C++ coroutines: Getting rid of our atomic variant discriminator

devblogs.microsoft.com/oldnewthing/20210420-28

April 20, 2021

Raymond Chen

We continue the refinement of our coroutine implementation by removing the atomic variable used as the discriminant of our result holder variant.

The discriminant needed to be atomic because we used it in `await_ready` to peek at whether the coroutine had completed. But we've switched over to using the atomic `m_waiting` member to track the coroutine state, which means that the use of the discriminant is now protected by the memory ordering requirements of `m_waiting`. The discriminant itself can now be a regular variable.

```
template<typename T>
struct simple_promise_holder
{
    ...
    // std::atomic<result_status>
    result_status status
        { result_status::empty };

    template<typename...Args>
    void set_value(Args&&... args)
    {
        new (std::addressof(result.wrap))
            wrapper<T>{ std::forward<Args>(args)... };
        status = result_status::value;
    }

    void unhandled_exception() noexcept
    {
        new (std::addressof(result.error))
            std::exception_ptr(std::current_exception());
        status = result_status::error;
    }

    // bool is_empty() const noexcept
    // {
    //     return status.load(std::memory_order_relaxed) ==
    //         result_status::empty;
    // }

    T get_value()
    {
        switch (status) {
        case result_status::value:
            return result.wrap.get_value();
        case result_status::error:
            std::rethrow_exception(
                std::exchange(result.error, {}));
        }
        assert(false);
        std::terminate();
    }

    ~simple_promise_result_holder()
    {
        switch (status) {
        case result_status::value:
            result.wrap.~wrapper();
            break;
        case result_status::error:
            if (result.error)
                std::rethrow_exception(result.error);
            result.error.~exception_ptr();
```

```
            }
        }
    };
```

What used to be `status.store` is now just an assignment, and what used to be `status.load` now just a read.

We can also get rid of the `is_empty` method, since it was used only by our previous version of `client_await_ready`, which we abandoned when we switched to using `m_waiting`.

Next time, we'll add support for cold-start coroutines.

Raymond Chen

Follow