The management of memory for resources in 16-bit Windows, redux

devblogs.microsoft.com/oldnewthing/20131001-00

October 1, 2013

Raymond Chen

Some time ago, I briefly ran down how 16-bit Windows managed memory for resources. But there's a detail that I neglected to mention: Ownership. As we saw, a resource handle `HRSRC` was really a pointer to the resource directory entry of the resource from the corresponding module. This could be done with a 16-bit pointer because the segment portion of the pointer could be inferred from the module the resource belonged to. In fact, since modules could be relocated in memory at run time due to compaction, you had better not try to remember the segment portion of the pointer since it could change! The `FindResource` function located the resource directory entry. The `LoadResource` function allocated memory for the resource and loaded it from disk. The `LockResource` function locked the memory so you could access it. If two people tried to load the same resource, the memory for the resource was re-used so there was only one copy in memory, and if both people free the resource, the resource is cached in case somebody asks for it again soon. Now things get interesting: When does the resource get removed from the cache? What actually controls the lifetime of the resource? Answer: The resource lifetime is tied to the module it came from. When the module is unloaded, all its resources are unloaded along with it. (Note that even if a resource is cached, its contents can get discarded if it is tagged as `DISCARDABLE`.) In Win32, modules are directly mapped into memory, and along with it, the resources. Therefore, accessing the resources of a module is a simple matter of figuring out where they got mapped (`Find-Resource` and friends will tell you), and then reading the memory directly. So despite the radical change to resources work, the basic rules haven't changed: The resources are good as long as the module is still in memory.

But there are resources and then there are resources. So far, we've been talking about resources in the sense of `FindResource`, which I will call *module resources* for lack of a better term. But people often work with objects like icons and bitmaps which are not literally resources but which are derived from resources. Next time, we'll look at the relationship between module resources and resource-derived objects in 16-bit Windows.

Raymond Chen

Follow