

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

Re: STA cannot prevent multiple client calls accessing at the same time??

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.atl/2005-02/0399.html>

From: Alexander Nickolov (agnickolov_at_mvps.org)

Date: 02/25/05

Date: Fri, 25 Feb 2005 09:54:38 -0800

When you call a method on the out-of-proc singleton you are making a cross-apartment call from an STA. At this point COM dispatches a message via some communication channel to your EXE server and spins a message loop waiting for the response. This is one form of yielding I mentioned earlier (by far the most common cause of reentrancy in STA COM). Within this message loop another COM request may come and be serviced – these come as messages posted to a hidden window created by COM as part of CoInitialize. Note there's only ever a single thread your DLL object (and most likely your client) runs on.

In short, the behavior you see is perfectly normal. Again, it's called reentrancy. And again, synchronization primitives like a mutex are useless since they are designed to be reentrant on a single thread. Use boolean flags. I believe I discussed this already in one of my earlier replies...

A fact you seem not to be aware of is that a window is firmly bound to the thread that created it – all messages for that window are always delivered on that single thread. It's said that the window has thread affinity.

As for the suddenly changing call stack – that's a debugger quirk – you don't always see the entire call stack because not all functions create stack frames. When there's a function without a stack frame on the call stack the debugger cannot parse the stack further up the call chain.

--

=====
Alexander Nickolov
Microsoft MVP [VC], MCS D
email: agnickolov@mvps.org
MVP VC FAQ: <http://www.mvps.org/vcfaq>
=====

"Angela Yan" <yanyan9@hotmail.com> wrote in message
news:%23AI125wGFHA.2360@TK2MSFTNGP12.phx.gbl...
> Hi,

Re: STA cannot prevent multiple client calls accessing at the same time??

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

```
>
> I've been searching for the STA info web sites these days, but it seems
> that they don't really help in my scenerio. And maybe I am not an Appz
> developer, while most of the web sites try to explain STA from
> Appz's/client's point of view, which makes me a bit confused.
>
> Let me revise my problem. I have an ATL com in-proc dll project. Appz will
> call AfxOleInit() before creating the dll object and start using the dll
> methods. And Appz use PostMessage() to windows message queue after
> receiving a call back. The receiving part of the message queue will also
> use the same dll interface pointer(ie. the interface pointer of the dll is
> globally shared). But I am not sure whether the "receiving part of message
> queue" is residing in the owning thread of my dll or not.
>
> There is another singleton EXE com server in the scene. Both the dll and
> Appz will hold the interface pointers of the EXE server. There is a
> callback machanism implemented between Appz and EXE. (The dll doesn't
> listen to callback from EXE)
>
> dll code:
>
> A()
> {
>     //Init and process some local variables and some class private member
>     ...
>     ...                                     <-----
> Location 1
>
>     IEXEPointer->DoSomethingSimple1();      <-----
> Location 2
>     IEXEPointer->DoSomethingSimple2();
>
>     ...
>     ...
> }
>
>
> The scenerio is when Appz calls dll method A(), I breakpoint at the first
> line of A(), and then trigger the EXE server to call back to the Appz using
> another Appz which does not involve the dll. When Appz receives the call
> back, it will try to call method B() of the dll. The dll will get method
> B() call in ALWAYS (seems like always) after Location 2
> (IEXEPointer->DoSomethingSimple1() call). The call stack in .Net debug
> window when B() is called shows:
>
> B() in
> ...           //some Appz calls
> ...
> ...
> A() in
> ...           //some Appz calls
> ...
>
> Is this a correct behaviour?
> Why the reentrancy always occurs after the call to the EXE server?
> How can I solve the reentrancy issue? (mutex... or... ?)
>
> =====
>
> After triggering the call back from EXE to the Appz, before Location 2
> call, I use "sleep(10000)" at location 1. During this period, no B() is
> received. And after location 2, when B() call is just received, the call
```

