# Inside C++/WinRT: Apartment switching: Unwinding the stack

devblogs.microsoft.com/oldnewthing/20230127-00

January 27, 2023

Raymond Chen

Last time, we found a case where we could avoid calling `IContextCallback::ContextCallback()`, thereby reducing stack usage.

Another case where we can avoid having to call `IContextCallback::ContextCallback()` is the case where we are resuming on the multi-threaded apartment (MTA). In that case, we can just resume on a new threadpool thread, since threadpool tasks run in the implicit multi-threaded apartment. This is an even better way of reducing stack usage, because it releases the original thread entirely.

In order to know what kind of apartment we captured, we'll have to remember the apartment type as well as the `IContextCallback`. We'll keep both of them in a new structure:

```
struct resume_apartment_context
{
    resume_apartment_context() = default;
    resume_apartment_context(std::nullptr_t) :
        m_context(nullptr) {}

    bool valid() const noexcept
    {
        return m_context != nullptr;
    }

    com_ptr<IContextCallback> m_context =
        capture<IContextCallback>(WINRT_IMPL_CoGetObjectContext);
    int32_t m_context_type = get_apartment_type().first;
};
```

When switching apartments, we add a check to see if we are switching to the MTA. If so, then we just queue the coroutine contination directly to a threadpool thread.

```
inline auto resume_apartment(
    resume_apartment_context const& context,
    coroutine_handle<> handle)
{
    WINRT_ASSERT(context.valid());
    if (context.m_context ==
        capture<IContextCallback>(WINRT_IMPL_CoGetObjectContext))
    {
        handle();
    }
    else if (context.m_context_type == 1 /* APTTYPE_MTA /)
    {
        resume_background(handle);
    }
    else if (is_sta_thread())
    {
        resume_apartment_on_threadpool(context.m_context, handle);
    }
    else
    {
        resume_apartment_sync(context.m_context, handle);
    }
}
```

The `resume_background()` function resumes the coroutine on a threadpool thread:

```
inline void __stdcall resume_background_callback(
    void*, void* context) noexcept
{
    coroutine_handle<>::from_address(context)();
};

inline auto resume_background(coroutine_handle<> handle)
{
    submit_threadpool_callback(
        resume_background_callback, handle.address());
}
```

The `apartment_context` holds this structure instead of just the `IContextCallback`.

```
struct apartment_context
{
    apartment_context() = default;
    apartment_context(std::nullptr_t) : context(nullptr) { }

    operator bool() const noexcept { return context.valid(); }
    bool operator!() const noexcept { return !context.valid(); }

    resume_apartment_context context;
};
```

And the Windows Runtime awaiter captures it into the lambda:

```
template <typename Async>
struct await_adapter
{
    await_adapter(Async const& async) : async(async) { }

    Async const& async;

    bool await_ready() const noexcept
    {
        return false;
    }

    void await_suspend(std::experimental::coroutine_handle<> handle) const
    {
        auto extend_lifetime = async;
        async.Completed([
            handle,
            context = resume_apartment_context()
        ](auto&& ...)
        {
            resume_apartment(context.context, handle);
        });
    }

    auto await_resume() const
    {
        return async.GetResults();
    }
};
```

Next time, we'll look at another feature of C++/WinRT that needs to be added to our apartment switching framework.