How does lock work exactly?

Asked 8 years, 7 months ago Active 2 months ago Viewed 304k times

486

100

I see that for using objects which are not thread safe we wrap the code with a lock like this:

```
private static readonly Object obj = new Object();

lock (obj)
{
    // thread unsafe code
}
```

So what happens when multiple threads access the same code (let's assume that it is running in a ASP.NET web application). Are they queued? If so how long will they wait?

What is the performance impact because of using locks?

c# .net synchronization locking thread-safety

edited Oct 13 '13 at 18:37
ChrisWue
15.8k 3 42 69

asked May 17 '11 at 10:55
NLV
18k 34 111 176

1 ^ dead link, see: jonskeet.uk/csharp/threads/index.html – Ivan Pavičić Oct 17 at 13:19

8 Answers

426

The `lock` statement is translated by C# 3.0 to the following:

```
var temp = obj;

Monitor.Enter(temp);
```

✓

```
try
{
    // body
}
finally
{
    Monitor.Exit(temp);
}
```

In C# 4.0 this has changed and it is now generated as follows:

```
bool lockWasTaken = false;
var temp = obj;
try
{
    Monitor.Enter(temp, ref lockWasTaken);
    // body
}
finally
{
    if (lockWasTaken)
    {
        Monitor.Exit(temp);
    }
}
```

You can find more info about what `Monitor.Enter` does here. To quote MSDN:

> Use `Enter` to acquire the Monitor on the object passed as the parameter. If another thread has executed an `Enter` on the object but has not yet executed the corresponding `Exit` , the current thread will block until the other thread releases the object. It is legal for the same thread to invoke `Enter` more than once without it blocking; however, an equal number of `Exit` calls must be invoked before other threads waiting on the object will unblock.

The `Monitor.Enter` method will wait infinitely; it will *not* time out.

edited Apr 10 '14 at 10:05
Umar Abbas
3,582 1 14 21

answered May 17 '11 at 10:57
Steven
139k 18 246 365

10 According to MSDN "Using the lock (C#) or SyncLock (Visual Basic) keyword is generally preferred over using the Monitor class directly, both because lock or SyncLock is more concise, and because lock or SyncLock insures that the underlying monitor is released, even if the protected code throws an

exception. This is accomplished with the finally keyword, which executes its associated code block regardless of whether an exception is thrown." msdn.microsoft.com/en-us/library/ms173179.aspx – Aiden Strydom Mar 7 '14 at 21:02

10 What's the point of the var temp = obj; line. since it's just a ref to begin with, what good does making another one do? – priehl Sep 25 '14 at 17:47

11 @priehl It allows the user to change `obj` without the whole system to deadlock. – Steven Sep 25 '14 at 20:03

6 @Joymon eventually, every language feature is syntactic sugar. Language features are about about making developers more productive and making applications more maintainable, as is the lock feature as well. – Steven Mar 16 '16 at 2:12

2 Correct. This is the entire purpose of the `lock` -statement and Monitor: so that you can perform an operation in one thread without having to worry about another thread mucking it up. – Dizzy H. Muffin Jun 30 '17 at 15:56

261

Its simpler than you think.

According to Microsoft: The `lock` keyword ensures that one thread does not enter a critical section of code while another thread is in the critical section. If another thread tries to enter a locked code, it will wait, block, until the object is released.

The `lock` keyword calls `Enter` at the start of the block and `Exit` at the end of the block. `lock` keyword actually handles `Monitor` class at back end.

For example:

```
private static readonly Object obj = new Object();

lock (obj)
{
    // critical section
}
```

In the above code, first the thread enters a critical section, and then it will lock `obj`. When another thread tries to enter, it will also try to lock `obj`, which is already locked by the first thread. Second thread will have to wait for the first thread to release `obj`. When the first thread leaves, then another thread will lock `obj` and will enter the critical section.

edited Oct 16 at 20:49
S.Pradeep
157 3 10

answered Apr 10 '14 at 9:54
Umar Abbas
3,582 1 14 21

6 should we create a dummy object to lock or can we lock an existing variable in the context? – batmaci Aug 16 '16 at 14:53

8 @batmaci - Locking on a separate private dummy object gives you a guarantee that no one else is locking in that object. If you lock on the data and that same piece of data is visible to the outside you lose that guarantee. – Umar Abbas Aug 17 '16 at 5:23

8 What happens if more than one process is waiting for the lock to release? Are the waiting processes queued so that they will be lock the critical section in FIFO order? – jstuardo Jul 13 '17 at 14:56

