Home
PUBLIC
Stack Overflow
Tags
Users
FIND A JOB
Jobs
Companies
TEAMS What's this?
Free 30 Day Trial

Can Multiple Threads Read The Same Class Member Variable?

Ask Question

Asked 12 months ago Active 12 months ago Viewed 204 times

2

Can multiple threads safely read the same class member variable without creating a race condition?

```
class foo {
    int x;
};

void Thread1(foo* bar) {
    float j = bar->x * 5;
}

void Thread2(foo* bar) {
    float k = bar->x / 5;
}
```

So if for example, we have two threads running `Thread1` and `Thread2`. If each thread is passed the *same* `foo` object, can they run independantly, without race conditions, because we are only reading the variable and not writing? Or is the act of accessing the object make this whole thing unsafe?

If the above *is* safe, can a third thread safely write to that same `foo` object as long as it doesn't touch `foo::x`?

```
#include <thread>

class foo {
public:
    int x = 1;
    int y = 1;
};

void Thread1(foo* bar) {
    int j;
    for (int i = 0; i < 1000; i++) {
        j = bar->x * 5;
    }
    printf("T1 - %i\n", j);
}

void Thread2(foo* bar) {
    int k;
    for (int i = 0; i < 1000; i++) {
        k = bar->x / 5;
    }
    printf("T2 - %i\n", k);
}

void Thread3(foo* bar) {
    for (int i = 0; i < 1000; i++) {
        bar->y += 3;
    }
    printf("T3 - %i\n", bar->y);
}

int main() {
    foo bar;

    std::thread t1(Thread1, &bar);
    std::thread t2(Thread2, &bar);
```

c++ thread-safety

share follow

edited Dec 5 '19 at 9:04

asked Dec 5 '19 at 8:51
KKlouzal
642 ● 5 ● 24

2 Yes, I strongly believe so. That, of course, includes that `foo::x` may not be written "indirectly" e.g. by overriding the `foo` completely somehow. – Scheff Dec 5 '19 at 8:56

2 You can read the same variable from as many threads as you like. Only writing requires synchronisation. – churill Dec 5 '19 at 8:59

1 The start of thread is a sufficient barrier. Read accesses to a certain address/range after start of thread (and before join) are safe and need not be guarded as long as nothing else writes to that specific address/range. – Scheff Dec 5 '19 at 8:59

Reading an aligned int is of atomic operation by default in most processors today, so it won't do harm, but still in C++ expressing it in a more explicit way using `std::atomic` with a corresponding memory fence parameter would be appreciated. – Dean Seo Dec 5 '19 at 9:19

1 @DeanSeo what has it to do with an aligned int and atomic operation, if the only thing I do is reading? Even if it would be a large array, there would be no race condition, if there is no writng at all? – RoQuOTriX Dec 5 '19 at 9:23

show 5 more comments

1 Answer

Active Oldest Votes

5

> Can multiple threads safely read the same class member variable without creating a race condition?

Yes and no.

Yes - The code you provided will not cause race condition, because race condition might occur when you have at least 2 threads working on the same shared resource, and at least one of those threads is writing to that resource.

No - your code isn't considered to be thread-safe, as it exposes x and y members for both read and write, and makes it possible (for you or other programmers that use your code) to cause a race condition. You rely on your knowledge (which you might forget over time) that x should only be read, and not written to, and that y should only be written by a single thread. You should enforce this by creating mutual exclusion in the critical code sections.

If you want the threads to only read from x and y, you should make this class immutable.

share follow

edited Dec 5 '19 at 9:35

answered Dec 5 '19 at 9:23
SubMachine
411 ● 3 ● 9

For the last sentence: Or use `std::atomic<int>` – Mike van Dyke Dec 5 '19 at 9:29

@MikevanDyke std::atomic is just another C++ tool to create mutual exclusion – SubMachine Dec 5 '19 at 9:32

1 @SubMachine Yes, atomics can be *used* to create mutual exclusion and critical code sections, but using an atomic variable does not in itself *imply* either of those. Mike is right - using an atomic is a valid alternative and probably preferred, as mutexes/critical sections tend to be orders of magnitude more expensive. – Max Langhof Dec 5 '19 at 9:37

I like your second part. This could be achieved e.g. by exposing the member `foo::x` as const reference to threads (or just as copied value). What I find more difficult to model in C++: data which is immutable just for the life-time of threads. – Scheff Dec 5 '19 at 9:38

A critical section is something which may cause additional performance impact. Hence, I understand that OP carefully asks how much safety is really necessary. (This is under the general assumption that C++ has been chosen because you can fine-grained manage what to use where to gain maximum performance. It's definitely a wrong choice to write bullet-proof multi-threading code as fast as possible.) ;-) – Scheff Dec 5 '19 at 9:46

add a comment

Your Answer

Sign up or log in

Sign up using Google

Sign up using Facebook

Sign up using Email and Password

Post as a guest

Name

Email
Required, but never shown

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged c++ thread-safety or ask your own question.

Linked
69 Multithreading program stuck in optimized mode but runs normally in -O0

Related
521 Templated check for the existence of a class member function?
471 error: request for member '..' in '..' which is of non-class type
461 Static constant string (class member)
2 Identify objects in boost::shared_ptr<boost::thread>
4 Thread-safe initialization of atomic variable in C++
0 Reading dynamic/changing data from multiple threads without locks? Can it cause a crash? Or just corrupted variables?
5 C++ multiple inheritance with base classes deriving from the same class
3 Call member method of a variadic class template with a member field
3 Accessing an atomic member of a class held by a shared_ptr

Hot Network Questions
Why did the 8087 need a special socket?
To become a better guitar player or musician, how do you balance your practice/training on lead playing and rhythm playing?
Can the Battle Master fighter's Precision Attack maneuver be used on a melee spell attack?
How to highlight "risky" action by its icon, and make it stand out from other icons?
Is it considered offensive to address one's seniors by name in the US?
I got my money returned for a product that I did not return
Mountain inside a "crater" after a meteor impact?
Why was the name of Discovery's most recent episode "Unification III"?
What happens if my Zurich public transportation ticket expires while I am traveling?
Joining Tikz paths seamlessly
Density of States of Supercells
What does the verb "to monograph" mean in documents context?
How does the title "Revenge of the Sith" suit the plot?
Have any other US presidents used that tiny table?
The strange pronunciations of "assume"
If someone had purchased some stocks prior to leaving California, then sold these stocks outside California, do they owe any tax to California?
Do PhD students sometimes abandon their original research idea? If so, how do they cope with it?
What does "中" as an independent verb means?
How many pawns make up for a missing queen in the endgame?
How to migrate data from MacBook Pro to new iPad Air
Motor will not go above half thrust. Why and how to fix?
Prison planet book where the protagonist is given a quota to commit one murder a week
Tower of Pisa function
How to backfill trench under slab in Los Angeles
Question feed