Getting MS-DOS games to run on Windows 95: Not enough XMS handles

devblogs.microsoft.com/oldnewthing/20160503-00

May 3, 2016

Raymond Chen

A report arrived on an MS-DOS game that crashed after you go through some preliminary game steps and then attempt a planetary landing. When you do this, the program blows up and starts corrupting memory.

Debugging the game revealed that performing a planetary landing causes the application to allocate some more XMS memory, and this happens to be its 32nd request for XMS memory. When run from a custom-made MS-DOS boot disk, this 32nd request succeeds, because `HIMEM.SYS`, the default MS-DOS XMS driver, gives you 32 handles. But when run under Windows 95, the 32nd call to allocate XMS memory fails.

Windows 95 could have been extra awesome, like it was with EMM memory, and offer a large number of handles. But it was also constrained by the MS-DOS XMS driver that it tries to take over from. If any MS-DOS drivers had allocated XMS memory from the MS-DOS XMS driver, then Windows 95 needs to respect those allocations when those drivers try to use that memory after the transition to 32-bit mode.

Those drivers don't even realize that the operating system changed out from under them. They allocated an XMS handle from one driver (`HIMEM.SYS`), then passed it to another driver (Windows 95's MS-DOS emulator), and the handle *still worked*.

The way this worked is that when a request to access XMS memory occurs, the Windows 95 XMS driver first checks if it is a handle that it knows about. If so, then it services the request. if not, then the Windows 95 XMS driver passes the XMS handle through to the MS-DOS driver to see if that driver recognizes it.

In order for this trick to work, Windows 95's XMS driver must not generate an XMS handle that happens to match a handle that was produced by the MS-DOS XMS driver. But how do you make sure that you don't generate a number that is different from an unknown set of numbers?

Here's the trick: One of the things that Windows 95 does when it starts up is allocate zero bytes of XMS memory from the MS-DOS XMS driver until it reports "no more handles". All of those handles that came out are now known to be different from any handles that the MS-DOS XMS driver handed to any 16-bit device driver. When an MS-DOS application asks for some XMS memory, Windows 95 hands out one of those "known not to be a global handle" handles.

What this means is that Windows 95 won't be able to give out any more XMS handles than the MS-DOS XMS driver did. On the other hand, each virtual machine gets its own set of handles, so it's not like there are 32 handles total. Rather, the calculation goes like this: If MS-DOS device drivers had allocated *N* XMS handles prior to the transition to 32-bit mode, then those handles are global. The remaining 32 – *N* handles are local.

Okay, back to our story.

The deal is that for some reason I have long forgotten, the value of *N* was 1, which meant that each MS-DOS virtual machine could allocate 31 more handles before it ran out.

Okay, so the game tries to allocate its 32nd handle, and the call fails. How does this lead to the program blowing up and corrupting memory?

The game had a helper library that provided a C interface to the ugly assembly-language-based XMS interface. The function to allocate a block of XMS memory returned the memory block handle on success, or zero on failure.

The program evidently never checked whether the call actually succeeded. It just assumed that the call succeeded, and used zero as the handle.

The excitement comes into play when the program tries to use that XMS memory handle. XMS has a single function for transferring memory, and the source and destination could be an XMS handle or conventional memory. (Technically, you could set both the source and destination to be conventional memory, but there'd be no need for that in practice, because you already can transfer that memory yourself without needing the help of the XMS driver. In practice, therefore, at least the source or the destination, and possibly both, will be an XMS handle.)

And the way you specified that the source or destination is conventional memory rather than an XMS handle is to set the corresponding *handle* field to zero.

Since the program never checked whether its XMS allocation succeeded, its attempt to copy memory into the XMS memory block that it just "allocated" actually copied memory into conventional memory. And since it's going to start by copying into the start of the XMS memory block (offset zero), what it actually ended up doing was copying into the start of conventional memory (`0000:0000`), which is the hardware interrupt table.

Scrambling the hardware interrupt table is a great way to blow up your system.

I marked the game as requiring MS-DOS mode with a custom configuration that set `/NUMHANDLES=64`, ensuring that the game got all the handles it needed and then some. I made a note of the problem, and the plan was that if more games appeared with the same problem, then maybe we would take a closer look at what we could do to get the program running under Windows 95 instead of requiring MS-DOS mode.

It turns out that no other games were found that had this problem.

Raymond Chen

Follow