Example: Windows

Windows History

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

- Windows 2000 additions
  - Plug-and-play
  - Network directory service
  - New GUI
- Vista additions
  - New GUI
  - More focus on security
  - Clean-up the code
- Windows 7
  - Focus on performance and compatibility

Different OS sizes

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

- Windows XP – 50 M
- Windows Vista – 70 M
Programming interface

Downward Compatibility

- Common API across both MS-DOS and NT based Windows
  - Important for downward compatibility
  - Increases the complexity of the API and its implementation
NT API: Programming layers

- Libraries (DLL) – as in most operating systems
- User mode service processes – accessed using RPC
- Upper layers interfaces public, but Native NT API not!

NT features

- Thread is the unit of concurrency
- Dynamic library is the unit of composition
- One command CreateProcess does both fork and exec
- Objects just for data hiding and abstraction
  - Object handles are specific to the process that created the object
  - Object is just a data structure
- Windows object naming
  - Unicode
  - Case-preserving, but case-insensitive!
Win32 API

- Public and fully documented
- Library routines to wrap native NT calls or to do the actual work in user mode
- Rich interface:
  - lot of parameters,
  - several functions for the same task
  - Mix of low-level and high-level functions

- Two special execution environments (Windows-on-Windows)
  - WOW32 – for 32-bit x86 systems to run 16-bit old windows 3.x applications
  - WOW64 – to allow 32-bit applications to run on x64 system

Example Win32 API Functions

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

- Cross between file system and database
- Used to store NT namespace
- Organised into separate volumes called hives
  - Each hive is kept in a separate file in directory `C:\Windows\system32\config`
- Loss of the registry requires reinstalling all software
- Regedit, procmon, PowerShell
- Current situation
  - Very disorganised, poorly defined conventions
  - Plans to use kernel-based transaction manager in the future

---

**System structure**
Windows kernel-mode layers

NTOS kernel layer

- Implements exception handling, trap and interrupt mechanisms – transition from user mode to kernel mode
- Scheduling and synchronizing threads
  - Switched at timer interrupts, starts waiting, higher-priority thread becomes ready-to-run
- Low-level support for synchronization
  - Control object
    - Including Deferred Procedure Call (DPC) object, Asynchronous Procedure Call (APC) object
  - Dispatcher object
DPC and APC

DPC - deferred proc. call
- Reduce the time to execute Interrupt Service Routines
- In any context (as interrupts)
- Only the critical operations are performed on high priority
- Rest is deferred in DPC object and executed later with lower priority
- Run level (2) between interrupts (3+) and normal processes (0)
- Executed immediately on a particular processor when level lowers to 2 (from 3+)

APC - asynchr. proc. call
- Somewhat similar to signals or notifications, defer processing of system routine, looks like unexpected procedure call
- In the thread context
- APCs queue to thread, executed in fifo order, when thread allows
- The kernel-mode APC continues DPCs work in the thread context
- User-mode APCs wake-up waiting thread

Dispatcher object

- A synchronization object, users can refer with handle
- A flag representing the signal state of the object
- A queue for threads waiting to be a signaled
- API call WaitForMultipleObjects allows thread to wait on any of the Dispatcher objects it has a handle to
- Notification - all waiting threads become runnable
- Synchronization - only the first waiting one become runnable
NTOS Executive Layer

- Contains most kernel service managers except drivers
  - Object manager
  - I/ O manager
  - Process manager
  - Memory manager
  - Cache manager
  - Security reference monitor
  - Configuration manager
  - LPC - local procedure call

Device driver

- Dynamic link libraries loaded by NTOS executive
- Load mechanism general, used also to extend the kernel

- A lot of objects:
  - Device object - for each element in the device stack (data flow path)
  - Driver object - contains the table of routines to use
  - Each device object is linked with a particular driver object
  - Some of these pairs are used just as filters
    - Pre- or postprocessing, provide new functionality, work around access rights
Device stack

HAL - Hardware abstraction layer

- Abstracts low-level hardware details
- Enhances portability across hardware platforms
- Cannot hide
  - Format of page-table entries
  - Physical memory page size, word length
  - Little-endian mode
  - Compare&swap synchronization primitive
- Can hide
  - Most other processor and chipset variation
- Main idea: less changes when porting
HAL

- Maps bus-related addresses onto system-wide logical addresses
- System-wide naming of interrupts
- DMA-transfers in device-independent way

Object manager & Object namespace

- Everything is object within Windows
  - An (executive) object is just a data structure in the virtual memory accessible in kernel mode
  - Object represents an abstract concept (file, process, ...)
- Object manager provides a uniform and consistent interface to managing all kinds of objects!
  - Creating, deleting and accounting
  - Objects can have names within NT namespace
  - Objects do not exist over reboots, they need to be created every time system is booted
Executive object

- Memory for kernel objects is allocated either from the pageable or nonpageable kernel memory heap by executive layer.
- Objects have a quota charge information.
- Process has limited quota.
- Objects are 'named' using handles, which user process can use to access a particular object.

