In C++/WinRT, you shouldn’t destroy an object while you’re co_awaiting it

devblogs.microsoft.com/oldnewthing/20240307-00

March 7, 2024

Raymond Chen

A customer ran into a problem where they were crashing inside a `co_await`. Here’s a minimal version:

```
struct MyThing : winrt::implements<MyThing, winrt::IInspectable>
{
    winrt::IAsyncAction m_pendingAction{ nullptr };

    winrt::IAsyncAction DoSomethingAsync() {
        auto lifetime = get_strong();
        m_pendingAction = LongOperationAsync();
        co_await m_pendingAction;
        PostProcessing();
    }

    void Cancel() {
        if (m_pendingAction) {
            m_pendingAction.Cancel();
            m_pendingAction = nullptr;
        }
    }
};
```

With this class, you can ask it to `DoSomethingAsync()`, and it will set into motion some long asynchronous operation, and then do some post-processing of the result. If you are impatient, you can call `MyThing::Cancel()` to give up on that long operation. For simplicity, we’ll assume that all calls occur on the same thread.

What the customer found was that after calling `MyThing::Cancel()`, the `co_await` crashed on a null pointer.

Some time ago, I set down some basic ground rules for function parameters, and one of them was that function parameters must remain stable for the duration of a function call.

In this case, it's not so much a function call as it is a function suspension, so the rule doesn't apply exactly, but the spirit is the same. The `MyThing::Cancel()` method cancels the `m_pendingAction`, which is fine, but it also modifies `m_pendingAction` out from under the `co_await` that is using it! When you call `m_pendingAction.Cancel()`, that cancels the `LongOperationAsync()`, which completes the `IAsyncAction`. At this point, the `co_await` will resume and call `m_pendingAction.GetResults()` to rethrown any errors.

But `m_pendingAction` was nulled out by the `MyThing::Cancel()`, so it's calling `GetResults()` on a null pointer, which leads to the null pointer crash.

The solution here is not to `co_await` the member variable directly, but rather to make a copy and `co_await` the copy. One way to do that is to copy to a local and await the local.

```
winrt::IAsyncAction DoSomethingAsync() {
    auto lifetime = get_strong();
    m_pendingAction = LongOperationAsync();
    // Await a copy of m_pendingAction because Cancel()
    // modifies m_pendingAction.
    auto pendingAction = m_pendingAction;
    co_await pendingAction;
    PostProcessing();
}
```

Another is to create a copy as an inline temporary by doing a conversion to itself.

```
winrt::IAsyncAction DoSomethingAsync() {
    auto lifetime = get_strong();
    m_pendingAction = LongOperationAsync();
    // Await a copy of m_pendingAction because Cancel()
    // modifies m_pendingAction.
    co_await winrt::IAsyncAction(m_pendingAction);
    PostProcessing();
}
```

We'll look some more at ways to force a copy next time.