How do I fix the problem of a long-running task running on the thread pool persistent thread?

devblogs.microsoft.com/oldnewthing/20170216-00

February 16, 2017

Raymond Chen

Last time, we diagnosed a problem caused by a long-running task on the thread pool persistent thread, which prevented other tasks which target the persistent thread from running. But what motivated the developers to write code that put a long-running task on the persistent thread in the first place?

My theory is that the developers saw this sentence in the documentation for `RegNotifyChangeKeyValue`:

> With the exception of **RegNotifyChangeKeyValue** calls with **REG_NOTIFY_THREAD_AGNOSTIC** set, this function must be called on persistent threads.

The reasoning was probably, "Well, the documentation says that this must be called on a persistent thread, so we have to schedule it with the `WT_EXECUTEINPERSISTENTHREAD` flag."

That's not what the documentation is trying to say, but I can't fault them for misinterpreting it that way.

What the documentation is trying to say is, "If you know what's good for you, you will call this function on a thread that will not exit until you close the registry key. If the thread exits prematurely, then the notification will stop working (in a specific way described below, though you would be best to just avoid the problem entirely)."

The documentation is using the word "persistent thread" here in a generic sense, meaning any thread that does not exit (until the thing you care about is over). It doesn't have to run on the thread pool's persistent thread; any persistent thread will do.

The callback function registered by the application does not close the registry key handle until it has finished with the change notification, so it's fine to run this code on any thread; it doesn't have to run on the thread pool's persistent thread.

Therefore, one fix for the problem is to remove the `WT_EXECUTEINPERSISTENTHREAD` flag. This runs the work item on a thread pool thread with no special attributes. You still want the `WT_EXECUTELONGFUNCTION` flag because the work item runs long: It runs indefinitely until the monitoring is stopped.

However, running a long function with indefinite lifetime isn't really the thread pool's bread and butter. The thread pool is really for running large numbers of short items. After all, scheduling a work item on the thread pool that runs indefinitely isn't really all that different from running a dedicated thread for the work item. The purpose of the thread pool is to amortize the cost of starting up a thread over many work items. Otherwise, the system will be spending more time starting up threads than it does actually performing work.

This benefit doesn't really help work items with indefinite running time. From a percentage standpoint, the added cost of starting up a new thread is not significant if the thread is going to be running for minutes.

But if you look at the work item that the customer was scheduling, it's not really doing work most of the time anyway. It spends most of its time waiting! Next time, we'll look at another way of designing the code so that instead of burning a thread for each active monitoring request, it pools the requests.

In other words, we're going to use the thread pool as a thread pool. Tune in next time for the exciting conclusion.

Raymond Chen

Follow