The case of the memory corruption from a coroutine that already finished

devblogs.microsoft.com/oldnewthing/20221014-00

October 14, 2022

Raymond Chen

A customer was getting sporadic crashes in the following code fragment:

```
class Widget : WidgetT<Widget>
{
public:
    winrt::IAsyncOperation<bool> InitializeAsync();

private:
    winrt::IAsyncAction GetHighScoreAsync();
    winrt::IAsyncAction GetNameAsync();
    winrt::IAsyncAction GetPictureAsync();

    winrt::hstring m_name{ L"(anonymous)" };
    winrt::SoftwareBitmap m_picture{ DefaultPicture() };
    std::optional<int32_t> m_highScore;
}

winrt::IAsyncAction Widget::GetHighScoreAsync()
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_highScore = co_await GetHighScoreFromServer();
}

winrt::IAsyncAction Widget::GetNameAsync()
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_name = co_await GetNameFromIdentityService();
}

winrt::IAsyncAction Widget::GetPictureAsync()
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_picture = co_await DecodePictureFromSettings();
}

winrt::IAsyncOperation<bool>
    Widget::InitializeAsync()
{
    auto lifetime = get_strong();

    try {
        // Get information in parallel. Faster!
        co_await winrt::when_all(
            GetHighScoreAsync(),
            GetNameAsync(),
            GetPictureAsyncAsync());
    } catch (...) {
        // Service is unavailable or something
        // else went wrong. Just proceed with whatever
        // worked.
    }

    ShowHighScore(m_highScore);
    BuildGreeting(m_name);
    CropPicture(m_picture);
}
```

The idea here is that they have an `InitializeAsync` coroutine function that wants to run a bunch of other coroutines to initialize stuff, and let those other coroutines run in parallel, since each one is doing something different. When all of the helper coroutines are done, we process the results. And if any of the helper coroutines fails, that's okay. We just proceed with what we were able to get.

The crashes, though, indicated that `BuildGreeting` or `CropPicture` were crashing on their accesses to `m_name` and `m_picture`.

Let's take a survey of how various programming languages allow you to wait for multiple asynchronous actions:

Language	Method	Result	If any fail
C++	Concurrency::when_all	`vector<T>`	fail immediately
C++	winrt::when_all	`void`	fail immediately
C#	Task.WhenAll	`T[]`	wait for others
JavaScript	Promise.all	`Array`	fail immediately
JavaScript	Promise.allSettled	`Array`	wait for others
Python	asyncio.gather	List	fail immediately by default
Rust	join!	tuple	wait for others
Rust	try_join!	tuple	fail immediately

Python's `asynchio.gather` lets you choose whether a failed coroutine causes `gather` to fail immediately or to wait for others before failing. The default is to fail immediately.

This customer is using `winrt::when_all`, which (consults table) fails as soon as any coroutine fails. (Our custom `when_all` has the same behavior.)

What happened is that one of the coroutines, let's say `GetHighScoreAsync` failed with an exception. That caused `winrt::when_all` to propagate the exception and abandon waiting on the other coroutines. The `InitializeAsync` method ignores the exception and then proceeds on the false assumption that all of the methods ran to complete (possibly with failure). When it tries to use the `m_name`, it races against the still-running `GetNameAsync` method, causing the `L"(anonymous)"` string to be destructed at the same time it is being copied, which does not end well. A similar race occurs when `CropPicture` reads the `m_picture` while `GetPictureAsync` is writing to it.

The simple solution here is to catch the exceptions in the coroutines so that they never produce a failure. That way, `winrt::when_all` never completes early.

```
winrt::IAsyncAction Widget::GetHighScoreAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_highScore = co_await GetHighScoreFromServer();
} catch (...)
{
}

winrt::IAsyncAction Widget::GetNameAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_name = co_await GetNameFromIdentityService();
} catch (...)
{
}

winrt::IAsyncAction Widget::GetPictureAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_picture = co_await DecodePictureFromSettings();
} catch (...)
{
}

winrt::IAsyncOperation<bool>
    Widget::InitializeAsync()
{
    auto lifetime = get_strong();

    // try {
    // Get information in parallel. Faster!
    co_await winrt::when_all(
        GetHighScoreAsync(),
        GetNameAsync(),
        GetPictureAsyncAsync());
    // } catch (...) {
    //     // Service is unavailable or something
    //     // else went wrong. Just proceed with whatever
    //     // worked.
    // }

    ShowHighScore(m_highScore);
    BuildGreeting(m_name);
    CropPicture(m_picture);
}
```

This code happens to use WIL, so there’s a helper macro for catching exceptions and logging them.

```
winrt::IAsyncAction Widget::GetHighScoreAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_highScore = co_await GetHighScoreFromServer();
} CATCH_LOG()

winrt::IAsyncAction Widget::GetNameAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_name = co_await GetNameFromIdentityService();
} CATCH_LOG()

winrt::IAsyncAction Widget::GetPictureAsync() try
{
    auto lifetime = get_strong();
    co_await winrt::resume_background();

    m_picture = co_await DecodePictureFromSettings();
} CATCH_LOG()
```

Raymond Chen

Follow