The registry hives in Windows Vista. HKLM is a short-hand for *HKEY_LOCAL_MACHINE.*

The Windows Registry (2)

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

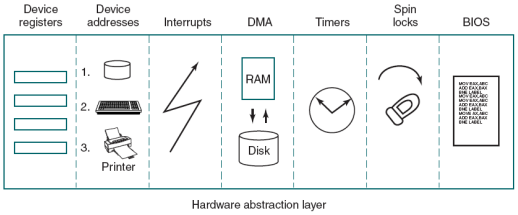
Some of the Win32 API calls for using the registry

Operating System Structure



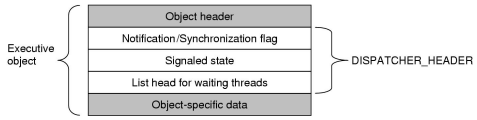
Windows kernel-mode organization.

The Kernel Layer



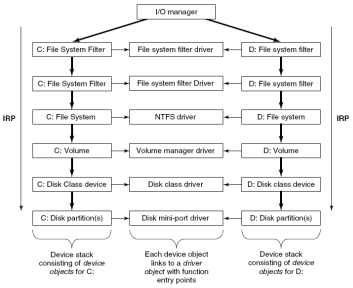
Some of the hardware functions the HAL manages.

Dispatcher Objects



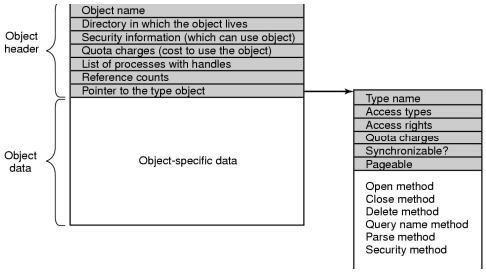
d*ispatcher_header* data structure embedded in many executive objects (*dispatcher objects*).

The Device Drivers



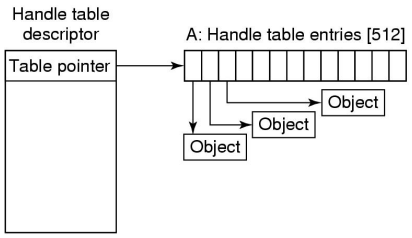
Simplified depiction of device stacks for two NTFS file volumes. I/O request packet passed from down the stack.

Implementation of the Object Manager



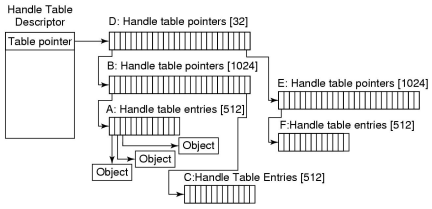
The structure of an executive object managed by the object manager.

Handles (1)



Handle table data structures for a minimal table using a single page for up to 512 handles.

Handles (2)



Handle table data structures for a maximal table of up to 16 million handles.

The Object Name Space (1)

Procedure	When called	Notes
Open	For every new handle	Rarely used
Parse	For object types that extend the namespace	Used for files and registry keys
Close	At last handle close	Clean up visible side effects
Delete	At last pointer dereference	Object is about to be deleted
Security	Get or set object's security descriptor	Protection
QueryName	Get object's name	Rarely used outside kernel

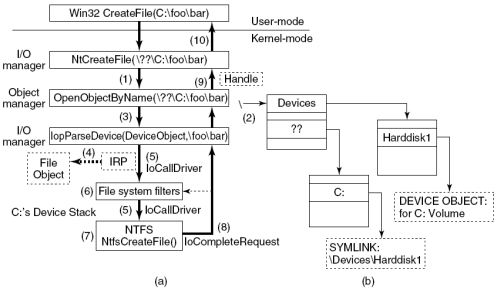
The object procedures supplied when specifying a new object type.

The Object Name Space (2)

Directory	Contents
??	Starting place for looking up MS-DOS devices like C:
DosDevices	Official name of ??, but really just a symbolic link to ??
Device	All discovered I/O devices
Driver	Objects corresponding to each loaded device driver
ObjectTypes	The type objects such as those listed in Fig. 11-22
Windows	Objects for sending messages to all the Win32 GUI windows
BaseNamedObjects	User-created Win32 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

Some typical directories in the object name space.

The Object Name Space (3)



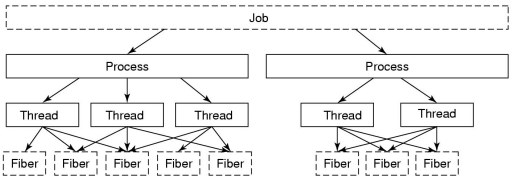
I/O and object manager steps for creating/opening a file and getting back a file handle.

