Why doesn't Windows 95 format floppy disks smoothly?

devblogs.microsoft.com/oldnewthing/20090102-00

January 2, 2009

Raymond Chen

Welcome, Slashdot readers. Remember, this Web site is for entertainment purposes only.

Who spends all day formatting floppy disks? From the reaction of geekdom, it appears that there are lots of geeks who sit around formatting disks all day. (Psst, you can buy them pre-formatted.) But why did Windows 95 get all sluggish when you formatted a floppy disk?

It's that pesky MS-DOS compatibility again.

As we saw a while ago, MS-DOS acted as the 16-bit legacy device driver layer for Windows 95. Even though the operation was handled by the 32-bit file system, all I/O calls were routed through 16-bit code (if only briefly) so that 16-bit drivers, TSR, and the like would see what appeared to be "normal 16-bit behavior" and continue operating in the manner to which they had become accustomed.

In the old 16-bit days, disk formatting was done via software interrupt 13h, and many programs took advantage of this by hooking the interrupt so they would know whenever a floppy disk was being formatted. Some TSRs did this, as did backup programs (including backup programs designed for Windows 3.0 which included 32-bit Windows 3.x drivers—VxDs they were called—to monitor the floppy drive). But that doesn't explain everything. After all, Windows 95 sent *all* disk I/O through the 16-bit vectors, not just floppy disk formatting. Why does floppy disk formatting take such a toll on the system?

As I noted in the linked article, the 32-bit file system did a lot of fakery to make 16-bit code believe that MS-DOS was in charge, even though it wasn't. Anybody who's done TSR programming (wow, the phrase *anybody who's done TSR programming* used to cover a lot of people but nowadays describes a few dozen geezers, most of whom are trying very hard to forget those days) knows all about the INDOS flag. This was a flag that MS-DOS set when an MS-DOS I/O call was active. Since MS-DOS was not re-entrant, TSRs had to pay close attention to that flag to know whether it was safe to issue MS-DOS calls or not. This INDOS flag was the 16-bit manifestation of what the 32-bit kernel called simply *The Critical Section*, with the definite article, because the 32-bit kernel kept the two critical sections (the 32-bit one and the 16-bit one) in sync so that MS-DOS drivers and TSRs wouldn't themselves get re-

entered. If one virtual machine claimed the critical section, another virtual machine that tried to claim it would wait until the first one released it. In that manner, the driver or TSR would not get re-entered.

As I already noted, back in the 16-bit days, the actual work of formatting was done by the ROM BIOS, and for compatibility reasons, floppy disk formatting was still sent through software interrupt 13h on the 16-bit side so any TSRs or drivers could see what was going on. There are a lot of crazy ROM BIOSes out there, and when a floppy disk format request was issued, the 32-bit kernel would do a bunch of extra work to ensure that the ROM BIOS got the execution environment it wanted. For example, the hardware timer ports were temporarily unvirtualized so as not to mess up the sensitive timing loops that ROM BIOSes used for floppy disk formatting.

Okay, let's add up the damage. When a floppy disk is formatting, the timer is unvirtualized so that the ROM BIOS timing loops will run accurately. Only the virtual machine that is formatting the floppy drive receives timer ticks; the others have to wait. No timer ticks means the scheduler doesn't get told when it's time to let another thread run. What's more, the critical section is held across this operation, which means that no other thread can issue I/O operations either. And on top of that, the floppy disk is a slow medium, so any operations that wait on the floppy disk will have to sit and wait for several seconds.

Well, at least floppy disks are formatted a track at a time, so the system doesn't get locked out for the entire duration of the format operation. The ROM BIOS would be told to format a track, and when it was done, the timers would be returned to normal (allowing the scheduler to do a little bit of scheduling), the critical section would be released (so that any pent-up I/O gets a chance to run), but then the `FORMAT.COM` program would turn around and format the next track, and the system would go back into *hang on, let's not disturb the ROM BIOS while it does its thing* mode for another track.

Now, as with the 32-bit file system, there was a 32-bit floppy driver that tried to catch the format operations on the back end, and if successful, it would take over the job of formatting one track from the ROM BIOS. It was a valiant effort, but it doesn't matter how high-performance your driver is; the speed of formatting a track is pretty much constrained by the mechanics of the floppy disk drive itself. (The person responsible for Windows 95's 32-bit floppy driver was no slouch. I'll try to remember to tell some more stories later.)

Sure, if Windows 95 didn't have to be compatible with 16-bit device drivers, TSRs, and squirly ROM BIOSes, it could have gone straight to the 32-bit floppy driver to do the formatting without having to do all this timer and critical section nonsense. But it turns out we already had a product that said good-bye to compatibility with 16-bit device drivers, TSRs, 16-bit Windows programs that talked to custom 32-bit Windows 3.x drivers, and squirly ROM BIOSes. It was called *Windows NT*.

If you want Windows NT you know where to find it.

Raymond Chen

Follow