# NT Timeline: the first 20 years

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/1989</td>
<td>Design/Coding Begins</td>
</tr>
<tr>
<td>7/1993</td>
<td>NT 3.1</td>
</tr>
<tr>
<td>9/1994</td>
<td>NT 3.5</td>
</tr>
<tr>
<td>5/1995</td>
<td>NT 3.51</td>
</tr>
<tr>
<td>7/1996</td>
<td>NT 4.0</td>
</tr>
<tr>
<td>12/1999</td>
<td>NT 5.0 Windows 2000</td>
</tr>
<tr>
<td>8/2001</td>
<td><strong>NT 5.1 Windows XP – ends Windows 95/98</strong></td>
</tr>
<tr>
<td>3/2003</td>
<td>NT 5.2 Windows Server 2003</td>
</tr>
<tr>
<td>8/2004</td>
<td>NT 5.2 Windows XP SP2</td>
</tr>
<tr>
<td>4/2005</td>
<td>NT 5.2 Windows XP 64 Bit Edition (&amp; WS03SP1)</td>
</tr>
<tr>
<td>10/2006</td>
<td>NT 6.0 Windows Vista (client)</td>
</tr>
<tr>
<td>2/2008</td>
<td>NT 6.0 Windows Server 2008 (Vista SP1)</td>
</tr>
<tr>
<td>10/2009</td>
<td>NT 6.1 Windows 7 &amp; Windows Server 2008 R2</td>
</tr>
</tbody>
</table>
Windows Academic Program

Historically little information on NT available
  – Microsoft focus was end-users and Win9x
  – Source code for universities was too encumbered

Much better internals information today
  – Windows Internals, 4th Ed., Russinovich & Solomon
  – Windows Academic Program (universities only):
    • CRK: Curriculum Resource Kit (NT kernel in PowerPoint)
    • WRK: Windows Research Kernel (NT kernel in source)
  – Chapters in leading OS textbooks (Tanenbaum, Silberschatz, Stallings)
Windows Architecture

- **Applications**
  - DLLs
  - System Services
  - Logon/GINA
  - Kernel32
  - Critical services
  - win32

- **Subsystem servers**
  - User-mode
    - System library (ntdll) / run-time library
  - Kernel-mode
    - NTOS kernel layer
      - Drivers
      - NTOS executive layer
      - HAL
    - Firmware, Hardware
Windows Kernel-mode Architecture

user mode

NT API stubs (wrap sysenter) -- system library (ntdll.dll)

kernel mode

NTOS kernel layer

- Trap/Exception/Interrupt Dispatch
- CPU mgmt: scheduling, synchr, ISRs/DPCs/APCs

Drivers
- Devices, Filters, Volumes, Networking, Graphics

Procs/Threads

Virtual Memory

Caching Mgr

IPC

glue

I/O

Object Mgr

Security

Registry

NTOS executive layer

Hardware Abstraction Layer (HAL): BIOS/chipset details

firmware/hardware

CPU, MMU, APIC, BIOS/ACPI, memory, devices
NT (Native) API examples

NtCreateProcess (&ProcHandle, Access, SectionHandle, DebugPort, ExceptionPort, …)
NtCreateThread (&ThreadHandle, ProcHandle, Access, ThreadContext, bCreateSuspended, …)
NtAllocateVirtualMemory (ProcHandle, Addr, Size, Type, Protection, …)
NtMapViewOfSection (SectHandle, ProcHandle, Addr, Size, Protection, …)
NtReadVirtualMemory (ProcHandle, Addr, Size, …)
NtDuplicateObject (srcProcHandle, srcObjHandle, dstProcHandle, dstHandle, Access, Attributes, Options)
Object Manager
NT Object Manager

Provides unified management of:

- kernel data structures
- kernel references
- user references (handles)
- namespace
- synchronization objects
- resource charging
- cross-process sharing
- central ACL-based security reference monitor
- configuration (registry)
Object Types