The Object Name Space (4)

Type	Description
Process	User process
Thread	Thread within a process
Semaphore	Counting semaphore used for inter-process synchronization
Mutex	Binary semaphore used to enter a critical region
Event	Synchronization object with persistent state (signaled/not)
ALPC Port	Mechanism for inter-process 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	Object used for representing mappable files
Key	Registry key, used to attach registry to object manager namespace
Object directory	Directory for grouping objects within the object manager
Symbolic link	Refers to another object manager object by pathname
Device	I/O device object for a physical device, bus, driver, or volume instance
Device driver	Each loaded device driver has its own object

Some common executive object types managed by object manager.

Processes and Threads in Windows Vista (1)



The relationship between jobs, processes, threads and fibers. Jobs and fibers are optional; not all processes are in jobs or contain fibers.

Processes and Threads in Windows Vista (2)

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

Basic concepts used for CPU and resource management.

Job, Process, Thread, and Fiber Management API Calls (1)

- Actual search path for finding program to execute buried in library code for Win32, but managed more explicitly in UNIX.
- Current working directory is kernel-mode concept in UNIX but user-mode string in Windows.
- UNIX parses command line and passes an array of parameters, Win32 leaves argument parsing up to individual program.
- Whether file descriptors can be inherited in UNIX is property of handle. In Windows it is property of both handle and parameter to process creation.
- Win32 is GUI-oriented, new processes directly passed information about their primary window

Job, Process, Thread, and Fiber Management API Calls (2)

- Windows has no SETUID bit as property of executable, one process can create a process that runs as a different user, as long as it can obtain a token with that user's credentials.
- Process and thread handle returned from Windows can be used to modify the new process/thread in many substantive ways. UNIX just makes modifications to new process between fork and exec calls.

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
SwitchToFiber	Run a different fiber on the current thread
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
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

Some of the Win32 calls for managing processes, threads, and fibers.

Scheduling (1)

The following conditions cause the currently running thread to execute the scheduler code:

- The currently running thread blocks on a semaphore, mutex, event, I/O, etc.
- The thread signals an object (e.g., does an up on a semaphore or causes an event to be signaled).
- The quantum expires.

Scheduling (2)

The scheduler is also called under two other conditions:

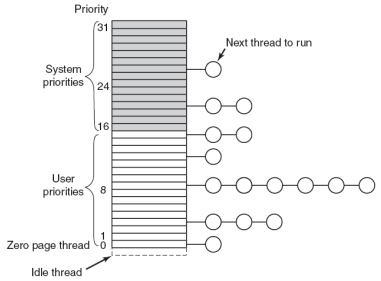
- An I/O operation completes.
- A timed wait expires.

Scheduling (3)

		Win32 process class priorities					
		Real-time	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

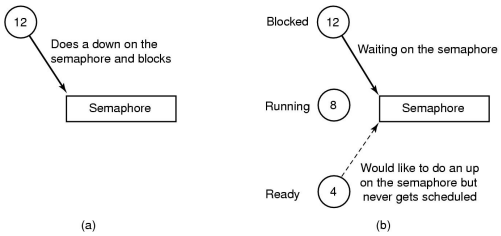
Mapping of Win32 priorities to Windows priorities.

Scheduling (4)



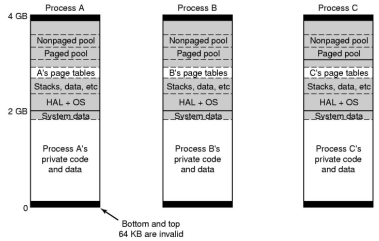
Windows Vista supports 32 priorities for threads.

Scheduling (5)



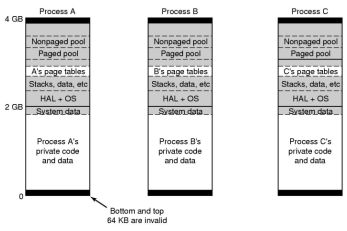
An example of priority inversion.

Memory Management(1)



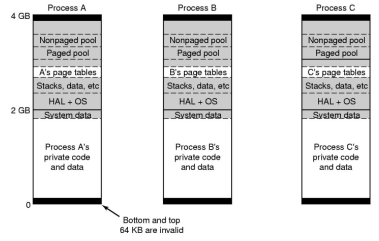
Virtual address space layout for three user processes on the x86. The white areas are private per process. The shaded areas are shared among all processes.

Memory Management (2)



Virtual address space layout for three user processes on the x86. The white areas are private per process. The shaded areas are shared among all processes.

Memory Management (3)



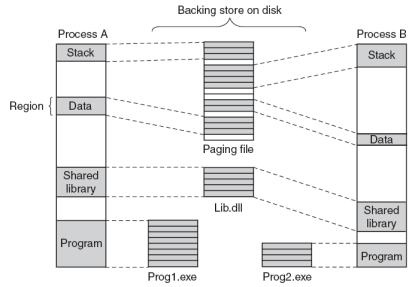
Virtual address space layout for three user processes on the x86. The white areas are private per process. The shaded areas are shared among all processes.

Addressing Large Physical Memories

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

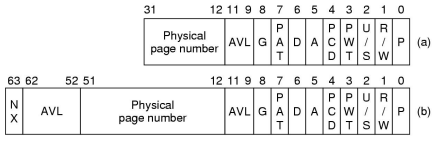
The principal Win32 API functions for managing virtual memory in Windows.

Implementation of Memory Management



Mapped regions with their shadow pages on disk. The *lib.dll* file mapped into two address spaces at same time.

Page Fault Handling (1)



NX – No eXecute
AVL – AVaiLable to the OS
G – Global page
PAT – Page Attribute Table
D – Dirty (modified)
A – Accessed (referenced)

PCD – Page Cache Disable
PWT – Page Write-Through
U/S – User/Supervisor
R/W – Read/Write access
P – Present (valid)

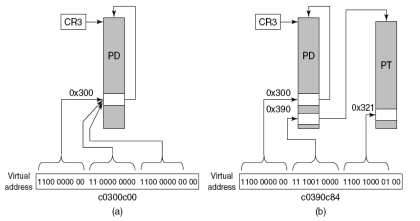
A page table entry (PTE) for a mapped page on the (a) Intel x86 and (b) AMD x64 architectures.

Page Fault Handling (2)

Each page fault can be considered as being in one of five categories:

- The page referenced is not committed.
- Attempted access to a page in violation of the permissions.
- A shared copy-on-write page was about to be modified.
- The stack needs to grow.
- The page referenced is committed but not currently mapped in.

Page Fault Handling (3)



Self-map: PD[0xc0300000>>22] is PD (page-directory)
Virtual address (a): (PTE *)(0xc0300c00) points to PD[0x300] which is the self-map page directory entry
Virtual address (b): (PTE *)(0xc0390c84) points to PTE for virtual address 0xe4321000

Windows self-map entry used to map the physical pages of page tables and page directory into kernel virtual addresses, for the x86.

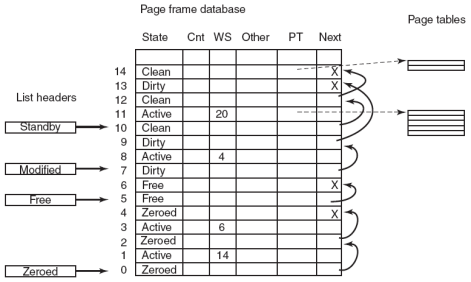
The Page Replacement Algorithm

- Three levels of activity by the working-set manager
  - Periodic based on a timer

New activity is added at each level:

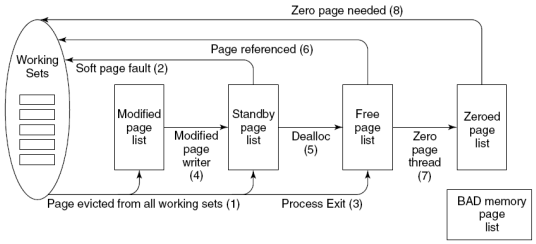
- Lots of memory available
- Memory getting tight
- Memory is tight

Physical Memory Management (1)



Some of the major fields in the page frame database for a valid page.

Physical Memory Management (2)



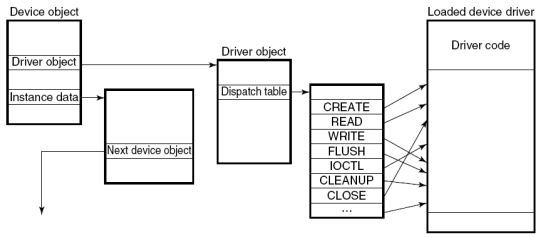
The various page lists and the transitions between them.

