Psychic debugging: Understanding DDE initiation

devblogs.microsoft.com/oldnewthing/20060621-17

June 21, 2006

Raymond Chen

You too can use your psychic powers to debug the following problem:

> We have a problem with opening documents with our application by double-clicking them in Explorer. What's really strange is that if we connect a debugger to Explorer and set a breakpoint on `kernel32!CreateProcessW`, then wait a moment after `CreateProcess` returns, then hit '`g`', then the document opens fine. But if we don't wait, then the application launches but the document does not open. Instead, you get the error message "Windows cannot find 'abc.lit'. Make sure you typed the name correctly, and then try again." Here is the command we are executing when we run into this problem:
>
> ```
> "F:\Program Files\LitSoft\LitWare\LitWare.exe" /dde
> ```
>
> What is wrong?

If you've been reading carefully and paid attention to the explanation of how document launching via DDE works, then you already know what the problem is.

Recall that launching documents via DDE is done by first looking for a DDE server and if none is found, launching a server manually and trying again. The command line above was clearly registered as the `command` associated with a `ddeexec`. There are two giveaway clues. First is the fact that the document name itself is not present anywhere on the command line. (This couldn't be a direct execution because the program wouldn't know what document it's supposed to be opening!) But the giveaway clue is the phrase `/dde` on the command line.

Clearly, something is going wrong when Explorer attempts the second DDE conversation to open the document. The fact that making Explorer wait a few seconds fixes the problem makes the cause obvious: The DDE server is being slow to get itself initialized and listening. Explorer launches the server and tries to talk to it, but the server is not yet ready and therefore doesn't respond to the DDE initiate.

But how do you fix this?

The shell assumes that a DDE server is ready to accept connections when it goes input idle. Once `WaitForInputIdle` on the DDE server returns, Explorer will make its second attempt at initiating a DDE conversation. The fix is for the application to get its DDE server up and running before it starts pumping messages. My guess is that the application moved its DDE server to a background thread to improve startup performance, since the DDE server is not involved in normal program operation. Too bad the program forgot to get the DDE server up and running prior to going input idle when the `/dde` flag is passed. The one time it's important to have the DDE server running and it misses the boat.

Moral of the story: If you're going to act as a DDE server, make sure you do so before your primary thread starts pumping messages. Otherwise you have a race condition between your application startup and the shell trying to talk to it.

Raymond Chen

Follow