Object Manager: Directory, SymbolicLink, Type
Processes/Threads: DebugObject, Job, Process, Profile, Section, Session, Thread, Token
Synchronization:
  Event, EventPair, KeyedEvent, Mutant, Semaphore, ALPC Port, IoCompletion, Timer, TpWorkerFactory
IO: Adapter, Controller, Device, Driver, File, Filter*Port
Kernel Transactions: TmEn, TmRm, TmTm, TmTx
Win32 GUI: Callback, Desktop, WindowStation
System: EtwRegistration, WmiGuid
Implementation: Object Methods

Note that the methods are unrelated to actual operations on the underlying objects:

OPEN: Create/Open/Dup/Inherit handle
CLOSE: Called when each handle closed
DELETE: Called on last dereference
PARSE: Called looking up objects by name
SECURITY: Usually SeDefaultObjectMethod
QUERYNAME: Return object-specific name
Naming example

L\\

<directory>

L“Global??”

<directory>

L“C:”

<symbolic link>

\Device\HarddiskVolume1

L\\

\Device\HarddiskVolume1

<directory>

L“Device”

<directory>

L“HarddiskVolume1”

<device>

implemented by I/O manager

© Microsoft Corporation 11
Object Manager Parsing example

\Global\C:\foo\bar.txt

(device object)

implemented by I/O manager

"\foo\bar.txt"

deviceobject->ParseRoutine == IopParseDevice

Note: namespace rooted in object manager, not FS
I/O Support: \textit{IopParseDevice}

- **Trap mechanism**
  - \texttt{NtCreateFile()}
    - \texttt{ObjMgr Lookup}
      - context
    - context
      - DevObj, context
      - \texttt{Dev Stack}
      - File System
      - \texttt{File Sys}
      - \texttt{File System Fills in File object}

- user
  - \textit{Returns handle to File object}

- kernel
  - Access check
  - Security RefMon

© Microsoft Corporation
Handle Table

- Every process has a handle table
  - System Process (kernel) has global handle table
  - Data structure also used to allocate/map process/thread IDs
- NT handles reference kernel data structures
  - Mostly used from user-mode
  - Kernel mode uses handles or referenced pointers
- NT APIs use explicit handles to refer to objects
  - Simplifies cross-process operations
  - Handles can be restricted and duplicated cross-process
- Handles can be used for synchronization
  - Any dispatcher object can be waited on
  - Multiple objects can be waited by single thread
One level: (to 512 handles)
Two levels: (to 512K handles)
Three levels: (to 16M handles)

Handle Table

TableCode

D: Handle Table Pointers [32 ]

B: Handle Table Pointers [1024 ]

A: Handle Table Entries [512 ]

E: Handle Table Pointers [1024 ]

F: Handle Table Entries

C: Handle Table Entries [512 ]
Thread Manager
Kernel Layer
CPU Control-flow

Normal threads are composed of kernel-threads and user-threads, each with stack, and scheduling/environmental info

APCs (Asynchronous Procedure Calls) interrupt the execution of a thread, not a processor

DPCs (Deferred Procedure Calls) interrupt the execution of a processor, not a thread

Thread Pools and Worker threads used to spawn work as tasks and reducing thread create/delete overhead
Threads

Unit of concurrency (abstracts the CPU)
Threads created within processes
System threads created within system process (kernel)
System thread examples:

  Dedicated threads
    Lazy writer, modified page writer, balance set manager,
    mapped pager writer, other housekeeping functions

  General worker threads
    Used to move work out of context of user thread
    Must be freed before drivers unload
    Sometimes used to avoid kernel stack overflows

  Driver worker threads
    Extends pool of worker threads for heavy hitters, like file server
Thread elements

user-mode

- user-mode stack
- Thread Environment Block (TEB)
  - most interesting: thread local storage

kernel-mode

- kernel-mode stack
- KTHREAD: scheduling, synchronization, timers, APCs
- ETHREAD: timestamps, I/O list, exec locks, process link
- thread ID
- impersonation token
Context-switching Kernel VM

