Low-level hooks have thread affinity, so make sure you keep an eye on the thread

devblogs.microsoft.com/oldnewthing/20150514-00

May 14, 2015

Raymond Chen

A customer was having a problem with their automated testing tool.

> We have an automation testing tool that, among other things, installs a low-level mouse hook. Sometimes, the hook takes too long to process an action, and it gets unhooked. We have a watchdog thread that tries to detect when this has happened, and in response, it kicks off a task on the thread pool to re-register the low-level hook. The call to register the low-level hook succeeds, but the hook apparently didn't get installed correctly because it never fires. What are we doing wrong?

Recall that low-level hooks have thread affinity. This is spelled out in the documentation.

> This hook is called in the context of the thread that installed it. The call is made by sending a message to the thread that installed the hook. Therefore, the thread that installed the hook must have a message loop.

So there are two mistakes here.

First, the hook is installed from a thread pool task, which means that the hook is associated with the thread pool thread. One of the characteristics of the thread pool is that threads come and go based on demand. If there is no thread pool activity for a while, the thread pool will probably start trimming threads, and it it decides to get rid of the thread that installed the hook, and the hook disappears with it.

The second mistake is that the hook is installed from a thread pool task. Sure, the hook registers successfully, but then when you return back to the thread pool, there's no guarantee that anybody on that thread is going to pump messages any time soon.

Indeed, odds are that it won't.

Tasks queued up on the thread pool tend not to be UI tasks, because, well, they're on the thread pool, not the UI thread. Therefore, there is no expectation that they will pump messages. Furthermore, if the thread goes idle, the thread pool is probably not going to pump

messages; it's just going to put the thread to sleep until the next task is queued up.

The customer thanked us for the explanation. I'm not sure what they are going to do about it, but I hope they're going to solve their problem not by patching up their watchdog thread but rather by fixing their low-level mouse hook so it doesn't exceeed the low-level hook timeout. For example, they could have the low-level hook post its events to another thread, then return immediately. That other thread can then do the expensive processing asynchronously. (This assumes that they are using the low-level hook only for monitoring the mouse rather than trying to intercept and block it.)

Raymond Chen

Follow