Why is kernel32.dll running in user mode and not kernel mode, like its name implies?

devblogs.microsoft.com/oldnewthing/20230926-00

September 26, 2023

Raymond Chen

So there's this DLL in Windows called `kernel32.dll`. From its name, you might expect it to be the code that controls 32-bit kernel mode. But in reality, it runs in user mode. Why is the "kernel" running in user mode?

Set the time machine to 1985.

Windows 1.0 consisted of three primary components:

- KERNEL: Provides low-level services like memory management, task scheduling, file I/O.
- GDI: Provides graphics services.
- USER: Provides user interface / window manager services.

The 8086 processor did not have privilege levels, so there was no privilege separation between programs and the operating system. Any separation between them was accomplished by convention and mutual respect.

When protected mode was introduced in the 80286 processor, it became possible to separate privileged from unprivileged code. The processor has four privilege levels, called *rings*, numbered from zero (most privileged) to three (least privileged). It also has a memory management unit that can be used to present separate address spaces to each application. However, for backward compatibility, 16-bit Windows didn't use the address space separation feature because existing programs had gotten into the habit of accessing each other's memory by just passing pointers directly back and forth.

Windows also held back on using privilege separation between applications and the Windows kernel, and this ended up coming in handy, because the first 80386 version of Windows used ring 0 for the 32-bit virtual machine manager. The 32-bit virtual machine manager allowed the creation of multiple MS-DOS sessions, each running in a separate virtual machine, and all running alongside a virtual machine in which all of your Windows programs ran.

But even though all of the Windows programs ran at the same privilege level as the Windows kernel, there was a virtual machine manager kernel that the Windows kernel in turn relied upon.

Ring 3	16-bit Windows app1 app2 KERNEL	MS-DOS session	MS-DOS session
Ring 0	Virtual machine manager		

It is the virtual machine manager that runs in ring zero. Initially, the virtual machine manager handled pre-emptive multi-tasking and virtual memory, but it left the file system in 16-bit code. In Windows 3.11 and Windows 95, even the file system itself moved into ring zero.

The job of what you think of as a modern operating system kernel has moved into ring zero, but applications still called functions in `KERNEL` for memory allocation and file I/O. Those functions in turned relied on ring zero services, but the interface to them is in ring three.

You can think of the names `KERNEL`, `GDI` and `USER` as shorthand for `KERNELSERVICES`, `GRAPHICSSERVICES`, and `USERINTERFACESERVICES`, abbreviated to fit inside the 8-character filename limitation of MS-DOS: The `KERNEL` module is not the ring zero kernel. Rather, it is the thing that gives you access to services which are associated with a modern operating system kernel.

The names `KERNEL`, `GDI`, and `USER`, carried forward to 32-bit Windows, leading to the somewhat confusing situation that `KERNEL32` runs in user mode. It isn't the part of the system that runs in kernel mode. Instead, it's the part of the operating system that provides kernel-ish services such as allocating memory, accessing files, and interacting with the scheduler.

Bonus chatter: The 80386 processor does not use the terms "kernel mode" and "user mode". It calls the different privilege levels "rings", and it so happens that Windows uses only rings zero and three.[1]

Windows NT consolidated the four rings into just two privilege levels because most non-Intel processors supported only two privilege levels anyway. Since Windows NT positioned itself as a portable operating system, it could not be architecturally dependent on a feature that was not present in most potential target processors.

[1] The OS/2 operating system used three of the four available rings. The privileged operating system code ran at ring zero, and most applications ran in ring three. However, an application could mark certain code segments to be run in ring two, which granted them access to I/O ports.

Ring one was intended to be used for device drivers, so they can be given access to more system services than regular applications, but without giving them full access to the entire system. I don’t know whether any operating system ever used ring one for that purpose.