

What is a race condition? [closed]

Asked 16 years, 3 months ago Modified 1 month ago

Viewed 908k times



1346



Closed. This question needs to be more [focused](#). It is not currently accepting answers.

💡 **Want to improve this question?** Update the question so it focuses on one problem only by [editing this post](#).

Closed last month.

The community reviewed whether to reopen this question last month and left it closed:

Original close reason(s) were not resolved

[Improve this question](#)

When writing multithreaded applications, one of the most common problems experienced is race conditions.

My questions to the community are:

- What is the race condition?
- How do you detect them?
- How do you handle them?
- Finally, how do you prevent them from occurring?

[multithreading](#)[concurrency](#)[terminology](#)[race-condition](#)[Share](#)[Improve this question](#)[Follow](#)

edited Oct 15, 2021 at 15:42

[Gray](#)

117k ● 24 ● 302 ● 359

asked Aug 29, 2008 at 15:55

[bmurphy1976](#)

31.1k ● 12 ● 34 ● 24

-
- 11 I'd like to mention that - without specifying the language - most parts of this question cannot be answered properly, because in different languages, the definition, the consequences and the tools to prevent them might differ.
– [MikeMB](#) Apr 21, 2015 at 17:18
-

@MikeMB. Agreed, except when analyzing byte code execution, like it is done by Race Catcher (see this thread stackoverflow.com/a/29361427/1363844) we can address all those approximately 62 languages that compile to byte code (see en.wikipedia.org/wiki/List_of_JVM_languages) – [Ben](#)
Aug 12, 2016 at 5:49 ✎

19 Answers

Sorted by:

Highest score (default)



1686



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.



Problems often occur when one thread does a "check-then-act" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g:

```
if (x == 5) // The "Check"
{
    y = x * 2; // The "Act"

    // If another thread changed x in between "if
    (x == 5)" and "y = x * 2" above,
    // y will not be equal to 10.
}
```

The point being, y could be 10, or it could be anything, depending on whether another thread changed x in between the check and act. You have no real way of knowing.

In order to prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time. This would mean something like this:

```
// Obtain lock for x
if (x == 5)
{
    y = x * 2; // Now, nothing can change x until
    the lock is released.
    // Therefore y = 10
}
```

```
}  
// release lock for x
```

Share Improve this answer

edited Apr 7, 2015 at 11:03

Follow



Amit Joki

59.2k ● 7 ● 79 ● 96

answered Aug 29, 2008 at 16:05



Lehane

48.6k ● 14 ● 55 ● 54

179 What does the other thread do when it encounters the lock? Does it wait? Error? – [Brian Ortiz](#) Oct 19, 2009 at 1:58

243 Yes, the other thread will have to wait until the lock is released before it can proceed. This makes it very important that the lock is released by the holding thread when it is finished with it. If it never releases it, then the other thread will wait indefinitely. – [Lehane](#) Oct 22, 2009 at 9:01

3 @Ian In a multithreaded system there will always be times when resources need to be shared. To say that one approach is bad without giving an alternative just isn't productive. I'm always looking for ways to improve and if there is an alternative I will gladly research it and weigh the pro's and cons. – [Despertar](#) May 3, 2012 at 5:16

3 @Despertar ...also, its not necessarily the case that resources will always need to be shared in a multi-threaded system. For example you might have an array where each element needs processing. You could possibly partition the array and have a thread for each partition and the threads can do their work completely independently of one another. – [Ian Warburton](#) May 3, 2012 at 17:29

26 For a race to occur it's enough that a single thread attempts to change the shared data while rest of the threads can either read or change it. – [SomeWittyUsername](#) Nov 9, 2012 at 16:13



262



A "race condition" exists when multithreaded (or otherwise parallel) code that would access a shared resource could do so in such a way as to cause unexpected results.

Take this example:



```
for ( int i = 0; i < 10000000; i++ )  
{  
    x = x + 1;  
}
```

If you had 5 threads executing this code at once, the value of x WOULD NOT end up being 50,000,000. It would in fact vary with each run.

This is because, in order for each thread to increment the value of x, they have to do the following: (simplified, obviously)

```
Retrieve the value of x  
Add 1 to this value  
Store this value to x
```

Any thread can be at any step in this process at any time, and they can step on each other when a shared resource

is involved. The state of x can be changed by another thread during the time between x is being read and when it is written back.

