# How did protected-mode 16-bit Windows fix up jumps to functions that got discarded?

devblogs.microsoft.com/oldnewthing/20141219-00

December 19, 2014

Raymond Chen

Commenter Neil presumes that Windows 286 and later simply fixed up the movable entry table with jmp selector:offset instructions once and for all.

It could have, but it went one step further.

Recall that the point of the movable entry table is to provide a fixed location that always refers to a specific function, no matter where that function happens to be. This was necessary because real mode has no memory manager.

But protected mode does have a memory manager. Why not let the memory manager do the work? That is, after all, its job.

In protected-mode 16-bit Windows, the movable entry table was ignored. When one piece of code needed to reference another piece of code, it simply jumped to or called it by its selector:offset.

```
    push    ax
    call    0987:6543
```

(Exercise: Why didn't I use `call 1234:5678` as the sample address?)

The selector was patched directly into the code as part of fixups. (We saw this several years ago in another context.)

When a segment is relocated in memory, there is no stack walking to patch up return addresses to point to thunks, and no editing of the movable entry points to point to the new location. All that happens is that the base address in the descriptor table entry for the selector is updated to point to the new linear address of the segment. And when a segment is discarded, the descriptor table entry is marked *not present*, so that any future reference to it will raise a *selector not present* exception, which the kernel handles by reloading the selector.

Things are a lot easier when you have a memory manager around. A lot of the head-exploding engineering in real-mode windows was in all the work of simulating a memory manager on a CPU that didn't have one!

Raymond Chen

**Follow**