Fibers aren't useful for much any more; there's just one corner of it that remains useful for a reason unrelated to fibers

devblogs.microsoft.com/oldnewthing/20191011-00

October 11, 2019

Raymond Chen

Fibers were the new hotness back in 1996, but the initial excitement was gradually met with the realizations that fibers are awful. Gor Nishanov has a fantastic write-up of the history of fibers and why they suck. Of particular note is that nearly all of the original proponents of fibers subsequently abandoned them.

Fibers make asynchronous functions appear to be synchronous. Depending on what color glasses you are wearing, this is either a cool trick or a hidden gotcha. Over time, the consensus of most of the computing community has settled on the side of "hidden gotcha".

But there's still one part of Windows fibers that is still useful: The fiber destruction callback.

Okay, let's step back and look at thread-local storage first. Windows thread-local storage works like this:

- You allocate a thread-local storage slot.
- You learn that a thread has been created via the `DLL_THREAD_ATTACH` notification. You can initialize the thread-local storage in response to this notification.
- You learn that a thread has been destroyed via the `DLL_THREAD_DETACH` notification. You can clean up the thread-local storage in response to this notification.
- You free the thread-local storage slot.

There are a few problems here.

One problem is the case of the thread that existed prior to the allocation of the thread-local storage slot. It's possible that you never received any `DLL_THREAD_ATTACH` notification for these pre-existing threads, because your DLL wasn't even loaded at the time. Even if your DLL did receive those notifications, you couldn't do anything to initialize a thread-local storage slot that didn't exist.

In practice, what happens is that thread-local storage is lazily allocated. The DLL ignores the `DLL_THREAD_ATTACH` notification and allocates the storage on demand the first time a thread does something that requires it.

Another problem is with threads that remain in existence at the time the thread-local storage slot is deallocated. You have to free the memory at deallocation time because even if you get a subsequent `DLL_THREAD_DETACH`, the slot is already gone, so you lost track of the memory you wanted to free.

A common workaround for this is to keep your own data structure that remembers all the data that has been allocated for thread-local storage, and free that data structure when the slot is deallocated. But if you're going to do this, then you really didn't need the thread-local storage slot in the first place. When a thread needs to access per-thread storage, you can just look it up in that data structure you're using to keep track of them!

A third problem is with code that doesn't have access to the `DllMain` function. Executables do not receive DLL notifications, so they never learn about thread creation or destruction. Static libraries do not have access to the `DllMain` function of their host.

A common workaround for this is to hire a lackey. Executables may have a lackey DLL whose purpose is to forward the notification back to the executable. Static libraries may require the host to forwared the notifications into a special function in the static library.

Even though workarounds for all these problems exist, they're still annoying problems.

An alternate workaround is to abandon thread-local storage and use fiber-local storage instead. But not because you care about fibers. Rather, it's because fiber-local storage has this nifty callback.

```
DWORD WINAPI FlsAlloc(
    _In_ PFLS_CALLBACK_FUNCTION lpCallback
);
```

The callback function is called when a fiber is destroyed. This is the fiber equivalent of `DLL_THREAD_DETACH`, except that it's not just for DLLs. Executables can use it too.

Even better: When you deallocate the fiber-local storage slot, the callback is invoked for every extant fiber. This lets you clean up all your per-fiber data before it's too late.

As a bonus you support fibers, in case anybody uses your code with fibers.

The new fancy pattern is now this:

- You allocate a filber-local storage slot with a callback function.
- The fiber-local storage is allocated on demand the first time a fiber needs access to it.
- The callback function frees the fiber-local storage, if it had been allocated.

- When finished, you free the thread-local storage slot. This will call the callback for each fiber still in existence, so you can clean up the data.

Even though fibers are basically dead, you can use fiber-local storage to get an improved version of thread-local storage.

Raymond Chen

Follow