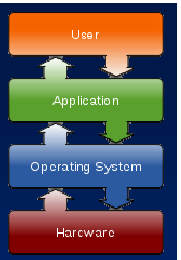
ENCE360 Operating Systems



Linux versus Windows

Philosophical differences in the design of Linux and Windows

- The Windows operating system is designed to support applications by moving more functionality into the operating system, and by more deeply integrating applications into the Windows kernel.
- *Linux differs from Windows in providing a clear separation between kernel space and user space*

System Requirements

Windows 7

- 1 GHz 32-bit or 64-bit processor
- 1 GB of system memory
- 16 GB of available disk space
- Support for DirectX 9 graphics with 128 MB memory (to enable the Aero theme)
- DVD-R/W Drive
- Internet access (to download the Beta and get updates)

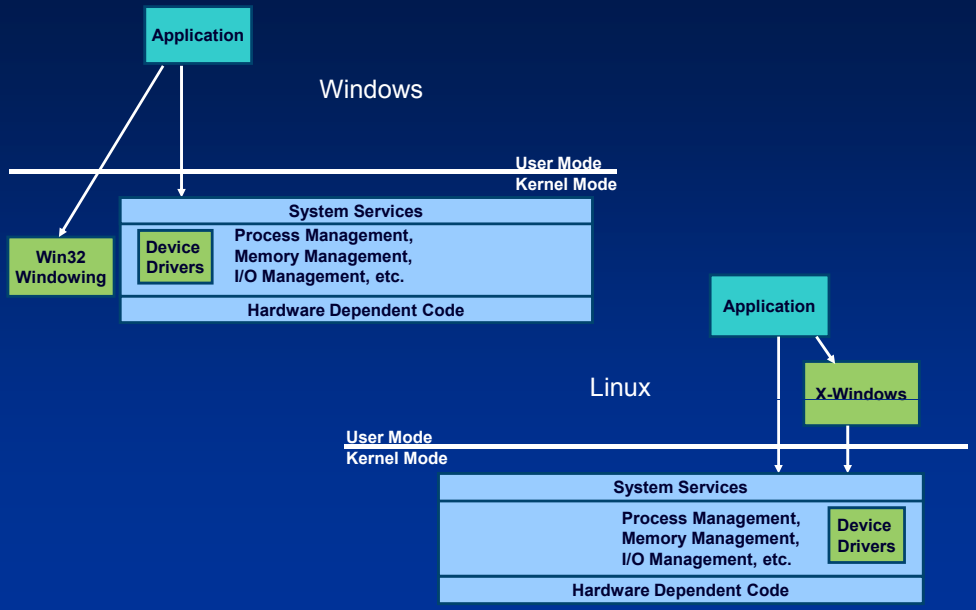
Ubuntu

- PC i386
- 256 MB
- 4 GB RAM
- New Release every six months. Each release supported for 18 months
- Software Free

Comparing the Architectures

- Both Linux and Windows are monolithic
  - All core operating system services run in a shared address space in kernel-mode
  - All core operating system services are part of a single module
    - Linux: vmlinuz
    - Windows: ntoskrnl.exe
- Windowing is handled differently:
  - Windows has a kernel-mode Windowing subsystem
  - Linux has a user-mode X-Windowing system

Kernel Architectures



Linux Kernel

- Linux is a monolithic but modular system
  - All kernel subsystems form a single piece of code with no protection between them
- Modularity is supported in two ways:
  - Compile-time options
  - Most kernel components can be built as a dynamically loadable kernel module (DLKM)
- DLKMs
  - Built separately from the main kernel
  - Loaded into the kernel at runtime and on demand (infrequently used components take up kernel memory only when needed)
  - Kernel modules can be upgraded incrementally
  - Support for minimal kernels that automatically adapt to the machine and load only those kernel components that are used

Windows Kernel

- Windows is a monolithic but modular system
  - No protection among pieces of kernel code and drivers
- Support for Modularity is somewhat weak:
  - Windows Drivers allow for dynamic extension of kernel functionality
  - Windows has special tools / packaging rules that allow coarse-grained configuration of the OS
- Windows Drivers are dynamically loadable kernel modules
  - Significant amount of code run as drivers (including network stacks such as TCP/IP and many services)
  - Built independently from the kernel
  - Can be loaded on-demand
  - Dependencies among drivers can be specified

