Avoiding double-destruction when an object is released

devblogs.microsoft.com/oldnewthing/20050928-10

September 28, 2005

Raymond Chen

As we saw last time, trying to do too much in one's destructor can lead to an object being destroyed twice. The standard way to work around this problem is to set an artificial reference count during destruction.

```
class MyObject : public IUnknown
{
 …
 ULONG Release()
 {
  LONG cRef = InterlockedDecrement(&m_cRef);
  if (cRef == 0) {
   m_cRef = DESTRUCTOR_REFCOUNT;
   delete this;
  }
  return cRef;
 }
 …
private:
 }
 enum { DESTRUCTOR_REFCOUNT = 42 };
 ~MyObject()
 {
  if (m_fNeedSave) Save();
  assert(m_cRef == DESTRUCTOR_REFCOUNT);
 }
};
```

If you have a common implementation of `IUnknown`, you can set the reference count to `DESTRUCTOR_REFCOUNT` in your implementation of `IUnknown::Release` like we did here, and assert that the value is correct in your implementation's destructor. Since C++ runs base class destructors after derived class destructors, your base class destructor will check the reference count after the derived class has done its cleanup.

By setting the reference count to an artificial non-zero value, any `AddRef()` and `Release()` calls that occur will not trigger a duplicate destruction (assuming of course that nobody in the destructor path has a bug that causes them to over-release). The assertion at the end ensures that no new references to the object have been created during destruction.

This is really more of a workaround than a rock-solid solution, because it assumes that no functions called during the destruction sequence retain a reference to the object beyond the function's return. This is in general not something you can assume about COM. In general, a method is free to call `AddRef` and hang onto a pointer to an object in order to complete the requested operation later. Some methods (such as the `IPersistPropertyBag::Load` method) explicitly forbid such behavior, but these types of methods are more the exception rather than the rule.

Exercise: Why is it safe to perform a simple assignment `m_cRef = DESTRUCTOR_REFCOUNT` instead of the more complicated `InterlockedExchangeAdd(&m_cRef, DESTRUCTOR_REFCOUNT)`?

Raymond Chen

Follow