Let's say a thread retrieves the value of x , but hasn't stored it yet. Another thread can also retrieve the **same** value of x (because no thread has changed it yet) and then they would both be storing the **same** value ($x+1$) back in x !

Example:

```
Thread 1: reads x, value is 7
Thread 1: add 1 to x, value is now 8
Thread 2: reads x, value is 7
Thread 1: stores 8 in x
Thread 2: adds 1 to x, value is now 8
Thread 2: stores 8 in x
```

Race conditions can be avoided by employing some sort of **locking** mechanism before the code that accesses the shared resource:

```
for ( int i = 0; i < 100000000; i++ )
{
    //lock x
    x = x + 1;
    //unlock x
}
```

Here, the answer comes out as 50,000,000 every time.

For more on locking, search for: mutex, semaphore, critical section, shared resource.

Share Improve this answer

edited Nov 12, 2015 at 19:31

Follow



IKavanagh

6,189 ● 11 ● 45 ● 48

answered Aug 29, 2008 at 17:01



privatehuff

2,879 ● 1 ● 15 ● 7

See jakob.engbloms.se/archives/65 for an example of a program to test how often such things go bad... it really depends on the memory model of the machine you are running on. – [jakobengblom2](#) Oct 12, 2008 at 19:54

1 How can it get to 50 million if it has to stop at 10 million?
– user4624979 Oct 26, 2015 at 14:24

12 @nocomprende: By 5 threads executing the same code at a time, as described directly below the snippet... – [Jon Skeet](#) Nov 12, 2015 at 18:47

4 @JonSkeet You are right, I confused the i and the x. Thank you. – user4624979 Nov 12, 2015 at 18:57

1 Double check locking in implementing Singleton pattern is such an example of preventing race condition.
– [Bharat Dodeja](#) Aug 31, 2016 at 8:02 ✎



What is a Race Condition?

201



You are planning to go to a movie at 5 pm. You inquire about the availability of the tickets at 4 pm. The representative says that they are available. You relax and reach the ticket window 5 minutes before the show. I'm



sure you can guess what happens: it's a full house. The problem here was in the duration between the check and the action. You inquired at 4 and acted at 5. In the meantime, someone else grabbed the tickets. That's a race condition - specifically a "check-then-act" scenario of race conditions.

How do you detect them?

Religious code review, multi-threaded unit tests. There is no shortcut. There are few Eclipse plugin emerging on this, but nothing stable yet.

How do you handle and prevent them?

The best thing would be to create side-effect free and stateless functions, use immutables as much as possible. But that is not always possible. So using `java.util.concurrent.atomic`, concurrent data structures, proper synchronization, and actor based concurrency will help.

The best resource for concurrency is JCIP. You can also get some more [details on above explanation here](#).

Share Improve this answer

Follow

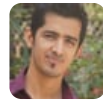
edited Aug 5, 2015 at 6:11



ekostadinov

6,942 ● 3 ● 30 ● 50

answered Oct 4, 2013 at 21:20



Vishal Shukla

2,988 ● 2 ● 18 ● 21

Code reviews and unit tests are secondary to modeling the flow between your ears, and making less use of shared memory. – [Asclepius](#) Nov 25, 2013 at 22:49

11 I appreciated the real world example of a race condition – [Tom O.](#) Jun 6, 2017 at 17:45

28 Like the answer *thumbs up*. Solution is: you lock the tickets between 4-5 with mutex (mutual exception, c++). In real world it is called ticket reservation :) – [Volt](#) Aug 25, 2017 at 12:11 ✎

2 would be a decent answer if you dropped the java-only bits (the question is not about Java, but rather race conditions in general) – [Corey Goldberg](#) Aug 26, 2018 at 7:12

1 Great answer indeed, Race conditions will occur if at all we come across "CHECK AND ACT" and "READ_MODIFY_UPDATE" situations – [Sumanth Varada](#) Jan 10, 2019 at 9:50



90



There is an important technical difference between race conditions and data races. Most answers seem to make the assumption that these terms are equivalent, but they are not.

A data race occurs when 2 instructions access the same memory location, at least one of these accesses is a write and there is no *happens before ordering* among these accesses. Now what constitutes a happens before ordering is subject to a lot of debate, but in general unlock-lock pairs on the same lock variable and wait-signal pairs

on the same condition variable induce a happens-before order.

A race condition is a semantic error. It is a flaw that occurs in the timing or the ordering of events that leads to erroneous program *behavior*.