Three regions of kernel VM are switched
- Page tables and page directory self-map
- Hyperspace (working set lists)
- Session space

• Session space
  - ‘Session’ is a terminal services session
  - Contains data structures for kernel-level GUI
  - Only switched when processes in different TS session

• Switched kernel regions not usually needed in other processes
  - *Thread attach* is used to temporary context switch when they are
  - Savings in KVA is substantial, as these are very large data structures
Kernel Thread Attach

Allows a thread in the kernel to temporarily move to a different process’ address space

• Used heavily in Mm and Ps, e.g.
  – Used to access process page tables, working set descriptors, etc
  – PspProcessDelete() attaches before calling ObKillProcess() to close/delete handles in proper process context

• Used to access the TEB/PEB in user-mode
  – (Thread/Process Environment Blocks)
NT thread priorities

worker threads

H
I
G
H
N
O
R
M
N
O
R
M
I
D
L
E

critical

normal (dynamic)

real-time (fixed)

idle

zero thread

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
Scheduling

Windows schedules threads, not processes
- Scheduling is preemptive, priority-based, and round-robin at the highest-priority
- 16 real-time priorities above 16 normal priorities
- Scheduler tries to keep a thread on its ideal processor/node to avoid perf degradation of cache/NUMA-memory
- Threads can specify affinity mask to run only on certain processors

Each thread has a current & base priority
- Base priority initialized from process
- Non-realtime threads have priority boost/decay from base
- Boosts for GUI foreground, waking for event
- Priority decays, particularly if thread is CPU bound (running at quantum end)

Scheduler is state-driven by timer, setting thread priority, thread block/exit, etc

Priority inversions can lead to starvation
- balance manager periodically boosts non-running runnable threads
Scheduler

Kernel Thread Transition Diagram
DavePr@Microsoft.com
2003/04/06 v0.4b
Asynchronous Procedure Calls

APCs execute routine in thread context
not as general as UNIX signals
user-mode APCs run when blocked & alertable
kernel-mode APCs used extensively: timers,
notifications, swapping stacks, debugging, set thread
ctx, I/O completion, error reporting, creating &
destroying processes & threads, …

APCs generally blocked in critical sections
  e.g. don’t want thread to exit holding resources
Asynchronous Procedure Calls

APCs execute code in context of a particular thread

APCs run only at PASSIVE or APC LEVEL (0 or 1)
  Interrupted by DPCs and ISRs

Three kinds of APCs
  **User-mode**: deliver notifications, such as I/O done
  **Kernel-mode**: perform O/S work in context of a process/thread, such as completing IRPs
  **Special kernel-mode**: used for process termination
Process Manager
Memory Manager
Cache Manager
Processes

• An environment for program execution
  – Namespaces (access to files & kernel objects)
  – virtual address mappings
  – ports (debug, exceptions)
  – threads
  – user authentication (token)
  – virtual memory data structures
  – PEB (Process Environment Block) in user-mode

• In Windows, a process abstracts the MMU, not the CPU
Process Lifetime

- Process created as an empty shell
- Address space created with only system DLL and the main image (including linked DLLs)
- Handle table created empty or populated via duplication of inheritable handles from parent
- Add environment, threads, map executable image sections (EXE and DLLs)

- Process partially destroyed ("rundown") at last thread exit
- Process totally destroyed on last dereference
System DLL

Core user-mode functionality in the system dynamic link library (DLL) *ntdll.dll*

Mapped during process address space setup by the kernel

Contains all core system service entry points

User-mode trampoline points for:
- Process/thread startup
- Exception dispatch
- User APC dispatch
- Kernel-user callouts
Process/Thread structure

