Adding state to the update notification pattern, part 7

devblogs.microsoft.com/oldnewthing/20240425-00

April 25, 2024

Raymond Chen

Last time, we refined our change counter-based stateful but coalescing update notification. This version still relies on a UI thread to do two things: (1) make the final final change counter check and the subsequent callback atomic, and (2) to serialize the callbacks.

If we don't have a UI thread, then we open a race condition.

```
class EditControl
{
    ⟦ ... existing class members ... ⟧

    std::atomic<unsigned> m_latestId;
};

winrt::fire_and_forget
EditControl::TextChanged(std::string text)
{
    auto lifetime = get_strong();

    auto id = m_latestId.fetch_add(1, std::memory_order_relaxed);

    co_await winrt::resume_background();

    if (!IsLatestId(id))) co_return;

    std::vector<std::string> matches;
    for (auto&& candidate : FindCandidates(text)) {
        if (candidate.Verify()) {
            matches.push_back(candidate.Text());
        }
        if (!IsLatestId(id))) co_return;
    }

    // co_await winrt::resume_foreground(Dispatcher());

    if (!IsLatestId(id))) co_return;

    SetAutocomplete(matches);
}
```

Another call to `TextChanged` could happen just before the `SetAutocomplete`, and its work could race ahead of the first task.

UI thread	Background thread 1	Background thread 2
`TextChanged("Bob")` `resume_background()`		

	(Bob's task)
	`id = m_latestId;` (id is 1)
	calculate matches for "Bob"
	`if (1 == m_latestId)`
	`(true)`	

`TextChanged("Alice");` `resume_background()`		

		(Alice's task)
		`id = m_latestId;` (id is 2)
		calculate matches for "Alice"
		`if (2 == m_latestId)`
		`(true)`
		`SetAutocomplete(alice's matches)`

	`SetAutocomplete(bob's matches)`	

One temptation is to fix this by restoring atomicity by adding a lock:

```
winrt::fire_and_forget
EditControl::TextChanged(std::string text)
{
    auto lifetime = get_strong();

    auto id = m_latestId.fetch_add(1, std::memory_order_relaxed);

    co_await winrt::resume_background();

    if (!IsLatestId(id))) co_return;

    std::vector<std::string> matches;
    for (auto&& candidate : FindCandidates(text)) {
        if (candidate.Verify()) {
            matches.push_back(candidate.Text());
        }
        if (!IsLatestId(id))) co_return;
    }

    auto lock = std::unique_lock(m_mutex);

    if (!IsLatestId(id))) co_return;

    SetAutocomplete(matches);
}
```

The good news is that this avoids the race condition. One thing to check is that nothing bad happens if `SetAutocomplete` itself triggers a recursive call to `TextChanged`: Since the `SetAutocomplete` is happening on a background thread, a call to `TextChanged` would hop to another background thread before eventually blocking on the mutex. Nobody is deadlocked, so we're okay there.

But there is a problem if the `TextChange` calls come in faster than `SetAutocomplete` can process them. In that case, each `TextChange` consumes a background thread and blocks on the mutex. There could be a lot of background threads all waiting for their turn to call `SetAutocomplete`, but not able to make progress because the current active call to `SetAutocomplete` is taking too long. In the worst case, `SetAutocomplete` takes so long that you end up consuming all the threads in the threadpool, which tends not to end well.

Instead of using a mutex, we can use an "async mutex", where each waiting coroutine just suspends until its turn to enter the protected region. Fairness is not important here, because all but one of the coroutines will bail out when they realize that their change counter doesn't match, so we can use simple task sequencer that uses a kernel event.

```
class EditControl
{
    ⟦ ... existing class members ... ⟧

    std::atomic<unsigned> m_latestId;
    wil::unique_event m_serializerEvent{ wil::EventOptions::Signaled };
};

winrt::fire_and_forget
EditControl::TextChanged(std::string text)
{
    auto lifetime = get_strong();

    auto id = m_latestId.fetch_add(1, std::memory_order_relaxed);

    co_await winrt::resume_background();

    if (!IsLatestId(id))) co_return;

    std::vector<std::string> matches;
    for (auto&& candidate : FindCandidates(text)) {
        if (candidate.Verify()) {
            matches.push_back(candidate.Text());
        }
        if (!IsLatestId(id))) co_return;
    }

    co_await winrt::resume_on_signal(m_serializerEvent.get());
    auto next = wil::SetEvent_scope_exit(m_serializerEvent.get());

    if (!IsLatestId(id))) co_return;

    SetAutocomplete(matches);
}
```