Why does my thread pool use only one thread?

devblogs.microsoft.com/oldnewthing/20070403-00

April 3, 2007

Raymond Chen

The thread pool is about reducing thread creating/termination overhead by consolidating work that would normally go onto separate threads into a small number of threads. In a sense, you shouldn't be surprised that the thread pool is using only one thread; instead, you should be happy!

> I switched to using the thread pool, and I'm finding that it's using only one thread. To demonstrate this, I wrote a test program that fires off a bunch of "work items" into the thread pool via `QueueUserWorkItem`. Each work item does some intensive computations. What I'm seeing is that they are all running serially on a single thread instead of running in parallel. Since I have a dual-processor machine, this leaves half of the computing capacity unutilized. If I create a separate thread for each "work item", then I get (not surprising) multiple threads and 100% CPU utilization. Why does my thread pool use only one thread?

The purpose of the thread pool, as I noted above, was to reduce the overhead of creating and terminating threads by running multiple tasks on a thread. For example, suppose you have three short tasks, say 1ms each. If you put each one on its own thread, you have

- Task1.CreateThread, Task1.Run, Task1.EndThread
- Task2.CreateThread, Task2.Run, Task2.EndThread
- Task3.CreateThread, Task3.Run, Task3.EndThread

Now suppose, for the purpose of this discussion, that creating and terminating a thread take 1ms each. if you create a separate thread for each task, you've spent 6ms on thread overhead and only 3ms doing actual work.

What if we could run multiple tasks on a single thread? That way, the cost of creating and terminating the thread could be amortized over all the tasks.

> ThreadPool.CreateThread, Task1.Run, Task2.Run, Task3.Run, ThreadPool.EndThread

Ah, now we have only 2ms of overhead for 3ms of work. Not great, but certainly better than what we had before. If we can pack more tasks into the thread pool, the fixed overhead of creating and terminating the thread becomes proportionally less.

The thread pool is designed for handling a collection of brief tasks, since those are the tasks that would best benefit from thread pooling. If you had a task that ran for ten seconds, putting it on the thread pool wouldn't yield much in the way of savings; that 2ms overhead you avoided is just noise compared to your ten seconds of running time. (Last year, we saw another case of a series of tasks ill-suited to thread pooling.)

As an accommodation for people who will put the occasional long-running task onto the thread pool (perhaps because it simplifies the program logic by treating everything as a work item), the thread pool allows you to give it a heads-up by passing the `WT_EXECUTELONGFUNCTION` flag. But that's not really what the thread pool is for. It's for quick-running tasks for which the overhead of creating a separate thread would be disproportionate to the work done by the task itself.

Raymond Chen

Follow