# If you want to track whether the current thread owns a critical section, you can use the critical section itself to protect it

devblogs.microsoft.com/oldnewthing/20130712-00

July 12, 2013

Raymond Chen

You may find yourself in the situation where you want to keep track of the owner of a critical section. This is usually for debugging or diagnostic purposes. For example, a particular function may have as a prerequisite that a particular critical section is held, and you want to assert this so that you can catch the problem when running the debug build.

```
class CriticalSectionWithOwner
{
public:
  CriticalSectionWithOwner() : m_Owner(0), m_EntryCount(0)
  {
    InitializeCriticalSection(&m_cs);
  }
  ~CriticalSectionWithOwner()
  {
    DeleteCriticalSection(&m_cs);
  }
  void Enter()
  {
    EnterCriticalSection(&m_cs);
#ifdef DEBUG
    m_Owner = GetCurrentThreadId();
    m_EntryCount++;
#endif
  }
  void Leave()
  {
#ifdef DEBUG
    if (--m_EntryCount == 0) {
      m_Owner = 0;
    }
#endif
    LeaveCriticalSection(&m_cs);
  }
#ifdef DEBUG
  bool IsHeldByCurrentThread()
  {
    return m_EntryCount &&
           m_Owner == GetCurrentThreadId();
  }
#endif
private:
  CRITICAL_SECTION m_cs;
#ifdef DEBUG
  DWORD m_Owner;
  int m_EntryCount;
#endif
};
```

After we successfully enter the critical section, we mark the current thread as the owner and increment the entry count. Before leaving the critical section, we see if this is the last exit, and if so, we clear the owner field.

Note that we update the owner and entry count while the critical section is held. We are using the critical section to protect its own diagnostic data.

The subtle part is the `IsHeldByCurrentThread` function. Let's look at the cases:

First, if the current thread is the owner of the critical section, then we know that the diagnostic data is safe to access because we own the critical section that protects it. That's not the subtle part.

The subtle part is the case where the current thread is *not* the owner of the critical section. A naïve analysis would say that the diagnostic data is off limits because you are trying to access it without owning the protective critical section. But what value can `m_Owner` have at this point?

1. If the critical section is not held, then `m_Owner` will be zero, which will be unequal to the current thread ID.
2. If the critical section is held, then `m_Owner` will be the owner of the critical section, which will also be unequal to the current thread ID.

But what if the value of `m_Owner` changes while we are looking at it? Well, since we are not the owner of the critical section, it can only change between the two states above (possibly from one state 2 to another state 2). In other words, it can only change from one value that is not equal to the current thread ID to another value that is *still* not equal to the current thread ID. Therefore, even if we race against another thread entering or leaving the critical section, the fact that the owner of the critical section is *not us* doesn't change.

Note that this analysis assumes that the `m_Owner` is a suitably-aligned value that can be updated atomically. (If not, then it's possible that a torn value will be read which coincidentally matches our thread ID.)

Since the `CRITICAL_SECTION` itself must already be suitably aligned, placing the `DWORD` up against it will also align the `DWORD`.

Raymond Chen

**Follow**