Why can’t you treat a FILETIME as an __int64?

devblogs.microsoft.com/oldnewthing/20040825-00

August 25, 2004

Raymond Chen

The FILETIME structure represents a 64-bit value in two parts:

```
typedef struct _FILETIME {
  DWORD dwLowDateTime;
  DWORD dwHighDateTime;
} FILETIME, *PFILETIME;
```

You may be tempted to take the entire `FILETIME` structure and access it directly as if it were an `__int64`. After all, its memory layout exactly matches that of a 64-bit (little-endian) integer. Some people have written sample code that does exactly this:

```
pi = (__int64*)&ft; // WRONG
(*pi) += (__int64)num*datepart; // WRONG
```

Why is this wrong?

Alignment.

Since a `FILETIME` is a structure containing two `DWORD`s, it requires only 4-byte alignment, since that is sufficient to put each `DWORD` on a valid `DWORD` boundary. There is no need for the first `DWORD` to reside on an 8-byte boundary. And in fact, you’ve probably already used a structure where it doesn’t: The WIN32_FIND_DATA structure.

```
typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD dwReserved0;
    DWORD dwReserved1;
    TCHAR  cFileName[ MAX_PATH ];
    TCHAR  cAlternateFileName[ 14 ];
} WIN32_FIND_DATA, *PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

Observe that the three `FILETIME` structures appear at offsets 4, 12, and 20 from the beginning of the structure. They have been thrown off 8-byte alignment by the `dwFileAttributes` member.

Casting a `FILETIME` to an `__int64` therefore can (and in the `WIN32_FIND_DATA` case, **will**) create a misaligned pointer. Accessing a misaligned pointer will raise a `STATUS_DATATYPE_MISALIGNMENT` exception on architectures which require alignment.

Even if you are on a forgiving platform that performs automatic alignment fixups, you can still run into trouble. More on this and other consequences of alignment in the next few entries.

Exercise: Why are the `LARGE_INTEGER` and `ULARGE_INTEGER` structures not affected?

Raymond Chen

Follow