Many race conditions can be (and in fact are) caused by data races, but this is not necessary. As a matter of fact, data races and race conditions are neither the necessary, nor the sufficient condition for one another. [This](#) blog post also explains the difference very well, with a simple bank transaction example. Here is another simple [example](#) that explains the difference.

Now that we nailed down the terminology, let us try to answer the original question.

Given that race conditions are semantic bugs, there is no general way of detecting them. This is because there is no way of having an automated oracle that can distinguish correct vs. incorrect program behavior in the general case. Race detection is an undecidable problem.

On the other hand, data races have a precise definition that does not necessarily relate to correctness, and therefore one can detect them. There are many flavors of data race detectors (static/dynamic data race detection, lockset-based data race detection, happens-before based data race detection, hybrid data race detection). A state of the art dynamic data race detector is [ThreadSanitizer](#) which works very well in practice.

Handling data races in general requires some programming discipline to induce happens-before edges between accesses to shared data (either during development, or once they are detected using the above mentioned tools). this can be done through locks, condition variables, semaphores, etc. However, one can also employ different programming paradigms like message passing (instead of shared memory) that avoid data races by construction.

Share Improve this answer

edited May 23, 2017 at 10:31

Follow



Community Bot

1 • 1

answered Aug 29, 2013 at 8:45



Baris Kasikci

3,197 • 1 • 17 • 8

The difference is critical to understand race condition.

Thanks! – ProgramCpp Apr 3, 2018 at 8:33

"It is a flaw that occurs in the timing or the ordering of events that leads to erroneous program behavior." Perfect definition! Indeed, there is no reason to assume that the events must occur within one instance of an application. Multiple instances are just as applicable. – truefusion Jul 8, 2020 at 4:11



49

A race condition is a situation on concurrent programming where two concurrent threads or processes compete for a resource and the resulting final state depends on who gets the resource first.



Share Improve this answer

edited Feb 9, 2019 at 17:35

Follow



answered Aug 29, 2008 at 16:07



Jorge Córdoba

52.1k ● 11 ● 82 ● 130

just brilliant explanation – [gokareless](#) Feb 9, 2019 at 17:12

Final state of what? – [Roman Alexandrovich](#) Jun 20, 2019 at 6:41

- 2 @RomanAlexandrovich The final state of the program. The state referring to things such as the values of variables, etc. See Lehane's excellent answer. The "state" in his example would refer to the final values of 'x' and 'y'. – [Alexander Terp](#) Nov 11, 2019 at 11:14
-



47



A sort-of-canonical definition is "*when two threads access the same location in memory at the same time, and at least one of the accesses is a write.*" In the situation the "reader" thread may get the old value or the new value, depending on which thread "wins the race." This is not always a bug—in fact, some really hairy low-level algorithms do this on purpose—but it should generally be avoided. @Steve Gury give's a good example of when it might be a problem.

Share Improve this answer


answered Aug 29, 2008 at 16:21

Follow



Chris Conway

56k ● 43 ● 131 ● 155

-
- 3 Could you please give an example of how race conditions can be useful? Googling didn't help. – [Alex V.](#) Dec 11, 2013 at 14:47 
-
- 3 @Alex V. At this point, I have no idea what I was talking about. I think this may have been a reference to lock-free programming, but it's not really accurate to say that depends on race conditions, per se. – [Chris Conway](#) Dec 12, 2013 at 15:31
-



A race condition is a kind of bug, that happens only with certain temporal conditions.

40

Example: Imagine you have two threads, A and B.



In Thread A:



```
if( object.a != 0 )  
    object.avg = total / object.a
```

In Thread B:

```
object.a = 0
```

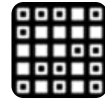
If thread A is preempted just after having check that `object.a` is not null, B will do `a = 0`, and when thread A will gain the processor, it will do a "divide by zero".

This bug only happen when thread A is preempted just after the if statement, it's very rare, but it can happen.

Share Improve this answer

Follow

edited May 16, 2013 at 5:08



Blorgbeard

103k ● 50 ● 235 ● 276

answered Aug 29, 2008 at 16:03



Steve Gury

15.6k ● 6 ● 39 ● 42



33

Many answers in this discussion explains what a race condition is. I try to provide an explanation why this term is called `race condition` in software industry.



