Awaiting a set of handles with a timeout, part 2: Continuing with two

devblogs.microsoft.com/oldnewthing/20240501-00

May 1, 2024

Raymond Chen

Last time, we tried to await two handles with a common timeout, but ran into a few problems. First, the timeouts didn't all start simultaneously, but rather sequentially. Second, we are taking a chance by saving the awaiter into a local variable rather than awaiting it immediately.

We can address both problems by awaiting the `resume_on_signal` immediately, but wrapping the whole thing inside another coroutine whose return type has well-defined copy behavior.

```
winrt::Windows::Foundation::IAsyncOperation<bool>
    resume_on_one_signal(HANDLE h,
        winrt::Windows::Foundation::TimeSpan timeout)
{
    co_return co_await winrt::resume_on_signal(h, timeout);
}

wil::task<std::array<bool, 2>>
    resume_on_both_signaled(HANDLE h1, HANDLE h2,
        winrt::Windows::Foundation::TimeSpan timeout = {})
{
    auto await1 = resume_on_one_signal(h1, timeout);
    auto await2 = resume_on_one_signal(h2, timeout);
    co_return std::array<bool, 2>
        { co_await await1, co_await await2 };
}
```

We solved one problem but introduced another: The C++/WinRT awaiter for `IAsyncOperation` (and all of the other asynchonous actions and operations in the Windows Runtime) resumes execution in the original apartment. In this case, it means that we end up hopping back to the original apartment, only to perform another `co_await` immediately. It would be better if we didn't have to keep bouncing through that original apartment, especially since we aren't even sure that the original apartment will still be there.

Fortunately, C++/WinRT has a way to say that you want to override the default `co_await` behavior for `IAsyncXxx` and allow the coroutine to resume in any apartment.

```
wil::task<std::array<bool, 2>>
    resume_on_both_signaled(HANDLE h1, HANDLE h2,
        winrt::Windows::Foundation::TimeSpan timeout = {})
{
    auto await1 = resume_on_one_signal(h1, timeout);
    auto await2 = resume_on_one_signal(h2, timeout);
    co_return std::array<bool, 2>
        { co_await winrt::resume_agile(await1),
          co_await winrt::resume_agile(await2) };
}
```

Next time, we'll try to generalize this to an arbitrary number of handles.