Lock-free algorithms: Choosing a unique value (solutions)

devblogs.microsoft.com/oldnewthing/20110406-01

April 6, 2011

Raymond Chen

Last time, I left a warm-up exercise consisting of a code fragment which tries to compute a unique process-wide value. Here it is again:

```
dwUniqueId = InterlockedCompareExchange(&g_dwUniqueId,
                                        g_dwUniqueId+1,
                                        g_dwUniqueId);
```

It may be easier to enumerate what the function does *right* rather than what it does wrong.

Um, the words are correctly-spelled.

That's about it.

Damien was the first to note that the author basically reimplemented `Interlocked-Increment`. Poorly.

As we saw earlier, the algorithm for performing complex calculations with interlocked functions is (capture, compute, compare-exchange, retry). But the above code didn't do any of these things.

By failing to capture the values, the code is vulnerable to another thread modifying the `g_dwUniqueId` value simultaneously. This means that the computation step can fail, because the inconsistent reads of `g_dwUniqueId` result in who-knows-what getting passed to the `InterlockedCompareExchange` function.

Okay, they managed to spell `InterlockedCompareExchange` correctly.

And then they forgot to retry the operation if the compare-exchange failed, which means that they will just proceed with whatever value the `g_dwUniqueId` variable held at the time of the `InterlockedCompareExchange` call. If it just got incremented by another thread, then this thread and the other thread will be using the same "unique" value.

Joshua points out that compiler optimization can prevent the capture from being a true capture. Though I would put the `volatile` keyword on `g_dwUniqueId` rather than `scv`, because the volatile object is the global variable, not the local. Marking the local as volatile forces all accesses to the local to be executed as written, but the compiler can still optimize the access to `g_dwUniqueId`. (It might, for example, propagate the value in from a previous read earlier in the function.)

And do take into consideration Leo Davidson's warning: This series of articles is a *peek behind the scenes* series, not a *here's how you should do it* series. We're taking apart a bunch of toasters to see how they work. When possible, take advantage of code written by people smarter than you.

Raymond Chen

Follow