If an event is signaled more than once, are they delivered in the order in which they were signaled?

devblogs.microsoft.com/oldnewthing/20090522-00

May 22, 2009

Raymond Chen

A customer asked the following question:

> Is it guaranteed that the events when signaled multiple times on an event object are delivered in the order in which they were signaled?
>
> For example, a thread is waiting on the event handle and performs some operation each time it is signaled. The ordering of those operations should be in the same order in which the event is signaled. Do events provide this guarantee? In case it matters, we're using an auto-reset event. We're finding issues when the system is under heavy load.

This question is confusing because events don't "remember" who signaled them. When an event is signaled, then a waiting thread is released. The waiting thread doesn't know "who" did it; there is no context associated with an event. So the answer is, "Sure, whatever, it comes in the order they were signaled, if that's how you want to think about it. If you want to think about it in reverse order, then sure, they come in reverse order. They come in whatever order you want since there's no way you can detect what order they arrived in, since they are identical."

You and your friend enter the museum at the same time. The museum attendance person clicks the number counter twice. "Was that first click for me or for my friend?" Who cares? All that got recorded is two clicks. There's nothing in the clicks that associates each one with you or your friend.

When asked for clarification, the customer explained:

> We have a producer process that signals the waiting thread when certain 'events' occur. The service maintains a single list of context structures for each of these 'events' and then signals the waiting thread to tell it to wake up and take some action based on the associated context. When the client wakes up, it calls back into the producer process to retrieve the details of the event that woke it up. We want to know whether the thread, when it wakes up, will receive those events in the same order they were generated.

This explanation makes clear that the issue is not the event at all. The issue is the order in which the context structures are added to the list. We have this:

- Thread A:
  - Add context to list in some manner
  - Signal event
- Thread B:
  - Add context to list in some manner
  - Signal event
- Waiting thread:
  - Wait for event
  - Retrieve context from list in some manner
  - Do something with context
  - Repeat

The question boils down to “If thread A adds its context and then signals the event, and simultaneously thread B adds its context and signals the event, then when the waiting thread wakes up, is it guaranteed that the waiting thread will retrieve thread A’s context first?”

The answer is “Well, it depends on how you added the context to the list and how you retrieved it.” If the race between thread A’s “Add context to list in some manner” and thread B’s “Add context to list in some manner” is won by thread A, and the waiting thread retrieves the context FIFO, then it will get thread A’s context. But the event has nothing to do with it. The race condition took place before the event was signaled.

The answer to the question is of no real value, however, because this design pattern is flawed from the start. Events are not counted. They have only two states: Signaled and not signaled. Consider the situation where three events are raised in rapid succession.

- Thread A:
  - Add context to list in some manner
  - Signal event
- Waiting thread:

	Wait for event completes; event returns to unsignaled
	Retrieve context
Thread A:	
	Add context to list in some manner
	Signal event
	Add context to list in some manner
	Signal event (has no effect since event is already signaled)
Waiting thread:	
	Do something with first context
	Wait for event (completes immediately since event is already signaled)
	Event returns to unsignaled
	Retrieve context (gets second context)
	Do something with second context
	Wait for event (waits since event is not signaled)

Congratulations, you just lost an event signal because setting an event that is already set is a no-op.

If you need the number of wakes to equal the number of signals, then use a semaphore. That's what it's for.

In response to this explanation, the customer simply wrote back, "Semaphores worked. Thanks."

Raymond Chen

Follow