Are DDE and WM_COPYDATA related as IPC mechanisms?

devblogs.microsoft.com/oldnewthing/20170202-00

February 2, 2017

Raymond Chen

A customer liaison asked whether DDE and the `WM_COPYDATA` message were related as IPC mechanisms. Specifically, are they dependent on each other, or are they independent?

The two communications mechanisms are independent. I mean, sure, they are related in the sense that they both use window messages, but there is no cross-dependency between them. You can use one without the other, and neither depends on the other.

In practice, you are likely to see one or the other, but not both. Very old programs will use DDE, because DDE was invented first. Newer programs will use `WM_COPYDATA` and ignore DDE because you are free to stop using DDE.

The customer liaison explained that the customer has a very old suite of applications which they are trying to migrate off their Windows Server 2003[1] systems to Windows Server 2008 R2, but the programs are getting into a hung state after an extended period of use. Looking at the memory dumps (filled with many ancient components, some provided from a third party, for which they have no symbols), reveals that two of the processes appear to be stuck sending a `WM_COPYDATA` message. The customer claims that their programs communicate with DDE, not `WM_COPYDATA`, which led to the customer liaison asking if DDE and `WM_COPYDATA` were somehow interdependent.

The two are not interdependent. They are two different ways of performing inter-process communication. Fortunately, `WM_COPYDATA` is easier to debug than DDE because `WM_COPYDATA` is a synchronous message. If a `WM_COPYDATA` is stuck, you can extract the window that is the target of the message, get the thread responsible for that window, and then study that thread to see why it is not responding.

My guess is that the customer has existing code that has taken a lock (let's call it Lock A), and then did something that processes inbound sent messages, say entering a message loop or sending a cross-thread message. While the thread is pumping messages or waiting for the cross-thread `SendMessage` to complete, another thread sends a `WM_COPYDATA` message, and now the window procedure is being re-entered. The `WM_COPYDATA` message tries to take a different lock (let's call it Lock B), and blocks. Meanwhile, the owner of Lock B wants to take Lock A, and we now have a deadlock. Reason: Pumping messages (or sending messages between threads) while holding a lock creates a lock inversion opportunity. Inspection of the stuck stack would reveal whether any window procedure re-entrancy is active.

This is a timing bug that could be the sort of thing exposed by a change in OS.

Bonus trivia: A colleague tells me that the `WM_COPYDATA` message was originally added in order to support the 32-bit version of MS Mail on the initial builds of Windows NT. Obviously, other people found other uses for the message since then.

[1] That is not a typo. They are running on a 13-year-old operating system which exited extended support over a year ago.

Raymond Chen

Follow