Adding state to the update notification pattern, part 8

devblogs.microsoft.com/oldnewthing/20240426-00

April 26, 2024

Raymond Chen

We’ve been developing two different algorithms for the update notification pattern, one based on a busy flag and a pending state (and also an alternate version that is not dependent upon a UI thread to do implicit serialization), and another based on a change counter. How do they stack up against each other?

When a new request comes in while a previous request is still being worked on, the pending state algorithm doesn’t begin working on it right away. It just saves the update for later, and when the previous request finishes (or is abandoned), the worker will start processing the new request. This means that you have only one worker at a time, and the worker is working on only one item at a time.

By comparison, the change counter algorithm does start working on it right away, so you have two threads both doing work, although one of them will eventually abandon its efforts once it learns that its request is no longer current. With the change counter algorithm, you could have many threads working on requests simultaneously, although only one of them will run to completion.

If your work supports concurrent operations (for example, maybe it’s performing a read-only search), then the change counter algorithm allows lower latency since the most recent request gets to start right away. However, it also comes at a higher CPU cost, since you can have multiple workers running, and all but one of them is doing pointless work.

If your work does not support concurrent operations (say, some operations use a mutex to ensure only one thread can be performing the operation at a time), then the threads participating in the change counter algorithm will all end up serializing on that mutex, so you end up burning a bunch of threads which spend most of their time waiting for their turn to perform the operation. In this case, the pending request algorithm seems better because it doesn’t spin up threads that “hurry up and wait”.

	Pending request	**Change counter**

Latency	Higher	Lower
CPU usage	Lower	Higher
Thread usage	Single thread	Multiple threads

If multiple operations cannot run in parallel on multiple threads, then the change counter algorithm will create multiple threads, even though only one of them can do work at a time.