Creating an awaitable lock for C++ PPL tasks

devblogs.microsoft.com/oldnewthing/20180802-00

August 2, 2018

Raymond Chen

The C# language (well, more accurately, the BCL) has the `ReaderWriterLockSlim` class which has a `WaitAsync` method which returns a task that completes asynchronously when the lock has been acquired. I needed an equivalent for the Parallel Patterns Library (PPL), and since I couldn't find one, I ended up writing one. (If you can find one, please let me know!)

```
// AsyncUILock is a nonrecursive lock that can be waited on
// asynchronously from a UI thread.
class AsyncUILock
{
public:
  Concurrency::task<void> WaitAsync()
  {
    std::lock_guard<std::mutex> guard(mutex);
    if (!locked) {
      // Lock is available. Acquire it.
      locked = true;
      return completed_apartment_aware_task();
    }

    // Lock is not available.
    return completed_apartment_aware_task()
      .then([captured_completion = completion] {
      // Wait for it to become available.
      return Concurrency::create_task(captured_completion);
    }).then([this] {
      // Then try again.
      return WaitAsync();
    });
  }

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

private:
  std::mutex mutex;
  bool locked = false;
  Concurrency::task_completion_event<void> completion;
};
```

The object consists of a `std::mutex` which protects the internal state, a flag that indicates whether the object has been claimed, and a task completion event that we use to signal anybody waiting on the lock that they should check again.

I could have used an `SRWLock` instead of a `std::mutex`, but I was lazy and wanted to take advantage of the existing `std::lock_guard`.

You can perform async waits on this object in the usual manner. For example:

```
AsyncUILock lock;

void DoSomething()
{
  lock.WaitAsync().then([]{
    // do something with the lock held.
    lock.Release();
  });
}
```

or if you prefer `co_await` (and you probably do):

```
AsyncUILock lock;

void DoSomething()
{
  co_await lock.WaitAsync();
  // do something with the lock held.
  lock.Release();
}
```

At this point, you might decide to return an RAII type to ensure that the lock doesn't leak. I'll leave that as an exercise.

Raymond Chen

Follow