Why is it possible to destroy a critical section while it is in use?

devblogs.microsoft.com/oldnewthing/20091223-00

December 23, 2009

Raymond Chen

Some time back, Stu wondered why it is possible to destroy a critical section while it is in use.

Well, there's nothing stopping you from creating a file that contains these lines:

```
#include <windows.h>
int __cdecl main(int, char**)
{
  CRITICAL_SECTION cs;
  InitializeCriticalSection(&cs);
  EnterCriticalSection(&cs);
  DeleteCriticalSection(&cs);
  return 0;
}
```

and then telling your compiler to turn it into a program. It's not like a bolt of lightning is going to come out of the sky and zap you before you hit the Enter key.

So obviously, it's possible.

On the other hand, it's a bug, just like closing a handle to a file that another thread is reading from, or like closing an event handle that another thread is waiting on.

Critical sections are one of those low-level *I sure hope you know what you're doing because I'm not going to help you if you mess up* pieces of functionality. If you use them incorrectly, then you will suffer the consequences, the same as if you tried to free memory twice or write to memory after freeing it or cast a `Gdiplus::Color*` to a `CComBSTR*`.

Are there any legitimate cases where you would delete a critical section while it is owned? I sure can't think of any.

If there were a legitimate case for deleting a critical section while it is owned, what could it be? Well, it can't be owned by the thread doing the deleting, because that would imply that you took it in order to prevent somebody else from entering it (while you deleted it), but that just creates another race condition: If you tinker the timing, then you can create this

scenario: That other thread gets pre-empted just as it was about to execute the first instruction of the `EnterCriticalSection` function. Meanwhile, the destroying thread enters the critical section, does whatever other stuff it wants to do, and then deletes the critical section. That other thread finally gets a chance to run and is now attempting to enter a deleted critical section, which is clearly not legal.

Okay, so if there were a legitimate case, it would have to be deleting a critical section owned by some other thread. Maybe that other thread enters the critical section, and then signals the main thread to delete the critical section. Why would it do that? Who knows. Maybe it wants to make sure only one thread signals the main thread. But you still have the same problem as with the previous case: You entered the critical section because you wanted to prevent a third thread from entering the protected region, but that third thread might have been pre-empted just as it transferred control to the first instruction of `EnterCriticalSection`, and when that third thread finally gets some CPU time, it proceeds to enter a deleted critical section.

So I can't think of a legitimate reason for deleting a critical section while it's in use. Maybe there's a flaw in my logic.

Raymond Chen

Follow