What is a mutex?

Asked 16 years, 3 months ago Modified 1 month ago

Viewed 512k times

996

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

What is a mutex and how do you use it?

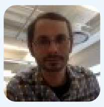
multithreading concurrency mutex

Share

Improve this question

Follow

asked Aug 29, 2008 at 15:59



bmurphy1976
31.1k ● 12 ● 34 ● 24

3 Here's a good article on the difference: barrgroup.com/Embedded-Systems/How-To/RTOS-Mutex-Semaphore – Adam Davis Apr 15, 2015 at 14:45

A tutorial on mutex can help clear things up: stackoverflow.com/questions/4989451/mutex-example-tutorial – Nav May 5, 2018 at 15:20

75 A mutex is like a bathroom key at a gas station, ensuring that only one person may use the bathroom at a time AND that no one else may use the toilet until the current occupant is finished and the key is returned. – jonschlinkert Jan 2, 2020 at 17:51

10 Answers

Sorted by: Highest score (default)

3035

When I am having a big heated discussion at work, I use a rubber chicken which I keep in my desk for just such occasions. The person holding the chicken is the only person who is allowed to talk. If you don't hold the chicken you cannot speak. You can only indicate that you want the chicken and wait until you get it before you speak. Once you have finished speaking, you can hand the chicken back to the moderator who will hand it to the next person to speak. This ensures that people do not speak over each other, and also have their own space to talk.

Replace Chicken with Mutex and person with thread and you basically have the concept of a mutex.

Of course, there is no such thing as a rubber mutex. Only rubber chicken. My cats once had a rubber mouse, but they ate it.

Of course, before you use the rubber chicken, you need to ask yourself whether you actually need 5 people in one room and would it not just be easier with one person in the room on their own doing all the work. Actually, this is just extending the analogy, but you get the idea.

Share Improve this answer Follow

edited May 3, 2023 at 10:01

answered Aug 29, 2008 at 16:07

Xetius
46.6k ● 25 ● 91 ● 124

7 @SirYakalot, you mean the chicken is the resource, and the moderator is the mutex? – Owen Oct 9, 2012 at 10:01

270 The chicken is the *mutex*. People hoilding the mu.. chicken are *competing threads*. The Moderator is the *OS*. When people requests the chicken, they do a lock request. When you call mutex.lock(), your thread stalls in lock() and makes a lock request to the OS. When the OS detects that the mutex was released from a thread, it merely gives it to you, and lock() returns - the mutex is now yours and only yours. Nobody else can steal it, because calling lock() will block him. There is also try_lock() that will block and return true when mutex is yours and immediately false if mutex is in use. – Петър Петров Sep 21, 2014 at 22:51

196 Sometimes the origin of some programming concepts is unclear. A newbie might go around wondering why everyone is talking about regex. It isn't apparent that regex is short for [reg]ular [ex]pression. Similarly, mutex is short for [mut]ual [ex]clusion. This might make the meaning of the term easier to digest. @TheSmurf linked to it in their answer, but it might be good to add it here for historical purposes. – Dodzi Dzakuma Dec 10, 2015 at 19:15

2 Someone read "Lord of the Flies" :-) RubberChicken == Conch sparknotes.com/lit/flies/central-idea-essay/…
– Cerniuk Nov 21, 2019 at 12:04

5 Not to be confused with *rubber duck debugging*.
– MC Emperor Jul 13, 2021 at 10:18

284

A Mutex is a **Mut**ually **ex**clusive flag. It acts as a gate keeper to a section (or sections) of code allowing one thread in and blocking access to all others. This ensures that the code being controlled will only be hit by a single thread at a time. Just be sure to release the mutex when you are done. :)

+50

Share Improve this answer Follow

edited Nov 16 at 12:13
paxdiablo
880k ● 241 ● 1.6k ● 2k

answered Aug 29, 2008 at 16:06
Craig
11.9k ● 13 ● 45 ● 62

27 A mutex has nothing to do with a section of code per se, it protects some *resource.* That resource may be a code segment if the mutex is only ever used around that code but, the minute you start using the mutex in multiple places in your code, your explanation fails. Typically it can also be used to protect some data structure, which may be accessed from many places in the code. – paxdiablo Nov 22, 2017 at 7:47

