Why is the LOADPARMS32 structure so messed up?

devblogs.microsoft.com/oldnewthing/20080707-00

July 7, 2008

Raymond Chen

If you look at the `LOADPARMS32` structure, you'll find a horrific mishmash. Double-null-terminated strings, a null-terminated string, some `WORD`s, and even a Pascal-style string. What's going on here?

Each of those members comes from a different era in time. The oldest member is the Pascal-style command line, which dates back to CP/M. On CP/M, command lines were stored at a fixed location, namely 0080h through 00FFh, in the form of a Pascal string. The byte at 0080h specified the length of the command line, and the bytes at 0081h through 00FFh contained the command line itself.

MS-DOS based much of its initial interface on CP/M in order to make porting to the new operating system easier, and part of what got carried over was the way command lines were passed from one program to another. The MS-DOS function to load a program took two parameters, a pointer to a null-terminated string (specifying the module to load) and a pointer to a parameter block which took the following form:

```
LOADPARMS       struc
loadp_environ   dw      ?       ; environment of new process
loadp_cmdline   dd      ?       ; command line of new process
loadp_fcb1      dd      ?       ; first FCB
loadp_fcb2      dd      ?       ; second FCB
LOADPARMS       ends
```

To ease the transition, Windows 1.0 used the same MS-DOS interface for launching programs: You loaded up the registers and issued an `int 21h` instruction. All the parameters had the same meaning. Generally speaking, 16-bit Windows used the old MS-DOS interface for a lot of functionality, especially disk access. Want to write to a file? Put the file handle in the `BX` register, the number of bytes in the `CX` register, a pointer to the buffer in the `DS:DX` registers, function code `40h` in the `AH` register, and issue an `int 21h`, just like in MS-DOS.

Why do this? Well, it saved the Windows team from having to invent a whole boatload of functions that duplicated what MS-DOS already did, and it meant that existing MS-DOS programs didn't need to change a thing in their file I/O code. If they used a runtime library

designed for MS-DOS (C or otherwise), that library would still write to files by setting registers and issuing an `int 21h`. If you want people to switch to your new platform, you need to make it easy, and "you don't have to change anything; it all just works" is pretty easy. (One minor change was that the first FCB was repurposed to contain the `nCmdShow`; the magic value of "2" in the first word of the FCB signals that it's not really an FCB.)

As a minor convenience, the `LoadModule` function provided a C-callable version of the low-level `int 21h`, but you still had to provide the parameters in the form of the MS-DOS exec structure. It wasn't until later versions of Windows that the `WinExec` function was added, thereby providing a much more convenient interface to starting a new program. No longer did you have to mess with the crazy MS-DOS exec structure and its strange way of passing the command line and `nCmdShow`.

The people who were designing Win32 created their own function `CreateProcess` to launch a new process, but for backward compatiblity, they retained the old `WinExec` and even older `LoadModule` mechanisms. The pointers in the crazy 16-bit exec block got converted to 32-bit, but the craziness of what they pointed to was retained to make porting old code easier. The `int 21h` interface no longer exists, of course. The craziness is just a leftover from the old MS-DOS days. The `WinExec` and `LoadModule` functions are now just stub functions that convert their parameters and call the `CreateProcess` function to do the real work.

Raymond Chen

Follow