How did Windows 95 rebase DLLs?

devblogs.microsoft.com/oldnewthing/20041217-00

December 17, 2004

Raymond Chen

Windows 95 handled DLL-rebasing very differently from Windows NT.

When Windows NT detects that a DLL needs to be loaded at an address different from its preferred load address, it maps the entire DLL as copy-on-write, fixes it up (causing all pages that contain fixups to be dumped into the page file), then restores the read-only/read-write state to the pages. (Larry Osterman went into greater detail on this subject earlier this year.)

Windows 95, on the other hand, rebases the DLL incrementally. This is another concession to Windows 95's very tight memory requirements. Remember, it had to run on a 4MB machine. If it fixed up DLLs the way Windows NT did, then loading a 4MB DLL and fixing it up would consume all the memory on the machine, pushing out all the memory that was actually worth keeping!

When a DLL needed to be rebased, Windows 95 would merely make a note of the DLL's new base address, but wouldn't do much else. The real work happened when the pages of the DLL ultimately got swapped in. The raw page was swapped off the disk, then the fix-ups were applied on the fly to the raw page, thereby relocating it. The fixed-up page was then mapped into the process's address space and the program was allowed to continue.

This method has the advantage that the cost of fixing up a page is not paid until the page is actually needed, which can be a significant savings for large DLLs of mostly-dead code. Furthermore, when a fixed-up page needed to be swapped out, it was merely discarded, because the fix-ups could just be applied to the raw page again.

And there you have it, demand-paging rebased DLLs instead of fixing up the entire DLL at load time. What could possibly go wrong?

Hint: It's a problem that is peculiar to the x86.

The problem is fix-up that straddle page boundaries. This happens only on the x86 because the x86 architecture is the weirdo, with variable-length instructions that can start at any address. If a page contains a fix-up that extends partially off the start of the page, you cannot

apply it accurately until you know whether or not the part of the fix-up you can't see generated a carry. If it did, then you have to add one to your partial fix-up.

To record this information, the memory manager associates a flag with each page of a relocated DLL that indicates whether the page contained a carry off the end. This flag can have one of three states:

- Yes, there is a carry off the end.
- No, there is no carry off the end.
- I don't know whether there is a carry off the end.

To fix up a page that contains a fix-up that extends partially off the start of the page, you check the flag for the previous page. If the flag says "Yes", then add one to your fix-up. If the flag says "No", then do not add one.

But what if the flag says "I don't know?"

If you don't know, then you have to go find out. Fault in the previous page and fix it up. As part of the computations for the fix-up, the flag will get to indicate whether there is a carry out the end. Once the previous page has been fixed up, you can check the flag (which will no longer be a "Don't know" flag), and that will tell you whether or not to add one to the current page.

And there you have it, demand-paging rebased DLLs instead of fixing up the entire DLL at load time, even in the presence of fix-ups that straddle page boundaries. What could possibly go wrong?

Hint: What goes wrong with recursion?

The problem is that the previous page might itself have a fix-up that straddled a page boundary at its start, and the flag for the page two pages back might be in the "I don't know" state. Now you have to fault in and fix up a third page.

Fortunately, in practice this doesn't go beyond three fix-ups. Three pages of chained fix-ups was the record.

(Of course, another way to stop the recursion is to do only a partial fix-up of the previous page, applying only the straddling fix-up to see whether there is a carry out and not attempting to fix up the rest. But Windows 95 went ahead and fixed up the rest of the page because it figured, hey, I paid for this page, I may as well use it.)

What was my point here? I don't think I have one. It was just a historical tidbit that I hoped somebody might find interesting.

Raymond Chen

Follow