Lecture 14:

Case Study: Windows NT

www.cl.cam.ac.uk/Teaching/2001/OSFounds/

Lecture 14: Monday 5th November 2001

Today's Lecture

Today we'll cover:

- Case Study: Windows NT
  - How is it structured?
  - Processes (and threads),
  - Virtual memory,
  - I/O and File management.

Windows NT: History

After OS/2, MS decide they need "**N**ew **T**echnology":

- 1988: Dave Cutler recruited from DEC.
- 1989: team ($\sim$ 10 people) starts work on a new OS with a micro-kernel architecture.
- July 1993: first version (3.1) introduced

Bloated and suckful $\Rightarrow$

- NT 3.5 released in September 1994: mainly size and performance optimisations.
- Followed in May 1995 by NT 3.51 (support for the Power PC, and more performance tweaks)
- July 1996: NT 4.0
  - new (windows 95) look 'n feel
  - various functions pushed back into kernel (most notably graphics rendering functions)
- Feb 2000: NT 5.0 aka Windows 2000
  - big push to finally kill DOS/Win 9x family

Windows XP (NT 6.0) coming June 2001. . .

NT Design Principles

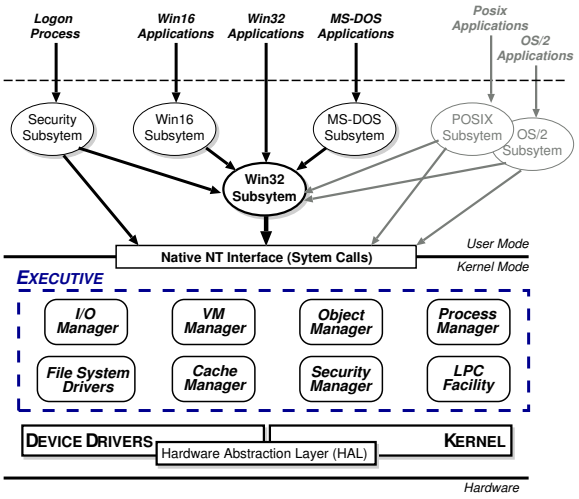
Key goals for the system were:

- Portability
- Security
- POSIX compliance
- Multiprocessor support
- Extensibility
- International support
- Compatibility with MS-DOS/Windows applications

This led to the development of a system which was:

- written in high-level languages (C and C++)
- based around a micro-kernel, and
- constructed in a layered/modular fashion.

Structural Overview



- Kernel Mode: HAL, Kernel, & Executive
- User Mode:
  - environmental subsystems
  - protection subsystem

HAL

- Layer of software (`HAL.DLL`) which hides details of underlying hardware

  e.g. interrupt mechanisms, DMA controllers, multiprocessor communication mechanisms
- Many HALs exist with same *interface* but different *implementation* (often vendor-specific)

Kernel

- Foundation for the executive and the subsystems
- Execution is **never** preempted.
- Four main responsibilities:
  1. CPU scheduling
  2. Interrupt and exception handling
  3. Low-level processor synchronisation
  4. Recovery after a power failure
- Kernel is **objected-oriented**; all objects either **dispatcher objects** or **control objects**

Processes and Threads

NT splits the "virtual processor" into two parts:

1. A **process** is the unit of resource ownership.
   Each process has:
   - a security token,
   - a virtual address space,
   - a set of resources (**object handles**), and
   - one or more *threads*.
2. A **thread** are the unit of dispatching.
   Each thread has:
   - a scheduling state (ready, running, etc.),
   - other scheduling parameters (priority, etc),
   - a context slot, and
   - (generally) an associated process.

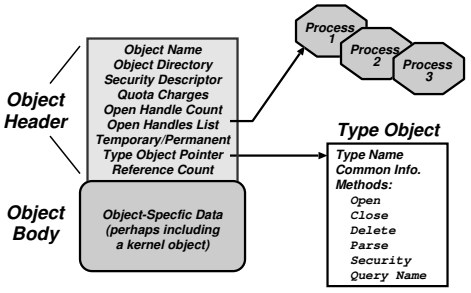
Threads are:

- co-operative: all threads in a process share the same address space & object handles.
- lightweight: require less work to create/delete than processes (mainly due to shared VAS).

