Using thread pool cleanup groups to clean up many things at once

devblogs.microsoft.com/oldnewthing/20150209-00

February 9, 2015

Raymond Chen

Today's Little Program demonstrates thread pool cleanup groups. When you associate a thread pool item with a cleanup group, you can perform bulk operations on the group. That can save you a lot of bookkeeping.

Remember that Little Programs do little to no error checking.

```
#include <windows.h>
#include <stdio.h> // horrors! Mixing stdio and C++!

VOID
CALLBACK
Callback(
    PTP_CALLBACK_INSTANCE Instance,
    PVOID                 /* Parameter */,
    PTP_TIMER             /* Timer */
    )
{
    // Say what time the callback ran.
    printf("%p at %d\n", Instance, GetTickCount());
}

int
__cdecl
main(int, char**)
{
    // Create an environment that we use for our timers.
    TP_CALLBACK_ENVIRON environ;
    InitializeThreadpoolEnvironment(&environ);

    // Create a thread pool cleanup group and associate it
    // with the environment.
    auto cleanupGroup = CreateThreadpoolCleanupGroup();
    SetThreadpoolCallbackCleanupGroup(&environ,
                                      cleanupGroup,
                                      nullptr);

    // Say what time we started
    printf("Start: %d\n", GetTickCount());

    // Ask for a one-second delay
    LARGE_INTEGER dueTime;
    dueTime.QuadPart = -10000LL * 1000; // one second
    FILETIME ftDue = { dueTime.LowPart, dueTime.HighPart };

    // Create ten timers to run after one second.
    for (int i = 0; i < 10; i++) {
        auto timer = CreateThreadpoolTimer(Callback,
                                           nullptr,
                                           &environ);
        SetThreadpoolTimer(timer, &ftDue, 0, 500);
    }

    // Wait a while - the timers will run.
    Sleep(1500);

    // Clean up the group.
    CloseThreadpoolCleanupGroupMembers(cleanupGroup,
                                       FALSE,
```

```
                                        nullptr);

    // Close the group.
    CloseThreadpoolCleanupGroup(cleanupGroup);
}
```

There is some trickiness in building the `FILETIME` structure to specify that we want to run after a one-second delay. First, the value is negative to indicate a relative timeout. Second, we cannot treat the `FILETIME` as an `__int64`, so we use a `LARGE_INTEGER` as an intermediary.

When we create the ten timers, we associate them with the environment, which is in turn associated with the cleanup group. This puts all the timers into the cleanup group, which is a good thing, because we didn't save the timer handles!

When it's time to clean up the timers, we use `CloseThreadpoolCleanupGroupMembers`, which does the work of closing each individual timer in the cleanup group. This saves us the trouble of having to remember all the timers ourselves and manually closing each one.

For our next trick, comment out the `Sleep(1500);` and run the program again. This time, the timers don't run at all. That's because we closed them before they reached their due time. We let the cleanup group do the bookkeeping for us.

Raymond Chen

Follow