C++ coroutines: Waiting synchronously for our coroutine to complete

devblogs.microsoft.com/oldnewthing/20210430-00

April 30, 2021

Raymond Chen

Last time, we added an extension point that permitted us to respond differently to the completion of the coroutine. We're going to put that extension point to good use by adding the ability to wait synchronously for the coroutine to complete.

```
namespace async_helpers::details
{
    template<typename T>
    struct simple_task_base
    {
        ...

        T get() &&
        {
            if (!promise->client_await_ready()) {
                bool completed = false;
                if (promise->client_await_suspend(
                    &completed, wake_by_address)) {
                    bool ready = true;
                    while (!completed) {
                        WaitOnAddress(&completed, &ready,
                            sizeof(completed), INFINITE);
                    }
                }
            }
            return std::exchange(promise, {})->client_await_resume();
        }

        ...
    private:
        ...

        static void CALLBACK
            wake_by_address(void* completed)
        {
            *reinterpret_cast<bool*>(completed) = true;
            WakeByAddressSingle(completed);
        }

    };
}
```

To wait synchronously for the coroutine to complete, we first check if the coroutine is already finished. If so, then we're done, and we can skip the waiting step.

Otherwise, the coroutine is still running, so we register our suspension with a pointer and a callback function. This time, the pointer is a pointer to a variable that we will set when the coroutine is complete, and the callback is a function that sets that variable. In our case, we use `WaitOnAddress` to create a synchronization object out of nothing.

If `client_await_suspend` returns `false`, then it means that the coroutine has already completed while we were preparing to suspend, so we should skip the suspend and go straight to the resume step.

Finally, we ask `client_await_resume` to obtain the completed value and return it. We use `std::exchange` to cause the `promise_ptr` to be emptied after we get the completed value, thereby consuming it and freeing the coroutine state.

Synchronously waiting for a coroutine is always a risky proposition because the coroutine you're waiting for might need to use the thread that you are waiting on. This is particularly true if you perform the wait from a single-threaded COM apartment, because the coroutine will probably need to get back to the original thread to continue its COM work, which it can't do because you've blocked the original thread in order to wait for the coroutine.

So don't do that.

Next time, we'll look at task interconvertibility.

Raymond Chen

Follow