1 @paxdiablo: allow me to disagree. You can have 2 pieces of code, one using the Mutex, the other doesn't. You access the data structure from both pieces of code. How does the Mutex protect the data structure? It doesn't. You will have data races as if no Mutex was in place at all. – Thomas Weller Feb 16, 2021 at 12:39

2 @Thomas, then your code has a bug. To say the mutex doesn't protect the data because you're not using the mutex at one of the points you need to is no different than saying your home security is deficient because, on Thursdays, you

leave the front door unlocked and open when you go to work :-) – paxdiablo Feb 16, 2021 at 21:53

2 @paxdiablo: that's my point: the developer has to follow a convention that he only accesses the data structure using *code* that is protected by the Mutex. The Mutex protects the code. It does not protect the data structure. I would say my home security is useless if I regularly leave the door open. I then have implemented it the wrong way, like I implemented code the wrong way. – Thomas Weller Feb 17, 2021 at 7:37

1 @Thomas, I suspect it's going to just come down to semantics. Forgetting to use a mutex to protect data is, in my view, no different to forgetting to use it to protect code. You've forgotten to protect it at some point, hilarity will ensue :-) I can see your viewpoint but I have to stand by my description because there's usually no problem at all with multiple threads running the same *code*, since the code itself doesn't change underneath them. It's only when the *data* changes unexpectedly will they run into trouble. – paxdiablo Feb 17, 2021 at 10:30

123

Mutual **Ex**clusion. Here's the Wikipedia entry on it.

The point of a mutex is to synchronize two threads. When you have two threads attempting to access a single resource, the general pattern is to have the first block of code attempting access to set the mutex before entering the code. When the second code block attempts access, it sees that the mutex is set and waits until the first block of code is complete (and unsets the mutex), then continues.

Specific details of how this is accomplished obviously varies greatly by programming language.

Share Improve this answer Follow

edited Oct 16, 2020 at 0:51
Pang
10.1k ● 146 ● 85 ● 124

answered Aug 29, 2008 at 16:05
TheSmurf
15.6k ● 3 ● 41 ● 48

I like how you explained this simply. Sometimes it's good to define the basic concept. I was trying to understand the context of the problem being solved, and how it's being solved, which is always a good start to deep dive into the technicalities if need be. Thank you' – Emmanuel Motsi Feb 14, 2023 at 7:47

1 This answer deserves the top spot. – gene.bustam Oct 23, 2023 at 15:35

81

When you have a multi-threaded application, the different threads sometimes share a common resource, such as a variable or similar. This shared source often cannot be accessed at the same time, so a construct is needed to ensure that only one thread is using that resource at a time.

The concept is called "mutual exclusion" (short Mutex), and is a way to ensure that only one thread is allowed inside that area, using that resource etc.

How to use them is language specific, but is often (if not always) based on a operating system mutex.

Some languages doesn't need this construct, due to the paradigm, for example functional programming (Haskell, ML are good examples).

Share Improve this answer Follow

edited Aug 3, 2019 at 11:44
Neurotransmitter
6,787 ● 2 ● 53 ● 40

answered Aug 29, 2008 at 16:04
Mats Fredriksson
20.1k ● 6 ● 38 ● 57

47

What is a Mutex?

The mutex (In fact, the term mutex is short for mutual exclusion) also known as spinlock is the simplest synchronization tool that is used to protect critical regions and thus prevent race conditions. That is a thread must acquire a lock before entering into a critical section (In critical section multi threads share a common variable, updating a table, writing a file and so on), it releases the lock when it leaves critical section.

What is a Race Condition?

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

Real life example:

> When I am having a big heated discussion at work, I use a rubber chicken which I keep in my desk for just such occasions. The person holding the chicken is the only person who is allowed to talk. If you don't hold the chicken you cannot speak. You can only indicate that you want the chicken and wait until you get it before you speak. Once you have finished speaking, you can hand the chicken back to the moderator who will hand it to the next person to speak. This ensures that people do not speak over each other, and also have their own space to talk.
>
> Replace Chicken with Mutex and person with thread and you basically have the concept of a mutex.
>
> @Xetius

Usage in C#:

This example shows how a local Mutex object is used to synchronize access to a protected resource. Because each calling thread is blocked until it acquires ownership of the mutex, it must call the ReleaseMutex method to release ownership of the thread.

```
using System;
using System.Threading;

class Example
{
    // Create a new Mutex. The creating thread does no
    private static Mutex mut = new Mutex();
    private const int numIterations = 1;
    private const int numThreads = 3;

    static void Main()
    {
        // Create the threads that will use the protec
        for(int i = 0; i < numThreads; i++)
        {
            Thread newThread = new Thread(new ThreadSt
            newThread.Name = String.Format("Thread{0}"
            newThread.Start();
        }

        // The main thread exits, but the application
        // run until all foreground threads have exite
    }

    private static void ThreadProc()
    {
        for(int i = 0; i < numIterations; i++)
        {
            UseResource();
        }
    }

    // This method represents a resource that must be
    // so that only one thread at a time can enter.
    private static void UseResource()
    {
        // Wait until it is safe to enter.
        Console.WriteLine("{0} is requesting the mutex
                          Thread.CurrentThread.Name);
        mut.WaitOne();

        Console.WriteLine("{0} has entered the protect
                          Thread.CurrentThread.Name);

        // Place code to access non-reentrant resource

        // Simulate some work.
        Thread.Sleep(500);

        Console.WriteLine("{0} is leaving the protecte
            Thread.CurrentThread.Name);

        // Release the Mutex.
        mut.ReleaseMutex();
        Console.WriteLine("{0} has released the mutex"
            Thread.CurrentThread.Name);
    }
}
// The example displays output like the following:
//       Thread1 is requesting the mutex
//       Thread2 is requesting the mutex
//       Thread1 has entered the protected area
//       Thread3 is requesting the mutex
//       Thread1 is leaving the protected area
//       Thread1 has released the mutex
//       Thread3 has entered the protected area
//       Thread3 is leaving the protected area
//       Thread3 has released the mutex
//       Thread2 has entered the protected area
//       Thread2 is leaving the protected area
//       Thread2 has released the mutex
```

MSDN Reference Mutex

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

Community Bot
1 ● 1

answered Jul 3, 2018 at 2:53

habib
2,446 ● 5 ● 27 ● 44

How is Mutex implemented though? Is it hardware based? Does it have some waiting mechanism to make sure all threads know what is the current state? – android developer Apr 1, 2021 at 23:11

What you explained in Race Condition is actually Data Race, Race Condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly. – Nedim AKAR Jan 14, 2023 at 11:00

35

There are some great answers here, here is another great analogy for explaining what **mutex** is:

Consider **single** toilet with a *key*. When someone enters, they take the key and the toilet is *occupied*. If someone else needs to use the toilet, they need to wait in a *queue*. When the person in the toilet is *done*, they pass the key to the next person in queue. Make sense, right?

Convert the *toilet* in the story to a *shared resource*, and the *key* to a *mutex*. Taking the key to the toilet (acquire a lock) permits you to use it. If there is no key (the lock is locked) you have to wait. When the key is returned by the person (*release the lock*) you're free to acquire it now.

answered Oct 9, 2017 at 8:54

Share Improve this answer Follow

Chen A.
11.2k ● 4 ● 51 ● 72

But the c# example does not support support your queue assertion, "pass the key to the next person in queue". The example is demonstrating a stack or random. 1, 2, & 3 all request access, in that sequence. One is first allowed into the protected area, and then three is allowed. A queue would have given it to the second. – donvnielsen Dec 26, 2019 at 22:32

7 I wasn't referring any concrete implementation, or concrete programming language. My example relates to high level abstraction of mutex as a principle. – Chen A. Dec 27, 2019 at 9:39

30

In C#, the common mutex used is the Monitor. The type is 'System.Threading.Monitor'. It may also be used implicitly via the 'lock(Object)' statement. One example of its use is when constructing a Singleton class.

```
private static readonly Object instanceLock = new Obje
private static MySingleton instance;
public static MySingleton Instance
{
    lock(instanceLock)
    {
        if(instance == null)
        {
            instance = new MySingleton();
        }
        return instance;
    }
}
```

The lock statement using the private lock object creates a critical section. Requiring each thread to wait until the previous is finished. The first thread will enter the section and initialize the instance. The second thread will wait, get into the section, and get the initialized instance.

Any sort of synchronization of a static member may use the lock statement similarly.