Handle table

- Each process has its own handle table.
Object name space

- Handles are process-specific, cannot be shared directly
- For sharing process can duplicate its handle only to processes it has a handle to
- Better way of sharing: give object a name
  - Arbitrary names can be given to any object
- Object exists only as long as there is at least one process having handle to it or it has been given a name
- Only some object types have interfaces supporting names
- Object manager has dirs and symbolic links for names

Typical directories in object name space

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>??</td>
<td>Starting place for looking up MS-DOS devices like C:</td>
</tr>
<tr>
<td>DosDevices</td>
<td>Official name of ??, but really just a symbolic link to ??</td>
</tr>
<tr>
<td>Device</td>
<td>All discovered I/O devices</td>
</tr>
<tr>
<td>Driver</td>
<td>Objects corresponding to each loaded device driver</td>
</tr>
<tr>
<td>ObjectType</td>
<td>The type objects such as those listed in Fig. 11-22</td>
</tr>
<tr>
<td>Windows</td>
<td>Objects for sending messages to all the Win32 GUI windows</td>
</tr>
<tr>
<td>BaseNamedObjects</td>
<td>User-created Win32 objects such as semaphores, mutexes, etc.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Partition names discovered by the boot loader</td>
</tr>
<tr>
<td>NLS</td>
<td>National Language Support objects</td>
</tr>
<tr>
<td>FileSystem</td>
<td>File system driver objects and file system recognizer objects</td>
</tr>
<tr>
<td>Security</td>
<td>Objects belonging to the security system</td>
</tr>
<tr>
<td>KnownDLLs</td>
<td>Key shared libraries that are opened early and held open</td>
</tr>
</tbody>
</table>

Tan08: Fig 11-21
Example: creating open-file object

Booting
Booting Windows Vista

- Initial startup phase: BIOS loads the small boot strap loader from disk
- Bootstrap loader load the BootMgr
- BootMgr can resume a hibernated system WinResume or load a new one WinLoad
- WinLoad loads:
  - Ntoskrnl.exe, hal.dll (hardware abstraction)
  - boot drivers (hard drive, file system, etc)
- Run smss - first program in user space (like init in UNIX)
- Windows boot procedure contains several recovery mechanism for possible problem situations encountered during boot

User-mode components

- Subsystems
  - Seen as a way to support multiple OS personalities
- Dynamic link libraries (D LL)
  - Linked to executable program at run-time not at compilation
  - Common code can be shared, versioning becomes an issue
  - D LLs form a graph, one may need others to be loaded
  - Allowed to run when a new process or thread is started (this attach is meant just for initialization purpose, but ...)
- User-mode services
  - Extends system’s functionality, strongly used in Vista
Process management

Process

- Just a container
  - Virtual address space
  - Handles to kernel-mode objects and threads
  - Hold common resources (quota structure, shared tokens)
- Process Environment Block (PEB)
  - List of loaded modules (dll & exe)
  - Memory containing environment strings
  - Current working directory
  - Data for managing the process' heap
- User shared data (kernel writes, threads only read)
  - Performance optimization: time formats, version info, flags
Thread

- Abstraction for scheduling the CPU
  - State
- Priorities assigned to thread based on the process priority value
- Affinitized to certain processors
- Thread Environment Block (TEB)
  - Thread Local Storage
  - Fields for localization (language, cultural)
- All system threads belong to special system process

Jobs & fibers

- Job is a group of processes sharing the same resource limits and restrictions (quota etc)
- A fiber has a stack and data structure for registers and data
  - Fibers do not belong to threads,
  - Threads are converted to fibers or fibers created independently of threads
  - Independent fibers can run only by thread’s explicit call SwitchToFiber
  - Overhead of switching between fibers is low, rarely used
Scheduling

- Priority based, round-robin on each priority level
- Default time quantum on user system 20ms and server 180 ms
- Threads must execute the scheduler code when they block, quantum expires or they signal an object
- Scheduling code is also run as DPC after I/O and timer interrupts

Memory management
Virtual address space layout

Physical Memory Management

Some of the major fields in the page frame database for a valid page
File system (and I/O)
**NTFS: Features**

