

Re: Unexpected reentrancy on Wait,etc...

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.clr/2007-11/msg00006.html>

- *From:* jaket <jtummond@xxxxxxxxxx>
 - *Date:* Tue, 06 Nov 2007 22:14:09 -0000
-

I suppose the non-reentrant code could be moved into a new thread with a queue. This doesn't solve the general problem that the GUI can reenter anytime it locks and it is very difficult to guarantee the rest of the GUI code is reentrant. Re your second point. The code is currently guarded and throws an exception indicating the caller violated the transaction discipline. Failure is not an option because the receiver could be a customer calling thru the API.

Thanks for your suggestions,

Jake

On Nov 5, 3:31 pm, "Frank Hileman"
<frank...@xx> wrote:

Hi Jake,

Usually some code using locks can be reworked such that each thread works with private copies of data only, returning data using a centralized queue, with only the centralized queue being thread safe (using locks or other techniques).

If you don't mind your reentrant function failing, set a boolean field to true on entry of the function, and set it to false in a finally statement at the end of the function (the body being in a try). Then add a check to the top of the function to bail out if you are already executing the function. That is the standard non-reentrancy guard.

Regards,
Frank Hileman

check out VG.net:<http://www.vgdotnet.com>
Animated vector graphics system
Integrated Visual Studio graphics editor

"jaket" <jtumm...@xxxxxxxxxx> wrote in message

<news:1194046798.871695.228490@xx>

Re: Unexpected reentrancy on Wait,etc...

Hello,

I have run into a serious reentrancy problem. It appears from a stack trace that lock, WaitOne and Thread.Join do some servicing of the message queue. Here is the setup:

The GUI thread uses a combination of Application.Idle and a timer to simulate the old MFC OnIdle. The Idle handling will periodically pick up some work that requires executing some non-reentrant code. This code uses several locks to synchronize with other threads.

Another background worker thread is calling thru the Application's API. The API must use Control.Invoke using the MainForm to marshal the calls to the GUI thread. It also calls the non-reentrant code.

While GUI thread is waiting on a lock (I presume that it is already held by a 3rd thread), the Control.Invoke can occasionally slip thru and cause reentrancy. Anyone familiar with the perils of DoEvents will understand why this is a major problem.

Any suggestions?

Sincerely, Jake– Hide quoted text –

– Show quoted text –