Myth: The /3GB switch expands the user-mode address space of all programs

devblogs.microsoft.com/oldnewthing/20040812-00

August 12, 2004

Raymond Chen

Only programs marked as /LARGEADDRESSAWARE are affected.

For compatibility reasons, only programs that explicitly indicate that they are prepared to handle a virtual address space larger than 2GB will get the larger virtual address space. Unmarked programs get the normal 2GB virtual address space, and the address space between 2GB and 3GB goes unused.

Why?

Because far too many programs assume that the high bit of user-mode virtual addresses is always clear, often unwittingly. MSDN has a page listing some of the ways programs make this assumption. One such assumption you may be making is taking the midpoint between two pointers by using the formula (a+b)/2. As I noted in a previous exercise, this is subject to integer overflow and consequently can result in an erroneous pointer computation. Consequently, you can't just take an existing program that you didn't write, mark it /LARGEADDRESSAWARE, and declare your job done. You have to check with the authors of that program that they verified that their code does not make any 2GB assumptions. (And the fact that the authors didn't mark their program as 3GB-compatible strongly suggests that no such verification has occurred. If it had, they would have marked the program /LARGEADDRESSAWARE!)

Marking your program /LARGEADDRESSAWARE indicates to the operating system, "Go ahead and give this program access to that extra gigabyte of user-mode address space," and as a result, addresses in the third gigabyte become possible return values from memory allocation functions. If you set the "Top down" flag in the memory manager allocation preferences mask (search for "top down"), you can instruct the memory manager to allocate high-address memory first, thereby forcing your program to deal with those addresses sooner than it normally would. This is very handy when testing your program in a /3GB configuration since it forces the troublesome memory addresses to be used sooner than normal.

Exercise: Find the bug in the following function. Hint: What's today's topic?

```
#define BUFFER_SIZE 32768
BOOL  IsPointerInsideBuffer(const BYTE *p, const BYTE *buffer)
{
  return p >= buffer && p - buffer < BUFFER_SIZE;
}
```

Raymond Chen

Follow