Share Improve this answer Follow

edited Apr 8, 2015 at 2:07
Radiodef
37.8k ● 14 ● 93 ● 126

answered Aug 29, 2008 at 16:11
Anthony Mastrean
22.4k ● 22 ● 115 ● 190

1 This is an implementation dependent answer. Also, In CS a monitor is different than mutex. Monitors has a synchronization mechanism but mutex just lock the thing until no longer needed. IDK about implementation details or C# semantics, but I think the context of the question is broader – marcoslhc Oct 3, 2015 at 16:21

22

To understand MUTEX at first you need to know what is "race condition" and then only you will understand why MUTEX is needed. Suppose you have a multi-threading program and you have two threads. Now, you have one job in the job queue. The first thread will check the job queue and after finding the job it will start executing it. The second thread will also check the job queue and find that there is one job in the queue. So, it will also assign the same job pointer. So, now what happens, both the threads are executing the same job. This will cause a segmentation fault. This is the example of a race condition.

The solution to this problem is MUTEX. MUTEX is a kind of lock which locks one thread at a time. If another thread wants to lock it, the thread simply gets blocked.

The MUTEX topic in this pdf file link is really worth reading.

Share Improve this answer Follow

answered Jun 27, 2014 at 9:57

user3751012
551 ● 1 ● 8 ● 21

By "MUTEX topic" you meant the section on semaphores, because its examples are of binary semaphores, right? – Carl G Dec 9, 2014 at 4:01

3 well a Mutex is just a semaphore with value 1 – marcoslhc Oct 3, 2015 at 16:22

What is the name of the book of that chapter you shared? Please – Omar Faroque Anik Feb 27, 2017 at 15:55

@OmarFaroqueAnik the book referenced is Advanced Linux Programming by CodeSourcery LLC, published by New Riders Publishing and can be accessed from the linked domain's home page. – RMart Mar 1, 2017 at 20:23

1 To understand "segmentation fault" at first you need to know what is... – Schrodo_Baggins Sep 3, 2020 at 13:38

11

Mutexes are useful in situations where you need to enforce exclusive access to a resource accross multiple processes, where a regular lock won't help since it only works accross threads.

Share Improve this answer Follow

answered Sep 2, 2008 at 18:57

18hrs
1,124 ● 1 ● 9 ● 10

1 Is that really true? Wont individual processes create their own mutex copy? – Leon Apr 8, 2015 at 18:09

I don't agree with "a regular lock won't help since it only works across threads" - this may be implementation specific , or a semantics/definition question. I worked in Intersystems Caché / M[UMPS] systems for many years, and Locks in M are cross-process. Or maybe a MUMPS lock is actually a mutex, and they just named it incorrectly back in the 70s. – nigh_anxiety Aug 12, 2023 at 19:28

6

Mutex: Mutex stands for **Mut**ual **Ex**clusion. It means only one process/thread can enter into critical section at a given time. In concurrent programming multiple threads/process updating the shared resource (any variable, shared memory etc.) may lead to some unexpected result. (As the result depends upon the which thread/process gets the first access).

In order to avoid such an unexpected result we need some synchronization mechanism, which ensures that only one thread/process gets access to such a resource at a time.

pthread library provides support for Mutex.

```
typedef union
{
  struct __pthread_mutex_s
  {
    ***int __lock;***
    unsigned int __count;
    int __owner;
#ifdef __x86_64__
    unsigned int __nusers;
#endif
int __kind;
#ifdef __x86_64__
    short __spins;
    short __elision;
    __pthread_list_t __list;
# define __PTHREAD_MUTEX_HAVE_PREV      1
# define __PTHREAD_SPINS             0, 0
#else
    unsigned int __nusers;
    __extension__ union
    {
      struct
      {
        short __espins;
        short __elision;
# define __spins __elision_data.__espins
# define __elision __elision_data.__elision
# define __PTHREAD_SPINS         { 0, 0 }
      } __elision_data;
      __pthread_slist_t __list;
    };
#endif
```

This is the structure for mutex data type i.e pthread_mutex_t. When mutex is locked, __lock set to 1.

When it is unlocked __lock set to 0.

This ensure that no two processes/threads can access the critical section at same time.

Share Improve this answer Follow

edited Aug 21, 2020 at 4:29
Community Bot
1 ● 1

answered Jan 23, 2020 at 12:34
Sandeep_black
1,431 ● 18 ● 19

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.