Comparing Portability

- Both Linux and Windows kernels are portable
  - Mainly written in C
  - Have been ported to a range of processor architectures
- Windows
  - i486, MIPS, PowerPC, Alpha, IA-64, x86-64, etc
  - > 64MB memory required
- Linux
  - Alpha, ARM, ARM26, CRIS, H8300, i386, IA-64, M68000, MIPS, PA-RISC, PowerPC, S/390, SuperH, SPARC, VAX, v850, x86-64, etc
  - DLKMs allow for minimal kernels for microcontrollers
  - > 4MB memory required

Comparing Layering, APIs, Complexity

- Windows
  - Kernel exports about 250 system calls (accessed via ntdll.dll)
  - Layered Windows/POSIX subsystems
  - Rich Windows API (17 500 functions on top of native APIs)
- Linux
  - Kernel supports about 200 different system calls
  - Layered BSD, Unix Sys V, POSIX shared system libraries
  - Compact APIs (1742 functions in Single Unix Specification Version 3; not including X Window APIs)

Comparing Architectures

- Processes and scheduling
- SMP support
- Memory management
- I/O
- File Caching
- Security

Process Management

Process Management

Windows

- Process
  - Address space, handle table, statistics and at least one thread
  - No inherent parent/child relationship
- Threads
  - Basic scheduling unit
  - Fibers - cooperative user-mode threads

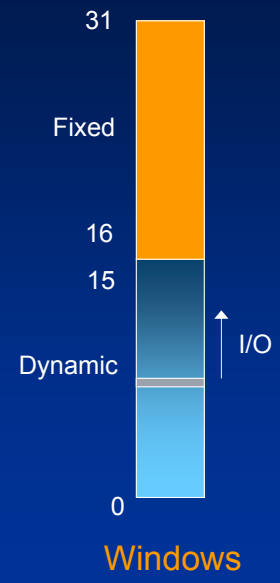
Linux

- Process is called a Task
  - Basic Address space, handle table, statistics
  - Parent/child relationship
  - Basic scheduling unit
- Threads
  - No threads per-se
  - Tasks can act like Windows threads by sharing handle table, PID and address space
  - PThreads – cooperative user-mode threads

Scheduling Priorities

Windows

- Two scheduling classes
  - "Real time" (fixed) - priority 16-31
  - Dynamic - priority 1-15
- Higher priorities are favored
  - Priorities of dynamic threads get boosted on wakeups
  - Thread priorities are never lowered



Windows

Scheduling Priorities

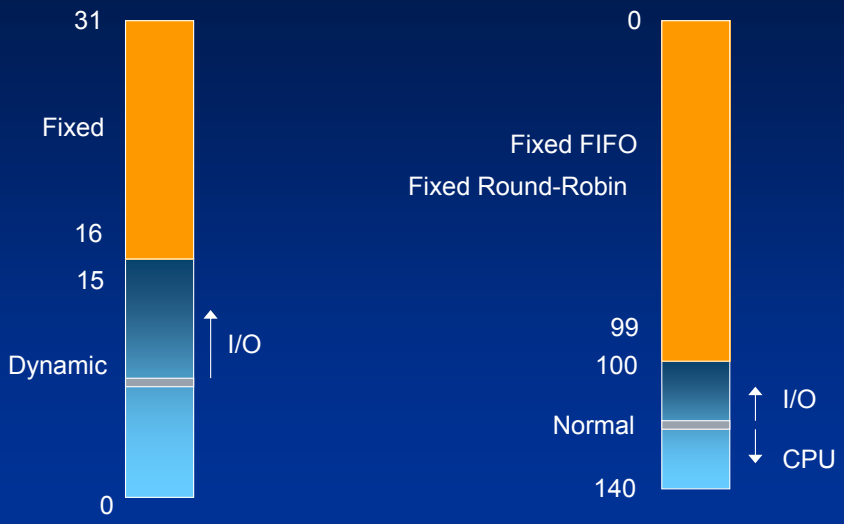
Windows

- Two scheduling classes
  - "Real time" (fixed) - priority 16-31
  - Dynamic - priority 1-15
- Higher priorities are favored
  - Priorities of dynamic threads get boosted on wakeups
  - Thread priorities are never lowered