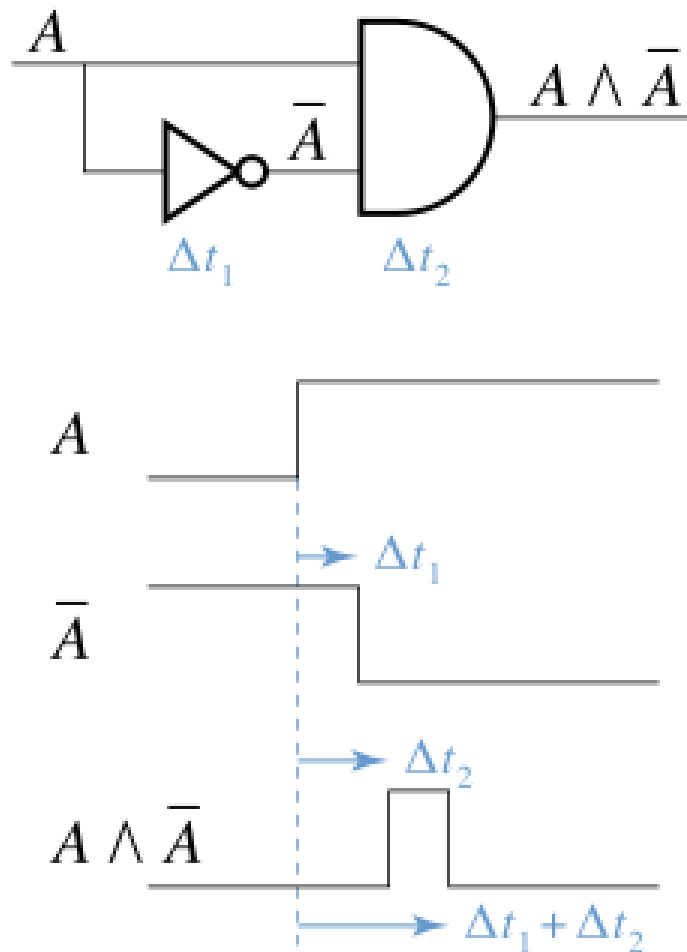
Why is it called `race condition` ?

Race condition is not only related with software but also related with hardware too. Actually the term was initially coined by the hardware industry.

According to [wikipedia](https://en.wikipedia.org/wiki/Race_condition):

The term originates with the idea of **two signals racing each other to influence the output first**.

Race condition in a logic circuit:



Software industry took this term without modification, which makes it a little bit difficult to understand.

You need to do some replacement to map it to the software world:

- "two signals" ==> "two threads"/"two processes"
- "influence the output" ==> "influence some shared state"

So race condition in software industry means "two threads"/"two processes" racing each other to "influence some shared state", and the final result of the shared state will depend on some subtle timing difference, which

could be caused by some specific thread/process launching order, thread/process scheduling, etc.

Share Improve this answer

edited Nov 3, 2022 at 1:28

Follow

answered Aug 4, 2017 at 3:57



nybon

9,561 ● 10 ● 62 ● 73



20



Race conditions occur in multi-threaded applications or multi-process systems. A race condition, at its most basic, is anything that makes the assumption that two things not in the same thread or process will happen in a particular order, without taking steps to ensure that they do. This happens commonly when two threads are passing messages by setting and checking member variables of a class both can access. There's almost always a race condition when one thread calls sleep to give another thread time to finish a task (unless that sleep is in a loop, with some checking mechanism).

Tools for preventing race conditions are dependent on the language and OS, but some common ones are mutexes, critical sections, and signals. Mutexes are good when you want to make sure you're the only one doing something. Signals are good when you want to make sure someone else has finished doing something. Minimizing shared resources can also help prevent unexpected behaviors

Detecting race conditions can be difficult, but there are a couple signs. Code which relies heavily on sleeps is prone to race conditions, so first check for calls to sleep in the affected code. Adding particularly long sleeps can also be used for debugging to try and force a particular order of events. This can be useful for reproducing the behavior, seeing if you can make it disappear by changing the timing of things, and for testing solutions put in place. The sleeps should be removed after debugging.

The signature sign that one has a race condition though, is if there's an issue that only occurs intermittently on some machines. Common bugs would be crashes and deadlocks. With logging, you should be able to find the affected area and work back from there.

Share Improve this answer

answered Aug 29, 2008 at 16:12

Follow



tsellon

2,426 ● 5 ● 25 ● 33



12

Microsoft actually have published a really detailed [article](#) on this matter of race conditions and deadlocks. The most summarized abstract from it would be the title paragraph:



A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to

see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.