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

```
> stack shows:
>
> B() in
> ....      //some Appz calls
> ....
>
> It does not have any information regarding A(). Until B() is completed,
> the call stack suddently becomes:
>
> A() in
> ...      //some Appz calls
> ...
>
> Any idea what is going on?
>
> Some more facts on the dll code:
> It doesn't have any windows message loop related stuff such as a message
> window or AtlWaitWithMessageLoop and CoWaitForMultipleHandles for example.
> The GetCurrentThreadID() call in both method A() and B() return the same
> value.
> The location 2 call in debug mode always makes a blink of the .Net frame.
> Possibely a context (thread/stack...?) switch?
>
>
> Thank you very much.
>
> Regards,
> Angela
>
>
>
>
> "Alexander Nickolov" <agnickolov@mvps.org> wrote in message
> news:ewQzUqRGFHA.3120@TK2MSFTNGP12.phx.gbl...
>> Yielding occurs when a message loop is spun within an STA
>> for any reason. Common causes are: calling a COM object
>> across apartment boundaries (including in another process),
>> displaying a dialog box or message box, calling SendMessage
>> to a window on another thread, and explicitly spinning a
>> message loop within your code (AtlWaitWithMessageLoop
>> and CoWaitForMultipleHandles for example). Note this all
>> is only of importance to objects running in an STA. It doesn't
>> matter what the threading model of the client is, except for
>> Both-threaded objects where this applies only for STA clients
>> (since then the object runs in an STA).
>>
>> Note that yielding does not break thread serialization for STA
>> objects. Your second request is delivered on the same thread,
>> so you don't need to do explicit synchronization (and it would
>> be useless anyway, since all kernel primitives are reentrant on
>> a single thread by design!). However, your code must be
>> desined to be reentrant, which is somewhat harder than making
>> it thread-safe...
>>
>> --
>> =====
>> Alexander Nickolov
>> Microsoft MVP [VC], MCSD
>> email: agnickolov@mvps.org
>> MVP VC FAQ: http://www.mvps.org/vcfaq
>> =====
```

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

```
>>
>> "Angela Yan" <yanyan9@hotmail.com> wrote in message
>> news:%23Cru5u$FFHA.1476@TK2MSFTNGP09.phx.gbl...
>>> Hi,
>>>
>>> I just read your first reply again. This is my literal understanding of
>>> the word "yield":
>>>
>>> Client calls my method A(), in A() there are calls to another COM object
>>> which is an out-of-process exe. Although the calls to the other COM
>>> object are not to display a dialog or message box etc, once the calls
>>> are made, serialization is not enforced any more and my dll can receive
>>> any interface method call again from client before A() finishes. (In
>>> this case method B() call is received.)
>>>
>>> Is this the correct understanding of the word "yield"?
>>>
>>> Does it matter if the client is single thread or multi thread in this
>>> case?
>>>
>>> So does it mean if the dll calls to any other out-of-process COM object,
>>> there is no more serialization? and what can I do about it?
>>>
>>> Thanks.
>>>
>>> Regards,
>>> Angela
>>>
>>>
>>> "Alexander Nickolov" <agnickolov@mvp.org> wrote in message
>>> news:%236azMD5EFHA.3200@TK2MSFTNGP10.phx.gbl...
>>>> Your call stack clearly shows that C yields. Check what statement
>>>> yields. Once that sinks in, you can use your choice of boolean
>>>> flags appropriate for your code. Another approach would be
>>>> to flag the deletion request and do nothing further if you are
>>>> called reentrantly (using another flag set at the beginning of A
>>>> and cleared at its end). You can check the deletion flag at
>>>> each iteration within C and abort it when set, then carry out
>>>> the deletion at the end of A. Just another idea. With some
>>>> experience with reentrancy you should be able to generate
>>>> these yourself...
>>>>
>>>> --
>>>> =====
>>>> Alexander Nickolov
>>>> Microsoft MVP [VC], MCSD
>>>> email: agnickolov@mvp.org
>>>> MVP VC FAQ: http://www.mvps.org/vcfaq
>>>> =====
>>>>
>>>> "Angela Yan" <yanyan9@hotmail.com> wrote in message
>>>> news:005cUnwEFHA.3276@TK2MSFTNGP10.phx.gbl...
>>>>> Hi,
>>>>>
>>>>> The possible solution, is it the 5th Feb reply? I've used
>>>>> "GetCurrentThreadID()" to print out the Thread ID while A() or B() is
>>>>> being called. And they turn out to be the same. For the time stamp, it
>>>>> has been confirmed that A() is called first, then followed by B() when
>>>>> A() has not finished its execution.
>>>>>
>>>>> For 4th Feb reply, the C method does not yield. It is just a simple
>>>>> loop to compare the link list elements. And I can't check for NULL
```