- Recovers from system failures and disk errors
  - LFS log file
- Access control
  - Security descriptor (pääsylistat)
- Allow large disks and files
  - FAT32 has only $2^{32}$ blocks, large allocation table
- File objects are pairs (value, attribute)
- Allows indexing for faster file access
- Block, cluster
  - One or more contiguous sectors (e.g. 512 B - 4 KB)
  - Unit for allocation and accounting
- Partition, volume
  - Logical part of the physical disk.
  - Has its own file system

---

**NTFS-partition**

<table>
<thead>
<tr>
<th>partition boot sector</th>
<th>Master File Table</th>
<th>System Files</th>
<th>File Area</th>
</tr>
</thead>
</table>

- Boot sector
  - Boot information, partition and file system structure
  - Location of MFT
- MFT
  - Information about files, folders and free blocks
- System Files (~ 1MB)
  - Duplicate of the early part of MFT
  - Log for recovery, bitmap free/allocated blocks, attribute definitions
- File Area

Sta05: Fig 12.17
NTFS - MFT (Master File Table)

- Linear sequence of 1-KB MFT records
  - Describes files or directories, one per file or directory
  - Contains sequence of (attribute header, value) pairs
    - variable-size of the record used
- 16 first records reserved for metadata
  - Names start with dollar sign $ 
- For small files, the MFT record contains also the data
- For large files, the data on a separate storage area
  - MFT-record has the sequence of block numbers
  - Can continue in other (even several) MFT records

![Diagram of NTFS MFT]

Fig. 11-34. The NTFS master file table.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard information</td>
<td>Flag bits, timestamps, etc.</td>
</tr>
<tr>
<td>File name</td>
<td>File name in Unicode, may be repeated for MS-DOS name</td>
</tr>
<tr>
<td>Security descriptor</td>
<td>Obsolete. Security information is now in $Extends$Secure</td>
</tr>
<tr>
<td>Attribute list</td>
<td>Location of additional MFT records, if needed</td>
</tr>
<tr>
<td>Object ID</td>
<td>64-bit file identifier unique to this volume</td>
</tr>
<tr>
<td>Reparse point</td>
<td>Used for mounting and symbolic links</td>
</tr>
<tr>
<td>Volume name</td>
<td>Name of this volume (used only in $Volume$)</td>
</tr>
<tr>
<td>Volume information</td>
<td>Volume version (used only in $Volume$)</td>
</tr>
<tr>
<td>Index root</td>
<td>Used for directories</td>
</tr>
<tr>
<td>Index allocation</td>
<td>Used for very large directories</td>
</tr>
<tr>
<td>Bitmap</td>
<td>Used for very large directories</td>
</tr>
<tr>
<td>Logged utility stream</td>
<td>Controls logging to $LogFile$</td>
</tr>
<tr>
<td>Data</td>
<td>Stream data, may be repeated</td>
</tr>
</tbody>
</table>

Fig. 11-35. The attributes used in MFT records.
[Tane 01]

MFT Record

Fig. 11-36. An MFT record for a three-run, nine-block file.
[Tane 01]
MFT record for small directory

- In small directories, the MFT index of files is sequential order.

- In large directories, the MFT index of files in B-tree.
  - Search for name no longer sequential.
<table>
<thead>
<tr>
<th>Win32 API function</th>
<th>UNIX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateFile</td>
<td>open</td>
<td>Create a file or open an existing file, return a handle</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>unlink</td>
<td>Destroy an existing file</td>
</tr>
<tr>
<td>CloseHandle</td>
<td>close</td>
<td>Close a file</td>
</tr>
<tr>
<td>ReadFile</td>
<td>read</td>
<td>Read data from a file</td>
</tr>
<tr>
<td>WriteFile</td>
<td>write</td>
<td>Write data to a file</td>
</tr>
<tr>
<td>SetFilePointer</td>
<td>seek</td>
<td>Set the file pointer to a specific place in the file</td>
</tr>
<tr>
<td>GetFileAttributes</td>
<td>stat</td>
<td>Return the file properties</td>
</tr>
<tr>
<td>LockFile</td>
<td>fcntl</td>
<td>Lock a region of the file to provide mutual exclusion</td>
</tr>
<tr>
<td>UnlockFile</td>
<td>fcntl</td>
<td>Unlock a previously locked region of the file</td>
</tr>
</tbody>
</table>

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

[Tane 01]

/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("new", 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, &cnt, 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.

[Tane 01]
NTFS: Recovery

- **Log FS**
  - Journaling, log all changes, only structural changes
  - Created in file cache as one transaction

- **Change disk content**
  1. Create the the log entry in file cache
  2. Make the file change in file cache first
  3. Store the log entry to disk from file cache
  4. Write the file changed from cache to disk
  5. **Commit** the changes

If the system crash happens before all changes are written to disk, the reboot can **rollback** to consistent situation using the log

- No guarantee that the file content remains
- System structure remains coherent