Why am I getting an unhandled exception from my C++ function that catches all exceptions?

devblogs.microsoft.com/oldnewthing/20230223-00

February 23, 2023

Raymond Chen

A customer had what they thought was a problem with C++/WinRT coroutines. “We are catching and handling all exceptions, but sometimes our program still crashes with an unhandled exception.”

```
winrt::fire_and_forget MyClass::DoSomethingAsync()
{
    auto lifetime = get_strong();
    try {
        auto name = co_await m_user.GetNameAsync();
        m_label.Text(name);
    } catch (...) {
        m_label.Text(L"unknown");
    }
}
```

The C++/WinRT knowledge you need to know here is that a coroutine that returns `fire_and_forget` terminates the application if an unhandled exception is encountered.

And you can see the unhandled exception in the stack trace:

```
KERNELBASE!RaiseFailFastException+0x15c
combase!RoFailFastWithErrorContextInternal2+0x43a
contoso!winrt::terminate+0x28
contoso!std::experimental::coroutine_traits<winrt::fire_and_forget>::promise_type::unh

contoso!`MyClass::DoSomethingAsync$_ResumeCoro$1'::`1'::catch$2+0x1f
VCRUNTIME140_1+0x1080
VCRUNTIME140_1!_NLG_Return2+0x1555
ntdll!RcFrameConsolidation+0x6
contoso!MyClass::DoSomethingAsync$_ResumeCoro$1+0xa9
contoso!std::experimental::coroutine_handle<void>::resume+0x5
contoso!std::experimental::coroutine_handle<void>::operator()+0x5
contoso!winrt::impl::resume_apartment_callback+0x9
...
```

But we did a `catch (...)`, which catches all exceptions. How did we get an unhandled exception?

The `catch (...)` catches all exceptions thrown in the preceding `try` block. But that's not where the unhandled exception is coming from.

The coroutine was just a red herring. Let's take coroutines out of the picture and make this a non-coroutine function.

```
void MyClass::DoSomething()
{
    try {
        auto name = m_user.GetName();
        m_label.Text(name);
    } catch (...) {
        m_label.Text(L"unknown");
    }
}
```

If an exception occurs in the `try` block, it is caught and handled by the `catch (...)` block. But if an exception occurs at the `m_label.Text(L"unknown")`, there's nobody around to catch the *second* exception.

You thought you caught the exception, but instead you merely caught *an* exception. If you don't want any exceptions to escape your function, you have to play Pokémon and catch them all.

```
winrt::fire_and_forget MyClass::DoSomethingAsync()
{
    auto lifetime = get_strong();
    try {
        auto name = co_await m_user.GetNameAsync();
        m_label.Text(name);
    } catch (...) {
        try {
            m_label.Text(L"unknown");
        } catch (...) {
            LOG_CAUGHT_EXCEPTION();
        }
    }
}
```

I'm assuming that if you can't even set the label to `L"unknown"` you just want to log the error and proceed anyway. For demonstration purposes, I'm using the WIL error handling helpers.

The nesting here is getting rather annoying, but you can make things a little less awkward by using a function try block.

```
winrt::fire_and_forget MyClass::DoSomethingAsync() try
{
    auto lifetime = get_strong();
    try {
        auto name = co_await m_user.GetNameAsync();
        m_label.Text(name);
    } catch (...) {
        m_label.Text(L"unknown");
    }
} catch (...) {
    // The function is best-effort. Ignore failures.
}
```

The function try lets you specify catch blocks that apply to the entire function body.

Next time, we'll look at some of the subtle exceptions that can come out of C++/WinRT and XAML.