Consequences of the scheduling algorithm: Sleeping doesn't always help

devblogs.microsoft.com/oldnewthing/20051004-09

October 4, 2005

Raymond Chen

More often I see the reverse of the "Low priority threads can run even when higher priority threads are running" problem. Namely, people who think that `Sleep(0)` is a clean way to yield CPU. For example, they might have run out of things to do and merely wish to wait for another thread to produce some work.

Recall that the scheduler looks for the highest priority runnable thread, and if there is a tie, all the candidates share CPU roughly equally. A thread can call `Sleep(0)` to relinquish its quantum, thereby reducing its share of the CPU. Note, however, that this does not guarantee that other threads will run.

If there is a unique runnable thread with the highest priority, it can call `Sleep(0)` until the cows come home, and it will nevertheless not relinquish CPU. That's because sleeping for zero milliseconds release the quantum but leaves the thread runnable. And since it is the only runnable thread with the highest priority, it immediately gets the CPU back. Sleeping for zero milliseconds is like going to back of the line. If there's nobody else in line, you didn't actually yield to anyone!

Therefore, if you use `Sleep(0)` as an ineffective yield, you will never allow lower priority threads to run. This means that various background activities (such as indexing) never get anywhere since your program is hogging all the CPU. What's more, the fact that your program never actually releases the CPU means that the computer will never go into a low-power state. Laptops will drain their batteries faster and run hotter. Terminal Servers will spin their CPU endlessly.

The best thing to do is to wait on a proper synchronization object so that your thread goes to sleep until there is work to do. If you can't do that for some reason, at least sleep for a nonzero amount of time. That way, for that brief moment, your thread is not runnable and other threads—including lower-priority threads—get a chance to run. (This will also reduce power consumption somewhat, though not as much as waiting on a proper synchronization object.)

Raymond Chen

Follow