Re-entrant read/write locking construct?

I'm an experienced .NET programmer and stretching my legs with iOS. One of my favorite multi-threading constructs in .NET is the ReaderWriterLock. It allows for multiple readers or a single writer. The one feature that I am really missing in iOS is that the locks are re-entrant. That is, reader threads can acquire the read lock multiple times as long as they release it the same number of times. Likewise, the single writer thread can acquire the lock multiple times as long as it releases the lock by an equivalent number.

I've looked into the iOS Framework and none of the constructs appear to offer the same support including the re-entrancy. I also looked into the pthread library. I found rwlock, but it does not allow re-entrancy.

Is there anything on iOS that allows for re-entrant read-write locks?

objective-c ios multithreading locking

edited Oct 7 '12 at 13:00
Gray
62.6k 8 103 162

asked Oct 7 '12 at 2:29
Askable
88 7

2 Answers

From the iOS Threading Programming Guide:

> The system supports read-write locks using POSIX threads only. For more information on how to use these locks, see the pthread man page.

So I guess if pthreads don't support re-entrancy, the answer is no.

answered Oct 7 '12 at 3:45
pepsi
3,400 3 23 53

Yes, the `@synchronized` directive is reentrant. See Using the @synchronized Directive in the Threading Programming Guide and also Threading in the ObjC Programming Language.

That said, you should almost never use this in iOS. In most cases you can avoid locks of all kinds, let alone very heavyweight (slow) locks like reentrant locks. See the Concurrency Programming Guide, and particularly the section Migrating Away from Threads, for extensive information the queue-based approaches that iOS favors over manual thread management and locking.

For example, the reader/writer lock works like this using Grand Central Dispatch:

```
- (id)init {
   ...
   _someObjectQueue = dispatch_queue_create("com.myapp.someObject",
                                            DISPATCH_QUEUE_CONCURRENT);
 }

// In iOS 5 you need to release disptach_release(_someObjectQueue) in dealloc,
// but not in iOS 6.

- (id)someObject {
  __block id result;
  dispatch_sync(self.someObjectQueue, ^{
    result = _someObject;
  });
  return result;
}

- (void)setSomeObject:(id)newValue {
  dispatch_barrier_async(self.queue, ^{
    _someObject = newValue;
});
```

This approach allows unlimited parallel readers, with exclusive writers, while ensuring that writers never starve and that writes and reads are serialized, all while avoiding any kernel calls unless there is actual contention. That's all to say it's very fast and simple.

When a reader comes along, you queue a request to read the value, and wait for it to process. When a writer comes along, it queues a barrier request to update it, which requires that no other requests from that queue are currently running. With this construct, the developer doesn't need to manage any locks. Just put things on the queue in the order you want them to run.

answered Oct 7 '12 at 19:58

Rob Napier
104k 12 135 199

`@synchronized` is re-entrant but doesn't support multiple readers. GCD would require some significant restructuring of my existing code. – Askable Oct 8 '12 at 18:06

You're correct that @synchronized doesn't support multiple readers. This approach is unusual in Cocoa, and I'm not familiar with a great solution to it beyond pthread_rwlock_init which I recall is not reentrant. This kind of locking, as I said, is very inefficient, and iOS and modern Mac have faster, safer, and simpler mechanisms in GCD and NSOperationQueue. In older Mac, cooperative (runloop) multitasking was strongly preferred. .NET inherits its love of threads and locks from Java. ObjC discourages explicit threading. Cocoa is not like .NET and it is worth learning the Cocoa approach in iOS. – Rob Napier Oct 8 '12 at 20:43

(Yeah; I know this doesn't really answer your question. Sorry about that.) – Rob Napier Oct 8 '12 at 20:46