

Re: ReaderWriterLockSlim + Dispose?

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2008-04/msg01184.html>

- *From:* "Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 08 Apr 2008 14:23:29 -0700
-

On Tue, 08 Apr 2008 11:11:20 -0700, NvrBst <nvrbst@xxxxxxxx> wrote:

I got a relation to Mac and Nazi in one post, your looking evil now, wuahah.

It is not true that "Money = Performance".

In a metaphore or analogy,

I know. But it's not a correct one. That's my point. Had you bothered to read the rest of what I wrote, you'd understand what I meant.

[...]

People think in different ways than others. Forcing someone to write code in a certain way will only prologe the development process.

It is objectively true that the design I've suggested is simpler to implement and simpler to maintain. For an inexperienced programmer it may well be that it might take them a little more time to learn the concept, but in the long run even they will spend less time on it.

If a person wants to dispose straight away and let threads do their thing, then thats just the way he thinks.

Computers only think one way. It is the job of the programmer to learn to think the way the computer thinks, so that he is not fighting the computer as he writes the code.

Beyond that, while it is not possible to write correct code that disposes objects threads might use before those threads exit, it's certainly possible to write "fire and forget" code that allows the clean-up to happen while the rest of the application continues on doing whatever it was doing. This should be sufficiently close to the

Re: ReaderWriterLockSlim + Dispose?

paradigm of "dispose straight away and let threads do their thing" that even the person for whom "that's just the way he thinks" could easily comprehend and use such a design.

For small classes it might even be the best way IMO. I'd like to see your proof on this method taking longer/cost more to develop, but I think we both know there isn't any

I guess that depends on what you consider proof. However, your proposed design can be implemented in a couple of ways: a single try/catch at the very top of the thread's call chain, or a try/catch around each and every possible use of a disposable object.

In the former case, the initial implementation is quick, but then you create a situation in which you can't tell whether you're catching an expected exception or one that represents some other bug. That's a situation that is demonstrably a huge maintenance risk.

In the latter case, you can ensure that you're not catching exceptions that shouldn't be hidden. But at the very least you've now committed to a huge amount of work to wrap every single one of your uses of a disposable object in a try/catch handler. In addition, you now introduce the risk of missing one of those uses.

These are clear code maintenance problems, ones that will necessarily and unavoidable result in higher maintenance costs for the code.

On the other hand, the design I've proposed carries no hidden performance costs, is 100% predictable, and as a result of that can be easily shown to correctly deal with tearing down worker threads.

(expecially if the developer already has that solution in his mind... you want to force a different thought process into this guys head with no "good" reason as I can see). You agree performance wise it isn't worst, you seem to think it'll be harder to maintain which isn't true either

I don't "seem to think". I know.

– as long as its commented right it'll be just as easy as any other implementation.

No, it won't. Because of the issues I describe above, the exception–based design inherently carries higher maintenance costs.

You seem to think it'll take longer to develop, for some maybe, the reverse is true for others.

I never said it would necessarily take longer to develop. In fact, if you just wrap the whole thread in a single

Re: ReaderWriterLockSlim + Dispose?

try/catch, the initial development cost is almost certainly lower with your proposed design. But that's not the only cost that is of concern, and in fact it's the least important cost to consider. The total cost of maintaining the code over the lifetime of the code is what you need to consider.

Now if you were giving valid reason's, even I'd convert and say "oww definally want to do it that way, I see why now", but the "nope, the design is flawed because it's not the way I do it.

I have provided numerous reasons, all of them valid. It's unfortunate that you are so committed to your previous position that you cannot see the validity of what I've written. But it doesn't change the facts.

By the way, much of what I've said is based on my own personal experience over the years. However, there's a very good book by Steve McConnell, called "Code Complete", that does a good job of covering these kinds of issues. I don't recall off the top of my head whether it addresses this specific kind of design issue, but a person who reads the book and takes to heart the general coding philosophy that McConnell is promoting will naturally understand what I'm talking about here.

You way works just as good, but costs more to develop." is a very hard argument stance to succeed in.

To the contrary. I've made very specific statements about where and why the design I propose is simpler and lower-cost to maintain over the long run than the design you propose. You have ignored those statements wholesale, rejecting them for reasons known only to you. But that doesn't mean they are invalid.

Pete

.