Aleksey Shipilev <aleksey.shipilev@gmail.com>

Please help clear up confusion re GetPrimitiveArrayCritical

9 messages

[redacted] Sun, Oct 8, 2017 at 12:35 PM

To: aleksey@shipilev.net

Hi,

First of all, thank you for your excellent articles, your participation in the community, and your work on the JVM. I'm contacting you to ask that you write an article or an SO Q/A resolving a very important issue that has caused me to battle with multiple library authors. Google sheds almost no light on the topic.

That issue is the use (or abuse) of GetPrimitiveArrayCritical, spurred by the awful documentation released by the JVM team.

1. Library authors refuse to believe that the sentence

> Inside a critical region, native code must not call other JNI functions, or any system call that may cause the current thread to block and wait for another Java thread.

forbids all JNI calls (aside from GetPrimitiveArrayCritical, GetStringCritical, and Release*Critical). It doesn't help that the English is a unclear, possibly referring to "(JNI functions and system calls) which may cause the current thread to block."

Some point out to the documentation of the `-Xcheck:jni` flag which states

> A JNI critical region is created when native code uses the JNI functions `GetPrimitiveArrayCritical` or `GetStringCritical` to obtain a reference to an array or string in the Java heap. The reference is held until the native code calls the corresponding release function. The code between the get and release is called a JNI critical section and during that time the HotSpot VM cannot bring the VM to a state that allows garbage collection to occur. The general recommendation is not to use other JNI functions within a JNI critical section, and in particular any JNI function that could potentially cause a deadlock.

It waffles about forbidding JNI calls, and refers to permitted JNI functions which won't cause deadlock. Of course, there is no such list in the JNI docs. This is completely ridiculous API documentation. The documentation should either forbid calling JNI, or explain how to do it safely and portably.

My understanding from reading the JVM sources, which I'd like you to confirm, is that deadlock occurs *only* because of waiting on an allocation. (Not, strictly, from blocking on a Java thread--which could be allocation-free.) And many JNI functions allocate local references in an unspecified way, causing deadlocking GCs. I suspect that using `EnsureLocalCapacity can prevent deadlock. However, nowhere is it specified which JNI functions only allocate local references, or how many they need.`

2. Library authors refuse to believe the sentence

> After calling `GetPrimitiveArrayCritical`, the native code should not run for an extended period of time before it calls `ReleasePrimitiveArrayCritical`.

They don't anticipate the thread stalling that occurs in multi-threaded code that is under allocation pressure.

Some examples of libraries which (ab)use Get*Critical are netlib-java and JCuda. Link to latest discussion: https://forum.byte-welt.net/t/code-crashes-when-run-in-multiple-threads-is-getprimitivearraycritical-to-blame/19535/10

Example of misinformation on SO: https://stackoverflow.com/questions/23258357/whats-the-trade-off-between-using-getprimitivearraycritical-and-getprimitivety I tried to add an answer, but it will forever be in the last place.

Please create an authoritative resource that I can point to when bringing up the issue. Library authors are so seduced by the performance of GetPrimitiveArrayCritical, that they don't want to listen to reason.

Thank you,
[redacted]

P.S. I see that you don't have any questions in SO. Did you know you can share knowledge by writing and answering your own questions?

Aleksey Shipilev <aleksey.shipilev@gmail.com> Sun, Oct 8, 2017 at 1:54 PM
[redacted]

On 10/08/2017 12:35 PM, wrote:
> It waffles about forbidding JNI calls, and refers to permitted JNI functions which won't cause
> deadlock. Of course, there is no such list in the JNI docs. This is completely ridiculous API
> documentation. The documentation should either forbid calling JNI, or explain how to do it safely
> and portably.

I think the wording there is deliberately vague to allow implementation freedom. Some JNI functions may cause deadlock, but it deliberately does not say what methods are. This is in line with the recommendation for not calling *any* JNI function withing the JNI critical region, even if the concrete implementation allows proceeding without a deadlock.

> My understanding from reading the JVM sources, which I'd like you to confirm, is that deadlock
> occurs *only* because of waiting on an allocation. (Not, strictly, from blocking on a Java
> thread--which could be allocation-free.) And many JNI functions allocate local references in an
> unspecified way, causing deadlocking GCs. I suspect that using |EnsureLocalCapacity can prevent
> deadlock. However, nowhere is it specified which JNI functions only allocate local references, or
> how many they need.

