How do mutexes really work?

Asked 12 years, 4 months ago Modified 1 year, 9 months ago

Viewed 32k times

46

The idea behind mutexes is to only allow one thread access to a section of memory at any one time. If one thread locks the mutex, any other lock attempts will block until the first one unlocks. However, how is this implemented? To lock itself, the mutex has to set a bit somewhere that says that it is locked. But what if the second mutex is reading at the same time the first is writing. Worse, what if they both lock the mutex at the same time? The mutex would succumb to the same problem it is meant to prevent.

How do mutexes really work?

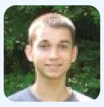
c++ multithreading concurrency mutex

Share

Improve this question

Follow

asked Aug 2, 2012 at 3:17



Joel

2,814 ● 6 ● 33 ● 46

4 Answers

Sorted by: Highest score (default)

30

Low-level atomic operations. These are essentially mutexes implemented in hardware, except you can only perform a very few operations atomically.

Consider the following equivalent pseudocode:

```
mutex global_mutex;
void InterlockedAdd(int& dest, int value) {
    scoped_lock lock(mutex);
    dest += value;
}
int InterlockedRead(int& src) {
    scoped_lock lock(mutex);
    return src;
}
void InterlockedWrite(int& dest, int value) {
    scoped_lock lock(mutex);
    dest = value;
}
```

These functions are implemented as instructions by the CPU, and they guarantee consistency between threads to various degrees. The exact semantics depend upon the CPU in question. x86 offers sequential consistency. This means that the operations act as if they were issued sequentially, in some order. This obviously involves blocking a little.

You may accurately surmise that atomic operations can be implemented in terms of mutexes, or vice versa. But usually, atomic ops are provided by hardware, and then mutexes and other synchronization primitives implemented on top of them by the operating system. This is because there are some algorithms that do not require a full mutex and can operate what is known as "locklessly", which means just using atomic operations for some inter-thread consistency.

Share Improve this answer Follow

answered Aug 2, 2012 at 3:24

Puppy
147k ● 40 ● 266 ● 477

26

A simple implementation that has been used in the past is to use a CPU level atomic "lock and exchange" instruction. This is a special instruction that atomically swaps a given value with a value in some memory location.

A thread could acquire such a mutex by trying to swap a 1 value into the memory location. If the value comes back as 0, then the thread would assume that it has the mutex and would continue. Otherwise, if the value returned is 1, then the thread would know that some *other* thread currently has the mutex. In that case it would wait until trying again.

The above is a highly simplified outline of what might happen in a simple system. Real operating systems are much more complex these days.

Share Improve this answer Follow

answered Aug 2, 2012 at 3:24

Greg Hewgill
990k ● 191 ● 1.2k ● 1.3k

1 This does not answer the question. What if two threads do "read and swap" at the same time. Then both get back 0. – user1146657 Feb 2, 2016 at 20:20

14 @user1146657: The point of the "lock and exchange" operation is that it is *atomic*, and the CPU(s) ensure that only one thread can do it at any given time. So by definition, two threads *cannot* do it at the same time. The CPU is specifically designed to work that way for exactly this purpose. – Greg Hewgill Feb 2, 2016 at 20:23

20

Here's a quick overview of what a mutex needs to work, it's a shortened form of my complete article How does a mutex work?

- There is an integer in memory that represents the locked state, with a value 1 or 0.
- The mutex needs an atomic `compare_and_swap` function that can atomically attempt to modify that value and report whether it succeeded. This allows a thread to both check and modify the state at the same time.
- The OS needs to provide a function to wait in the case the mutex is locked. On Linux the low-level function is `futex`. This will place the thread in a queue, and also monitor the integer in memory.
- The operations involved also include data fences, to prevent modifications in memory from being visible prior to the lock, and being completely available after the lock.

Share Improve this answer Follow

edited Feb 28, 2023 at 10:24

answered Jul 21, 2017 at 3:43

edA-qa mort-ora-y
31.8k ● 43 ● 150 ● 278

1 Futex explanation – yzabalotski Sep 28, 2022 at 22:03

@eda-qa-mort-ora-y, update the link to the article, pls: mortoray.com/how-does-a-mutex-work-what-does-it-cost – ox160d05d Feb 27, 2023 at 16:29

5

All you need is to do it atomically. It can be provided by hardware, such as atomic compare-and-exchange instructions, or by the operating system through system calls. Once it's in the OS domain it's fairly easy to make sure only a single thread is trying to lock the mutex.

In practice both approaches are combined. See for example Linux's futexes.

Share Improve this answer Follow

answered Aug 2, 2012 at 3:32

DanielKO
4,517 ● 20 ● 30