Acquire and release sound like bass fishing terms, but they also apply to memory models

devblogs.microsoft.com/oldnewthing/20081003-00

October 3, 2008

Raymond Chen

Many of the normal interlocked operations come with variants called `InterlockedXxxAcquire` and `InterlockedXxxRelease`. What do the terms `Acquire` and `Release` mean here?

They have to do with the memory model and how aggressively the CPU can reorder operations around it.

An operation with *acquire* semantics is one which does not permit subsequent memory operations to be advanced before it. Conversely, an operation with *release* semantics is one which does not permit preceding memory operations to be delayed past it. (This is pretty much the same thing that MSDN says on the subject of Acquire and Release Semantics.)

Consider the following code fragment:

```
int adjustment = CalculateAdjustment();
while (InterlockedCompareExchangeAcquire(&lock, 1, 0) != 0)
  { /* spin lock */ }
for (Node *node = ListHead; node; node = node->Next)
   node->value += adjustment;
InterlockedExchangeRelease(&lock, 0);
```

Applying *Acquire* semantics to the first operation operation ensures that the operations on the linked list are performed only after the `lock` variable has been updated. This is obviously desired here, since the purpose of the updating the `lock` variable is ensure that no other threads are updating the list while we're walking it. Only after we have successfully set the lock to 1 is it safe to read from `ListHead`. On the other hand, the *Acquire* operation imposes no constraints upon when the store to the `adjustment` variable can be completed to memory. (Of course, there may very well be other constraints on the `adjustment` variable, but the Acquire does not add any new constraints.)

Conversely, *Release* semantics for an interlocked operation prevent pending memory operations from being delayed past the operation. In our example, this means that the stores to `node->value` must all complete before the interlocked variable's value changes back to zero. This is also desired, because the purpose of the lock is to control access to the linked list. If we had completed the stores after the lock was released, then somebody else could have snuck in, taken the lock, and, say, deleted an entry from the linked list. And then when our pending writes completed, they would end up writing to memory that has been freed. Oops.

The easy way to remember the difference between *Acquire* and *Release* is that *Acquire* is typically used when you are acquiring a resource (in this case, taking a lock), whereas *Release* is typically used when you are releasing the resource.

As the MSDN article on acquire and release semantics already notes, the plain versions of the interlocked functions impose both acquire and release semantics.

Bonus reading: Kang Su discusses how VC2005 converts volatile memory accesses into acquires and releases.

[Raymond is currently away; this message was pre-recorded.]

Raymond Chen

Follow