If you’re going to use an interlocked operation to generate a unique value, you need to use it before it’s gone

devblogs.microsoft.com/oldnewthing/20130425-00

April 25, 2013

Raymond Chen

Is the `InterlockedIncrement` function broken? One person seemed to think so.

> We’re finding that the `InterlockedIncrement` is producing duplicate values. Are there are any know bugs in `InterlockedIncrement`?

Because of course when something doesn’t work, it’s because you are the victim of a vast conspiracy. There is a fundamental flaw in the `InterlockedIncrement` function that only you can see. *You are not a crackpot*.

```
LONG g_lNextAvailableId = 0;


DWORD GetNextId()
{
  // Increment atomically
  InterlockedIncrement(&g_lNextAvailableId);


  // Subtract 1 from the current value to get the value
  // before the increment occurred.
  return (DWORD)g_lNextAvailableId – 1;
}
```

Recall that `InterlockedIncrement` function increments a value atomically and returns the incremented value. If you are interested in the result of the increment, you need to *use the return value directly* and not try to read the variable you incremented, because that variable may have been modified by another thread in the interim.

Consider what happens when two threads call `GetNextId` simultaneously (or nearly so). Suppose the initial value of `g_lNextAvailableId` is 4.

- First thread calls `InterlockedIncrement` to increment from 4 to 5. The return value is 5.
- Second thread calls `InterlockedIncrement` to increment from 5 to 6. The return value is 6.
- First thread ignores the return value and instead reads the current value of `g_lNext-AvailableId`, which is 6. It subtracts 1, leaving 5, and returns it.
- Second thread ignores the return value and instead reads the current value of `g_lNextAvailableId`, which is still 6. It subtracts 1, leaving 5, and returns it.

Result: Both calls to `GetNextId` return 5. Interpretation: "`InterlockedIncrement` is broken."

Actually, `InterlockedIncrement` is working just fine. What happened is that the code threw away the unique information that `InterlockedIncrement` returned and instead went back to the shared variable, even though the shared variable changed its value in the meantime.

Since this code cares about the result of the increment, it needs to use the value returned by `InterlockedIncrement`.

```
DWORD GetNextId()
{
  // Increment atomically and subtract 1 from the
  // incremented value to get the value before the
  // increment occurred.
  return (DWORD)InterlockedIncrement(&g_lNextAvailableId) – 1;
}
```

Exercise: Criticize this implementation of `IUnknown::Release`:

```
STDMETHODIMP_(ULONG) CObject::Release()
{
 InterlockedDecrement(&m_cRef);
 if (m_cRef == 0)
 {
  delete this;
  return 0;
 }
 return m_cRef;
}
```

Raymond Chen

Follow