You cannot globally reserve user-mode address space

devblogs.microsoft.com/oldnewthing/20050215-00

February 15, 2005

Raymond Chen

Occasionally, somebody asks for a way to reserve user-mode address space globally. In other words, they want to allocate address space in all processes in the system (current and future). Typically this is because they want to map some memory into each process and don't want to go through the trouble of designing the shared memory blocks so that they can be relocated. This is obviously not possible. Why obviously? Well, imagine if this were possible. "Imagine if this were possible" is one of the logic tests you can apply to a theory to see if it can possibly be true. Here, we're using it to determine whether a proposed behavior is possible. [Typo fixed 10am.] (There is a corresponding thought experiment, "Imagine if things actually worked that way.") What are the consequences of global address space allocation? Well, first of all, there's no guarantee that by the time you request your global address space, there will be any available addresses at all. Consider a program that uses every last scrape of user-mode address space. (It can do this by just calling `VirtualAlloc(MEM_RESERVE)` in a loop; since no memory is being committed, this doesn't actually require 2GB of RAM.) Run such a program and no global address space allocations are possible until that program exits. So even if it were possible, it wouldn't be reliable. Your program would have to be prepared for the situation where no global address space was available. Since you're going to have to write fallback code anyway, you didn't save yourself any work. Next, suppose it were possible, that there were some imaginary `GlobalVirtualAlloc` function. Well, then I can write a program that calls this imaginary function in a loop, sucking up all available global virtual address space, and run it as a non-administrator. I just violated security. (The general principle here is that a non-administrator should not be allowed to affect other users. We've already seen one scenario where a non-administrator can crash a program running as administrator due to insecure use of shared memory.) My imaginary program sucked up all global virtual address space, reducing the address space available to programs running as administrator. If there aren't many programs running on the system, my imaginary program will probably be able to suck up quite a lot of address space this way, which in turn will cause a corresponding reduction in address space to those administrative programs. I can therefore cause those programs to run out of address space sooner, resulting in premature failure (denial of service). Yes, you could decide that "global" address space reservation is available only to administrators, but that wouldn't help a lazy programmer, since the program would not work when run as a non-administrator – you have to write the fallback code anyway. Or

you could decide that “global” address space reservation applies only to users within the same session with the same security token, but again, you have to write the fallback code anyway if the global reservation fails; you didn’t save yourself any work. When the time comes, you can use `VirtualAlloc` and pass a preferred address to try to get the memory at that address; if it fails, then use the fallback code that you already wrote.

The moral of the story is that each process gets its own address space and each process manages its address space independently of other processes. (Of course, a process can grant a user `PROCESS_VM_OPERATION` permission, which gives that user permission to mess with the address space of that process. But changes to that process’s address space does not affect the address space of other processes.)

Raymond Chen

Follow