Input/Output API Calls

I/O system call	Description
NtCreateFile	Open new or existing files or devices
NtReadFile	Read from a file or device
NtWriteFile	Write to a file or device
NtQueryDirectoryFile	Request information about a directory, including files
NtQueryVolumeInformationFile	Request information about a volume
NtSetVolumeInformationFile	Modify volume information
NtNotifyChangeDirectoryFile	Complete when any file in the directory or sub-tree is modified
NtQueryInformationFile	Request information about a file
NtSetInformationFile	Modify file information
NtLockFile	Lock a range of bytes in a file
NtUnlockFile	Remove a range lock
NtFsControlFile	Miscellaneous operations on a file
NtFlushBuffersFile	Flush in-memory file buffers to disk
NtCancelIoFile	Cancel outstanding I/O operations on a file
NtDeviceIoControlFile	Special operations on a device

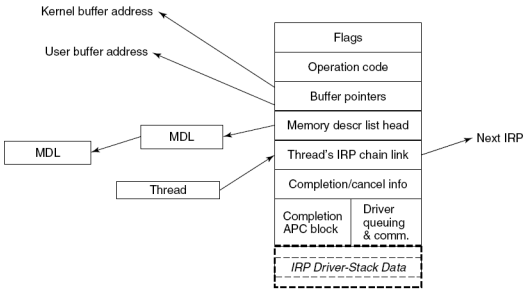
Native NT API calls for performing I/O.

Device Drivers



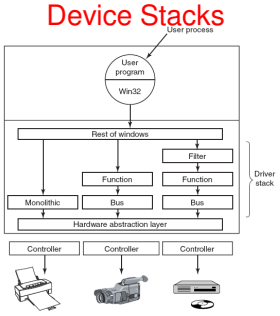
A single level in a device stack.

I/O Request Packets



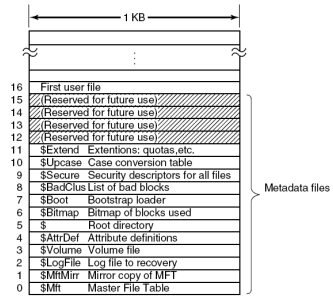
The major fields of an I/O Request Packet.

Device Stacks



Windows allows drivers to be stacked to work with a specific instance of a device. The stacking is represented by device objects.

File System Structure (1)



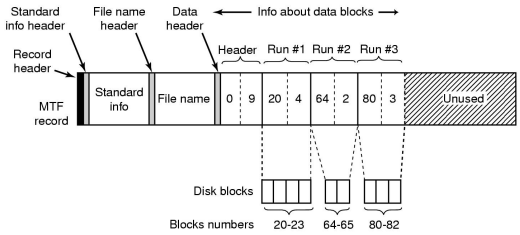
The NTFS master file table.

File System Structure (2)

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

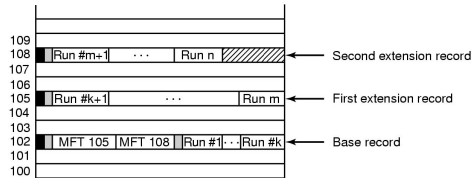
The attributes used in MFT records.

Storage Allocation (1)



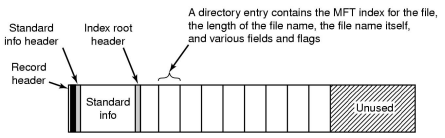
An MFT record for a three-run, nine-block stream.

Storage Allocation (2)



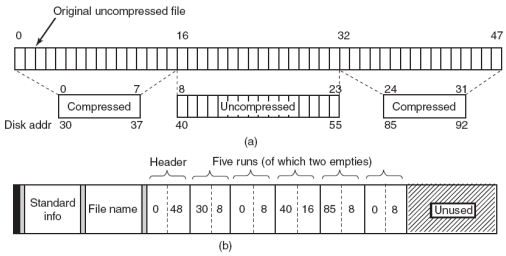
A file that requires three MFT records to store all its runs.

Storage Allocation (3)



The MFT record for a small directory.

File Compression



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

Security in Windows Vista (1)

Security properties inherited from the original security design of NT:

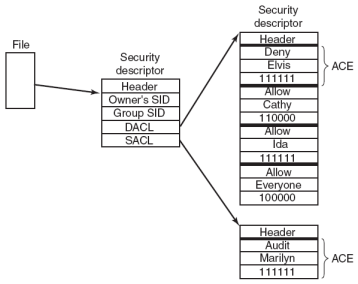
- Secure login with anti-spoofing measures.
- Discretionary access controls.
- Privileged access controls.
- Address space protection per process.
- New pages must be zeroed before being mapped in.
- Security auditing.

Security in Windows Vista (2)

Header	Expiration time	Groups	Default CACL	User SID	Group SID	Restricted SIDs	Privileges	Impersonation level	Integrity level

Structure of an access token.

Security in Windows Vista (3)



An example security descriptor for a file.

Security API Calls

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

The principal Win32 API functions for security.