Um. In current *implementation*, that might be true. But, from portability standpoint, any call to JNI method may cause trouble.

So, you want to be portable? Do not use *Critical.

You want to be portable *and* performant? Use *Critical, but only within the tight codepaths that do not call into advanced machinery. I see no point in enumerating all the cases where *Critical may work with JNI functions calls, because at very best that is guesswork around the specific implementation.

> 2. Library authors refuse to believe the sentence
>
>> After calling |GetPrimitiveArrayCritical|, the native code should not run for an extended period
> of time before it calls |ReleasePrimitiveArrayCritical|.
>
> They don't anticipate the thread stalling that occurs in multi-threaded code that is under
> allocation pressure.

...

> Please create an authoritative resource that I can point to when bringing up the issue. Library
> authors are so seduced by the performance of GetPrimitiveArrayCritical, that they don't want to
> listen to reason.

No sure what you want from me here :) No amount of "authoritative source" would break through the wishful thinking of developers who are staring into performance improvements. I did this example before:
https://shipilev.net/jvm-anatomy-park/9-jni-critical-gclocker/

...and still think this is as far as I can or should get about it. Documentation says "Inside a critical region, native code must not call other JNI functions", and that is it.

-Aleksey

---

signature.asc
1K

[redacted]

Mon, Oct 9, 2017 at 12:03 AM

To: Aleksey Shipilev <aleksey.shipilev@gmail.com>

I'm trying to understand where our conversation is going wrong. Do you not believe that the documentation is confusing/bad and people are misunderstanding it? Or are you unsure what I'm asking of you?

I can point to specific conversations with developers who are confused about the documentation. Otherwise, I hope I don't need to explain that vague API specifications are bad API specifications. I think we both agree that the API specification should be interpreted as forbidding all JNI calls, except calls to GetPrimitiveArrayCritical, GetStringCritical, ReleasePrimitiveArrayCritical, and ReleaseStringCritical. And that taking too long inside the critical region will stall other threads (under all current GC implementations).

What I would like is for the JNI documentation to be made clearer. I can make specific suggestions, if you like. But since that might be a long and difficult process, a public post by someone like you confirming the two previous points would go a long way.

> No sure what you want from me here :) No amount of "authoritative source" would break through the wishful thinking of developers who are staring into performance improvements.

That's sort of a hateful sentence.

> I did this example before: https://shipilev.net/jvm-anatomy-park/9-jni-critical-gclocker/

Thank you for that. It's one of the few search results for "GetPrimitiveArrayCritical." Unfortunately, it's basically an advertisement for Shenandoah, and not a warning about the dangers of using GetPrimitiveArrayCritical.

[Quoted text hidden]

Aleksey Shipilev <aleksey.shipilev@gmail.com>

Mon, Oct 9, 2017 at 12:24 AM

[redacted]

On 10/09/2017 12:03 AM, wrote:
> I'm trying to understand where our conversation is going wrong. Do you not believe that the
> documentation is confusing/bad and people are misunderstanding it? Or are you unsure what I'm asking
> of you?

I am not sure what can we do about it. The specification text looks clear enough to me: "Inside a critical region, native code must not call other JNI functions" -- everything else is trying to bend the rules to reap the performance benefits without blowing your feet off, by exploiting particular implementation quirks.

> I can point to specific conversations with developers who are confused about the documentation.
> Otherwise, I hope I don't need to explain that vague API specifications are bad API specifications.
> I think we both agree that the API specification should be interpreted as forbidding all JNI calls,
> except calls to GetPrimitiveArrayCritical, GetStringCritical, ReleasePrimitiveArrayCritical, and
> ReleaseStringCritical. And that taking too long inside the critical region will stall other threads
> (under all current GC implementations).

That is what spec already says, no?

> What I would like is for the JNI documentation to be made clearer. I can make specific suggestions,
> if you like. But since that might be a long and difficult process, a public post by someone like you
> confirming the two previous points would go a long way.

Make the suggestions at hotspot-dev@, maybe? I assume that would be the list to talk about JNI spec.

