What is mutex and semaphore in Java ? What is the main difference?

Asked 15 years, 8 months ago Modified 6 years ago Viewed 110k times

120

What is mutex and semaphore in Java ? What is the main difference ?

java multithreading concurrency mutex semaphore

Share

Improve this question

Follow

edited Oct 9, 2012 at 8:54

Jan Hrcek

673 ● 11 ● 25

asked Apr 21, 2009 at 6:51

Isabel Jinson

8,661 ● 16 ● 61 ● 76

Similar query in stackoverflow.com/questions/4039899/… – Karthik Balaguru Oct 10, 2014 at 17:59

10 Answers

Sorted by: Highest score (default)

152

Unfortunately everyone has missed the most important difference between the semaphore and the mutex; the concept of "**ownership**".

Semaphores have no notion of ownership, this means that any thread can release a semaphore (this can lead to many problems in itself but can help with "death detection"). Whereas a mutex does have the concept of ownership (i.e. you can only release a mutex you have acquired).
Ownership is incredibly important for safe programming of concurrent systems. I would always recommend using mutex in preference to a semaphore (but there are performance implications).

Mutexes also may support priority inheritance (which can help with the priority inversion problem) and recursion (eliminating one type of deadlock).

It should also be pointed out that there are "binary" semaphores and "counting/general" semaphores. Java's semaphore is a counting semaphore and thus allows it to be initialized with a value greater than one (whereas, as pointed out, a mutex can only a conceptual count of one). The usefulness of this has been pointed out in other posts.

So to summarize, unless you have multiple resources to manage, I would always recommend the mutex over the semaphore.

edited Dec 4, 2011 at 21:22

Piotr Nowicki
18.2k ● 9 ● 65 ● 83

answered Apr 22, 2009 at 8:52

feabhas

1,521 ● 1 ● 8 ● 2

1 Feabhas's answer is quite important - the mutex checks the thread attempting to release the mutex actually owns it. I've had this as an interview question so its worth trying to remember it. – andrew pate Feb 18, 2011 at 21:09

119

Semaphore can be counted, while mutex can only count to 1.

Suppose you have a thread running which accepts client connections. This thread can handle 10 clients simultaneously. Then each new client sets the semaphore until it reaches 10. When the Semaphore has 10 flags, then your thread won't accept new connections

Mutex are usually used for guarding stuff. Suppose your 10 clients can access multiple parts of the system. Then you can protect a part of the system with a mutex so when 1 client is connected to that sub-system, no one else should have access. You can use a Semaphore for this purpose too. A mutex is a "Mutual Exclusion Semaphore".

Share Improve this answer

Follow

answered Apr 21, 2009 at 6:58

Eric

19.9k ● 19 ● 88 ● 149

4 This is not quite true. The same thread can enter the same mutex more than once, so a count needs to be maintained to

ensure entries `& exits are balanced. – finnw Aug 3, 2009 at 9:56

1 @finnw, in general there are two types of mutexes, recursive and non-recursive. Does Java by default use the recursive type? – edA-qa mort-ora-y Apr 24, 2011 at 12:10

2 @edA-qa mort-ora-y, the term "Mutex" is not used in the Java VM or API spec so I am assuming it refers to the monitor built into every object, which is also similar to the Win32 object called a Mutex. The same applies to a `ReentrantLock`. All of these are recursive. I am not aware of any "real-world" examples of non-recursive mutexes (I have seen them only in textbooks) so I did not consider them. – finnw Apr 24, 2011 at 14:12

2 Non-recursive mutexes can be implemented by using a Semaphore with count one. This can be useful if you want to prevent recursive calls. This has practical uses, I've personally used it in big projects to detect loops in initialization code (A initializes B which tries to initialize A again). – Alexander Torstling Nov 23, 2011 at 16:51

1 In C++11 (C++0x) standard, mutex is non-recursive. They also provide a separate 'recursive_mutex' for those that need that. I know we are talking Java here, but then many of us code across languages now. – Aditya Kumar Pandey Jan 3, 2013 at 14:47

42

Mutex is basically mutual exclusion. Only one thread can acquire the resource at once. When one thread acquires the resource, no other thread is allowed to acquire the resource until the thread owning the resource releases. All threads waiting for acquiring resource would be blocked.

Semaphore is used to control the number of threads executing. There will be fixed set of resources. The resource count will gets decremented every time when a thread owns the same. When the semaphore count reaches 0 then no other threads are allowed to acquire the resource. The threads get blocked till other threads owning resource releases.

In short, the main difference is *how many threads are allowed to acquire the resource at once ?*

- Mutex --its ONE.
- Semaphore -- its DEFINED_COUNT, (as many as semaphore count)

Share Improve this answer Follow

edited Apr 21, 2009 at 7:09

answered Apr 21, 2009 at 6:59

aJ.

