When should we use mutex and when should we use semaphore

When should we use mutex and when should we use semaphore ?

linux multithreading synchronization mutex semaphore

edited Oct 1 '13 at 11:33
legends2k
8,217 6 27 64

asked Oct 28 '10 at 5:00
Karthik Balaguru
1,205 3 14 35

1 possible duplicate of What is mutex and semaphore in Java ? What is the main difference ? – Merlyn Morgan-Graham Oct 28 '10 at 5:20

8 Answers

A mutex is a mutual exclusion *semaphore*, a special variant of a semaphore that only allows one locker at a time and whose ownership restrictions may be more stringent than a normal semaphore.

In other words, it's equivalent to a normal counting semaphore with a count of one and the requirement that it can only be released by the same thread that locked it.

A semaphore, on the other hand, has a count and can be locked by that many lockers concurrently. And it may not have a requirement that it be released by the same thread that claimed it (but, if not, you have to carefully track who currently has responsibility for it, much like allocated memory).

So, if you have a number of instances of a resource (say three tape drives), you could use a semaphore with a count of 3. Note that this doesn't tell you which of those tape drives you have, just that you have a certain number.

Also with semaphores, it's possible for a single locker to lock multiple instances of a resource, such as for a tape-to-tape copy. If you have one resource (say a memory location that you don't want to corrupt), a mutex is more suitable.

Equivalent operations are:

```
Counting semaphore          Mutual exclusion semaphore
--------------------------  --------------------------
  Claim/decrease (P)                  Lock
  Release/increase (V)                Unlock
```

Aside: in case you've ever wondered at the bizarre letters used for claiming and releasing semaphores, it's because the inventor was Dutch. Probeer te verlagen means to try and decrease while verhogen means to increase.

edited Nov 4 '10 at 4:26

answered Oct 28 '10 at 5:02
paxdiablo
379k 81 733 1173

Okay, I came across binary semaphore too. When should we need to go in for binary semaphore and when should we use mutex ? – Karthik Balaguru Oct 28 '10 at 5:08

Conceptually, a binary semaphore *is* a mutex, and it's equivalent to a normal semaphore with a one-count. There may be differences in *implementations* of the concept such as efficiency, or ownership of the resource (can be released by someone other than the claimer, which I don't agree with BTW - a resource should only be releasable by the thread that claimed it). – paxdiablo Oct 28 '10 at 5:12

Another potential implementation difference is the recursive mutex. Because there's only one resource, a single thread may be allowed to lock it multiple times (as long as it releases it that many times as well). This isn't so easy with a multiple-instance resource since you may not know whether the thread wants to claim *another* instance or the *same* instance again. – paxdiablo Oct 28 '10 at 5:21

1 They solve a specific problem. The fact that the problem they solve is people who don't *quite* grok mutexes, should in no way belittle the solution :-) – paxdiablo Oct 28 '10 at 7:05

2 A mutex is totally different from a binary semaphore. Sorry but this definition is wrong – Peer Stritzinger Oct 28 '10 at 14:37

Here is how I remember when to use what -

Semaphore: Use a semaphore when you (thread) want to sleep till some other thread tells you to wake up. Semaphore 'down' happens in one thread (producer) and semaphore 'up' (for same semaphore) happens in another thread (consumer) e.g.: In producer-consumer problem, producer wants to sleep till at least one buffer slot is empty - only the consumer thread can tell when a buffer slot is empty.

Mutex: Use a mutex when you (thread) want to execute code that should not be executed by any other thread at the same time. Mutex 'down' happens in one thread and mutex 'up' *must* happen in the same thread later on. e.g.: If you are deleting a node from a global linked list, you do not want another thread to muck around with pointers while you are deleting the node. When you acquire a mutex and are busy deleting a node, if another thread tries to acquire the same mutex, it will be put to sleep till you release the mutex.

Spinlock: Use a spinlock when you really want to use a mutex but your thread is not allowed to sleep. e.g.: An interrupt handler within OS kernel must never sleep. If it does the system will freeze / crash. If you need to insert a node to globally shared linked list from the interrupt handler, acquire a spinlock - insert node - release spinlock.

edited Aug 22 '12 at 7:29

j0k
14.4k 10 38 51

answered Aug 22 '12 at 3:12

Annu Gogatya
171 1 3

It is very important to understand that a mutex ***is not*** a semaphore with count 1!