>> No sure what you want from me here :) No amount of "authoritative source" would break through the
> wishful thinking of developers who are staring into performance improvements.
>
> That's sort of a hateful sentence.

I am telling you this from experience: no matter how harsh the spec wording is, developers would find excuses that their particular case is exempt from the rule, if doing so gives them substantial performance improvement. See e.g. Unsafe.

> <https://shipilev.net/jvm-anatomy-park/9-jni-critical-gclocker/>
> Thank you for that. It's one of the few search results for "GetPrimitiveArrayCritical."
> Unfortunately, it's basically an advertisement for Shenandoah, and not a warning about the dangers
> of using GetPrimitiveArrayCritical.

This, I think, underlines the fact that within the bounds of allowed specification, quality of implementation defines how bad it would be to break the specification. Some GCs may let more things slide, others may be more restrictive. If you want to be portable, adhere to the specification requirements, not implementation details.

I added the paragraph above to the post.

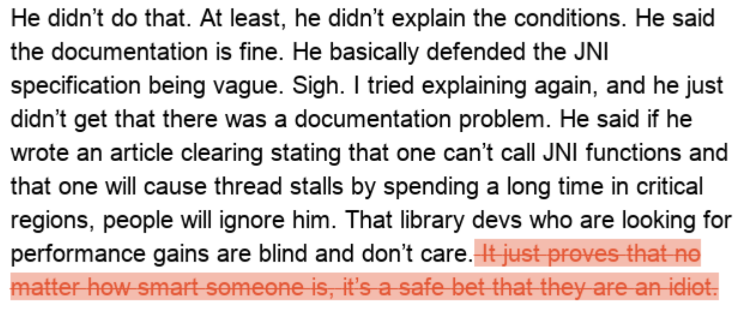
-Aleksey

signature.asc
1K

[redacted]
To: Aleksey Shipilev <aleksey.shipilev@gmail.com>

No, it's not what the spec says. Thanks for nothing.
[Quoted text hidden]

This bit is from another forum
https://forum.byte-welt.net/t/multiple-threads-getprimitivearraycritical/19535/15
(strikethrough is the edit from moderator)



He didn't do that. At least, he didn't explain the conditions. He said the documentation is fine. He basically defended the JNI specification being vague. Sigh. I tried explaining again, and he just didn't get that there was a documentation problem. He said if he wrote an article clearing stating that one can't call JNI functions and that one will cause thread stalls by spending a long time in critical regions, people will ignore him. That library devs who are looking for performance gains are blind and don't care. ~~It just proves that no matter how smart someone is, it's a safe bet that they are an idiot.~~

Aleksey Shipilev <aleksey.shipilev@gmail.com>
To: [redacted]

On 10/09/2017 12:51 AM, wrote:
> No, it's not what the spec says. Thanks for nothing.

Good luck calling people idiots, or just "thank for nothing" them, and then wondering why they "don't want to listen to reason". That might not be a coincidence after all ;)

-Aleksey

signature.asc
1K

[redacted] Mon, Oct 9, 2017 at 11:38 AM
To: Aleksey Shipilev <aleksey.shipilev@gmail.com>

It was a mistake coming to you. You're just an optimization guy. You don't have an appreciation for specifications or documentation. There is such a world of difference between the guys working on the JVM and the guys who write in Java. I need to contact one of the Java guys.

And, на правду не обижаются.

[Quoted text hidden]

Aleksey Shipilev <aleksey.shipilev@gmail.com> Mon, Oct 9, 2017 at 11:42 AM
To: [redacted]

Then you would not mind if I publish this entire conversation, and let others judge it for themselves?

-Aleksey

On 10/09/2017 11:38 AM, [redacted] wrote:
> It was a mistake coming to you. You're just an optimization guy. You don't have an appreciation for
> specifications or documentation. There is such a world of difference between the guys working on the
> JVM and the guys who write in Java. I need to contact one of the Java guys.
>
> And, на правду не обижаются.
>
> On Mon, Oct 9, 2017 at 11:50 AM Aleksey Shipilev <aleksey.shipilev@gmail.com

[Quoted text hidden]

signature.asc
1K

[redacted] Mon, Oct 9, 2017 at 11:49 AM
To: Aleksey Shipilev <aleksey.shipilev@gmail.com>

If you make no omissions, I do not.

[Quoted text hidden]