@jstuardo - They are queued, but the order is not guaranteed to be FIFO. Check out this link: albahari.com/threading/part2.aspx – Umar Abbas Sep 27 at 12:57

Copied without attribution from net-informations.com/faq/qk/lock.htm – Martijn Pieters ♦ Nov 23 at 17:14

45

No, they are not queued, they are sleeping

A lock statement of the form

```
lock (x) ...
```

where x is an expression of a reference-type, is precisely equivalent to

```
var temp = x;
System.Threading.Monitor.Enter(temp);
try { ... }
finally { System.Threading.Monitor.Exit(temp); }
```

You just need to know that they are waiting to each other, and only one thread will enter to lock block, the others will wait...

Monitor is written fully in .net so it is enough fast, also look at class Monitor with reflector for more details

edited Sep 18 '13 at 10:33
Tolga Evcimen
5,778 10 43 74

answered May 17 '11 at 10:56
Arsen Mkrtchyan
44k 28 132 172

6 Note that the code emitted for the `lock` statement changed slightly in C#4: blogs.msdn.com/b/ericlippert/archive/2009/03/06/… – LukeH May 17 '11 at 11:09

@ArsenMkrt, they are not kept in "Blocked" state "Queue?. I think there are some difference between Sleep and block state, is not it? – Mohanavel Jan 13 '14 at 6:56

what difference you mean @Mohanavel? – Arsen Mkrtchyan Jan 13 '14 at 9:40

1 That was not the question. The question was regarding "lock" keyword. Suppose a process enter a "lock" section. That means that process blocks that piece of code and no other process may be able to enter that section until that lock is released. Well.... now, 2 more processes try to enter the same block. Since it is protected by "lock" keyword, they will wait, according to what was said in this forum. When the first process releases the lock. What process enters the block? the first one that tried to enter or the last one? – jstuardo Jul 14 '17 at 0:03

1 I guess you mean thread instead of process... if so, than the answer is No, there is no guarantee which one will enter... more here stackoverflow.com/questions/4228864/… – Arsen Mkrtchyan Jul 14 '17 at 10:03

27

Locks will block other threads from executing the code contained in the lock block. The threads will have to wait until the thread inside the lock block has completed and the lock is released. This does have a negative impact on performance in a multithreaded environment. If you do need to do this you should make sure the code within the lock block can process very quickly. You should try to avoid expensive activities like accessing a database etc.

edited Oct 24 '13 at 9:20

answered May 17 '11 at 11:01

Andrew
4,764 1 19 37

10

The performance impact depends on the way you lock. You can find a good list of optimizations here: http://www.thinkingparallel.com/2007/07/31/10-ways-to-reduce-lock-contention-in-threaded-programs/

Basically you should try to lock as little as possible, since it puts your waiting code to sleep. If you have some heavy calculations or long lasting code (e.g. file upload) in a lock it results in a huge performance loss.

answered May 17 '11 at 11:08

Simon Woker
4,740 22 38

1 But trying to write low-lock code can often result in subtle, hard-to-find-and-fix bugs, even if you're an expert in the field. Using a lock is often the lesser of two evils. You should lock exactly as much as you need to, no more, no less! – LukeH May 17 '11 at 11:13

1 @LukeH: There are some usage patterns where low-lock code can be very simple and easy [`do { oldValue = thing; newValue = updated(oldValue); } while (CompareExchange(ref thing, newValue, oldValue) != oldValue`]. The biggest danger is that if the requirements evolve beyond what such techniques can be handle, it may be hard to adapt the code to handle that. – supercat Nov 13 '13 at 19:45

Link has broken. – CarenRose Jan 31 at 20:16

7

The part within the lock statement can only be executed by one thread, so all other threads will wait indefinitely for it the thread holding the lock to finish. This can result in a so-called deadlock.

answered May 17 '11 at 10:58
Mr47
2,530 1 14 24

7

The `lock` statement is translated to calls to the `Enter` and `Exit` methods of [`Monitor`](#) .

The `lock` statement will wait indefinitely for the locking object to be released.

edited Apr 10 '14 at 10:50
Umar Abbas
3,582 1 14 21

answered May 17 '11 at 10:59
Paolo Tedesco
45.6k 26 122 172

3

[lock](#) is actually hidden [Monitor](#) class.

edited Dec 8 '17 at 22:09
johnnyRose
5,299 12 38 56

answered May 17 '11 at 10:58
Euphoric
11.3k 1 22 41