This is the reason there are things like binary semaphores (which are really semaphores with count 1).

The difference between a Mutex and a Binary-Semaphore is the principle of ownership:

A mutex is acquired by a task and therefore must also be released by the same task. This makes it possible to fix several problems with binary semaphores (Accidential release, recursive deadlock and priority inversion).

Caveat: I wrote "makes it possible", if and how these problems are fixed is up to the OS implementation.

Because the mutex is has to be released by the same task it is not very good for synchronization of tasks. But if combined with condition variables you get very powerful building blocks for building all kinds of ipc primitives.

So my recommendation is: if you got cleanly implemented mutexes and condition variables (like with POSIX pthreads) use these.

Use semaphores only if they fit exactly to the problem you are trying to solve, don't try to build other primitives (e.g. rw-locks out of semaphores, use mutexes and condition variables for these)

There is a lot of misunderstanding mutexes and semaphores. The best explanation I found so far is in this 3-Part article:

Mutex vs. Semaphores – Part 1: Semaphores

Mutex vs. Semaphores – Part 2: The Mutex

Mutex vs. Semaphores – Part 3 (final part): Mutual Exclusion Problems

edited Oct 28 '10 at 14:34

answered Oct 28 '10 at 14:08

Peer Stritzinger
5,233 15 29

The urls to this site contain funky characters and do not work therefore... I'm working on it – Peer Stritzinger Oct 28 '10 at 14:15

Fixed the unicode characters in the urs, links now work. – Peer Stritzinger Oct 28 '10 at 14:23

While @opaxdiablo answer is totally correct I would like to point out that the usage scenario of both things is quite different. The mutex is used for protecting parts of code from running concurrently, semaphores are used for one thread to signal another thread to run.

```
/* Task 1 */
pthread_mutex_lock(mutex_thing);
    // Safely use shared resource
pthread_mutex_unlock(mutex_thing);



/* Task 2 */
pthread_mutex_lock(mutex_thing);
   // Safely use shared resource
pthread_mutex_lock(mutex_thing);
```

The semaphore scenario is different:

```
/* Task 1 - Producer */
sema_post(&sem);   // Send the signal

/* Task 2 - Consumer */
sema_wait(&sem);   // Wait for signal
```

See http://www.netrino.com/node/202 for further explanations

edited Nov 14 '12 at 9:00

rbrito
1,162 1 7 20

answered Oct 28 '10 at 6:04

tristopia
5,506 19 26

1 You're right. Even if you are using a semaphore with a count of one, you are implying something about what you're doing than if you used a mutex. – Omnifarious Oct 28 '10 at 6:40

I'm not sure I agree with that, though I don't *disagree* so vehemently that I'll downvote you :-) You say that the usage pattern of semaphores is to notify threads but that's exactly what mutexes do when there's another thread waiting on it, and exactly what semaphores *don't* when there's no threads in `sema_wait` :-) In my opinion, they're *both* about resources and the notification handed to other threads is a side-effect (very important, performance-wise) of the protection. – paxdiablo Oct 28 '10 at 7:07

`You say that the usage pattern of semaphores is to notify threads` One point about notifying threads. You can call `sem_post` from a signal handler safely (pubs.opengroup.org/onlinepubs/009695399/functions/…) but it is not recomended to call `pthread_mutex_lock` and `pthread_mutex_unlock` from signal handlers (manpages.ubuntu.com/manpages/lucid/man3/…) – skwllsp Mar 5 '13 at 12:13

Thanks, interesting. – tristopia Mar 5 '13 at 12:40

@paxdiablo: There is one major difference between this mutex binary semaphore is maintaining the reference count. Mutex or you can say any conditional mutex does not maintain any count related to lock where as sempahore use to maintain the count. So sem_wait and sem_post are maintaining the count. – Prak Jul 14 '13 at 12:56

Trying not to sound zany, but can't help myself.

Your question should be what is the difference between mutex and semaphores ? And to be more precise question should be, 'what is the relationship between mutex and semaphores ?'

