# Some remarks on VirtualAlloc and MEM_LARGE_PAGES

devblogs.microsoft.com/oldnewthing/20110128-00

January 28, 2011

Raymond Chen

If you try to run the sample program demonstrating how to create a file mapping using large pages, you'll probably run into the error `ERROR_NOT_ALL_ASSIGNED` (Not all privileges or groups referenced are assigned to the caller) when calling `AdjustTokenPrivileges`. What is going on?

The `AdjustTokenPrivileges` enables privileges that you already have (but which are masked). Sort of like how a super hero can't use super powers while disguised as a normal mild-mannered citizen. In order to enable the `SeLockMemoryPrivilege` privilege, you must already have it. But where do you get it?

You do this by using the group policy editor. The list of privileges says that the `SeLock-MemoryPrivilege` corresponds to "Lock pages in memory".

Why does allocating very large pages require permission to lock pages in memory?

Because very large pages are not pageable. This is not an inherent limitation of large pages; the processor is happy to page them in or out, but you have to do it all or nothing. In practice, you don't want a single page-out or page-in operation to consume 4MB or 16MB of disk I/O; that's a thousand times more I/O than your average paging operation. And in practice, the programs which use these large pages are "You paid $40,000 for a monster server whose sole purpose is running my one application and nothing else" type applications, like SQL Server. Those applications don't want this memory to be pageable anyway, so adding code to allow them to be pageable is not only a bunch of work, but it's a bunch of work to add something *nobody who uses the feature actually wants*.

What's more, allocating very large pages can be time-consuming. All the physical pages which are involved in a very large page must be contiguous (and must be aligned on a large page boundary). Prior to Windows XP, allocating a very large page can take 15 seconds or more if your physical memory is fragmented. (And even machines with as much as 2GB of memory will probably have highly fragmented physical memory once they're running for a little while.) Internally, allocating the physical pages for a very large page is performed by the kernel function which allocates physically contiguous memory, which is something device

drivers need to do quite often for I/O transfer buffers. Some drivers behave "highly unfavorably" if their request for contiguous memory fails, so the operating system tries very hard to scrounge up the memory, even if it means shuffling megabytes of memory around and performing a lot of disk I/O to get it. (It's essentially performing a time-critical defragmentation.)

If you followed the discussion so far, you'll see another reason why large pages aren't paged out: When they need to be paged back in, the system may not be able to find a suitable chunk of contiguous physical memory!

In Windows Vista, the memory manager folks recognized that these long delays made very large pages less attractive for applications, so they changed the behavior so requests for very large pages from applications went through the "easy parts" of looking for contiguous physical memory, but gave up before the memory manager went into desperation mode, preferring instead just to fail. (In Windows Vista SP1, this part of the memory manager was rewritten so the really expensive stuff is never needed at all.)

Note that the `MEM_LARGE_PAGES` flag triggers an exception to the general principle that `MEM_RESERVE` only reserves address space, `MEM_COMMIT` makes the memory manager guarantee that physical pages will be there when you need them, and that the physical pages aren't actually allocated until you access the memory. Since very large pages have special physical memory requirements, the physical allocation is done up front so that the memory manager knows that when it comes time to produce the memory on demand, it can actually do so.

Raymond Chen

**Follow**