OS Tech Report

Windows 7

Matthew Allen

The operating system that I have chosen is the, objectively, best operating system to be released in all of mankind: Windows 7. Yes, this is objective, you would think it's subjective, but I can't have my mind changed; therefore, this is objective. 😊

Windows 7 was created by Microsoft and support began on October 22, 2009[1]. Sadly, support ended January 14, 2020—just two months before COVID-19 shut down the world. Coincidence? I think not.

Windows 7's kernel type is a hybrid kernel that combines monolithic kernels and microkernels[2]. Monolithic kernels are where operating systems work entirely in kernel space[4]. Microkernels are the near-minimum amount of software used to provide an operating system[5]. The mechanisms for this type of operating system include low-level address space management, thread management, and inter-process communication[5]. Microsoft's Windows NT kernel—the first hybrid kernel, was designed by Richard Rashid at Carnegie Mellon University[3].

Process/Thread Management:

Windows 7 came with a bunch of new functions regarding threading. 64-bit versions of Windows 7 will support more than 64 logical processors on one computer[7]. Each process contains the resources needed to execute a program. It has a virtual address space, executable code, system objects, security context, unique identifier, environment variables, priority class, minimum and maximum set sizes, and at least one thread of execution[8]. A process starts with one primary thread but can support multiple. All threads share the process' virtual address space and system resources[8]. Additionally, a thread maintains exception handlers, scheduling priority, local storage, a unique identifier, machine registers, kernel stack, TCB, user stack, and their own

security context. Windows 7 also supports fibers which is a unit of execution that must be manually scheduled by the application[8]. A thread can schedule multiple fibers. There are no advantages to fibers over a well-programmed multithreaded application[8], but they are useful for porting applications designed to schedule their own threads[8]. Windows 7 supports preemptive multitasking[8] which is what we learned in class when it comes to multithreading. Multiple threads will be executing on all possible processors on the computer. Windows 7 uses a thread pool to reduce the number of application threads and provide management of worker threads[8]. Threads can be queued, wait, and be automatically queued to work on a timer and bind with I/O[8]. Threading in Windows 7 is practically the same as what we learned in class.

Scheduling:

Windows 7 supports a priority scheduler where threads run based on their priority. 0 is the lowest priority a thread can have while 31 is the highest priority a thread can have[9]. All threads that have the same priority are scheduled in a round-robin scheduler with the highest priority. If the threads with the highest priority are not ready to run, the next highest priority threads get assigned a time slice to run in round-robin[9]. If a higher priority thread is ready to run, it will preempt the currently running thread, not allowing it to finish, and will then run the higher priority thread. Thread priority is determined by the priority class of its process and the priority level of the thread within the priority class of its process[9]. If the thread has the priority level THREAD_PRIORITY_IDLE, its base priority is 1. If the thread has the priority level THREAD_PRIORITY_TIME_CRITICAL, its base priority is 31. The base priority is a combination of its priority class and thread priority. So, each class has a THREAD_PRIORITY_IDLE with a base priority of 1. The THREAD_PRIORITY_TIME_CRITICAL is a member of the REALTIME_PRIORITY_CLASS[9]. The only thread that can have 0 priority is the zero-page thread. This is a system thread responsible for zeroing any free pages when there are no other threads needing to run[9].

The scheduler maintains a queue of ready threads based on priority. Window 7's steps for context switching are as follows[10]:

1. Save context of thread that finished executing.
2. Place thread that finished executing at the end of the queue for its priority.
3. Find the highest priority queue that contains ready threads.
4. Remove the thread at the head of the queue, load its context, and execute.

Microsoft says that there are three reasons to context switch[10]:

1. Time slice has elapsed.
2. Thread with higher priority is ready to run.
3. Running thread needs to wait.
    a. This is done by yielding the CPU.