(I would have added that question but I'm hundred % sure some overzealous moderator would close it as duplicate without understanding difference between difference and relationship.)

In object terminology we can observe that :

observation.1 Semaphore contains mutex

observation.2 Mutex is not semaphore and semaphore is not mutex.

There are some semaphores that will act as if they are mutex, called binary semaphores, but they are freaking NOT mutex.

There is a special ingredient called Signalling (posix uses condition_variable for that name), required to make a Semaphore out of mutex. Think of it as a notification-source. If two or more threads are subscribed to same notification-source, then it is possible to send them message to either ONE or to ALL, to wakeup.

There could be one or more counters associated with semaphores, which are guarded by mutex. The simple most scenario for semaphore, there is a single counter which can be either 0 or 1.

This is where confusion pours in like monsoon rain.

A semaphore with a counter that can be 0 or 1 is NOT mutex.

Mutex has two states (0,1) and one ownership(task). Semaphore has a mutex, some counters and a condition variable.

Now, use your imagination, and every combination of usage of counter and when to signal can make one kind-of-Semaphore.

1. Single counter with value 0 or 1 and signaling when value goes to 1 AND then unlocks one of the guy waiting on the signal == Binary semaphore
2. Single counter with value 0 to N and signaling when value goes to less than N, and locks/waits when values is N == Counting semaphore
3. Single counter with value 0 to N and signaling when value goes to N, and locks/waits when values is less than N == Barrier semaphore (well if they dont call it, then they should.)

Now to your question, when to use what. (OR rather correct question version.3 when to use mutex and when to use binary-semaphore, since there is no comparison to non-binary-semaphore.) Use mutex when 1. you want a customized behavior, that is not provided by binary semaphore, such are spin-lock or fast-lock or recursive-locks. You can usually customize mutexes with attributes, but customizing semaphore is nothing but writing new semaphore. 2. you want lightweight OR faster primitive

Use semaphores, when what you want is exactly provided by it.

If you dont understand what is being provided by your implementation of binary-semaphore, then IMHO, use mutex.

And lastly read a book rather than relying just on SO.

answered Feb 8 '13 at 8:23

community wiki
Ajeet

As was pointed out, a semaphore with a count of one is the same thing as a 'binary' semaphore which is the same thing as a mutex.

The main things I've seen semaphores with a count greater than one used for is producer/consumer situations in which you have a queue of a certain fixed size.

You have two semaphores then. The first semaphore is initially set to be the number of items in the queue and the second semaphore is set to 0. The producer does a P operation on the first semaphore, adds to the queue. and does a V operation on the second. The consumer does a P operation on the second semaphore, removes from the queue, and then does a V operation on the first.

In this way the producer is blocked whenever it fills the queue, and the consumer is blocked whenever the queue is empty.

edited Oct 28 '10 at 13:52

answered Oct 28 '10 at 5:35

Omnifarious
26.9k 5 50 118

See "The Toilet Example" - http://pheatt.emporia.edu/courses/2010/cs557f10/hand07/Mutex%20vs_%20Semaphore.htm:

Mutex:

Is a key to a toilet. One person can have the key - occupy the toilet - at the time. When finished, the person gives (frees) the key to the next person in the queue.

Officially: "Mutexes are typically used to serialise access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section." Ref: Symbian Developer Library

(A mutex is really a semaphore with value 1.)

Semaphore:

Is the number of free identical toilet keys. Example, say we have four toilets with identical locks and keys. The semaphore count - the count of keys - is set to 4 at beginning (all four toilets are free), then the count value is decremented as people are coming in. If all toilets are full, ie. there are no free keys left, the semaphore count is 0. Now, when eq. one person leaves the toilet, semaphore is increased to 1 (one free key), and given to the next person in the queue.

Officially: "A semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore)." Ref: Symbian Developer Library

answered Oct 28 '10 at 14:02

fornwall
1,506 1 7 19

I think the question should be the difference between mutex and binary semaphore.

Mutex = It is a ownership lock mechanism, only the thread who acquire the lock can release the lock.

binary Semaphore = It is more of a signal mechanism, any other higher priority thread if want can signal and take the lock.

answered Dec 22 '14 at 16:52

Saurabh Sengar
26 5
