Creating other types of synchronization objects that can be used with co_await, part 10: Wait for an event to clear

devblogs.microsoft.com/oldnewthing/20210322-00

March 22, 2021

Raymond Chen

We've been looking at creating different types of awaitable synchronization objects. This time, we're going to create something that doesn't exist in the normal Win32 repertoire: An event where you can wait for the event to be in a desired state, either set or reset. Normal Win32 events allow you to wait for them to be set, but you cannot wait for Win32 event to be reset. The usual workaround is to have two events.

Let's go.

```
struct awaitable_signal_state;

template<>
struct async_helpers::
    extra_await_data<awaitable_signal_state>
{
    extra_await_data(bool value = true) : desired(value) {}
    bool desired;
};
```

If you don't say whether you are awaiting the signal to be set or reset, we will assume set.

```
struct awaitable_signal_state :
    async_helpers::awaitable_state<awaitable_signal_state>
{
    awaitable_signal_state(bool initial)
    : signaled(initial) {}

    std::atomic<bool> signaled;

    bool fast_claim(extra_await_data const& e) noexcept
    {
        return signaled.load(std::memory_order_acquire)
            == e.desired;
    }

    bool claim(extra_await_data const& e) noexcept
    {
        return signaled.load(std::memory_order_relaxed)
            == e.desired;
    }

    void set_state(node_list& list, bool value) noexcept
    {
        if (signaled.load(std::memory_order_relaxed)
            != value) {
            signaled.store(value, std::memory_order_relaxed);
            resume_all(list);
        }
    }
};
```

If the signal is in the desired state, then we allow the await to complete immediately. Otherwise, we suspend the coroutine.

If the state changes, we resume everybody who was waiting for the state to change.

```
struct awaitable_signal
    : async_helpers::awaitable_sync_object<awaitable_signal_state>
{
    awaitable_signal(bool initial = false) :
        awaitable_sync_object(initial) { }

    auto when_state(bool desired)
    { return make_awaiter(desired); }

    auto when_set() { return when_state(true); }
    auto when_reset() { return when_state(false); }

    void set_state(bool desired) noexcept
    {
        action_impl(&state::set_state, desired);
    }

    void set() noexcept
    {
        action_impl(&state::set_state, true);
    }

    void reset() noexcept
    {
        action_impl(&state::set_state, false);
    }
};
```

The main class supports a plain `co_await` which defaults to waiting for the signal to be set. You can `co_await signal.when_set()` to wait for it to be set, or `co_await signal.when_reset()` to wait for it to be reset, or `co_await signal.when_state()` to wait for a specific state.

All of this playing around with awaitable synchronization objects is really just warm-up for the case of an awaitable object that produces a result. We'll take that up next time.

Raymond Chen

Follow