Example: Windows

Lecture 11, Monday 10.10.2011

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>Initial release for IBM PC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1983</td>
<td>MS-DOS 2.0</td>
<td>Support for PC/XT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1984</td>
<td>MS-DOS 3.0</td>
<td>Support for PCAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>Windows 3.0</td>
<td>Ten million copies in 2 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1991</td>
<td>MS-DOS 5.0</td>
<td>Added memory management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1992</td>
<td>Windows 3.1</td>
<td>Runs only on 286 and later</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1993</td>
<td>Windows NT 3.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1995</td>
<td>MS-DOS 7.0</td>
<td>Windows 95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>Windows NT 4.0</td>
<td>MS-DOS embedded in Win 92</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, Windows 2000, Windows XP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>Windows XP</td>
<td>Replaced Windows ME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Windows Vista</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2009 Windows 7

Features added

- Windows2000 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

- Windows XP - 50 M
- Windows Vista - 70 M

Programming interface

- 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!

**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>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>CreateFile</td>
<td>Create a new file</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>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, rules, etc.</td>
</tr>
<tr>
<td>WaitForMultipleObjects</td>
<td>Block on a set of objects whose handles are given</td>
</tr>
<tr>
<td>PostEvent</td>
<td>Set an event to signal 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

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 signaled
- The kernel-mode APC continues DPCs work in the thread context
- User-mode APCs wake-up waiting thread

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

- 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

Example: creating open-file object

Booting

- 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 (DLL)
  - Linked to executable program at run-time not at compilation
  - Common code can be shared, versioning becomes an issue
  - DLLs form a graph, one may need others to be loaded
  - Allowed to run when a new process or thread is started (this stack 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
Operating Systems, Fall 2011

10.10.2011

Lecture 11, Tiina Niklander 7

Virtual address space layout

Physical Memory Management

Windows: page frames

File system (and I/O)

NTFS: Features

NTFS-partition
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

![Fig. 11-34. The NTFS master file table.](Tan01)

---

<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 LCUOS name</td>
</tr>
<tr>
<td>Security descriptor</td>
<td>Object ID, security information is now in SDDL (Security Description List)</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 reinterpreting symbolic links</td>
</tr>
<tr>
<td>Volume information</td>
<td>Name of this volume (used only in $volumes)</td>
</tr>
<tr>
<td>Index root</td>
<td>Used for directories</td>
</tr>
<tr>
<td>Index allocation</td>
<td>Used for files, not directories</td>
</tr>
<tr>
<td>Bitmap</td>
<td>Used for very large directories</td>
</tr>
<tr>
<td>Log record stream</td>
<td>Common log recording log file</td>
</tr>
<tr>
<td>Data</td>
<td>Data stream, may be repeated</td>
</tr>
</tbody>
</table>

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

![Fig. 11-36. An MFT record for a three-run, nine-block file.](Tan01)

MFT record for small directory

- In small directories, the MFT index of files in sequential order
  - In large directories, the MFT index of files in B-tree
    - Search for name no longer sequential

![Fig. 11-45. A directory entry contains the MFT index for the file.](Tan01)
NTFS: Recovery

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

- **Change disk content**
  1. Create 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 change 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

---

Guest lecture

- **NOW**: Monday 10.10. at 16:15 in C222

- Dr. Pan Hui (Deutche Telecom): “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading”