Overview

Objective:
• To explore the principles upon which Windows 7 is designed and the specific components involved in the system.
• To understand how Windows 7 can run programs for other operating systems.
• To cover the interface available to system and application programmers.

• History
• Design Principles
• System Components
• Programmer Interface
Overview

• History
• Design Principles
• System Components
• Programmer Interface
History

• 32-bit preemptive multitasking operating system for Intel microprocessors

• Key goals for the system:
  • portability
  • security
  • POSIX compliance
  • multiprocessor support
  • extensibility
  • international support
  • compatibility with MS-DOS and MS-Windows applications.

• Uses a micro-kernel architecture

• Available in six client versions, Starter, Home Basic, Home Premium, Professional, Enterprise and Ultimate. With the exception of Starter edition (32-bit only) all are available in both 32-bit and 64-bit.
History

• In 1988, Microsoft decided to develop a “new technology” (NT) portable operating system that supported both the OS/2 and POSIX APIs.

• Originally, NT was supposed to use the OS/2 API as its native environment but during development NT was changed to use the Win32 API, reflecting the popularity of Windows 3.0.
Overview

• History
• **Design Principles**
• System Components
• Programmer Interface
Design Principles

• **Extensibility** — layered architecture
  • **Executive**, which runs in protected mode, provides the basic system services
  • On top of the executive, several server subsystems operate in user mode
  • **Modular structure** allows additional environmental subsystems to be added without affecting the executive

• **Portability** — Windows 7 can be moved from one hardware architecture to another with relatively few changes
  • Written in C and C++
  • Processor-specific portions are written in assembly language for a given processor architecture (small amount of such code).
  • Platform-dependent code is isolated in a **dynamic link library** (DLL) called the “**hardware abstraction layer**” (HAL)
Design Principles

- **Reliability** — Windows 7 uses hardware protection for virtual memory, and software protection mechanisms for operating system resources

- **Compatibility** — applications that follow the IEEE 1003.1 (POSIX) standard can be complied to run on 7 without changing the source code

- **Performance** — Windows 7 subsystems can communicate with one another via high-performance message passing
  - Preemption of low priority threads enables the system to respond quickly to external events
  - Designed for symmetrical multiprocessing

- **International support** — supports different locales via the national language support (NLS) API
Overview

• History
• Design Principles
• **System Components**
• Programmer Interface
System Components: Windows 7 Architecture

• Layered system of module
• Protected mode — hardware abstraction layer (HAL), kernel, executive
• User mode — collection of subsystems
  • Environmental subsystems emulate different operating systems
  • Protection subsystems provide security functions
System Components: Windows 7 Architecture
System Components: Kernel

• Foundation for the executive and the subsystems
• Never paged out of memory; execution is never preempted

• Four main responsibilities:
  • thread scheduling
  • interrupt and exception handling
  • low-level processor synchronization
  • recovery after a power failure

• Kernel is object-oriented, uses two sets of objects
  • dispatcher objects control dispatching and synchronization (events, mutants, mutexes, semaphores, threads and timers)
  • control objects (asynchronous procedure calls, interrupts, power notify, power status, process and profile objects)
System Components: Process and Threads

- The process has a virtual memory address space, information (such as a base priority), and an affinity for one or more processors.
- Threads are the unit of execution scheduled by the kernel’s dispatcher.
- Each thread has its own state, including a priority, processor affinity, and accounting information.
- A thread can be one of six states: ready, standby, running, waiting, transition, and terminated.
The dispatcher uses a 32-level priority scheme to determine the order of thread execution.

- Priorities are divided into two classes
  - The real-time class contains threads with priorities ranging from 16 to 31
  - The variable class contains threads having priorities from 0 to 15

- Characteristics of Windows 7’s priority strategy
  - Tends to give very good response times to interactive threads that are using the mouse and windows
  - Enables I/O-bound threads to keep the I/O devices busy
System Components: Scheduling (2/2)

- **Scheduling can occur** when a thread enters the ready or wait state, when a thread terminates, or when an application changes a thread’s priority or processor affinity.

