What is the Re-entrant lock and concept in general?

I am always get confused . Would someone example what Reentrant means in different contexts? And why would you want to use reentrant vs. non-reentrant.

Say pthread (posix) locking primitives, are the re-entrant or not? What pitfalls should be avoided when using them

Is mutex re-entrant? Thanks

multithreading locking pthreads

edited Jan 17 '11 at 14:50
Javache
1,762 1 11 21

asked Aug 21 '09 at 14:22
vehomzzz
8,189 33 93 169

3 Answers

Re-entrant locking

A reentrant lock is one where a process can claim the lock multiple times without blocking on itself. It's useful in situations where it's not easy to keep track of whether you've already grabbed a lock. If a lock is non re-entrant you could grab the lock, then block when you go to grab it again, effectively deadlocking your own process.

Reentrancy in general is a property of code where it has no central mutable state that could be corrupted if the code was called while it is executing. Such a call could be made by another thread, or it could be made recursively by an execution path originating from within the the code itself.

If the code relies on shared state that could be updated in the middle of its execution it is not re-entrant, at least not if that update could break it.

A use case for re-entrant locking

A (somewhat generic and contrived) example of an application for a re-entrant lock might be:

- You have some computation involving an algorithm that traverses a graph (perhaps with cycles in it). A traversal may visit the same node more than once due to the cycles or due to multiple paths to the same node.

- The data structure is subject to concurrent access and could be updated for some reason, perhaps by another thread. You need to be able to lock individual nodes to deal with potential data corruption due to race conditions. For some reason (perhaps performance) you don't want to globally lock the whole data structure.

- You computation can't retain complete information on what nodes you've visited, or you're using a data structure that doesn't allow 'have I been here before' questions to be answered quickly.

  An example of this situation would be a simple implementation of Dijkstra's algorithm with a priority queue implemented as a binary heap or a breadth-first search using a simple linked list as a queue. In these cases, scanning the queue for existing insertions is O(N) and you may not want to do it on every iteration.

In this situation, keeping track of what locks you've already acquired is expensive. Assuming you want do the locking at the node level a re-entrant locking mechanism alleviates the need to tell whether you've visited a node before. You can just blindly lock the node, perhaps unlocking it after you pop it off the queue.

Re-entrant mutexes

A simple mutex is not re-entrant as only one thread can be in the critical section at a given time. If you grab the mutex and then try to grab it again a simple mutex doesn't have enough information to tell who was holding it previously. To do this recursively you need a mechanism where each thread had a token so you could tell who had grabbed the mutex. This makes the mutex mechanism somewhat more expensive so you may not want to do it in all situations.

IIRC the POSIX threads API does offer the option of re-entrant and non re-entrant mutexes.

edited Aug 21 '09 at 16:44

answered Aug 21 '09 at 14:25
ConcernedOfTunbridgeW
39.2k 10 93 161

1 Although such situations should usually be avoided anyway, as it makes it hard to avoid deadlock etc as well. Threading is hard enough anyway without being in doubt as to whether you've already got a lock. – Jon Skeet Aug 21 '09 at 14:28

+1, also consider the case where the lock is NOT reentrant, you can block on yourself if you're not careful. Plus in C, you don't have the same mechanisms other languages do to ensure the lock is Released as many times as it is Acquired. This can lead to big problems. – user7116 Aug 21 '09 at 14:30

1 that's exact ly what happened to me yesterday: I didn't take the issue of re-entrancy into a consideration and ended up debugging a deadlock for 5 hours... – vehomzzz Aug 21 '09 at 14:34

@Jon Skeet - I think there are probably situations (see my somewhat contrived example above) where keeping track of locks is impractical due to performance or other considerations. – ConcernedOfTunbridgeWells Aug 21 '09 at 16:20

A re-entrant lock lets you write a method `M` that puts a lock on resource `A` and then call it from code that already holds a lock on `A`. With a non re-entrant lock, you would need 2 versions of `M`.

edited Feb 4 '13 at 15:32
Hans Kesting
18.1k 2 30 60

answered Aug 21 '09 at 14:33
Henk Holterman
160k 11 132 268

Pictorial explanation would be very helpful here... – Rachel Aug 20 '13 at 18:29

Reentrant lock is very well described in this tutorial.

The example in the tutorial is far less contrived than in the answer about traversing a graph. A reentrant lock is useful in very simple cases.

answered Mar 5 '14 at 12:35
Ratna Beresford
51 1 3