The case of the COM reference that suddenly went bad in the middle of a coroutine

devblogs.microsoft.com/oldnewthing/20220601-00

June 1, 2022

Raymond Chen

A customer reported a crash in a coroutine and needed help understanding it.

Here's the crash:

```
contoso!winrt::impl::consume_Contoso_IWidgetOptions<winrt::Contoso::IWidgetOptions>::
Name+0x15:
00007ff8`fa6c77e1 mov     rax,qword ptr [rcx] ds:00000000`00000000=????????????????
```

And the stack trace:

```
contoso!winrt::impl::consume_Contoso_IWidgetOptions<winrt::Contoso::IWidgetOptions>::
Name+0x15
contoso!winrt::Contoso::implementation::Widget::CreateAsync$_ResumeCoro$1+0xfe
contoso!winrt::Contoso::implementation::Widget::CreateAsync$_InitCoro$2+0x95
contoso!winrt::Contoso::implementation::Widget::CreateAsync+0x63
contoso!winrt::Contoso::implementation::Gadget::CreateWidgetAsync+0x7e
contoso!winrt::impl::produce<winrt::Contoso::implementation::Gadget,winrt::Contoso::
IGadget>::CreateWidgetAsync+0x35
litware!winrt::impl::consume_Contoso_Gadget<winrt::Contoso::Gadget>::
CreateWidgetAsync+0x47
litware!winrt::LitWare::implementation::GadgetViewer::
CreateWidgetsAsync$_ResumeCoro$2+0x343
litware!std::experimental::coroutine_handle<void>::resume+0xc
litware!std::experimental::coroutine_handle<void>::operator()+0xc
litware!winrt::impl::resume_background_callback+0x10
ntdll!TppSimplepExecuteCallback+0xa3
ntdll!TppWorkerThread+0x686
kernel32!BaseThreadInitThunk+0x10
ntdll!RtlUserThreadStart+0x2b
```

This is crashing inside the C++/WinRT projection, which you can infer because we crashed inside a `consume_` function. The `consume_` functions are provided by the C++/WinRT library to convert your C++/WinRT calls into low-level ABI calls. So something happened inside that projection.

Here's the consume function up to the point of the crash:

```
    push    rbx
    sub     rsp,20h
    mov     rcx,qword ptr [rcx]   ; get the raw pointer from the IWidgetOptions
    and     qword ptr [rsp+30h],0 ; pre-null the output parameter
    mov     rbx,rdx
    mov     rax,qword ptr [rcx]   ; Load the vtable
```

The problem it seems is that the `this` parameter to `IWidgetOptions::Name()` is a COM wrapper around a null pointer.

We are called from `_InitCoro`, which runs the initial synchronous portion, so we are still in the synchronous portion of the coroutine. That's nice, because it means that the caller is still on the stack:

```
IAsyncOperation<Widget> Widget::CreateAsync(const WidgetOptions& options)
{
    auto name = options.Name();
```

Let's see what we got as the `options`:

```
0:018> .frame 3
03 000000bb`dccff630 00007ff8`fa6c7b32     contoso!Widget::CreateAsync+0x63
0:018> dv
     options = 0x000000bb`dccff978
0:018> ?? options
struct winrt::Contoso::IWidgetOptions * 0x000000bb`dccff978
   +0x000 m_ptr            : (null)
```

Yes indeed, the `options` is null. That's why we crash trying to call a method on it.

The caller of `Widget::CreateAsync` is `Gadget::CreateWidgetAsync`:

```
IAsyncOperation<Widget> Gadget::CreateWidgetAsync()
{
    return Widget::CreateAsync(m_options);
}
```

The customer suspected that this function was incorrectly implemented. Shouldn't it be this?

```
IAsyncOperation<Widget> Gadget::CreateWidgetAsync()
{
    co_return co_await Widget::CreateAsync(m_options);
}
```

"The immediate return might be losing some necessary coroutine frame lifetime, so that when `GadgetViewer::CreateWidgetsAsync` completes, it's operating on already-freed memory. However, I'm not confident in this analysis."

The function is fine. While it's true that the common way of producing a `IAsync-Operation<Widget>` is to autogenerate one from a coroutine, it is also perfectly legal to just create one by other means and just return it.

Let's try to walk back up the stack to find out what the `m_options` were that we *thought* we were passing in.

```
0:018> .frame 4
04 000000bb`dccff930 00007ff8`fa6c7a55 contoso!winrt::Contoso::implementation::
Gadget::CreateWidgetAsync+0x7e
0:018> dv
           this = <value unavailable>
```

Rats, the `this` pointer got optimized out. Keep going.

```
0:018> .frame 5
05 000000bb`dccff970 00007ff8`be19b2ff contoso!winrt::impl::produce<winrt::Contoso::
implementation::Gadget,winrt::Contoso::IGadget>::CreateWidgetAsync+0x35
0:018> dv
           this = <value unavailable>
      operation = 0x000000bb`dccff9d0

0:018> .frame 6
06 000000bb`dccff9a0 00007ff8`be109963     litware!winrt::impl::
consume_Contoso_Gadget<winrt::Contoso::Gadget>::CreateWidgetAsync+0x47
0:018> dv
           this = <value unavailable>
      operation = 0x00000000`00000000

0:018> .frame 7
07 000000bb`dccff9f0 00007ff8`be0d4b40     litware!winrt::LitWare::implementation::
GadgetViewer::CreateWidgetsAsync$_ResumeCoro$2+0x343
0:018> dv
      <coro_frame_ptr> = 0x00000276`15223b00
            strongThis = struct winrt::com_ptr<winrt::LitWare::implementation::
GadgetViewer>
```

We had to chase all the way back to `GadgetViewer::CreateWidgetsAsync` before we got a foothold into the thing that will lead us to the `options`. Now we can start working our way back in.

