On writing loops in PPL and continuation-passing style, part 3

devblogs.microsoft.com/oldnewthing/20230824-00

August 24, 2023

Raymond Chen

Last time, we wrote a task-based `while` loop using recursion, using a `shared_ptr` to pass state, and we noted a redundancy in that we created a `shared_ptr` to a lambda that in turn held a `shared_ptr`.

We can eliminate the nested shared pointers by requiring that all the state live inside a caller-provided `shared_ptr`, levaing the callable stateless.

```
template<typename State>
task<void> do_while_task(
    std::shared_ptr<State> const& state,
    bool (*f)(std::shared_ptr<State> const&)
{
    return f(state).then([state, f](bool loop) {
        return loop ? do_while_task(state, f) :
                      task_from_result();
    });
}

struct lambda_state
{
    lambda_state(Widgets* w) : widgets(w) {}
    Widgets* widgets;
    int i = 0;
};

auto state = std::make_shared<lambda_state>(widgets);

do_while_task(state, [](auto&& state)
{
    if (state->i >= 3) return task_from_result(false);
    return create_widget().then([state](auto widget)
    {
        state->widgets[state->i] = widget;
        state->i++;
        return true;
    }
}).then([] {
    printf("Done!\n");
});
```

We can get rid of all the `state->` prefixes by making the state be invocable.

```
template<typename State>
task<void> do_while_task(
    std::shared_ptr<State> const& state)
{
    return (*state)().then([state](bool loop) {
        return loop ? do_while_task(state) :
                      task_from_result();
    });
}

struct lambda_state
    : std::enable_shared_from_this<lambda_state>
{
    lambda_state(Widgets* w) : widgets(w) {}
    Widgets* widgets;
    int i = 0;

    task<bool> operator()()
    {
        if (i >= 3) return task_from_result(false);
        return create_widget().then(
            [this, lifetime = shared_from_this()](auto widget)
        {
            widgets[i] = widget;
            i++;
            return true;
        });
    }
};

auto state = std::make_shared<lambda_state>(widgets);

do_while_task(state).then([] {
    printf("Done!\n");
});
```

Hey, we've come full circle! The only difference is that we used `enable_shared_from_this` so that the `lambda_state` could obtain a `shared_ptr` to itself.

Next time, we'll translate all this nonsense into C# and JavaScript.