Lock free many-producer/single-consumer patterns: A work queue with task coalescing

devblogs.microsoft.com/oldnewthing/20161121-00

November 21, 2016

Raymond Chen

Today we're going to implement a very simple producer/consumer pattern: There are many producers who are producing work, and a single consumer who processes the work. The work is all identical, and it is idempotent. That means that doing the work twice is the same as doing it once.

You see this pattern when the work is something like *Refresh*. Refreshing once is the same as refreshing twice, as long as the single refresh is the last one.

The naïve way of doing this is to avoid the coalescing at all and have each refresh request notify the consumer to perform a refresh. This means, however, that if a thousand refresh requests come in rapid succession, the consumer thread will perform a thousand refreshes, nearly all of which were unnecessary.

A less naïve solution is to have a flag that says whether a refresh was requested. The first person to request a refresh wakes up the consumer. The second and subsequent people to request a refresh merely reminds the consumer that another refresh request arrived. When the consumer wakes up, it performs an atomic test-and-reset of the flag. If set, then it performs a refresh, and then it goes back and performs another test-and-reset of the flag to see whether a refresh request arrived while the previous one was in progress.

```
LONG RefreshRequested;

void RequestRefresh()
{
 if (!InterlockedExchange(&RefreshRequested, TRUE)) {
  // You provide the WakeConsumer() function.
  WakeConsumer();
 }
}

// You call this function when the consumer receives the
// signal raised by WakeConsumer().
void ConsumeRefresh()
{
 while (InterlockedExchange(&RefreshRequested, FALSE)) {
  Refresh();
 }
}
```

Note that we wake the consumer only on the transition from `FALSE` to `TRUE`. (In the lingo, the consumer is *edge-triggered* rather than *level-triggered*.) This reduces traffic between the two threads, which is important if your communication channel is something like `PostMessage`, because you don't want to spam the message queue with 10,000 identical messages. Not only does this cause `ConsumeRefresh` to run 9,999 times more than necessary, but you run the risk of overflowing your message queue.

Remember, there is only one consumer, so if `WakeConsumer` is called while `ConsumeRefresh` is still running, the wake will not start a new consumer immediately. It will wait for the existing `ConsumeRefresh` to complete before starting a new `ConsumeRefresh`.

Okay, that scenario was easy. We're still warming up. More next time.

Raymond Chen

Follow