Why did the 16-bit _lopen and _lcreat function return -1 on failure instead of 0?

devblogs.microsoft.com/oldnewthing/20230919-00

September 19, 2023

Raymond Chen

Some time ago, I discussed why `HANDLE` return values are so inconsistent, and I traced it all the way back to the 16-bit `_lopen` and `_lcreat` functions, which returned `-1` on failure.

But why do those functions return `-1` on failure instead of zero?

The `_lopen` and `_lcreat` functions were Windows versions of the C runtime `_open` and `_creat` functions. The C runtime functions came in four different versions depending on which MS-DOS memory model you were using, and the convention was that when Windows adopted a C runtime function, it used the "large" version with the `L` prefix, since that is the most general version.

Okay, so why did `_open` and `_creat` return `-1` on failure?

Because they were MS-DOS-compatible versions of the Unix functions `open` and `creat`. They even preserve the dropped silent "e" at the end of `creat`.

Okay, so why do those functions return `-1` on failure?

On Unix, the return value is an integer that represents a file descriptor, valid file descriptors are integers starting with zero. Every process comes with three predefined file descriptors:

Descriptor	Meaning
0	stdin (standard input)
1	stdout (standard output)
2	stderr (standard error)

Files opened by the program begin with file descriptor 3.

The value `-1` is used to represent failure because 0 was already taken.

And that value of `-1` carried forward, through a chain of backward compatibility, to Win32 as the numeric value of `INVALID_HANDLE_VALUE`. We saw a little while ago one of the consequences.