Comparing fibers to threads for the purpose of expanding a thread's stack at runtime

devblogs.microsoft.com/oldnewthing/20200608-00

June 8, 2020

Raymond Chen

For the past week, we've been looking at using a fiber to allow execution to continue on a larger stack. Instead of creating a fiber, we could also have created a thread. What are the pros and cons?

Fibers are user-mode concepts, so switching fibers will save you the overhead of a kernel transition. This is probably not a big deal since we are using the fiber to run a function that consumes a lot of stack, and the running time of that function is probably going to overwhelm the cost of a kernel transition.

On the other hand, the savings of a fiber are more significant when compared to the cost of a new thread. (And we have to create a new thread, rather than using a thread from the thread pool, because we need to control the size of the thread's stack, which is something you can do only with threads that you created.)

Creating a new thread involves a lot of kernel-mode machinery, as well as a good amount of user-mode overhead. In particular, creating a thread requires the loader lock because DLL thread notifications are serialized by the loader lock. The loader lock can be a high-contention lock under certain workloads; for example, calling `GetModuleFileName` will probably require the loader lock to ensure that the module table is stable for the duration of the lookup. Even if it's not high-contention, the existence of the lock means that the start of each of your library calls is effectively serialized.

Using fibers also permits you to operate on objects that have thread affinity, such as UI objects or apartment-threaded COM objects, because the fiber runs on the same thread. You also have to worry about how to sleep the original thread while waiting for the worker thread to finish. UI threads, for example, need to pump messages.

On the other hand, fibers are not as well understood as threads. This means that maintaining code that users fibers may be a challenge to future developers. And as we noted in the opening to this series, if you manage the fibers incorrectly, you end up creating a lot of confusion.

As a general rule, most code does not expect to be run on a fiber. Therefore, if you're going to be running foreign code on your fiber, you need to be careful not to use any fiber special powers while that code is running. If the foreign code uses callbacks, then your callback shouldn't move the fiber to another thread. And it shouldn't suspend the fiber and then call back into the library.

Fortunately, if the sole purpose of using a fiber is to expand the stack, you are unlikely to be tempted to pull any of these fancy fiber tricks. You just want to get a bigger stack.

Those are the pros and cons I could come up with off the top of my head.

Raymond Chen

Follow