The `GadgetViewer::CreateWidgetsAsync` coroutine looks like this:

```
IAsyncOperation<IVectorView<MenuItem>> GadgetViewer::CreateWidgetsAsync()
{
    ...

    const auto strongThis{ get_strong() };

    co_await winrt::resume_background();

    std::vector<winrt::Contoso::Widget> widgets;

    if (const auto widget1 = co_await m_gadget.CreateWidgetAsync())
    {
        ...
```

Aha, so the outbound call to `CreateWidgetAsync` is made on the `GadgetViewer`'s `m_gadget`. Follow the call:

```
0:018> ?? strongThis
struct winrt::com_ptr<winrt::Contoso::implementation::GadgetViewer>
   +0x000 m_ptr            : 0x00000276`184121f0 winrt::Contoso::implementation::
GadgetViewer
0:018> ??((winrt::Contoso::implementation::GadgetViewer*) 0x00000276`184121f0)-
>m_gadget
struct winrt::Contoso::Gadget
   +0x000 m_ptr            : 0x00000276`0f68d4e0 winrt::impl::abi<winrt::Windows::
Foundation::IUnknown,void>::type
```

Okay, we've found the `m_gadget`. Now to the options.

```
0:018> dt contoso!winrt::Contoso::implementation::Gadget
   +0x010 vtable           : winrt::impl::produce<winrt::Contoso::implementation::
Gadget,winrt::Contoso::IGadget>
   +0x000 __VFN_table : Ptr64
   +0x008 m_references     : std::atomic<unsigned __int64>
   +0x018 m_options        : winrt::Contoso::WidgetOptions
   +0x020 m_source         : winrt::Contoso::GadgetSource
   +0x028 m_home           : std::basic_string<wchar_t,std::char_traits<wchar_t>,
std::allocator<wchar_t> >
```

We see that the vtable for the producer of `IGadget` is at offset `0x010`, so we subtract that amount from our `winrt::Contoso::Gadget` pointer to get a pointer to the implementation.

```
0:018> ?? ((contoso!winrt::Contoso::implementation::Gadget*)(0x00000276`0f68d4e0-
0x10))
struct winrt::Contoso::implementation::Gadget * 0x00000276`0f68d4d0
   +0x010 vtable           : winrt::impl::produce<winrt::Contoso::implementation::
Gadget,winrt::Contoso::IGadget>
   +0x000 __VFN_table : 0x00007ff8`fa6df2e8
   +0x008 m_references     : std::atomic<unsigned __int64>
   +0x018 m_options        : winrt::Contoso::WidgetOptions
   +0x020 m_source         : winrt::Contoso::GadgetSource
   +0x028 m_home           : std::basic_string<wchar_t,std::char_traits<wchar_t>,
std::allocator<wchar_t> >

0:018> ?? &((contoso!winrt::Contoso::implementation::Gadget*)(0x00000276`0f68d4e0-
0x10))->m_options
struct winrt::Contoso::WidgetOptions * 0x00000276`0f68d4e8
   +0x000 m_ptr            : 0x00000276`152f6270 winrt::impl::abi<winrt::Windows::
Foundation::IUnknown,void>::type
```

The address of this `m_options` is `0x00000276`0f68d4e8` , which is not the same as the address that was passed to `Widget::CreateAsync` :

```
0:018> .frame 3
03 000000bb`dccff630 00007ff8`fa6c7b32     contoso!Widget::CreateAsync+0x63
0:018> dv
     options = 0x000000bb`dccff978
```

What happened? How did the address of a variable change?

Studying the code in the `GadgetViewer` that uses the `m_gadget` member variable, we see that the member variable is used from both the foreground thread as well as from a background thread:

```
// Property setter: Set the "Gadget" property of the GadgetViewer
void GadgetViewer::Gadget(winrt::Contoso::Gadget const& value)
{
    VerifyUIThread();

    if (m_gadget != value)
    {
        m_gadget = value;
        ...
    }
}
```

Recall that one of the principles of debugging somebody else's code is to assume that the code is mostly correct. The problem is likely in a small detail, an edge case, or a peculiar combination of factors. After all, if the problem was in a common case, they probably wouldn't have had to ask an outsider for help.

The `VerifyUIThread` call tells us that the expectation is that the gadget is changed only from the UI thread. But there is no synchronization to protect access to this variable from multiple threads, even though we are accessing it from a background thread:

```
IAsyncOperation<IVectorView<MenuItem>> GadgetViewer::CreateWidgetsAsync()
{
    ...

    const auto strongThis{ get_strong() };

    co_await winrt::resume_background(); // ← hop to background thread

    std::vector<winrt::Contoso::Widget> widgets;

    if (const auto widget1 = co_await m_gadget.CreateWidgetAsync())
    {
        ...
```

What may have happened is that while `CreateWidgetsAsync` was using `m_gadget` on the background thread, the foreground thread changed the `m_gadget`, causing the old gadget (and its `m_options`) to be destructed while the background thread was still using it.

The customer provided some history: As originally written, the `GadgetViewer` accessed the `m_gadget` only from the foreground thread, but a subsequent change moved some of the work to a background thread, and that introduced concurrency into code that was written on the assumption that there was no concurrency.

One possible solution is for `GadgetViewer::CreateWidgetsAsync` to capture the member variables it intends to use (the `m_gadget`, in this case, but possibly other member variables not seen here) before going to the background thread, and operating entirely on the captured variables. It means that when you call `CreateWidgetsAsync`, you get the widgets associated with the gadget that you were viewing at the point you called `CreateWidgetsAsync`, even if you changed the gadget while the `CreateWidgetsAsync` was still working.

Raymond Chen

Follow