If there's already a bug, it's not surprising that there's a possibility for error

devblogs.microsoft.com/oldnewthing/20081103-00

November 3, 2008

Raymond Chen

It's great to think about all the things that can go wrong but you also have to think about the situations that could lead to those bad things. In particular, you have to recognize when you are trying to avoid a bug that is ultimately outside your component and which you can't fix anyway.

For example, consider this multithreaded race condition:

> Why is `InterlockedDecrement` used in the implementation of `IUnknown::Release`? The only reason I can think of is for multithread safety. But that `Release` function doesn't look multithread safe—what if another thread was about to increment `m_cRef`? Does the `AddRef` refcount incrementer have a special interlocked check for zero to catch this case?

What if another thread was about to increment `m_cRef`? In other words, what if another thread was about to call `IUnknown::AddRef`? In other words, you have two threads and an object with a refcount of one. One thread calls `Release` and the other thread calls `AddRef`. The concern is that the thread calling `AddRef` may execute after the thread that calls `Release`, thereby "rescuing" the reference count from zero back to one.

But this scenario you're worried about *is already a bug*. Suppose the second thread runs just a smidgen slower than the scenario you described, calling `AddRef` after the `Release` returns instead of while it is executing. Well, now, that's obviously a bug in the caller, isn't it? It's using a pointer after destroying it.

This happens a lot: You're worrying about not introducing a bug into a hypothetical situation that is already a bug. The answer to that is "Fix the original bug."

In this specific situation of reference counting, a useful rule of thumb is "If you're worrying about the possibility of a reference count incrementing from zero to one, then you already have a bug somewhere else."

Raymond Chen

Follow