C++ coroutines: Making the promise itself be the shared state, the inspiration

devblogs.microsoft.com/oldnewthing/20210402-00

April 2, 2021

Raymond Chen

Earlier, we improved our simple coroutine promise by delaying the resumption of awaiting coroutines until local variables have destructed. This time, we'll look at another improvement.

Recall that our coroutine is structured like this:

	Coroutine state				Caller
	bookkeeping				
promise	`holder`	→	`result_holder` state	←	`holder`
	stack frame				

There are two allocations, one for the coroutine state, and one for the shared state internal to the `result_holder`. But what if we put the `result_holder` shared state inside the promise? In other words, what if we made the promise *be* the `result_holder` shared state?[1]

This trick takes advantage of the fact that you are permitted to suspend in the `final_suspend`. This lets you pause the coroutine execution before it gets to the point where it destroys the coroutine state.

The idea is that we move into the promise object all of the `result_holder` shared state, including the reference count hiding inside the `shared_ptr`.

Let's make the original diagram a bit more honest about the shared pointer control block. Recall that a `shared_ptr` is a pair of pointers, one to a control block and one to the shared data, and the control block consists of two reference counts, one for strong references and one for weak references.

	Coroutine state				Caller
	bookkeeping				
promise	`holder`	→	refcounts	←	`holder`
		→	`result_holder` state	←	
	stack frame				

What we're doing is moving the shared pointer control block and the shared state into the promise.

	Coroutine state		Caller
	bookkeeping		
promise	refcount	←	`holder`
	`result_holder` state		
	stack frame		

We don't need to support weak references at all, so we are down to just one reference count.

A running coroutine has a reference to its own state, and any outstanding `holder` objects also have a reference to the coroutine state. Only when all references go away do we destroy the coroutine state.

We're going to have to rewrite a bunch of stuff basically from scratch, seeing as we're abandoning the entire `shared_ptr` model that we had been using up until now. Let's hope it's worth it.

Bonus chatter: I figured I'd do the whole `shared_ptr` thing first, since it makes the several-week-long path to this point easier to follow. If I had started directly with the "result holder state embedded in the coroutine state", it would probably have been too confusing.

[1] Thanks to Gor Nishanov for providing this inspiration.

Raymond Chen

Follow