Rough edges in the when_all coroutine, part 2: Overloaded comma operator

devblogs.microsoft.com/oldnewthing/20200904-00

September 4, 2020

Raymond Chen

Last time, we looked at a problematic edge case in our `when_all` coroutine: The empty parameter list.

There's another edge case that can cause trouble, and that's the case where the comma operator itself has been overloaded.[1]

```
struct S
{
  void Detonate();
  S operator,(S right) { Detonate(); return right; }
};

struct async_s : std::experimental::suspend_never
{
  S await_resume() { return {}; }
};

when_all(async_s(), async_s()); // kaboom
```

We start by defining a type `S` that has a comma operator. When you comma two `S` objects together, the first one explodes.

Next, we define an awaitable object `async_s`: When you `co_await`, an `S` comes out.

And then we pass two of these objects to `when_all`. The expectation is that the `when_all` awaits the two objects, throws away the results, and returns.

Instead, what happens is that the `S` object explodes.

What went wrong is that our fold expression expanded to

```
IAsyncAction when_all(async_s v1, async_s v2)
{
  (co_await v1, co_await v2);
  co_return;
}
```

The intent of the comma in the fold expression was to throw away the left-hand operand, leaving the last surviving operand to be thrown away by the statement-expression. But thanks to the custom comma operator, it actually causes the left-hand operand to explode.

To suppress any custom comma operators, we can cast the result of the `co_await` to `void`. Since you cannot overload the comma operator for `void`, this forces the use of the default comma operator, so we just comma-combine a bunch of `void`s, which is harmless.

```
template <typename... T>
Windows::Foundation::IAsyncAction when_all(T... async)
{
    (void(co_await async), ...);
    co_return;
}
```

Here is the PR that fixes the empty parameter list and comma operator issues, and a follow-up.

Bonus chatter: We could also have used a right fold:

```
    (co_await async, ..., void());
```

which expands to

```
    (co_await v1, (co_await v2, void()));
```

But I think casting away the value is simpler.

[1] Shame on you.

Raymond Chen

Follow