What do HeapLock and HeapUnlock do, and when do I need to call them?

devblogs.microsoft.com/oldnewthing/20121228-00

December 28, 2012

Raymond Chen

You never need to call the `HeapLock` and `HeapUnlock` functions under normal operation. Assuming the heap is serialized (which is the default), all the standard heap functions like `HeapAllocate` and `HeapFree` will automatically serialize.

In fact, the way they serialize is by calling the¹ `HeapLock` and `HeapUnlock` functions!

Nearly all heap operations complete in a single call. If your heap is serialized, this means that the heap operation takes the heap lock, does its work, and then releases the heap lock and returns. If *all* heap operations were like this, then there would be no need for `HeapLock` or `HeapUnlock`.

Unfortunately, there is also the `HeapWalk` function, which does a little bit of work, and then returns with a partial result. The design for `HeapWalk` is that the application calls the function repeatedly until it either gets all the results it wants, or gets bored and gives up. But wait, what if the heap changes while the application is trying to walk through it? To prevent that from happening, the program can call `HeapLock` before starting the enumeration, and `HeapUnlock` when it is done. During the time the heap is locked, other threads which attempt to call a `HeapXxx` function with that same heap will block until the heap is unlocked.

The ability to lock the heap creates a lot of potential for craziness, because the heap is a high-traffic area. As a result, it is very important that any code which calls `HeapLock` do very little while the lock is held. Take the lock, do your thing, and get out quickly.

But wait, there's more. Holding the heap lock blocks all other threads from allocating or freeing memory. This puts the heap lock very low in your lock hierarchy. Therefore, while you hold the heap lock, you cannot block on synchronization objects whose owners might try to access the heap you just locked. Consider the following:

```
// Code in italics is wrong.
void BadIdea()
{
 HeapLock(GetProcessHeap());
 SendMessage(...);
 HeapUnlock(GetProcessHeap());
}
```

Sending a message is a big deal. The thread that is the target of the message may be waiting for the heap lock, and now you've created a deadlock. You won't proceed until that thread processes the message, but that thread can't process the message until you unlock the heap.

You might accidentally do something wrong while hold the heap lock if you happen to trigger a delay-loaded DLL, in which case your call into that other DLL turns into a call to `Load-Library`, and now you've lost control. In practice, the only thing you should be doing while holding the heap lock is calling `HeapWalk` and saving the results locally, and in a way that doesn't allocate or free memory on the heap you are walking! Wait until after you unlock the heap to start studying the results you collected or transfer the raw data into a more suitable data structure.

Bonus chatter

Note that if you call `HeapLock` or `HeapUnlock` on a heap that was created without serialization (`HEAP_NO_SERIALIZATION`), then the results are undefined. That's because passing the `HEAP_NO_SERIALIZATION` flag means "Hey, Heap Manager, don't bother locking this heap. I will take responsibility for ensuring that only one thread operates on this heap at a time." If you later call `HeapLock` on a no-serialization heap, the heap manager will say, "Wha? You said that *you* would take care of serialization, not me!"

It's like ordering a car and saying, "Don't bother installing door locks. I will take responsibility for ensuring the safety of the car. (Say, by never letting the car leave a secured facility.)" And then a month later, calling OnStar and saying, "Hi, can you remotely lock my car for me? Thanks." Dude, you explicitly opted out of door locks.

(Amazingly, I encountered one developer who thought that calling `HeapLock` on a no-serialization heap would cause other heap operations on the heap to be blocked, even if they passed the `HEAP_NO_SERIALIZATION` flag to those operations. Um, no, the `HeapLock` function cannot lock a no-serialization heap because a no-serialization heap *doesn't have lock in the first place, at your request*.)

Nitpicker's corner

[1] s/the/the functional equivalents of/

Raymond Chen

Follow