Adding state to the update notification pattern, part 3

devblogs.microsoft.com/oldnewthing/20240419-00

April 19, 2024

Raymond Chen

Last time, we developed a stateful but coalescing update notification, and we noted that the code does a lot of unnecessary work because the worker thread calculates all the matches, even if the work has been superseded by another request.

We can add an optimization to abandon the background work if it notices that its efforts are going to waste: Periodically check whether there is any pending text. This will cost us a mutex, however, to protect access to `m_pendingText` from multiple threads.

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

    bool m_busy = false;
    std::mutex m_mutex;
    std::optional<string> m_pendingText;
};
winrt::fire_and_forget
EditControl::TextChanged(std::string text)
{
    auto lifetime = get_strong();

    ExchangePendingText(std::move(text));
    if (std::exchange(m_busy, true)) {
        co_return;
    }

    while (auto pendingText = ExchangePendingText(std::nullopt);
           pendingText) {

        co_await winrt::resume_background();

        auto matches = BuildMatches(*pendingText);

        co_await winrt::resume_foreground(Dispatcher());

        if (matches) {
            SetAutocomplete(*matches);
        }

    }
    m_busy = false;
}

template<typename T = std::optional<std::string>>
std::optional<std::string>
    EditControl::ExchangePendingText(T&& pending)
{
    auto lock = std::unique_lock(m_mutex);
    return std::exchange(m_pendingText, std::forward<T>(pending));
}

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

```
        if (auto lock = std::unique_lock(m_mutex);
            m_pendingText) {
            return std::nullopt;
        }
    }
    return matches;
}
```

Last time, I noted that the UI thread is doing a lot of work for us, since it is implicitly ensuring that the updates to `m_busy` and `m_pendingText` are atomic.

If our background work doesn't dispatch back to the UI thread, then we will be responsible for our own locking. We'll look at that next time.