Share Improve this answer

answered Sep 14, 2012 at 8:00

Follow



Konstantin Dinev

34.9k ● 14 ● 78 ● 102



What is a race condition?

12



The situation when the process is critically dependent on the sequence or timing of other events.



For example, Processor A and processor B **both needs** identical resource for their execution.



How do you detect them?

There are tools to detect race condition automatically:

- [Lockset-Based Race Checker](#)
- [Happens-Before Race Detection](#)
- [Hybrid Race Detection](#)

How do you handle them?

Race condition can be handled by **Mutex** or **Semaphores**. They act as a lock allows a process to acquire a resource based on certain requirements to prevent race condition.

How do you prevent them from occurring?

There are various ways to prevent race condition, such as **Critical Section Avoidance**.

1. No two processes simultaneously inside their critical regions. (**Mutual Exclusion**)
2. No assumptions are made about speeds or the number of CPUs.
3. No process running outside its critical region which blocks other processes.
4. No process has to wait forever to enter its critical region. (A waits for B resources, B waits for C resources, C waits for A resources)

Share Improve this answer

Follow

edited Mar 26, 2019 at 18:14



Trieu Toan

9 ● 2

answered Nov 14, 2014 at 13:43



Adnan Qureshi

562 ● 1 ● 6 ● 16



3

You can *prevent race condition*, if you use "Atomic" classes. The reason is just the thread don't separate operation get and set, example is below:



```
AtomicInteger ai = new AtomicInteger(2);  
ai.getAndAdd(5);
```

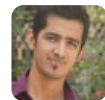


As a result, you will have 7 in link "ai". Although you did two actions, but the both operation confirm the same thread and no one other thread will interfere to this, that means no race conditions!

Share Improve this answer

Follow

edited Aug 19, 2017 at 18:19



Vishal Shukla

2,988 ● 2 ● 18 ● 21

answered Aug 12, 2017 at 14:48

Aleksei Moshkov

91 ● 11



3

I made a video that explains this.



Essentially it is when you have a state with is shared across multiple threads and before the first execution on a given state is completed, another execution starts and the new thread's initial state for a given operation is wrong because the previous execution has not completed.



Because the initial state of the second execution is wrong, the resulting computation is also wrong. Because

eventually the second execution will update the final state with the wrong result.



You can view it here. <https://youtu.be/RWRicNoWKoY>

Share Improve this answer

edited May 8, 2022 at 8:00

Follow

answered May 8, 2022 at 6:36



zacksiri

191 ● 1 ● 4



The answers given above are excellent enough to explain technically what is race condition.

1



I just want to add explanation from a layman's term/perspective:



'Race condition' is like in a race which involves more than 1 participant. Whoever reach the finishing line first is the only winner. We assume chances of winning are equal among participants. Let say the race is repeated more than once. So we can't predict exactly who will be the winner in each race. There is always a **potential** that different winner will win in each race.

Here where the problem comes in - If the finishing line is a resource, and a participant is a process, **potentially** different process will reach the resource at the end of every race and become a winner.

The problem involving race condition is that if process A changed the value in the beginning of 'race', it is not guaranteed that process A will reach the same value in the resource again in the end (finishing line), since A **potentially** might lose the race. If other process e.g. B become the winner, then B **may** change the value set by A before process A reach it. If this happened, process A lose its value and will cause problem to process A.

So issue with 'race condition' is the **potential** of a process lost its value from shared resource, caused by

the modification by other process. Race condition is not a problem/issue, if

1. no change/update happened, or
2. if the same process able to regain and control its value, or
3. no sharing resource involved.

Problem with race condition can be solved by adding an 'assurance' that no other process can access the shared resource while a process is using it (read or write). The period of time for the assurance is called the 'critical section'.

This assurance can be provided by creating a lock. E.g. If a process need to use a shared resource, it can lock the resource and release it when it is done, as the steps shown below. The lock may use the mechanism called Semaphore or Mutex. Meanwhile other process that need to use the shared resource will do the same steps.

```
wait until Mutex is unlocked
set Mutex=lock
begin read/write value in shared resource
..... do something
finish read/write value in shared resource
set Mutex=unlock
```

This way a process A can ensure no other process will update the shared resource while A is using the resource. The same issue will apply for thread.