35.4k ● 22 ● 87 ● 131

11

A mutex is used for serial access to a resource while a semaphore limits access to a resource up to a set number. You can think of a mutex as a semaphore with an access count of 1. Whatever you set your semaphore count to, that may threads can access the resource before the resource is blocked.

Share Improve this answer

answered Apr 21, 2009 at 6:57

Follow

Jason Coco
78.3k ● 21 ● 186 ● 180

3

A semaphore is a counting synchronization mechanism, a mutex isn't.

Share Improve this answer

Follow

answered Apr 21, 2009 at 6:56

Brian Rasmussen
116k ● 34 ● 224 ● 323

3

A mutex is often known as a binary semaphore. Whilst a semaphore can be created with any non-zero count a mutex is conceptually a semeaphore with an upper count of 1.

Share Improve this answer

Follow

answered Apr 21, 2009 at 15:08

Sean
62.4k ● 11 ● 99 ● 138

2

Semaphore:

> A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each `acquire()` blocks if necessary until a permit is available, and then takes it. Each `release()` adds a permit, potentially releasing a blocking

> acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource

Java does not have built-in Mutex API. But it can be implemented as binary semaphore.

> A semaphore initialized to one, and which is used such that it only has at most one permit available, can serve as a mutual exclusion lock. This is more commonly known as a binary semaphore, because it only has two states: one permit available, or zero permits available.

When used in this way, the binary semaphore has the property (unlike many Lock implementations), that *the "lock" can be released by a thread other than the owner (as semaphores have no notion of ownership)*. This can be useful in some specialized contexts, such as deadlock recovery.

So key differences between Semaphore and Mutex:

1. Semaphore restrict number of threads to access a resource throuhg permits. Mutex allows only one thread to access resource.

2. No threads owns Semaphore. Threads can update number of permits by calling `acquire()` and `release()` methods. Mutexes should be unlocked only by the thread holding the lock.
3. When a mutex is used with condition variables, there is an implied bracketing—it is clear *which part of the program is being protecte*d. This is not necessarily the case for a semaphore, which might be called the *go to of concurrent programming*—it is powerful but too easy to use in an unstructured, indeterminate way.

Share Improve this answer

Follow

answered Jul 3, 2017 at 13:54

Ravindra babu

38.9k ● 11 ● 256 ● 219

0

The object of synchronization **Semaphore** implements a classical traffic light. A traffic light controls access to a resource shared by a counter. If the counter is greater than zero, access is granted; If it is zero, access is denied. The counter counts the permissions that allow access to the shared resource. Then, to access the resource, a thread must receive permission from the traffic light. In general, to use a traffic light, the thread that wants to access the shared resource tries to acquire a permit. If the traffic light count is greater than zero, the thread acquires a permit, and the traffic light count is decremented. Otherwise the thread is locked until it can get a permission. When the thread no longer needs to access the shared resource, it releases the permission,

so the traffic light count is increased. If there is another thread waiting for a permit, it acquires a permit at that time. The Semaphore class of Java implements this mechanism.

Semaphore has two builders:

```
Semaphore(int num)
Semaphore(int num, boolean come)
```

num specifies the initial count of the permit. Then num specifies the number of threads that can access a shared resource at a given time. If num is one, it can access the resource one thread at a time. By setting **come** as true, you can guarantee that the threads you are waiting for are granted permission in the order they requested.

Share Improve this answer

Follow

answered Jul 2, 2017 at 21:01

Amarildo

268 ● 4 ● 20

0

You compare the incomparable, technically there is no difference between a Semaphore and mutex it doesn't make sense. Mutex is just a significant name like any name in your application logic, it means that you initialize a semaphore at "1", it's used generally to protect a resource or a protected variable to ensure the mutual exclusion.

Share Improve this answer

edited Dec 17, 2018 at 3:09

Follow

Pang
10.1k ● 146 ● 85 ● 124

answered Dec 7, 2012 at 19:08

Marwen Trabelsi
4,237 ● 8 ● 40 ● 81

-1

Mutex is binary semaphore. It must be initialized with 1, so that the First Come First Serve principle is met. This brings us to the other special property of each mutex: *the one who did **down**, must be the one who does **up***. Ergo we have obtained mutual exclusion over some resource.

Now you could see that a mutex is a special case of general semaphore.

Share Improve this answer

Follow

answered Feb 28, 2017 at 16:32

Dimitar
4,753 ● 6 ● 33 ● 48

This is incorrect. The difference concerns ownership. A mutex can only be unlocked by the thread that currently holds the lock. That is not true for a semaphore. Check this other related SO question – cmhteixeira May 14, 2022 at 14:24