Why doesn't TryEnterCriticalSection try harder?

devblogs.microsoft.com/oldnewthing/20100426-00

April 26, 2010

Raymond Chen

Bart wants to know why the `TryEnterCriticalSection` gives up if the critical section is busy instead of trying the number of times specified by the critical section spin count.

Actually, there was another condition on the proposed new behavior: "but does not release its timeslice to the OS if it fails to get it while spinning." This second condition is a non-starter because you can't prevent the operating system from taking your timeslice away from you. The best you can do is detect that you lost your previous timeslice when you receive the next one. And even that is expensive: You have to keep watching the CPU cycle counter, and if it jumps by too much, then you lost your timeslice. (And you might have lost it due to a hardware interrupt or paging. Good luck stopping those from happening.)

Even if there were a cheap way of detecting that the operating system was about to take your timeslice away from you, what good would it do? "Oh, my calculations indicate that if I spin one more time, I will lose my timeslice, so I'll just fail and return." Now the application regains control with 2 instructions left in its timeslice. That's not even enough time to test the return value and take a conditional jump! Even if the `TryEnterCriticalSection` managed to return just before the timeslice expired, that's hardly any consolation, because the timeslice is going to expire before the application can react to it. Whatever purpose there was to "up to the point where you're about to release the timeslice" is lost.

Okay, maybe the intention of that clause was "without intentionally releasing its timeslice (but if it loses its timeslice in the normal course of events, well that's the way the cookie crumbles)." That brings us back to the original question. Why doesn't `TryEnterCritical-Section` try harder? Well, because if it tried harder, then the people who didn't want it to try hard at all would complain that it tried too hard.

The function `TryEnterCriticalSection` may have been ambiguously named, because it doesn't describe *how hard* the function should try. Though in general, functions named `TryXxx` try only once, and that's the number of times `TryEnterCriticalSection` tries. Perhaps a clearer (but bulkier name) would have been `EnterCriticalSectionIfNotOwned-ByAnotherThread`.

The `TryEnterCriticalSection` function represents the core of the `EnterCritical-Section` function. In pseudocode, the two functions work like this:

```
BOOL TryEnterCriticalSection(CRITICAL_SECTION *CriticalSection)
{
  atomically {
   if (CriticalSection is free or is owned by the current thread) {
     claim the critical section and return TRUE;
   }
  }
  return FALSE;
}
void EnterCriticalSection(CRITICAL_SECTION *CriticalSection)
{
 for (;;) {
  DWORD SpinTimes = 0;
  do {
    if (TryEnterCriticalSection(CriticalSection)) return;
  } while (++SpinTimes < GetSpinCount(CriticalSection));
  WaitForCriticalSectionOwnerToLeave(CriticalSection);
 }
}
```

The `TryEnterCriticalSection` function represents the smallest meaningful part of the `EnterCriticalSection` process. If you want it to spin, you can write your own `TryEnter-CriticalSectionWithSpinCount` function:

```
BOOL TryEnterCriticalSectionWithSpinCount(
    CRITICAL_SECTION *CriticalSection,
    DWORD SpinCount)
{
  DWORD SpinTimes = 0;
  do {
    if (TryEnterCriticalSection(CriticalSection)) return TRUE;
  } while (++SpinTimes < SpinCount);
  return FALSE;
}
```

(Unfortunately, there is no `GetCriticalSectionSpinCount` function, so you'll just have to keep track of it yourself.)

Raymond Chen

Follow