Linux

- Has 3 scheduling classes:
  - Normal – priority 100-139
  - Fixed Round Robin – priority 0-99
  - Fixed FIFO – priority 0-99
- Lower priorities are favored
  - Priorities of normal threads go up (decay) as they use CPU
  - Priorities of interactive threads go down (boost)

Scheduling Priorities (cont)



Linux Scheduling Details

- Most threads use a dynamic priority policy
  - Normal class - similar to the classic UNIX scheduler
  - A newly created thread starts with a base priority
  - Threads that block frequently (I/O bound) will have their priority gradually increased
  - Threads that always exhaust their time slice (CPU bound) will have their priority gradually decreased
- "Nice value" sets a thread's base priority
  - Larger values = less priority, lower values = higher priority
  - Valid nice values are in the range of -20 to +20
  - Nonprivileged users can only specify positive nice value
- Dynamic priority policy threads have static priority zero
  - Execute only when there are no runnable real-time threads

Real-Time Scheduling on Linux

- Linux supports two static priority scheduling policies:
  - Round-robin and FIFO (first in, first out)
    - Selected with the sched-setscheduler( ) system call
    - Use static priority values in the range of 1 to 99
    - Executed strictly in order of decreasing static priority
  - FIFO policy lets a thread run to completion
    - Thread needs to indicate completion by calling the sched-yield( )
  - Round-robin lets threads run for up to one time slice
    - Then switches to the next thread with the same static priority
  - RT threads can easily starve lower-prio threads from executing
    - Root privileges or the CAP-SYS-NICE capability are required for the selection of a real-time scheduling policy
- Long running system calls can cause priority-inversion
  - Same as in Windows; but cmp. rtLinux

Windows Scheduling Details

- Most threads run in variable priority levels
  - Priorities 1-15;
  - A newly created thread starts with a base priority
  - Threads that complete I/O operations experience priority boosts (but never higher than 15)
  - A thread's priority will never be below base priority
- The Windows API function SetThreadPriority() sets the priority value for a specified thread
  - This value, together with the priority class of the thread's process, determines the thread's base priority level
  - Windows will dynamically adjust priorities for non-realtime threads

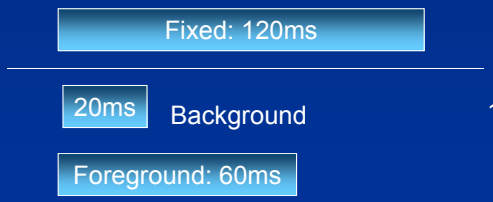
Real-Time Scheduling on Windows

- Windows supports static round-robin scheduling policy for threads with priorities in real-time range (16-31)
  - Threads run for up to one quantum
  - Quantum is reset to full turn on preemption
  - Priorities never get boosted
- RT threads can starve important system services
  - Such as CSRSS.EXE
  - SeIncreaseBasePriorityPrivilege required to elevate a thread's priority into real-time range (this privilege is assigned to members of Administrators group)
- System calls and DPC/APC handling can cause priority inversion

Scheduling Timeslices

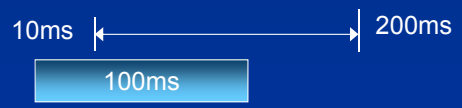
Windows

- The thread timeslice (quantum) is 10ms-120ms
  - When quanta can vary, has one of 2 values
- Reentrant and preemptible



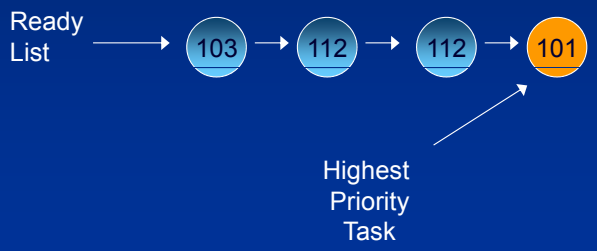
Linux

- The thread quantum is 10ms-200ms
  - Default is 100ms
  - Varies across entire range based on priority, which is based on interactivity level
- Reentrant and preemptible



