What is a semaphore?

A semaphore is a programming concept that is frequently used to solve multi-threading problems. My question to the community:

What is a semaphore and how do you use it?

multithreading concurrency semaphore

asked Aug 29 '08 at 15:58
Dalroth
3,033 7 22 20

1 a boolean flag whose value is based on whether an integer counter has reached its designated upper limit. Obfuscation to the max! – Sam Sep 19 '11 at 1:02

If you found an answer useful/satisfactory, please accept it. – aspen100 May 8 '14 at 15:25

12 Answers

Think of semaphores as bouncers at a nightclub. There are a dedicated number of people that are allowed in the club at once. If the club is full no one is allowed to enter, but as soon as one person leaves another person might enter.

It's simply a way to limit the number of consumers for a specific resource. For example, to limit the number of simultaneous calls to a database in an application.

Here is a very pedagogic example in C# :-)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace TheNightclub
{
    public class Program
    {
        public static Semaphore Bouncer { get; set; }

        public static void Main(string[] args)
        {
            // Create the semaphore with 3 slots, where 3 are available.
            Bouncer = new Semaphore(3, 3);

            // Open the nightclub.
            OpenNightclub();
        }

        public static void OpenNightclub()
        {
            for (int i = 1; i <= 50; i++)
            {
                // Let each guest enter on an own thread.
                Thread thread = new Thread(new ParameterizedThreadStart(Guest));
                thread.Start(i);
            }
        }

        public static void Guest(object args)
        {
            // Wait to enter the nightclub (a semaphore to be released).
            Console.WriteLine("Guest {0} is waiting to entering nightclub.", args);
            Bouncer.WaitOne();
            // Do some dancing.
            Console.WriteLine("Guest {0} is doing some dancing.", args);
            Thread.Sleep(500);

            // Let one guest out (release one semaphore).
            Console.WriteLine("Guest {0} is leaving the nightclub.", args);
            Bouncer.Release(1);
        }
    }
}
```

edited Jan 27 '12 at 3:44
Cody Gray
111k 14 201 291

answered Sep 2 '08 at 20:13
Patrik
5,763 7 30 60

14 one upvote for comapring bouncers of nightclub with Semaphore – Duraiamuthan.H Feb 15 '14 at 7:51

1 Yes, the bouncer analogy is epic! – Mike Caron Mar 7 '14 at 13:05

if it's like bouncers at a night club, it should let the guests go in sequentially, but when i tried it out, it's random. Eg. Guest 40 came in first before Guest 39. Is there anything we could do to control this? – TNA May 31 '14 at 9:12

@TNA: Yes, that has to do with the way new threads are started in this example, and not really in the scope of the answer. – Patrik May 31 '14 at 19:57

coverity Coverity is the leader in development testing, Use static analysis to find issues earlier. LEARN MORE

Good article on Mutex's and Semaphores - what makes them different, and why they might or might not be used given various conditions.

answered Sep 2 '08 at 18:37
Adam Davis
50.3k 39 190 292

6 Can I up-vote this twice? That article clearly and concisely spelled out the difference between Mutex's and Semaphores... concepts that had always been explained to me with very mirky distinctions. Thanks! – embedded_guy Jan 19 '12 at 23:54

Mutex: exclusive-member access to a resource

Semaphore: n-member access to a resource

That is, a mutex can be used to syncronize access to a counter, file, database, etc.

A sempahore can do the same thing but supports a fixed number of simultaneous callers. For example, I can wrap my database calls in a semaphore(3) so that my multithreaded app will hit the database with at most 3 simultaneous connections. All attempts will block until one of the three slots opens up. They make things like doing naive throttling really, really easy.

answered Sep 2 '08 at 18:49
Michael Haren
46.2k 27 118 175

3 According to Richard W. Stevens, a mutex is actually a binary semaphore, with only two possible values: 0 and 1. – Qiang Xu Oct 12 '12 at 21:14

@Craig:

> A semaphore is a way to lock a resource so that it is guaranteed that while a piece of code is executed, only this piece of code has access to that resource. This keeps two threads from concurrently accesing a resource, which can cause problems.

This is not restricted to only one thread. A semaphore can be configured to allow a fixed number of threads to access a resource.

answered Aug 29 '08 at 16:07
Mats Fredriksson
8,039 6 21 46

This is a commentary, not an answer. – gg.kaspersky May 19 '13 at 17:15

2 Yeah, but I think I wrote this before comments got added to Stack Overflow. Or I didn't, don't really remember. This time I answered in a comment though. :-) – Mats Fredriksson Sep 15 '13 at 21:36

Semaphore can also be used as a ... semaphore. For example if you have multiple process enqueuing data to a queue, and only one task consuming data from the queue. If you don't want your consuming task to constantly poll the queue for available data, you can use semaphore.

Here the semaphore is not used as an exclusion mechanism, but as a signaling mechanism. The consuming task is waiting on the semaphore The producing task are posting on the semaphore.

This way the consuming task is running when and only when there is data to be dequeued

answered Sep 16 '08 at 15:10
shodanex
7,639 5 28 64

A semaphore is an object containing a natural number (i.e. a integer greater or equal to zero) on which two modifying operations are defined. One operation, `V` , adds 1 to the natural. The other operation, `P` , decreases the natural number by 1. Both activities are atomic (i.e. no other operation can be executed at the same time as a `V` or a `P`).

Because the natural number 0 cannot be decreased, calling `P` on a semaphore containing a 0 will block the execution of the calling process(/thread) until some moment at which the number is no longer 0 and `P` can be successfully (and atomically) executed.

As mentioned in other answers, semaphores can be used to restrict access to a certain resource to a maximum (but variable) number of processes.

answered Sep 2 '08 at 19:46
mweerden
4,536 2 17 28

There are two essential concepts to building concurrent programs - synchronization and mutual exclusion. We will see how these two types of locks (semaphores are more generally a kind of locking mechanism) help us achieve synchronization and mutual exclusion.

A semaphore is a programming construct that helps us achieve concurrency, by implementing both synchronization and mutual exclusion. Semaphores are of two types, Binary and Counting.

A semaphore has two parts : a counter, and a list of tasks waiting to access a particular resource. A semaphore performs two operations : wait (P) [this is like acquiring a lock], and release (V)[similar to releasing a lock] - these are the only two operations that one can perform on a semaphore. In a binary semaphore, the counter logically goes between 0 and 1. You can think of it as being similar to a lock with two values : open/closed. A counting semaphore has multiple values for count.

What is important to understand is that the semaphore counter keeps track of the number of tasks that do not have to block, i.e., they can make progress. Tasks block, and add themselves to the semaphore's list only when the counter is zero. Therefore, a task gets added to the list in the P() routine if it cannot progress, and "freed" using the V() routine.

Now, it is fairly obvious to see how binary semaphores can be used to solve synchronization and mutual exclusion - they are essentially locks.

ex. Synchronization:

