A survey of the various ways of declaring pages of memory to be uninteresting

devblogs.microsoft.com/oldnewthing/20170113-00

January 13, 2017

Raymond Chen

The list of ways a program can declare pages of memory to be uninteresting seems to be growing steadily. Let's look at what we have so far today.

The most old-fashioned way of declaring a page to be uninteresting is to free it. The catch with that is that freeing the memory with the `VirtualFree` function and the `MEM_RELEASE` flag frees the entire allocation, not individual pages. If you allocated a 64KB chunk of memory, then you have to release the whole thing. You can't release half of it.

But all is not lost. Because while you cannot free a single page from a larger allocation, you *can* decommit it, which is almost as good. Decommitting page is like freeing it, except that the address space is still reserved. To decommit a page, call `VirtualFree` with the `MEM_DECOMMIT` flag.

For quite some time, those were the only tools you had available. Around the Windows NT 4 era, a new trick arrived on the scene: You could `VirtualUnlock` memory that was already unlocked in order to remove it from your working set. This was a trick, because it took what used to be a programming error and gave it additional meaning, but in a way that didn't break backward compatibility because the contractual behavior of the memory did not change: The contents of the memory remain valid and the program is still free to access it at any time. The new behavior is that unlocking unlocked memory also takes it out of the process's working set, so that it becomes a prime candidate for being paged out and used to satisfy another memory allocation.

The fact that it preserved contractual behavior means that you could scatter `VirtualUnlock` calls randomly throughout the program and have no effect on correctness. It might run slower (or faster), but it will still run.

Around the Windows 2000 era, the `MEM_RESET` flag was added to `VirtualAlloc`. If you pass this flag, this tells the memory manager that the memory in question is no longer interesting to your program, and the memory manager is free to discard it without saving the contents. The memory itself remains accessible to the program, and doing so before the

memory gets discarded will read the old values. On the other hand, if the memory manager decides that it needs to evict the memory (in order to satisfy a memory request elsewhere), it will throw away the contents without saving it, and then turn the page into a demand-zero-initialized page. Later, if your program tries to access the memory, it will see a page full of zeroes.

Windows 8 added the `MEM_RESET_UNDO` flag which says, "Hey, um, I changed my mind. I don't want you to discard the contents of the memory after all." If the memory hasn't yet been discarded, then it is "rescued" and behaves like normal memory again. But if the memory has already been discarded, then the memory manager will say, "Sorry, too late."

And then at some point, I don't know exactly when, my colleague Adrian added code to check if a page of memory is all zeroes before paging it out, and turning it into a demand-zero-initialized page if so. So another way to say that you are not interested in a page of memory is to explicitly zero it. That causes it to turn into a demand-zero-initialized page at page-out time, which avoids the I/O of writing a page full of zeroes to disk. This is another one of those things that has no effect on the programming model; it's just an optimization. If you are running on a system that doesn't perform this optimization, everything still behaves the same as before, just a little slower.

Note that writing the zeroes to the page does have its own side effects. (Well, aside from the obvious side effect of, y'know, *filling the page with zeroes*.) Writing to the page will set both the Dirty and Accessed bits in the page table, which will bring it into the process's working set, and therefore will reduce its likelihood of being selected for eviction. In other words, zeroing out the page "resets the clock" on the eviction calendar. Therefore, if you're going to do this, do it as soon as you're done with the memory.

In Windows 8.1 we got the function `OfferVirtualMemory` which mixes in a few new wrinkles. First of all, when you call `OfferVirtualMemory`, you pass a flag that says how much you don't care about this memory: You can say that you totally don't care, you mostly don't care, you sort of don't care, or you have no opinion on the concept of caring.

Okay, formally, what you're doing is saying how to prioritize the memory for discarding. At one extreme, you can make it a prime candidate for discarding. At the other extreme, you can say, "No special priority here. Just prioritize it according to the standard rules, as if it were plain old regular process memory."

The other wrinkle to the `OfferVirtualMemory` function is that once you offer the memory, it is no longer accessible to your program. Trying to access memory that has been offered will take an access violation.

If you later decide that you want the memory back, you can call `ReclaimVirtualMemory`, which will try to bring the memory back into your process. If it fails, then the contents are garbage.

There's also a companion function `DiscardVirtualMemory` which forces an immediate discard and leaves the page contents undefined. It's the equivalent of `OfferVirtual-Memory`, and then calling `ReclaimVirtualMemory`, and forcing the reclaim to fail.

Okay, so here we go with the table.

	VirtualFree + MEM_RELEASE	VirtualFree + MEM_DECOMMIT	Virtual-Unlock	VirtualAlloc + MEM_RESET	Zero-Memor
Is address space still reserved?	N	Y	Y	Y	Y
Is memory accessible?	N	N	Y	Y	Y
Is memory removed from working set?	Y	Y	Y	N[1]	N
Can control eviction priority?	N	N	N	N	N
Are previous contents recoverable?	N	N	Y	Y until eviction	N
Contents if recovery failed	N/A	N/A	N/A	Zeroes	Zeroes

Bonus chatter: The flip side of discarding memory is prefetching it. I've discussed the `PrefetchVirtualMemory` before, so I'll leave it at a mention this time. (And here's a non-mention.)

[1] The fact that `MEM_RESET` does not remove the page from the working set is not actually mentioned in the documentation for the `MEM_RESET` flag. Instead, it's mentioned in the documentation for the `OfferVirtualMemory` function, and in a sort of backhanded way:

> Note that offering and reclaiming virtual memory is similar to using the MEM_RESET and MEM_RESET_UNDO memory allocation flags, except that **OfferVirtualMemory** removes the memory from the process working set and restricts access to the offered pages until they are reclaimed.

Raymond Chen

Follow