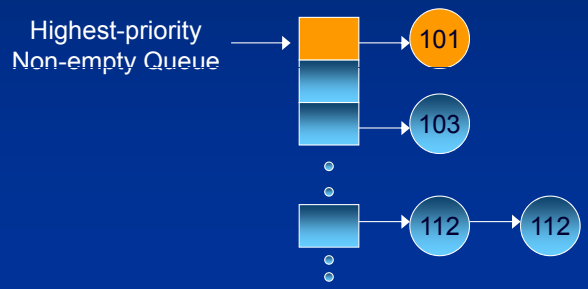
Scheduling

- The Linux scheduler is O(n)
  - If there are 10 active tasks, it scans 10 of them in a list in order to decide which should execute next
  - This means long scans and long durations under the scheduler lock



Scheduling

- Linux has a scheduler that's O(1) from Ingo Molnar that:
  - Calculates a task's priority at the time it makes scheduling decision
  - Has per-CPU ready queues where the tasks are pre-sorted by priority

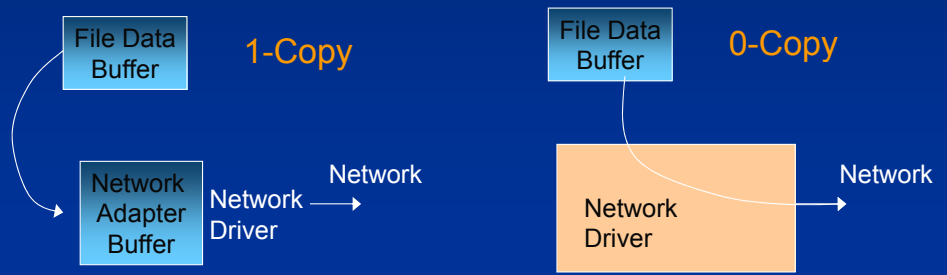


Scheduling

- Windows has an O(1) scheduler based on pre-sorted thread priority queues
- Per-CPU ready queues
  - Linux load balances queues
  - Windows does not
    - Not seen as an issue in performance testing by Microsoft
    - Applications where it might be an issue are expected to use affinity

Zero-Copy Sendfile

- Linux efficiently sends file data over a socket - introduced zero-copy Sendfile
- Windows pioneered zero-copy file sending with TransmitFile, the Sendfile equivalent in Windows



Light-Weight Synchronization

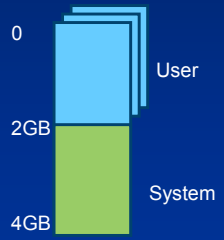
- Linux has Futexes
  - There's only a transition to kernel-mode when there's contention
- Windows has CriticalSections
  - Same behavior
- Futexes go further:
  - Allow for prioritization of waits
  - Works interprocess as well

Memory Management

Virtual Memory Management

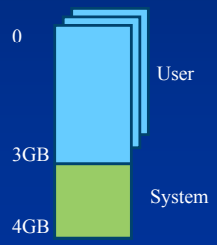
Windows

- 32-bit versions split user-mode/kernel-mode from 2GB/2GB to 3GB/1GB
- Demand-paged virtual memory
  - 32 or 64-bits
  - Copy-on-write
  - Shared memory
  - Memory mapped files



Linux

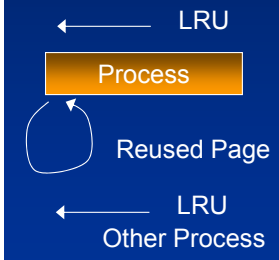
- Splits user-mode/kernel-mode from 1GB/3GB to 3GB/1GB
  - 2.6 has "4/4 split" option where kernel has its own address space
- Demand-paged virtual memory
  - 32-bits and/or 64-bits
  - Copy-on-write
  - Shared memory
  - Memory mapped files



Physical Memory Management

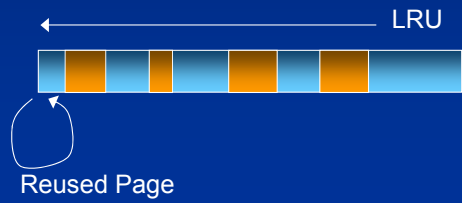
Windows

- Per-process working sets
  - Working set tuner adjust sets according to memory needs using the "clock" algorithm
- No "swapper"