Windows 7 scheduling uses a combination of priority queues and round-robin to schedule threads. With priority queues comes priority inversion. However, Windows 7 avoids this with priority boosts. A thread has something called a dynamic priority that is set equal to its base priority initially[11]. Threads' dynamic priority will be boosted or lowered so threads don't starve and are responsive[11]. Threads with a base priority between 0 and 15 can only be boosted while threads with a base priority between 16 and 31 cannot be boosted. Only the dynamic priority is changes, not the base priority. Here is when the OS will boost a thread[11]:

- A thread using the NORMAL_PRIORITY_CLASS tag is brought into the foreground. The priority is boosted to be greater than or equal to any priority class in the background.
- When a window receives any sort of input including timers, mouse, and keyboard. The thread that owns the window will have its priority boosted.
- When the wait conditions for waiting threads (blocked threads) are satisfied. I.E.: when keyboard or disk I/O finishes.

This method ensures that no thread will starve. The other part of the scheduler is purely round-robin which favors I/O bound jobs over CPU bound jobs. The scheduling is fair. Everything depends on the time slice of the timer. Lastly, the scheduler supports thread affinity[12].

Memory Management:

Windows 7 memory management support include virtual memory, memory mapped files, copy-on-write memory, large memory support, and support for the cache manager[13]. The virtual addressing in Windows is the same as what we learned in class: the virtual address doesn't represent the physical address of an object's memory[14]. A page table is used for each process to translate virtual addresses to physical addresses. Windows 32-bit systems have 4GB of virtual address space divided into two partitions. One is used for the process while the other is used for the OS.

Windows memory management holds something called Working Sets. "The working set of a process is the set of pages in the virtual address space of the process that are currently resident in physical memory[15]." Only pageable memory allocations are included in the working set. Page faults occur when a process tries to reference memory not in the working set[15]. The system will try to resolve the page fault, and if it succeeds, it will add the page to the working set.

Virtual addresses can be in one of three states[16]:

- Free state
    - The page is not accessible by a process. Reading or writing creates an access violation.
- Reserved state
    - The page is reserved for future use. The addresses in this page can't be used, and the page is not accessible or has no physical memory.
- Committed state
    - Changes to the memory have been allocated from the RAM and paging files on disk. The page is accessible and controlled by memory protection. The system loads the page into physical memory during the first attempt at reading or writing the page. When finished, the system releases the storage required for this page so other pages can use it.

- A page can be either reserved or committed, or both reserved and committed. A page can't be free along with any other state.

Windows also has a layer of memory protection built into the OS. The Copy-on-Write Protection allows multiple processes to map virtual address spaces so they can share a page[17]. They share a physical page until one of any of the processes modifies the page[17]. The OS uses this process as lazy evaluation[17] so it can conserve physical memory and time by not performing an operation until necessary.

The second method is Loading Applications and DLLs. If multiple windows of the same application are running, each instance has its own virtual address space while their instance handles have the same value[17]. The value represents the base address of the application in the virtual space. If each instance can be loaded into its base address, it can map and share the same physical pages with other instances[17] until one instance modifies the page. Then it will follow the Copy-on-Write method. If an instance can't be loaded at its base address, it will receive its own physical page. The same is done for loading DLLs—dynamically linked libraries.

This set of memory loading and saving style done with paging was used because Microsoft knew that systems would be unable to keep up with the information produced using segmentation or other types of memory loading.

Resources

1. https://learn.microsoft.com/en-us/lifecycle/products/windows-7
2. https://en.wikipedia.org/wiki/Windows_7
3. https://en.wikipedia.org/wiki/Hybrid_kernel
4. https://en.wikipedia.org/wiki/Monolithic_kernel
5. https://en.wikipedia.org/wiki/Microkernel
6. https://en.wikipedia.org/wiki/Fiber_(computer_science)
7. https://learn.microsoft.com/en-us/windows/win32/procthread/what-s-new-in-processes-and-threads
8. https://learn.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads
9. https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities
10. https://learn.microsoft.com/en-us/windows/win32/procthread/context-switches
11. https://learn.microsoft.com/en-us/windows/win32/procthread/priority-boosts
12. https://learn.microsoft.com/en-us/windows/win32/procthread/multiple-processors
13. https://learn.microsoft.com/en-us/windows/win32/memory/memory-management
14. https://learn.microsoft.com/en-us/windows/win32/memory/virtual-address-space
15. https://learn.microsoft.com/en-us/windows/win32/memory/working-set
16. https://learn.microsoft.com/en-us/windows/win32/memory/page-state
17. https://learn.microsoft.com/en-us/windows/win32/memory/memory-protection