How to wait for multiple C++ coroutines to complete before propagating failure, initial plunge

devblogs.microsoft.com/oldnewthing/20230626-00

June 26, 2023

Raymond Chen

I'll start by repeating a very handy cheat sheet from a debugging case study of memory corruption from a coroutine that already finished.

Language	Method	Result	If any fail
C++	Concurrency::when_all	`vector<T>`	fail immediately
C++	winrt::when_all	`void`	fail immediately
C#	Task.WhenAll	`T[]`	wait for others
JavaScript	Promise.all	`Array`	fail immediately
JavaScript	Promise.allSettled	`Array`	wait for others
Python	asyncio.gather	List	fail immediately by default
Rust	join!	tuple	wait for others
Rust	try_join!	tuple	fail immediately

Python's `asynchio.gather` lets you choose whether a failed coroutine causes `gather` to fail immediately or to wait for others before failing. The default is to fail immediately.

The problem we saw was that the C++/WinRT `when_all` fails immediately, but the code wanted it to wait for the others before failing. This is a potentially useful general pattern, so let's try to write our own `when_all_completed` function.

It'll take a few tries to get there.

The idea behind the function is simple. Here's the pseudocode:

```
template<typename... T>
IAsyncAction when_all_complete(T... asyncs)
{
    std::exception_ptr eptr;

    /* Repeat for each element "async" of asyncs... */
    try {
        co_await async;
    } catch (...) {
        if (!eptr) {
            eptr = std::current_exception();
        }
    }
    ...

    if (eptr) std::rethrow_exception(eptr);
}
```

The idea is that we `co_await` each of the passed-in coroutines, but do so inside a `try`/`catch` block. If an exception occurs, then we save it, assuming we don't have an exception already: The first exception thrown is the one that is reported. (Naturally, you can remove the `if (!eptr)` if you want to report the last exception thrown.)

The "repeat for…" part can be solved with expansion statements:

```
template<typename... T>
IAsyncAction when_all_complete(T... asyncs)
{
    std::exception_ptr eptr;

    for... (auto& async : async) {
        try {
            co_await async;
        } catch (...) {
            if (!eptr) {
                eptr = std::current_exception();
            }
        }
    }

    if (eptr) std::rethrow_exception(eptr);
}
```

But before we clap the dust off our hands, note that expansion statements failed to be completed in time for C++20 and were postponed to C++23. And even if it were available in C++20, it will take time for projects to migrate off of C++17, so we'll have to find a solution that at least works on C++17.

We'll start our explorations next time.

Warning: There will be a lot of failure. If you're looking for an answer to be handed to you, you'll have to skip ahead to the end of the series, whenever that ends up happening.

Bonus chatter: Also, there's a frustrating edge case in the above code, but I don't want to try to fix it yet.