

Re: Are static array pointers thread safe?

## Re: Are static array pointers thread safe?

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2007-11/msg01693.html>

---

- *From:* Joseph M. Newcomer <[newcomer@xxxxxxxxxxxxx](mailto:newcomer@xxxxxxxxxxxxx)>
  - *Date:* Fri, 30 Nov 2007 12:30:39 -0500
- 

I'm not sure how this "common knowledge" came about, since it is obvious from the name 'static' that it is NOT allocated on the stack! That is 'auto'. On the other hand, there are another set of myths, that data types like CString, CArray, std::string, std::vector, etc. are allocated on the stack, and those are equally untrue; what is allocated on the stack is a reference to heap-allocated storage. There seems to be endless confusion about how allocation works; I've even had one correspondent assure me that he would not use malloc because that caused the space to be allocated on the disk, which would really slow things down!

joe

On Fri, 30 Nov 2007 08:34:07 -0800 (PST), flowstudioLA@xxxxxxxxxx wrote:

Yes, I think I would be a good example of someone who picked up this information incorrectly as common knowledge. There are quite a few sites a discussion boards that reference static data as always being allocated on the stack. I do, however, remember a few stating what Doug mentioned about it being stored in the BSS section of the exe, but seeing as how the overwhelming majority thought otherwise I assumed the same: dangerous behavior!

John

On Nov 29, 11:21 pm, Joseph M. Newcomer <[newco...@xxxxxxxxxxxxx](mailto:newco...@xxxxxxxxxxxxx)> wrote:

What amazed me recently was that a student in my course explained to me how static data was always allocated on the stack, because he'd been assured by "my professor" that this was so. I suggested that he might have misunderstood the explanation, but he insisted that this is what he'd been taught. Kind of scary if true.

On Thu, 29 Nov 2007 22:49:15 -0600, "Doug Harrison [MVP]" <[d...@xxxxxxxxxx](mailto:d...@xxxxxxxxxx)> wrote:

On Thu, 29 Nov 2007 19:51:14 -0800 (PST),

Re: Are static array pointers thread safe?

flowstudi...@xxxxxxxxx wrote:

No, it's not dynamically resized. I was under the impression that you are in fact initializing it when using an actual array variable (int foo::a[26][26][26];) and all values are defaulting to zero.

Yes, objects with static storage duration are zero-initialized during program startup. I was talking about non-default initialization. In old-timey terms, your (uninitialized) array's storage would be reserved in the BSS section of the executable, where it takes up no space, as opposed to the DATA section, where it would tend to be a full-sized image of the array (at least in older linkers).

Also, I was under the impression that static arrays are put on the stack, so that's exactly what I was worried about (stack overflow).

Objects with static storage duration do not live on the stack. They live in a separate area of memory dedicated to them. These objects include all globals, static data members of classes, and local statics.

The array is actually a bit larger, out to 5 dimensions, and so was a bit concerned, which is why I thought it might be best to create it dynamically where it would be put on the heap. I've probably confused a few things though, this is my first forray in

Re: Are static array pointers thread safe?

thinking about memory  
allocation. If my worries however were  
unfounded, you're right, I  
would probably just do it that way.

As long as you don't explicitly initialize it, and all the array  
bounds are  
constants, I think you'd be fine using a static array. I suppose  
there  
might be a small delay zero-initializing a huge array at  
program startup,  
but I find it hard to imagine it would be noticeable. For  
security reasons,  
NT-based Windows only provides zero-initialized pages of  
memory to  
processes, so I wonder, does the NT loader even have to zero  
out the BSS?  
It would seem to be redundant.

Ok, thank you. Does it make a difference if I  
just use a static array  
variable (int foo::a[26][26][26];) as you  
suggested above?

Nope. (Of course, all the createArray concerns become  
moot.)

Also,  
could you please expound on what you mean  
by reliably observe, like  
what the results of an unreliable observation  
would be?

Multiple CPUs in a weakly-ordered memory system (e.g.  
SMP Itanium) can  
observe all sorts of strange things in the absence of  
synchronization, such  
as reads/writes appearing out of order, writes being  
indefinitely delayed,  
etc. So by "reliable", I mean the participating threads should  
observe

Re: Are static array pointers thread safe?

memory consistently. Google for "memory visibility", "mutex", and "memory barrier", and you will turn up a lot of hits on this topic.

I should explain with that last question... I'm not so interested that the value be absolutely correct, the values are continuously being updated and read. So if reader thread is reading while writer thread is updating the value, and reader thread reads the old value instead of the new one, then that's ok if that is the result of an unreliable observation. My concern is more with that same situation causing an application failure.

In and of itself, it won't cause a crash, and since ints are read/written atomically, you won't get any word-tearing. You would have problems, though, if you are doing read-modify-write cycles, such as ++x. When two unsynchronized threads do that at the same time, a typical failure mode is for x to appear to have been incremented only once.

I suppose my real question would be, what happens in between the time an array value is updated with another value (a[0] = 1;a[0] = 2;), is that memory space emptied and then filled, leaving junk in the interim, is it seamless, or...?

Given that we're talking about ints, because int writes are atomic, another thread reading a[0] after one or the other of those writes will observe it to have either the value 1 or 2, or possibly some previous value if the

Re: Are static array pointers thread safe?

memory system is weakly ordered. That's assuming the writes actually happen as you wrote them, but they may not:

1. Because "a" isn't volatile, writing to it doesn't constitute "observable behavior". This means the compiler could omit writing 1 and postpone writing 2 until you do something that causes observable behavior, such as calling an opaque library function that could conceivably have access to a[0]. Similar concerns apply to writing to different elements of the array; the compiler can reorder these writes as it sees fit, as long as it doesn't change the observable behavior of the abstract machine upon which the language is based. (Significantly, that abstract machine is single-threaded.)

2. Even if you make "a" volatile, unless "volatile" has memory barrier semantics, it won't help much in a weakly-ordered memory system. So if the compiler doesn't get you with its reordering, the hardware will.

To sum up, if you're not using synchronization, you need to use volatile, and you should be well aware of the pitfalls inherent to not using synchronization. However, if you're using synchronization, you don't need volatile, because synchronization provides the desired memory consistency between the threads; indeed, the only thing using volatile on top of synchronization does is kill performance.

Joseph M. Newcomer [MVP]  
email: newco...@xxxxxxxxxxxxx  
Web:<http://www.flounder.com>  
MVP Tips:[http://www.flounder.com/mvp\\_tips.htm](http://www.flounder.com/mvp_tips.htm)

Re: Are static array pointers thread safe?

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxx

Web: <http://www.flounder.com>

MVP Tips: [http://www.flounder.com/mvp\\_tips.htm](http://www.flounder.com/mvp_tips.htm)

.