I saw a pinvoke signature that passed a UInt64 instead of a FILETIME, what's up with that?

devblogs.microsoft.com/oldnewthing/20150820-00

August 20, 2015

Raymond Chen

A customer had a question about a pinvoke signature that used a `UInt64` to hold a `FILETIME` structure.

> ```
> [DllImport("kernel32.dll", SetLastError = true)
> static external bool GetProcessTimes(
>     IntPtr hProcess,
>     out UInt64 creationTime,
>     out UInt64 exitTime,
>     out UInt64 kernelTime,
>     out UInt64 userTime);
> ```
>
> Is this legal? The documentation for `FILETIME` says
>
> > Do not cast a pointer to a **FILETIME** structure to either a **ULARGE_INTEGER*** or **__int64*** value because it can cause alignment faults on 64-bit Windows.
>
> Are we guilty of this cast in the above code? After all you can't treat a `FILETIME` as an `__int64`.

There are two types of casts possible in this scenario.

- Casting from `FILETIME*` to `__int64*`.
- Casting from `__int64*` to `FILETIME*`.

The `FILETIME` structure requires 4-byte alignment, and the `__int64` data type requires 8-byte alignment. Therefore the first cast is unsafe, because you are casting from a pointer with lax alignment requirements to one with stricter requirements. The second cast is safe because you are casting from a pointer with strict alignment requirements to one with laxer requirements.

4-byte aligned	8-byte aligned

Everything in the blue box is also in the pink box, but not vice versa.

Which cast is the one occurring in the above pinvoke signature?

In the above signature, the `UInt64` is being allocated by the interop code, and therefore it is naturally aligned for `UInt64`, which means that it is 8-byte aligned. The `GetProcess-Times` function then treats those eight bytes as a `FILETIME`. So we are in the second case, where we cast from `__int64*` to `FILETIME*`.

Mind you, you can avoid all this worrying by simply declaring your pinvoke more accurately. The correct solution is to declare the last four parameters as `ComTypes.FILETIME`. Now there are no sneaky games. Everything is exactly what it says it is.

Bonus reading: The article Use PowerShell to access registry last-modified time stamp shows how to use the `ComTypes.FILETIME` technique from PowerShell.

Raymond Chen

Follow