# Lock free many-producer/single-consumer patterns: A work queue of distinct events, FIFO

devblogs.microsoft.com/oldnewthing/20161125-00

November 25, 2016

Raymond Chen

At last, our long national nightmare is over: The end of the miniseries on lock-free many-producer/single-consumer patterns. We'll finish up with the case where there are many producers generating work, and one consumer performing the work, and the work items must be processed in the order they were submitted.

We can build on the algorithm we used last time, when the order of processing was not important. We just have to remember to process the work items FIFO rather than LIFO.

```
SLIST_HEADER WorkQueue;

struct alignas(MEMORY_ALLOCATION_ALIGNMENT)
WorkItem : SLIST_ENTRY
{
    ... other stuff ...
};

void Initialize()
{
 InitializeSListHeader(&WorkQueue);
}

void RequestWork(WorkItem* work)
{
 if (InterlockedPushEntrySList(&WorkQueue, work)
                                                == nullptr) {
  // You provide the WakeConsumer() function.
  WakeConsumer();
 }
}

// You call this function when the consumer receives the
// signal raised by WakeConsumer().
void ConsumeWork()
{
 PSLIST_ENTRY entry = InterlockedFlushSList(&WorkQueue);
 entry = ReverseLinkedList(entry);
 while (entry != null) {
  ProcessWorkItem(static_cast<WorkItem*>(entry));
  delete entry;
 }
}
```

I leave the `ReverseLinkedList` function as an exercise. (Answer.)

The conditions of the exercise are that the order of operations is important, but the linked list is LIFO. We solve this by detaching the list from the `WorkQueue`, so that it can no longer be affected by `RequestWork`, and then we reverse the (now-private) list, thereby converting it from LIFO to FIFO, at which point we can process it.

Any new work that gets queued up will be added to the the (now-empty) `WorkQueue`. The first such new work will cause `WakeConsumer` to be called, upon which a new cycle will begin as soon as the current `ConsumeWork` finishes.

That's all I have on the topic of lock-free many-producer/single-consumer patterns. I hope you weren't too bored.

Raymond Chen

**Follow**