Fixing a floating point exception when operating on NaN

devblogs.microsoft.com/oldnewthing/20160325-00

March 25, 2016

Raymond Chen

A customer had a DLL would sometimes fail to load into certain clients. Further investigation revealed that it failed to load into programs that had enabled floating point exceptions, perhaps inadvertently. The DLL initialized some global variables to hold convenient values, and one of the values it needed was NaN.

This particular DLL did not use the C++ standard library, so it did not have access to `numeric_limits<double>::quiet_NaN()`. (And even if it did, those values are not compile-time constants.) Instead, it tried to generate a NaN on its own:

```
double const NaN = HUGE_VAL * 0;
```

The gotcha here is that `HUGE_VAL` is a static global variable in the Microsoft C runtime, so this performs a runtime calculation inside of `DLL_PROCESS_ATTACH`, and if floating point exceptions are enabled, the NaN-generating calculation causes the exception to be raised, which then causes the loader to treat the DLL as having failed to initialize.

The DLL was precalculating NaN because there was a method that returns NaN to indicate that something isn't available. They precalculated the value so that that method could return it.

The customer's solution was to move the NaN-generating calculation into the code that needed it. Unfortunately, that still raises the "invalid calculation" exception in the case where the function wants to report "No value", which isn't really a case of an invalid calculation; it's just a sentinel value.

Instead, the code can generate a NaN at compile time. Everybody wins: There is no exception at DLL load time because there is no code running at DLL load time! It also means that the page containing the `NaN` variable does not get dirtied, which avoids a copy-on-write charge.

Raymond Chen

Follow