A correction to the awaitable lock for C++ PPL tasks

devblogs.microsoft.com/oldnewthing/20241010-00

October 10, 2024

Some time ago, I created an awaitable lock for PPL tasks. But it turns out that there's a bug in that code.

The idea behind the awaitable lock was that everybody who was awaiting the lock subscribed to a completion event, and when the owner of the lock released the lock, the code signaled the completion event, which then woke up all of the waiters, one of whom would get the lock and proceed, and the others would loop back and wait some more.

The code that releases the lock enters a private mutex, marks the lock as available, swaps in a new completion event, and then signals everyone waiting on the old one.

```
void Release()
{
  std::lock_guard<std::mutex> guard(mutex);
  locked = false;
  auto previousCompletion = completion;
  completion = Concurrency::task_completion_event<void>();
  previousCompletion.set();
}
```

Unfortunately, there's a bug here: The previous subscribers are woken while still holding the private mutex. This means that you are now running arbitrary code while holding a private mutex, which is a bad idea. In particular, one of the items that was waiting for the completion might try to enter that same mutex from the same thread, and now we have an illegal recursive acquisition.[1]

We need to drop the lock before signaling the completion.

```
void Release()
{
  Concurrency::task_completion_event<void> previousCompletion;
  {
      std::lock_guard<std::mutex> guard(mutex);
      locked = false;
      previousCompletion = completion;
      completion = Concurrency::task_completion_event<void>();
  }
  previousCompletion.set();
}
```

This could be tightened up to

```
void Release()
{
  auto previousCompletion = [&] {
      auto guard = std::lock_guard(mutex);
      locked = false;
      return std::exchange(completion, {});
  }();
  previousCompletion.set();
}
```

or if you're really in a mood:

```
void Release()
{
  [&] {
      auto guard = std::lock_guard(mutex);
      locked = false;
      return std::exchange(completion, {});
  }().set();
}
```

The original article has been retroactively updated.

[1] Indeed, we *know* that all of them will try to enter the same mutex, because that's the point! What we don't know is what thread it will happen on.