- Real-time threads are given preferential access to the CPU; but 7 does not guarantee that a real-time thread will start to execute within any particular time limit.
  - This is known as **soft realtime**.
## System Components: Windows 7 Interrupt Request Levels

<table>
<thead>
<tr>
<th>interrupt levels</th>
<th>types of interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>machine check or bus error</td>
</tr>
<tr>
<td>30</td>
<td>power fail</td>
</tr>
<tr>
<td>29</td>
<td>interprocessor notification (request another processor to act; e.g., dispatch a process or update the TLB)</td>
</tr>
<tr>
<td>28</td>
<td>clock (used to keep track of time)</td>
</tr>
<tr>
<td>27</td>
<td>profile</td>
</tr>
<tr>
<td>3–26</td>
<td>traditional PC IRQ hardware interrupts</td>
</tr>
<tr>
<td>2</td>
<td>dispatch and deferred procedure call (DPC) (kernel)</td>
</tr>
<tr>
<td>1</td>
<td>asynchronous procedure call (APC)</td>
</tr>
<tr>
<td>0</td>
<td>passive</td>
</tr>
</tbody>
</table>
System Components: Kernel – Trap Handling

- The kernel provides *trap handling* when exceptions and interrupts are generated by hardware or software.
- Exceptions that cannot be handled by the trap handler are handled by the kernel's *exception dispatcher*.
- The interrupt dispatcher in the kernel handles interrupts by calling either an *interrupt service routine* (such as in a device driver) or an *internal kernel routine*.
- The kernel uses spin locks that reside in global memory to achieve multiprocessor mutual exclusion.
System Components: Executive – Object Manager

- Windows 7 uses objects for all its services and entities; the object manager supervises the use of all the objects
  - Generates an object handle
  - Checks security
  - Keeps track of which processes are using each object
- Objects are manipulated by a standard set of methods, namely create, open, close, delete, query name, parse and security.
System Components: Executive – Naming Objects

- The Windows 7 executive allows almost any object to be given a name, which may be either permanent or temporary. Exceptions are process, thread and some others object types.
- Object names are structured like file path names in MS-DOS and UNIX.
- Windows 7 implements a symbolic link object, which is similar to symbolic links in UNIX that allow multiple nicknames or aliases to refer to the same file.
- A process gets an object handle by creating an object, by opening an existing one, by receiving a duplicated handle from another process, or by inheriting a handle from a parent process.
System Components: Executive - Virtual Memory Manager

• The design of the VM manager assumes that the underlying hardware supports virtual to physical mapping a paging mechanism, transparent cache coherence on multiprocessor systems, and virtual addressing aliasing.

• The VM manager in Windows 7 uses a page-based management scheme with a page size of 4 KB.

• The Windows 7 VM manager uses a two step process to allocate memory
  • The first step reserves a portion of the process’s address space
  • The second step commits the allocation by assigning space in the system’s paging file(s)
System Components: Executive – Virtual Memory Layout
System Components:
Virtual Memory Manager

• The virtual address translation in Windows 7 uses several data structures
  • Each process has a page directory that contains 1024 page directory entries of size 4 bytes.
  • Each page directory entry points to a page table which contains 1024 page table entries (PTEs) of size 4 bytes.
  • Each PTE points to a 4 KB page frame in physical memory.

• A 10-bit integer can represent all the values from 0 to 1023, therefore, can select any entry in the page directory, or in a page table.

• This property is used when translating a virtual address pointer to a byte address in physical memory.

• A page can be in one of six states: valid, zeroed, free, standby, modified and bad.
System Components: Virtual-to-Physical Address Translation

10 bits for page directory entry, 20 bits for page table entry, and 12 bits for byte offset in page
System Components: Page File Page-Table Entry

5 bits for page protection,
20 bits for page frame address,
4 bits to select a paging file,
3 bits that describe the page state.
V = 0
System Components: Executive – Process Manager

• Provides services for creating, deleting, and using threads and processes

• Issues such as parent/child relationships or process hierarchies are left to the particular environmental subsystem that owns the process.
System Components: Executive – Local Procedure Call Facility

