If each thread's TEB is referenced by the fs selector, does that mean that the 80386 is limited to 1024 threads?

devblogs.microsoft.com/oldnewthing/20190520-00

May 20, 2019

Raymond Chen

Commenter Waleri Todorov recalled that the global descriptor table (GDT), which is one of the places that selectors are defined, is limited to 1024 selectors. Does that mean that there is a hard limit of 1024 threads?

The question was in the context of how Windows NT for the 80386 managed the thread environment block (TEB), namely, by using the `fs` register to point to the per-thread data. The point is that there are at most 1024 possible distinct values for the `fs` register to have, so does this implicitly limit the number of threads to 1024?

No, it doesn't, because nobody said that the distinct values had to be different simultaneously.

Let's start with a single-processor system. That single processor is executing only one thread at a time, so there needs to be only one valid value for `fs` at a time. When the processor changes threads, the definition of that selector is updated to refer to the TEB for the incoming thread. Using selectors to access another thread's TEB is not part of the ABI; all that is required is that you can use `fs` to access *your own* TEB.

You can see this in the debugger. Break into a multithreaded program and look at the value of the `fs` register. On my system it's `0x0053`. Switch to another thread and look at the value of the `fs` register. It's the same value: `0x0053`. Every thread has the same selector in `fs`. What happens is that each time the processor changes threads, the GDT entry for `0x0053` is updated to refer to the TEB of the thread that is being scheduled.[1]

This trick works even on multiprocessor systems. Each processor has its own GDTR internal register, so instead of sharing a single GDT for all processors, you can give each processor its own GDT.

So I guess this puts the theoretical maximum number of processors supported by an x86-based system at around twenty-four million, because that would exhaust all of kernel mode address space just for GDTs.[2]

No, wait, that's still not the limit, because each processor also gets its own page table. After all, that's how two processors can be executing threads from different processes (and therefore in different address spaces). So the theoretical limit is basically *until you run out of memory*.

But I suspect you'll run into other problems long before you add that twenty-four-millionth processor.

[1] Bit 2 is clear for GDT selectors and set for LDT selectors, so you can infer that `0x0053` is a GDT selector.

[2] I calculated this by dividing 2^{31} by `0x60`, which is my presumed minimum size for a GDT. A selector whose numeric value is `0x0053` implies that the GDT is at least `0x0058` bytes in size, because that's how big you need to be to get to a selector value of `0x0053` in the first place.

Raymond Chen

Follow