Adding state to the update notification pattern, part 5

devblogs.microsoft.com/oldnewthing/20240423-00

April 23, 2024

Raymond Chen

We've been looking at the problem of a stateful but coalescing update notification, where multiple requests for work can arrive, and your only requirement is that you send a notification for the last one.

This time, we'll apply the trick of using a counter to record who is doing the work on behalf of the most recent change. Here's our first attempt:

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

    unsigned m_latestId;
};

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

    co_await winrt::resume_background();

    auto id = ++m_latestId;

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

    co_await winrt::resume_foreground(Dispatcher());

    if (id != m_latestId) co_return;

    SetAutocomplete(matches);
}
```

Just before we set the results, we check if the `m_latestId` has changed since when we started. If so, then our calculations are stale, and we abandon the operation. Otherwise, we proceed with the results. And we use an unsigned integer for the change counter to avoid undefined behavior on signed overflow.

There is still a problem here: We hop to a background thread *before* increment the counter. This means that the counter is not necessarily assigned sequentially to each call to `TextChanged`. Suppose the `m_latestId` is initially 0.

UI thread	Background thread 1	Background thread 2
`TextChanged("Bob")` `resume_background()`		
	(Bob's task)	
`TextChanged("Alice");` `resume_background()`		
		(Alice's task) `id = m_latestId;` (id is 1)
	`id = m_latestId;` (id is 2) calculate matches for "Bob" `resume_foreground()`	
(Bob's task) `if (2 == m_latestId)` (true) `SetAutocomplete(bob's matches)` (Bob's task completes)		calculate matches for "Alice"
		`resume_foreground()`

(Alice's task)
`if (1 == m_latestId)`
`(false)`
(Alice's task completes)

We incremented the change counter *after* hopping to the background thread, which means that it happens after we have lost control over the ordering of the work. We have to update the change counter from the UI thread so that the counter values correspond to the order in which the calls arrived, not the order in which the work began.

The result of this fix is this:

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

    auto id = ++m_latestId;

    co_await winrt::resume_background();

    std::vector<std::string> matches = FindMatches(text);

    co_await winrt::resume_foreground(Dispatcher());

    if (id != m_latestId) co_return;

    SetAutocomplete(matches);
}
```

This approach works, but it is a bit wasteful: If there are multiple changes in rapid succession, we do the work of finding matches for every text change, and throw away all but the last one.

Next time, we'll work toward making this more efficient.