- The **LPC** passes requests and results between client and server processes within a single machine.
- In particular, it is used to request services from the various Windows 7 subsystems.
- When a LPC channel is created, one of three types of message passing techniques must be specified.
  - First type is **suitable for small messages**, up to 256 bytes; port's message queue is used as intermediate storage, and the messages are copied from one process to the other.
  - Second type **avoids copying large messages** by pointing to a shared memory section object created for the channel.
  - Third method, called **quick LPC** was used by graphical display portions of the Win32 subsystem.
Overview

• History
• Design Principles
• System Components
• Programmer Interface
Programmer Interface: Access to Kernel Object

• A process gains access to a kernel object named XXX by calling the `CreateXXX` function to open a handle to XXX; the handle is unique to that process.

• A handle can be closed by calling the `CloseHandle` function; the system may delete the object if the count of processes using the object drops to 0.

• Windows 7 provides three ways to share objects between processes
  • A child process inherits a handle to the object
  • One process gives the object a name when it is created and the second process opens that name
  • `DuplicateHandle` function:
    • Given a handle to process and the handle’s value a second process can get a handle to the same object, and thus share it
Programmer Interface: 
Process Management

• Process is started via the CreateProcess routine which loads any dynamic link libraries that are used by the process, and creates a primary thread.

• Additional threads can be created by the CreateThread function.

• Every dynamic link library or executable file that is loaded into the address space of a process is identified by an instance handle.
Programmer Interface: Process Management

• Scheduling in Win32 utilizes four priority classes:
  1. IDLE_PRIORITY_CLASS (priority level 4)
  2. NORMAL_PRIORITY_CLASS (level 8 — typical for most processes)
  3. HIGH_PRIORITY_CLASS (level 13)
  4. REALTIME_PRIORITY_CLASS (level 24)

• To provide performance levels needed for interactive programs, 7 has a special scheduling rule for processes in the NORMAL_PRIORITY_CLASS
  • 7 distinguishes between the foreground process that is currently selected on the screen, and the background processes that are not currently selected.
  • When a process moves into the foreground, 7 increases the scheduling quantum by some factor, typically 3.
Programmer Interface: Process Management

• The kernel *dynamically adjusts the priority of a thread* depending on whether it is I/O-bound or CPU-bound.

• To synchronize the concurrent access to shared objects by threads, the kernel provides synchronization objects, such as semaphores and mutexes
  
  • In addition, threads can synchronize by using the `WaitForSingleObject` or `WaitForMultipleObjects` functions.

  • Another method of synchronization in the Win32 API is the critical section.
Programmer Interface: Process Management

- A fiber is user-mode code that gets scheduled according to a user-defined scheduling algorithm.
  - Only one fiber at a time is permitted to execute, even on multiprocessor hardware.
  - Windows 7 includes fibers to facilitate the porting of legacy UNIX applications that are written for a fiber execution model.
- Windows 7 also introduced user-mode scheduling for 64-bit systems which allows finer grained control of scheduling work without requiring kernel transitions.
Programmer Interface: Interprocess Communication

- Win32 applications can have interprocess communication by sharing kernel objects.
- An alternate means of interprocess communications is message passing, which is particularly popular for Windows GUI applications.
- Every Win32 thread has its own input queue from which the thread receives messages.
Programmer Interface: Memory Management

- Virtual memory:
  - `VirtualAlloc` reserves or commits virtual memory
  - `VirtualFree` decommits or releases the memory
  - These functions enable the application to determine the virtual address at which the memory is allocated

- An application can use memory by memory mapping a file into its address space
  - Multistage process
  - Two processes share memory by mapping the same file into their virtual memory
Programmer Interface: Memory Management

• A heap in the Win32 environment is a region of reserved address space
  • A Win 32 process is created with a 1 MB default heap
  • Access is synchronized to protect the heap’s space allocation data structures from damage by concurrent updates by multiple threads

• Because functions that rely on global or static data typically fail to work properly in a multithreaded environment, the thread-local storage mechanism allocates global storage on a per-thread basis
  • The mechanism provides both dynamic and static methods of creating thread-local storage
Credits for slides

Silberschatz, Galvin and Gagne