CPU Scheduling

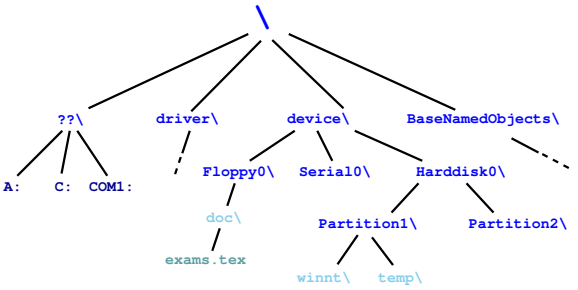
- Hybrid static/dynamic priority scheduling:
  - Priorities 16–31: "real time" (static priority).
  - Priorities 1–15: "variable" (dynamic) priority.
- Default quantum 2 ticks (∼20ms) on Workstation, 12 ticks (∼120ms) on Server.
- Threads have **base** and **current** ($\geq$ base) priorities.
  - On return from I/O, current priority is *boosted* by driver-specific amount.
  - Subsequently, current priority decays by 1 after each completed quantum.
  - Also get boost for GUI threads awaiting input: current priority boosted to 14 for one quantum (but quantum also doubled)
  - Yes, this is true.
- On Workstation also get quantum stretching:
  - ". . . performance boost for the foreground application" (window with focus)
  - fg thread gets double or triple quantum.

Object Manager



- Every resource in NT is represented by an **object**
- The **Object Manager** (part of the Executive) is responsible for:
  - creating objects and **object handles**
  - Performing security checks
  - Tracking which processes are using each object
- Typical operation:
  - `handle = open(objectname, accessmode)`
  - `result = service(handle, arguments)`

Object Namespace



- Objects (optionally) have a **name**
- Object Manger manages a hierarchical namespace:
  - shared between all processes ⇒ sharing
  - implemented via directory objects
  - each object protected by an access control list
  - naming domains (implemented via `parse`) mean file-system namespaces can be integrated
- Also get symbolic link objects: allow multiple names (aliases) for the same object.
- Modified view presented at API level. . .

Process Manager

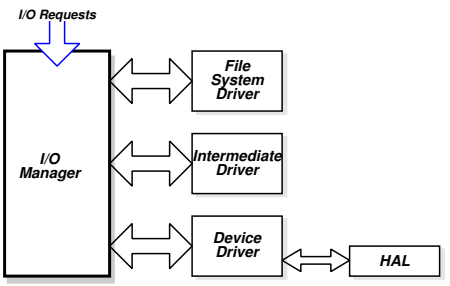
- Provides services for creating, deleting, and using threads and processes.
- Very flexible:
  - no built in concept of parent/child relationships or process hierarchies
  - processes and threads treated orthogonally.

⇒ can support Posix, OS/2 and Win32 models.

Virtual Memory Manager

- NT employs **paged virtual memory management**
- The VMM provides processes with services to:
  - allocate and free virtual memory
  - modify per-page protections
- Can also share portions of memory:
  - use **section objects** (≈ software segments)
  - based verus non-based.
  - also used for **memory-mapped files**

I/O Manager



- The I/O Manager is responsible for:
  - file systems
  - cache management
  - device drivers
- Basic model is **asynchronous**:
  - each I/O operation explicitly split into a request and a response
  - **I/O Request Packet** (IRP) used to hold parameters, results, etc.
- File-system & device drivers are **stackable**. . .

File System

- The fundamental structure of the NT filing system (NTFS) is a **volume**
  - created by the NT disk administrator utility
  - based on a logical disk partition
  - may occupy a portion of a disk, and entire disk, or span across several disks.
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a **structured** object consisting of **attributes**.
- Every file in NTFS is described by one or more records in an array stored in a special file called the **Master File Table** (MFT).
- NTFS has a number of advanced features, e.g.
  - security (access checks on open)
  - unicode based names
  - use of a log for efficient recovery
  - support for sparse and compressed files

  (but only recently are features being used)

Summary

- Main Windows NT features are:
  - layered/modular architecture:
  - generic use of objects throughout
  - multi-threaded processes
  - multiprocessor support
  - asynchronous I/O subsystem
  - advanced filing system
  - preemptive priority-based scheduling
- Design essentially more advanced than Unix.
- Implementation of lower levels (HAL, kernel & executive) actually rather decent.
- But: has historically been crippled by
  - almost exclusive use of Win32 API
  - legacy device drivers (e.g. VXDs)
  - lack of demand for "advanced" features
- Windows XP + Luna might finally break free. . .

Next lecture: Last year's exam questions!

Background Reading:

- Solomon – "Inside Windows NT (2nd Edition)"
- Silberschatz et al.: – Chapter 21 (Win2K)

Course Review

- **Part I: Computer Organisation**
  - "how does a computer work?"
  - fetch-execute cycle, data representation, etc
  - NB: 'circuit diagrams' *not* examinable
- **Part II: Operating System Functions**
  - OS structures: h/w support, kernel vs. μ-kernel
  - Processes: states, structures, scheduling
  - Memory: virtual addresses, sharing, protection
  - I/O subsytem: polling/interrupts, buffering.
  - Filing: directories, meta-data, file operations.
- **Part III: Case Studies**
  - Unix: file abstraction, command 'extensibility'
  - Windows NT: layering, objects, asynch. I/O.