- Any Handle Table
- Object Manager
- Process Object
- Process’ Handle Table
- Virtual Address Descriptors
- Memory Manager Structures
- Files
- Events
- Devices
- Drivers
- Threads
Physical Frame Management

- Table of PFN (Physical Frame Number) data structures
  - Represent all pageable pages
  - Synchronize page-ins
  - Linked to management lists

- Page Tables
  - Hierarchical index of page directories and tables
  - Leaf-node is page table entry (PTE)
  - PTE states:
    - Active/valid
    - Transition
    - Modified-no-write
    - Demand zero
    - Page file
    - Mapped file
Virtual Memory Management

Working-set Manager

Working-set list

VAD tree

Sections

Image

c-o-w Data

File Data

File Data

Data

Modified List

Standby List

Free List

Modified Page Writer

App

© Microsoft Corporation
Physical Frame State Changes

- Standby List
- Modified List
- Modified Pagewriter
- Zero List
- Zero Thread
- Process (or System) Working Set

States and Transitions:
- MM Low Mem
- Standby List to Modified List: Soft fault
- Modified List to Modified Pagewriter: Trim Dirty
- Modified Pagewriter to Standby List: Trim Clean
- Standby List to Free List: MM Low Mem
- Delete page from Free List to Zero List
- Hard fault (I/O): Free List to Process (or System) Working Set
- Process (or System) Working Set to Modified List: Zero fill fault
- Process (or System) Working Set to Modified Pagewriter: Zero fill fault

© Microsoft Corporation
32b Virtual Address Translation

CR3 ➔ PD (1024 PDEs) ➔ PT (1024 PTEs) ➔ page (4096 bytes) ➔ DATA

© Microsoft Corporation 37
Self-mapping page tables

Virtual Access to PageDirectory[0x300]

Phys: PD[0xc0300000 >> 22] = PD
Virt: *((0xc0300c00) == PD

© Microsoft Corporation
Self-mapping page tables
Virtual Access to PTE for va 0xe4321000

GetPteAddress: 0xe4321000 => 0xc0390c84

CR3

PD

PT

0x300
0x390

1100 0000 0011 1001 0000 1100 1000 0100

PTE
I/O
I/O Model

- Extensible filter-based I/O model with driver layering
- Standard device models for common device classes
- Support for notifications, tracing, journaling
- Configuration store
- File caching is virtual, based on memory mapping
- Completely asynchronous model (with cancellation)
I/O Request Packet (IRP)

- Flags
- Buffer Pointers
- MDL Chain
- Thread’s IRPs
- Completion/Cancel Info
  - Completion
  - APC block
  - Driver Queuing & Comm.

**IRP Stack Locations**

System -> User

MDL
Thread

© Microsoft Corporation
Layering Drivers

Device objects attach one on top of another using IoAttachDevice* APIs creating device stacks
- I/O manager sends IRP to top of the stack
- Drivers store next lower device object in their private data structure
- Stack tear down done using IoDetachDevice and IoDeleteDevice

Device objects point to driver objects
- Driver represent driver state, including dispatch table
File System Device Stack

Application

Kernel32 / ntdll

NT I/O Manager

File System Filters

File System Driver

Partition/Volume Storage Manager

Cache Manager

Virtual Memory Manager

Disk Class Manager

Disk Driver

DISK
I/O Manager

NtCreateFile

Object Manager

ObOpenObjectName

I/O Manager

IopParseDevice

IRP

FS filter drivers

IoCallDriver

NTFS

IoCallDriver

Volume Mgr

IoCallDriver

Disk Driver

IoCallDriver

HAL

Result: *File Object*

filled in by NTFS
I/O Completions

- Receiving notification for asynchronous I/O completion:
  - poll status variable
  - wait for the file handle to be signalled
  - wait for an explicitly passed event to be signalled
  - specify a routine to be called on the originating ports
  - use an I/O completion port
I/O Completion Ports

normal completion

I/O completion ports
Questions