Linux

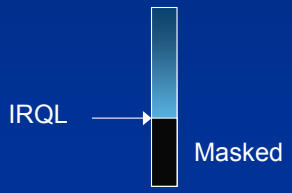
- Global working set management uses "clock" algorithm
- No "swapper" (the working set trimmer code is called the swap daemon, however)



I/O Management

Windows

- Centered around the file object
- Layered driver architecture throughout driver types
- Most I/O supports asynchronous operation
- Internal interrupt request level (IRQL) controls interruptability
- Interrupts are split between an Interrupt Service Routine (ISR) and a Deferred Procedure Call (DPC)
- Supports plug-and-play



Linux

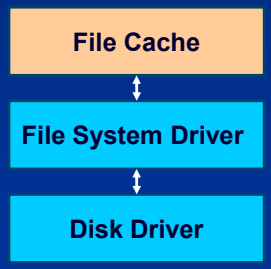
- Centered around the vnode
- No layered I/O model
- Most I/O is synchronous
  - Only sockets and direct disk I/O support asynchronous I/O
- Internal interrupt request level (IRQL) controls interruptability
- Interrupts are split between an ISR and soft IRQ or tasklet
- Supports plug-and-play

I/O & File System Management

File Caching

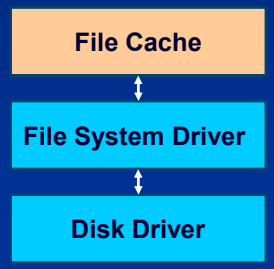
Windows

- Single global common cache
- Virtual file cache
  - Caching is at file vs. disk block level
  - Files are memory mapped into kernel memory
- Cache allows for zero-copy file serving



Linux

- Single global common cache
- Virtual file cache
  - Caching is at file vs. disk block level
  - Files are memory mapped into kernel memory
- Cache allows for zero-copy file serving



Monitoring - Linux procfs

- Linux supports a number of special filesystems
  - Like special files, they are of a more dynamic nature and tend to have side effects when accessed
- Prime example is procfs (mounted at /proc)
  - provides access to and control over various aspects of Linux (I.e.; scheduling and memory management)
    - /proc/meminfo contains detailed statistics on the current memory usage of Linux
    - Content changes as memory usage changes over time
- Services for Unix implements procfs on Windows

I/O Processing

- Linux 2.2 had the notion of bottom halves (BH) for low-priority interrupt processing
  - Fixed number of BHs
  - Only one BH of a given type could be active on a SMP
- Linux 2.4 introduced *tasklets*, which are non-preemptible procedures called with interrupts enabled
- Tasklets are the equivalent of Windows Deferred Procedure Calls (DPCs)

Asynchronous I/O

- Linux 2.2 only supported asynchronous I/O on socket connect operations and tty's
- Linux 2.6 added asynchronous I/O for direct-disk access
  - AIO model includes efficient management of asynchronous I/O
    - Also added alternate epoll model
  - Useful for database servers managing their database on a dedicated raw partition
  - Database servers that manage a file-based database suffer from synchronous I/O
- Windows I/O is inherently asynchronous
- Windows has had completion ports since NT 3.5
  - More advanced form of AIO

Security

Security

Windows

- Very flexible security model based on Access Control Lists
- Users are defined with
  - Privileges
  - Member groups
- Security can be applied to any Object Manager object
  - Files, processes, synchronization objects, …
- Supports auditing

Linux

- Two models:
  - Standard UNIX model
  - Access Control Lists (SELinux)
- Users are defined with:
  - Capabilities (privileges)
  - Member groups
- Security is implemented on an object-by-object basis
- Had no built-in auditing support
- Version 2.6 included Linux Security Module framework for add-on security models

Further Reading

- Transaction Processing Council: www.tpc.org
- SPEC: www.spec.org
- NT vs Linux benchmarks: www.kegel.com/nt-linux-benchmarks.html
- The C10K problem: http://www.kegel.com/c10k.html
- Linus Torvald's home: http://www.osdl.org/
- Linux Kernel Archives: http://www.kernel.org/
- Linux history: http://www.firstmonday.dk/issues/issue5_11/moon/
- Veritest Netbench result: http://www.veritest.com/clients/reports/microsoft/ms_netbench.pdf
- The Open Group's Single UNIX Specification: http://www.unix.org/version3/