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

```
>>>> condition of the linkList, because after I 'delete' the LinkList in
>>>> B(), all the contents in the link list will be 0xcdcdcdcd or
>>>> 0xdddddddd. Currently I can even repro the whole thing such that B( )
>>>> comes in before C( ) gets call.
>>>>
>>>>
>>>> Call Stack:
>>>>
>>>> C()in
>>>> B()out
>>>> B()in
>>>> A()in
>>>>
>>>> So I assume it has nothing to do with the linklist. In this case the
>>>> program will not crash but still the calling sequence is not a desired
>>>> behaviour.
>>>>
>>>> Looking forward to your reply.
>>>>
>>>> Thanks.
>>>> Angela
>>>>
>>>>
>>>> "Alexander Nickolov" <agnickolov@mvps.org> wrote in message
>>>> news:%230w1wbsEFHA.3200@TK2MSFTNGP10.phx.gbl...
>>>>> If you refer to my very first reply, that's exactly what I told
>>>>> you - you have reentrancy. I even gave you a possible solution
>>>>> in one of the later replies. You haven't done anything about it
>>>>> yet I assume.
>>>>>
>>>>> --
>>>>> =====
>>>>> Alexander Nickolov
>>>>> Microsoft MVP [VC], MCS D
>>>>> email: agnickolov@mvps.org
>>>>> MVP VC FAQ: http://www.mvps.org/vcfaq
>>>>> =====
>>>>>
>>>>> "Angela Yan" <yanyan9@hotmail.com> wrote in message
>>>>> news:0%23PGfGnEFHA.3536@TK2MSFTNGP15.phx.gbl...
>>>>>> Hi,
>>>>>>
>>>>>> Sorry for replying so late. Just came back from holiday... :p
>>>>>>
>>>>>> I've tried using "GetCurrentThreadId" in A( ) and B( ). They return
>>>>>> the same value. But I am very sure that A( ) will not call B( ). In
>>>>>> fact, in the whole dll, there is no B( ) call is made. B( ) is only
>>>>>> called by the client. And I don't use windows message loop in the
>>>>>> dll, only the Client uses it. Neither any thread message call is in
>>>>>> the dll.
>>>>>>
>>>>>> About the call stack, I can see A( ) is being called, then some
>>>>>> Cleint calls, then B( ) is called.
>>>>>>
>>>>>> B( )
>>>>>> client call1
>>>>>> client call2
>>>>>> client call3
>>>>>> client call4
>>>>>> client call5
>>>>>> ..
>>>>>> ..
```

microsoft.public.vc.atl: Re: STA cannot prevent multiple client calls accessing at the same time??

```
>>>>>> ..
>>>>>> C( )
>>>>>> A( )
>>>>>> client call100
>>>>>> client call101
>>>>>> client call102
>>>>>> ..
>>>>>> ..
>>>>>> ..
>>>>>>
>>>>>> Please help.
>>>>>>
>>>>>> Thank you and
>>>>>>
>>>>>> Happy Chinese new year!!! ^_^
>>>>>>
>>>>>> Angela
>>>>>>
>>>>>>
>>>>>> "Rossen Tzonev" <rossenbg@hotmail.com> wrote in message
>>>>>> news:opslo421ol8xxqlh@rossenbg-homepc...
>>>>>>
>>>>>> I will add to Alexander suggestions. This is also possible:
>>>>>>
>>>>>>> 1. "GetCurrentThreadId" returns the same value when "A" and "B"
>>>>>>> are - variant 1. Then I can bet you are calling "B" somewhere
>>>>>>> during the call to "A". Because your LOG shows that you are in
>>>>>>> "A", but "B" is being called
>>>>>>>
>>>>>>> 2. "GetCurrentThreadId" returns the same value when "A" and "B"
>>>>>>> are - variant 2. If somewhere durring "A" execution you are calling
>>>>>>> something like "AtlWaitWithMessageLoop" or any code which gets
>>>>>>> current thread message in the message queue and dispatches it,
>>>>>>> then it is possible that "B" is called during "A" execution.
>>>>>>> That's because of COM rules for STA. STA has message loop and all
>>>>>>> COM calls to any methods in objects created in STA are done via
>>>>>>> COM private messages to the thread.
>>>>>>>
>>>>>>> 3. "A" and "B" are called from different threads. Then your problem
>>>>>>> is obvious. But this means that your code for creating the COM
>>>>>>> object is located not where it should be.
>>>>>>>
>>>>>>> Can you try to put breakpoint at the beginning and at the end of
>>>>>>> "A" and "B" and examine the call stack. If "B" is currently called
>>>>>>> and "A" is in the call stack then you have case (1). If "B" is
>>>>>>> called via the thread message loop code and after it there is a
>>>>>>> call to "A" in the stack then you have (2).
>>>>>>>
>>>>>>>
>>>>>>> On Fri, 4 Feb 2005 09:43:20 -0800, Alexander Nickolov
>>>>>>> <agnickolov@mvp.org> wrote:
>>>>>>>
>>>>>>>> Why don't you log the thread ID and time stamp in your traces
>>>>>>>> to be sure what's going on? If your client is misbehaving, there's
>>>>>>>> nothing for you to fix...
>>>>>>>>
>>>>>>>>
>>>>>>>>
>>>>>>>>
>>>>>>>> --
>>>>>>>> Rossen Tzonev
>>>>>>>> Sofia, Bulgaria
```

Re: STA cannot prevent multiple client calls accessing at the same time??

