How to get your C++/WinRT asynchronous operations to respond more quickly to cancellation, part 2

devblogs.microsoft.com/oldnewthing/20200723-00

July 23, 2020

Raymond Chen

We saw last time that you can hasten the cancellation of your C++/WinRT coroutine by polling for cancellation. But that works only if it's the top-level coroutine that needs to respond to the cancellation. But often, your coroutine calls out to other coroutines, and if one of those other coroutines takes a long time, your main coroutine won't get a chance to respond to the cancellation until it regains control.

Can we get the main coroutine to respond more quickly to cancellation?

Sure.

You can pass a custom delegate to the the C++/WinRT cancellation token's `callback` method to be notified immediately when the coroutine is cancelled.

```
IAsyncAction ProcessAllWidgetsAsync()
{
    auto cancellation = co_await get_cancellation_token();

    cancellation.callback([] { /* zomg! cancelled! */ });

    auto widgets = co_await GetAllWidgetsAsync();
    for (auto&& widget : widgets) {
        if (cancellation()) co_return;
        ProcessWidget(widget);
    }
    co_await ReportStatusAsync(WidgetsProcessed);
}
```

When the coroutine is cancelled, the cancellation callback is called immediately. This is your chance to hasten the death of your coroutine. For example, we could do so by cancelling the `GetAllWidgetsAsync` call.

```
IAsyncAction ProcessAllWidgetsAsync()
{
    auto cancellation = co_await get_cancellation_token();

    auto operation = GetAllWidgetsAsync();
    cancellation.callback([operation] { operation.Cancel(); });
    auto widgets = co_await operation;

    for (auto&& widget : widgets) {
        if (cancellation()) co_return;
        ProcessWidget(widget);
    }
    co_await ReportStatusAsync(WidgetsProcessed);
}
```

If the `ProcessAllWidgetsAsync` is cancelled, we propagate that cancellation to the `GetAllWidgetsAsync` operation, in the hopes that it will abandon its attempt to get all the widgets and give control back to `ProcessAllWidgetsAsync`. The `co_await` will fail with `hresult_canceled`, which will then propagate out of the coroutine, causing the entire coroutine to become cancelled.

This is a common enough pattern that you could write a wrapper for it:

```
template<typename Async, typename Token>
std::decay_t<Async> MakeCancellable(Async&& async, Token&& token)
{
    token.callback([async] { async.Cancel(); });
    return std::forward<Async>(async);
}
```

Now we just wrap our asynchronous operations inside a `MakeCancellable`:

```
IAsyncAction ProcessAllWidgetsAsync()
{
    auto cancellation = co_await get_cancellation_token();

    auto widgets = co_await MakeCancellable(GetAllWidgetsAsync(), cancellation);

    for (auto&& widget : widgets) {
        if (cancellation()) co_return;
        ProcessWidget(widget);
    }
    co_await MakeCancellable(ReportStatusAsync(WidgetsProcessed), cancellation);
}
```

Exercise: What happens if the `ProcessAllWidgetsAsync` is cancelled after `GetAllWidgets` has completed?

Don't give up yet: There's part 3.

Raymond Chen

Follow