The case of COM failing to pump messages in a single-threaded COM apartment

devblogs.microsoft.com/oldnewthing/20250314-00

Raymond Chen March 14, 2025

A customer encountered a hang caused by COM not pumping messages while waiting for a cross-thread operation to complete. They were using the `task_sequencer` class for serializing asynchronous operations on a UI thread they created to handle accessibility callbacks.

The hang stack looked like this:

```
ntdll!ZwWaitForMultipleObjects+0x4
KERNELBASE!WaitForMultipleObjectsEx+0xe0
combase!MTAThreadDispatchCrossApartmentCall+0x3a0
combase!CSyncClientCall::SendReceive2+0x65c
combase!DefaultSendReceive+0x88
combase!CSyncClientCall::SendReceive+0x390
combase!CClientChannel::SendReceive+0xc0
combase!NdrExtpProxySendReceive+0x68
rpcrt4!NdrpClientCall3+0x764
combase!ObjectStublessClient+0x180
combase!ObjectStubless+0x34
combase!CObjectContext::InternalContextCallback+0x3f0
combase!CObjectContext::ContextCallback+0x80
contoso!winrt::impl::resume_apartment_sync+0x58
contoso!winrt::impl::resume_apartment+0xe8
contoso!winrt::impl::apartment_awaiter::await_suspend+0x6c
contoso!〚lambda...〛::operator()<〚...〛>+0x1c8
contoso!task_sequencer::chained_task::continue_with+0x38
contoso!task_sequencer::QueueTaskAsync<〚...〛>+0xd0
contoso!〚lambda...〛::<lambda_invoker_cdecl>+0xa0
user32!__ClientCallWinEventProc+0x34
ntdll!KiUserCallbackDispatcherReturn
win32u!ZwUserGetMessage+0x4
user32!GetMessageW+0x28
contoso!〚lambda...〛::operator()+0x204
contoso!std::thread::_Invoke<〚lambda...〛>+0x24
ucrtbase!thread_start<unsigned int (__cdecl*)(void *),1>+0x48
kernel32!BaseThreadInitThunk+0x40
ntdll!RtlUserThreadStart+0x44
```

We see that we have a UI thread (notice the `GetMessage` at the bottom of the stack), yet COM decided to block without pumping messages (`WaitForMultipleObjectsEx` instead of (`MsgWaitForMultipleObjectsEx`).

Is this a bug in the task sequencer?

Let's look at the stack more closely. A message arrived via `__ClientCallWinEventProc`, and that then queued a task into the task sequencer. The `continue_with` saw that the task sequencer had no active task, so it ran the new task immediately. That new task wants to run on a different thread, so C++/WinRT's apartment-switching code kicked in.

The apartment-switching code went to resume_apartment_sync, which in turn called our friend `IContextCallback::ContextCallback`, and that called into the COM thread-switching infrastructure, which doesn't pump messages while wiating for the destination apartment to respond.

Now, COM is a rather mature technology, and this code path is execised constantly throughout the system, so it's unlikely that it simply "forgot" to pump messages. The function name `MTAThreadDispatchCrossApartmentCall` strongly suggests that COM thinks that the

thread is in an MTA. And the use of `resume_apartment_sync` suggests that C++/WinRT also thinks that the thread is in an MTA:

```
else if (is_sta_thread())
{
    resume_apartment_on_threadpool(
        context.m_context, handle, failure);
    return true;
}
else
{
    return resume_apartment_sync(
        context.m_context, handle, failure);
}
```

If this were an STA thread, then we would have called `resume_apartment_on_threadpool` instead of `resume_apartment_sync`.

Let's take a closer look at this thread:

```
// Create a thread to receive accessibility notifications.
m_thread = std::thread([this] {
    ::SetThreadDescription(::GetCurrentThread(), L"Accessibility STA");

    ⟦ ... ⟧

    wil::unique_hwineventhook hook(SetWinEventHook(⟦...⟧));
    THROW_LAST_ERROR_IF_NULL(hook);

    MSG msg;
    while (!m_stop && GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
});
```

Ah, so there's your problem.

The thread claims to be an STA thread:

```
    ::SetThreadDescription(::GetCurrentThread(), L"Accessibility STA");
```

But there is nothing in the thread procedure that actually makes it an STA thread. It never initialized COM in single-threaded mode.

The thread merely engaged in wishful thinking, proclaming itself to be an STA thread without actually becoming one. (Or maybe it believed in nominative determinism: The mere act of calling itself an STA thread was sufficient to make it true.)

Since COM is already initialized elsewhere in the process, the new thread gets put into the implicit MTA by default, and it took no action to leave it, so from COM's point of view, this thread is an MTA thread. And MTA threads are allowed to block without pumping messages.

What they need to do is actually make it an STA thread, say, by calling `CoInitializeEx` with the `COINIT_APARTMENTTHREADED` flag, and then uninitializing COM before the thread exits to return the thread to its original state. You can kill two birds with one stone with the help of the WIL RAII type.

```
// Create a thread to receive accessibility notifications.
m_thread = std::thread([this] {
    auto uninit = wil::CoInitializeEx(COINIT_APARTMENTTHREADED);

    ::SetThreadDescription(::GetCurrentThread(), L"Accessibility STA");

    ⟦ ... ⟧

    wil::unique_hwineventhook hook(SetWinEventHook(⟦...⟧));
    THROW_LAST_ERROR_IF_NULL(hook);

    MSG msg;
    while (!m_stop && GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
});
```