Share Improve this answer

edited Feb 11 at 12:02

Follow

answered Feb 11 at 0:55



Yunus

84 ● 5



0



Here is the classical Bank Account Balance example which will help newbies to understand Threads in Java easily w.r.t. race conditions:

```
public class BankAccount {

    /**
     * @param args
     */
    int accountNumber;
    double accountBalance;

    public synchronized boolean Deposit(double amount){
        double newAccountBalance=0;
        if(amount<=0){
            return false;
        }
        else {
            newAccountBalance = accountBalance+amount;
            accountBalance=newAccountBalance;
            return true;
        }
    }

    public synchronized boolean Withdraw(double amount){
        double newAccountBalance=0;
        if(amount>accountBalance){
            return false;
        }
        else{
            newAccountBalance = accountBalance-amount;
        }
    }
}
```



```

        accountBalance=newAccountBalance;
        return true;
    }
}

public static void main(String[] args) {
    // TODO Auto-generated method stub
    BankAccount b = new BankAccount();
    b.accountBalance=2000;
    System.out.println(b.Withdraw(3000));
}

```

Share Improve this answer

edited May 16, 2013 at 5:12

Follow



Blorgbeard

103k ● 50 ● 235 ● 276

answered Nov 22, 2011 at 5:35



realPK

2,941 ● 31 ● 24

on deposit method if amount in negative value , people can deposit right – [Parthasarathy B](#) Jun 28, 2020 at 5:46



Try this basic example for better understanding of race condition:

0



```

public class ThreadRaceCondition {

    /**
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args)
        throws InterruptedException {

```

```
Account myAccount = new
Account(22222222);

// Expected deposit: 250
for (int i = 0; i < 50; i++) {
    Transaction t = new
Transaction(myAccount,
Transaction.TransactionType.DEPOSIT, 5.00);
    t.start();
}

// Expected withdrawal: 50
for (int i = 0; i < 50; i++) {
    Transaction t = new
Transaction(myAccount,
Transaction.TransactionType.WITHDRAW, 1.00);
    t.start();
}

// Temporary sleep to ensure all
threads are completed. Don't use in
// realworld :-)
Thread.sleep(1000);
// Expected account balance is 200
System.out.println("Final Account
```

[Share](#) [Improve this answer](#)

answered May 31, 2013 at 15:51

[Follow](#)



Morsu

1 ● 1



0



You don't always want to discard a race condition. If you have a flag which can be read and written by multiple threads, and this flag is set to 'done' by one thread so that other thread stop processing when flag is set to 'done', you don't want that "race condition" to be eliminated. In fact, this one can be referred to as a benign race condition.

However, using a tool for detection of race condition, it will be spotted as a harmful race condition.

More details on race condition here,
<http://msdn.microsoft.com/en-us/magazine/cc546569.aspx>.

Share Improve this answer

answered Sep 7, 2014 at 9:11

Follow



kiriloff

26.3k ● 39 ● 161 ● 233

What language is your answer based on? – [MikeMB](#) Apr 21, 2015 at 17:14

Frankly it seems to me that if you have race conditions *per se*, you are not architecting your code in a tightly-controlled manner. Which, while it may not be an issue in your theoretical case, is evidence of larger issues with the way you design & develop software. Expect to face painful race condition bugs sooner or later. – [Engineer](#) Jul 13, 2016 at 12:57



0



Consider an operation which has to display the count as soon as the count gets incremented. ie., as soon as **CounterThread** increments the value **DisplayThread** needs to display the recently updated value.

```
int i = 0;
```

Output

```
CounterThread -> i = 1
DisplayThread -> i = 1
CounterThread -> i = 2
CounterThread -> i = 3
CounterThread -> i = 4
DisplayThread -> i = 4
```

Here **CounterThread** gets the lock frequently and updates the value before **DisplayThread** displays it. Here exists a Race condition. Race Condition can be solved by using Synchronization

Share Improve this answer

edited Jul 15, 2015 at 8:06

Follow

answered Jul 15, 2015 at 8:00



[bharanitharan](#)

2,619 ● 5 ● 33 ● 30



A race condition is an undesirable situation that occurs when two or more process can access and change the

0



shared data at the same time. It occurred because there were conflicting accesses to a resource. Critical section problem may cause race condition. To solve critical condition among the process we have taken out only one process at a time which executes the critical section.

Share Improve this answer

edited Dec 5, 2018 at 7:02

Follow

answered Dec 5, 2018 at 6:55



rashedcs

3,739 ● 2 ● 42 ● 44



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.