```
thread A{
```

```
semaphore &s; //locks/semaphores are passed by reference! think about why this is so.
A(semaphore &s): s(s){} //constructor
foo(){
...
s.P();
;// some block of code B2
...
}

//thread B{
semaphore &s;
B(semaphore &s): s(s){} //constructor
foo(){
...
...
// some block of code B1
s.V();
..
}

main(){
semaphore s(0); // we start the semaphore at 0 (closed)
A a(s);
B b(s);
}
```

In the above example, B2 can only execute after B1 has finished execution. Let's say thread A comes executes first - gets to sem.P(), and waits, since the counter is 0 (closed). Thread B comes along, finishes B1, and then frees thread A - which then completes B2. So we achieve synchronization.

Now let's look at mutual exclusion with a binary semaphore:

```
thread mutual_ex{
semaphore &s;
mutual_ex(semaphore &s): s(s){} //constructor
foo(){
...
s.P();
//critical section
s.V();
...
...
s.P();
//critical section
s.V();
...

}

main(){
semaphore s(1);
mutual_ex m1(s);
mutual_ex m2(s);
}
```

The mutual exclusion is quite simple as well - m1 and m2 cannot enter the critical section at the same time. So each thread is using the same semaphore to provide mutual exclusion for its two critical sections. Now, is it possible to have greater concurrency? Depends on the critical sections. (Think about how else one could use semaphores to achieve mutual exclusion.. hint hint : do i necessarily only need to use one semaphore?)

Counting semaphore: A semaphore with more than one value. Let's look at what this is implying - a lock with more than one value?? So open, closed, and ...hmm. Of what use is a multi-stage-lock in mutual exclusion or synchronization?

Let's take the easier of the two:

Synchronization using a counting semaphore: Let's say you have 3 tasks - #1 and 2 you want executed after 3. How would you design your synchronization?

