

Re: Thread Pre-Emption Question . . .

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2007-11/msg00113.ht>

- *From:* Pops <dude@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 11 Nov 2007 09:59:08 -0500
-

Tell you what. When we get a report of a problem, and if we find it is related to critsect.h, then we will fix it. It hasn't happen yet in 12 years with hundred of thousands of customers, so we are not going to worry about it. We do not recognized there is a problem because if there is an issue, it generally means there is other outside issues involved, not the critsect.h code itself. Our system has been put throught extensive design, profiling, testing and most issues, completely unrelated to locks, we engineering out thru the years and since there are MOST more experience here than you can provide or provided, we will take out chances with experience.

One day you will learn what I am talking about.

See ya

--

HLS

Chris Thomasson wrote:

"m" <m@xxx> wrote in message news:udKFfc9IIHA.484@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

I am not sure why you are so defensive about your code – I haven't attacked it at all. I haven't said anything about it at all except that I hadn't reviewed it in detail and that at first glance it looked odd.

[...]

All of that being said, you code still looks odd and, based on my analysis, making assumptions about how the code is used, has both bugs and a high probability of correct execution. Specific bugs:

- 1) Not checking for failure of InitializeCriticalSection,

You can't really check if InitializeCriticalSection fails because it returns void... However, you can use InitializeCriticalSectionAndSpinCount to make sure that everything is correctly initialized at constructor time.

Re: Thread Pre-Emption Question . . .

CreateEvent or CreateSemaphore in the constructors. These are C APIs that do not raise exceptions on failure so their return codes must be checked and an exception thrown on failure to abort the constructor. BTW: not checking deallocateor failure is okay because the program will continue to run correctly although it is good practice to check this in debug builds because deallocators usually only fail because of invalid arguments.

All correct indeed. However, not only does he fail to check any return values in the constructor, he fails to check any of them within in the class body... For instance, he make numerous calls to SetEvent, ResetEvent, ReleaseSemaphore and WaitForSingleObject without checking for any errors. I hope that he corrects these issues because its a very bad habit to get into simply because the end result is incorrect code.

Specific performance issues:

- 1) ReaderEnter need not grab WriterMutex and could be much more efficient if it employed an optimistic locking strategy (assume that the lock state is consistent with desired access until it is discovered otherwise) using InterlockedIncrement & InterlockedDecrement and a loop
- 2) In WriterEnter, ResetEvent need not be called every time. It will be a NOOP if WriterRecursion is ≥ 1 .

Before you start at me and say again that this has run for years and years with no problem, let me remind you that the bugs I've identified have a very low probability of occurrence (should only occur during a low memory condition) and only exist in the constructors so the number of calls is few.

True.

And depending on what you do while holding this lock may not prevent the correct execution of error handling logic (if the access causes unplanned exceptions before data corruption and these unplanned exceptions are caught in the same place as planned exceptions for example).

Re: Thread Pre-Emption Question . . .

Also true. The most dangerous bug I found in his code is in the following snippets:

```
inline void TReaderWriter::ReaderEnter()
{
    TCriticalSectionGrabber grab(WriterMutex);
    WaitForSingleObject(ReadingOk, INFINITE);
    if (InterlockedIncrement(&Readers) == 0) {
        WaitForSingleObject(ReadingActiveSemaphore, INFINITE);
    }
}

inline void TReaderWriter::WriterEnter()
{
    WriterMutex.Enter();
    WriterRecursion++;
    ResetEvent(ReadingOk);
    if (WriterRecursion == 1) {
        WaitForSingleObject(ReadingActiveSemaphore, INFINITE);
    }
}
```

If one of those `WaitForSingleObject` calls just happens to fail for any reason whatsoever, the code will simply enter the write and/or read critical section when its clearly not suppose to. I say and/or because this could allow a reader and writer to access the data concurrently. This can, and will, corrupt real data in any number of random ways depending on the work done in the writer. This means that if you use this code to protect some of your data, well, its like playing Russian Roulette with a double barreled shotgun.

I have asked "Pops" about this issue several times and he refuses to answer. I sure do hope that he decides to all of the bugs in his code, for his customers sake. I don't know why he starts to shout and get angry, as we are only trying to help him by pointing out the obvious mistakes.

He should be thanking us.

[...]

As for the advise you gave to the OP, well, your correct... The OP simply cannot rely on processing to complete within a 40ms window. The OP must realize that he will experience random failures due to the processing taking longer than 40ms, period.

Re: Thread Pre-Emption Question . . .