How can I close a thread pool and cancel all work that had been queued to it?

devblogs.microsoft.com/oldnewthing/20240205-00

February 5, 2024

Raymond Chen

When you close a custom Win32 thread pool, the call to `CloseThreadpool` returns immediately, but the actual closure doesn't occur until all pending work has completed. What if you want to abandon any work that hasn't yet begun?

What you can do is put the work items in a thread pool cleanup group, and when you want to abandon all of the pending callbacks, call `CloseThreadpoolCleanupGroupMembers` and pass `TRUE` for `fCancelPendingCallbacks`. The function abandons any callbacks that haven't started but waits for for callbacks that have already started.[1]

Here's a sample, with no error checking.

```
auto pool = CreateThreadpool(0);
auto group = CreateThreadpoolCleanupGroup();

TP_CALLBACK_ENVIRON env;
InitializeThreadpoolEnvironment(&env);
SetThreadpoolCallbackPool(&env, pool);
SetThreadpoolCallbackCleanupGroup(&env, group, nullptr);

// Queue up 100 slow-running tasks.
auto callback = [](auto instance, auto context, auto work)
    {
        printf("%d\n", PtrToInt(context));
        Sleep(100);
    };
for (int i = 0; i < 100; i++)
{
    auto work = CreateThreadpoolWork(callback, IntToPtr(i), &env);
    SubmitThreadpoolWork(work);
}

// Shut down the entire group.
// Any callbacks that haven't started will be abandoned.
CloseThreadpoolCleanupGroupMembers(group, TRUE, nullptr);

// Clean up the thread pool when done.
CloseThreadpool(pool);
```

[1] If you want the ones that have already started to respond to the shutdown, you can create a custom signal and have each callback explicitly periodically check that custom signal themselves.