```
thread t1{
...
s.P();
//block of code B1

thread t2{
...
s.P();
//block of code B2

thread t3{
...
//block of code B3
s.V();
s.V();
}
```

So if your semaphore starts off closed, you ensure that t1 and t2 block, get added to the semaphore's list. Then along comes all important t3, finishes its business and frees t1 and t2. What order are they freed in? Depends on the implementation of the semaphore's list. Could be FIFO, could be based some particular priority,etc. (Note : think about how you would arrange your P's and V;s if you wanted t1 and t2 to be executed in some particular order, and if you weren't aware of the implementation of the semaphore)

(Find out : What happens if the number of V's is greater than the number of P's?)

Mutual Exclusion Using counting semaphores: I'd like you to construct your own pseudocode for this (makes you understand things better!) - but the fundamental concept is this : a counting semaphore of counter = N allows N tasks to enter the critical section freely. What this means is you have N tasks (or threads, if you like) enter the critical section, but the N+1th task gets blocked (goes on our favorite blocked-task list), and only is let through when somebody V's the semaphore at least once. So the semaphore counter, instead of swinging between 0 and 1, now goes between 0 and N, allowing N tasks to freely enter and exit, blocking nobody!

Now gosh, why would you need such a stupid thing? Isn't the whole point of mutual exclusion to not let more than one guy access a resource?? (Hint Hint...You don't always only have one drive in your computer, do you...?)

To think about : Is mutual exclusion achieved by having a counting semaphore alone? What if you have 10 instances of a resource, and 10 threads come in (through the counting semaphore) and try to use the first instance?

answered May 8 '14 at 15:23
aspen100
102 8

A hardware or software flag. In multi tasking systems , a semaphore is as variable with a value that indicates the status of a common resource.A process needing the resource checks the semaphore to determine the resources status and then decides how to proceed.

answered May 12 '11 at 8:40
MABUTOBRIGHTON
11 1

---

https://www.youtube.com/watch?v=VQFJQJGmixM Should give you basic idea of semaphore and mutes. it explains the concept in simplified terms.

answered Jan 13 '13 at 23:20
Vijay
21 2

1 Could you, perhaps, expand on this answer a bit? It's a bit... lacking for the breadth of the question. – Emrakul Jan 13 '13 at 23:38

It would help if you gave a summary of the 8 minute video in your answer, unless your aim is to get people to watch the video... – assylias Jan 14 '13 at 7:51

The video explains the concept in a more simple way. It relates it to more on a day to day concept which makes understanding easy. I have no intention that people watch this video. I am not being paid by the guy who made it. Watch it if you want else who cares. I found its easy for me to understand so gave the link. – Vijay Jan 21 '13 at 23:43

A semaphore is a way to lock a resource so that it is guaranteed that while a piece of code is executed, only this piece of code has access to that resource. This keeps two threads from concurrently accesing a resource, which can cause problems.

answered Aug 29 '08 at 16:01
Craig H
4,330 13 35 53

4 sounds like a mutex not a semaphore – Sam Sep 19 '11 at 0:57

This is an old question but one of the most interesting uses of semaphore is a read/write lock and it has not been explicitly mentioned.

The r/w locks works in simple fashion: consume one permit for a reader and all permits for writers. Indeed, a trivial implementation of a r/w lock but requires metadata modification on read (actually twice) that can become a bottle neck, still significantly better than a mutex or lock.

Another downside is that writers can be started rather easily as well unless the semaphore is a fair one or the writes acquire permits in multiple requests, in such case they need an explicit mutex between themselves.

Further read:

answered Feb 1 '12 at 21:58
bestsss
6,826 2 29 47

So imagine everyone is trying to go to the bathroom and there's only a certain number of keys to the bathroom. Now if there's not enough keys left, that person needs to wait. So think of semaphore as representing those set of keys available for bathrooms (the system resources) that different processes (bathroom goers) can request access to.

Now imagine two processes trying to go to the bathroom at the same time. That's not a good situation and semaphores are used to prevent this. Unfortunately, the semaphore is a voluntary mechanism and processes (our bathroom goers) can ignore it (i.e. even if there are keys, someone can still just kick the door open).

There are also differences between binary/mutex & counting semaphores.

Check out the lecture notes at http://www.cs.columbia.edu/~jae/4118/lect/L05-ipc.html.

answered Feb 13 '14 at 17:44
Columbia Student in Jae
11