Creating a co_await awaitable signal that can be awaited multiple times, part 1

devblogs.microsoft.com/oldnewthing/20210301-00

March 1, 2021

Raymond Chen

C++/WinRT asynchronous activities can be awaited only once. This is consistent with their intended usage pattern, which is for an application to start the activity, `co_await` the result, and then continue.

But maybe you want something like a Win32 event, where any number of people can `co_await` the event, and then once it is signaled, all the awaiters are resumed.

Well, an easy way to do this is simply to have a Win32 event!

```
struct awaitable_event
{
  void set() const noexcept
  { SetEvent(os_handle()); }

  auto operator co_await() const noexcept
  { return winrt::resume_on_signal(os_handle()); }

private:
  HANDLE os_handle() const noexcept
  { return handle.get(); }

  winrt::handle handle{
    winrt::check_pointer(CreateEvent(nullptr,
      /* manual reset */ true, /* initial state */ false,
      nullptr)) };
};
```

This class is just a wrapper around a Win32 manual-reset event handler. You can call the `set` method to set the event, and you can `co_await` it to wait for the event.

The traditional way of supporting `co_await` is to implement the trio of methods `await_ready`, `await_suspend`, and `await_resume`. But another way is to define the `co_await` operator so it returns an awaiter. We implement our custom `co_await` operator

by propagating the awaiter returned by `resume_on_signal`. Basically, awaiting the `awaitable_event` is the same as awaiting a call of `resume_on_signal` with the hidden handle.

For simple scenarios, this might be all you need. You can define a global `awaitable_event` and have as many people as you like `co_await` it.

If you want the object not to have static storage duration (say, because it's a member of another class which is dynamically-allocated), then you will encounter lifetime issues because you can't destruct the `awaitable_event` while somebody else is still awaiting it.

We'll continue investigating this issue next time.

Raymond Chen

Follow