

# Re: DirectShow base classes and the possible deadlocks

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.directx.video/2005-04/msg00>

---

- *From:* Thore Karlsen [MVP DX] <[sid@xxxxxxxx](mailto:sid@xxxxxxxx)>
  - *Date:* Thu, 28 Apr 2005 10:38:19 -0500
- 

On Thu, 28 Apr 2005 02:58:07 -0700, "Cyril"  
<[Cyril@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:Cyril@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)> wrote:

>> So, according to your analysis, what will happen if one thread tries to  
>> start the filter, and another thread tries to stop the filter, is:  
>>  
>> Thread A:  
>>  
>> - Calls Run()  
>> - Run() grabs FilterLock  
>> - Run() holds FilterLock until finished  
>>  
>> Thread B:  
>>  
>> - Calls Stop()  
>> - Stop() tries to grab FilterLock, but cannot get it until Run() is  
>> finished  
>>  
>> Consequently, Active() and Inactive() cannot be called at the same time.  
>>  
>> Or am I misunderstanding you?

> I was trying to create a basefilter class by myself, not use the current  
> one (to add some additional functionality to the reference implementation).  
> What I was reporting is mainly the need for the base filter to be like the  
> current implementation (and to recursive lock critical sections) because of a  
> bad design in the CSource implementation. If you move the line like said  
> above, the CBaseFilter no longer need to lock the FilterLock before  
> inactiv'ing the source pin. You are then locking less often, (and no more  
> when not needed).

I checked the source codes, and you seem to be missing one thing. The critical sections you call FilterLock and FilterStateLock are both the same critical section. Yes, there will be recursive locking, but that's harmless. There's no problem with the order of locking, since you're only dealing with one critical section.

## Re: DirectShow base classes and the possible deadlocks

>This could result in a deadlock in that case:  
>Let's say I have a filter graph with 2 filters (Source and Renderer)  
>The filter graph create the worker thread (call to CSource::Active) and it  
>works.  
>Then during execution, the application thread spawns a main thread to handle  
>graph event, streaming, whatever. This main thread locks a CS needed by the  
>source filter (like FilterStateLock). The worker thread is then blocked  
>(usual behavior).  
>  
>The application thread is then scheduled, and due to a message being  
>received in WndProc, (like WM\_DESTROY), ask the filter graph to stop (by  
>calling CoUninitialize, or even ME->Stop, filter->release, etc). The worker  
>thread cannot stop because it is waiting for its FilterStateLock. The main  
>thread is blocked because it is waiting for an event in FillBuffer to be set.  
>The application deadlocks.

That is true if the worker thread is indeed waiting for the filter state lock, but at least for CSource/CSourceStream that's not the case as far as I can see. Active()/Inactive() aren't called on the worker thread, and FillBuffer() doesn't lock anything, nor does it wait for any events. FillBuffer() doesn't do anything at all, it's up to derived classes to implement it properly.

>I think you should specify in the documentation not to use any CS of the  
>CBaseFilter in any code for the filter (and neither rely on them).

I agree with this as a general rule of thumb. Of course sometimes you have to use the critical sections for things to work properly, but I generally use my own critical sections for all my own variables. It's too easy to make a mistake using the critical sections provided by the base classes.

>Moreover,  
>the FillBuffer shouldn't be using any event to communicate with other thread  
>(nor OnThreadCreate, OnThreadDestroy). If the lock would have be moved like I  
>said above, then the main thread would only have blocked in Inactive method  
>(and not the Stop method), thus allowing the FillBuffer to be called, then  
>allowing the main thread to be unblocked, then releasing the blocked CS in  
>Inactive and then allowing the worker thread to exit.  
>  
>(The FillBuffer is not called because the FilterGraph is not sending any  
>request because it is blocked in CBaseFilter waiting in Stop)

FillBuffer() is called continuously by the worker thread while there are no requests pending. That doesn't matter, though, because FillBuffer() doesn't have anything to do with the blocking or unblocking of anything. (Again, unless you specifically make it so in your own override.)

So, given that there is only one critical section, and that the only threads using that critical section are your own threads, do you still think that there can be a problem?

## Re: DirectShow base classes and the possible deadlocks

>So to conclude (this is a bit long, sorry):

> If the locking is really needed then it must always follow the same  
>order. Then it should be specified in the documentation what is the locking  
>order, where we could use locks, and why not using them. I have found this  
>document :

><http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/stoppingthreads.asp>

>but it is not enough to avoid such cases.

>In FillBuffer it is impossible not to wait for external event (FillBuffer is  
>used for a push model, most of time waiting for a IO thread (network,  
>peripheral) to deliver sample, so it can be block externally). If the IO  
>thread is terminating, and then terminating the graph by notifying the main  
>thread, it is very difficult to avoid deadlock without clear explanations of  
>"don't do that".

Well, you have to provide your own synchronization for FillBuffer()  
that's independent of the critical sections, and you need to be able to  
break out of FillBuffer() when the filter stops. One way to do that is  
to use an event to signal that FillBuffer() should stop waiting for some  
other event, and setting that event in e.g. Stop().

It's not always easy to get this stuff right, but these are the kinds of  
things you need to know to make complex filters in DirectShow.

>Moreover, the sample code I have (PushSource from Directshow SDK, 10/17/04)  
>is even creating a deadlock here:

```
>  
>// This is where we insert the DIB bits into the video stream.  
>// FillBuffer is called once for every sample in the stream.  
>HRESULT CPushPinDesktop::FillBuffer(IMediaSample *pSample)  
>{  
> BYTE *pData;  
> long cbData;  
>  
> CheckPointer(pSample, E_POINTER);  
>  
> CAutoLock cAutoLockShared(m_pFilter->pStateLock()); <= Deadlock, as the  
>filter could never stop
```

Yes, this is very bad. I agree this is a deadlock waiting to happen.

>while it should have been :

```
>// This is where we insert the DIB bits into the video stream.  
>// FillBuffer is called once for every sample in the stream.  
>HRESULT CPushPinDesktop::FillBuffer(IMediaSample *pSample)  
>{  
> BYTE *pData;  
> long cbData;  
>  
> CheckPointer(pSample, E_POINTER);  
>
```

Re: DirectShow base classes and the possible deadlocks

> CAutoLock cAutoLockShared(some\_internal\_CS);

>It tooks me 2 weeks to figure out that the code was wrong because of a bad  
>locking order (or bad locked objects), because I couldn't think your code was  
>wrong.

Just so we're clear, I don't work for Microsoft, so it's not my code. :)  
And there's several problems with the DirectShow code. I've found  
confirmed deadlocks myself in the base classes.

—  
New to newsgroups? Read: <http://dev.6581.com/newsgroups.html>

---

- **Follow-Ups:**
  - ◆ ***Re: DirectShow base classes and the possible deadlocks***
    - ◇ From: Cyril
  
- **References:**
  - ◆ ***DirectShow base classes and the possible deadlocks***
    - ◇ From: Cyril
  - ◆ ***Re: DirectShow base classes and the possible deadlocks***
    - ◇ From: Thore Karlsen [MVP DX]
  - ◆ ***Re: DirectShow base classes and the possible deadlocks***
    - ◇ From: Cyril
  
- Prev by Date: ***Re: Resizing Frames Filter***
- Next by Date: ***Re: How to write a 2 input pin filter?***
- Previous by thread: ***Re: DirectShow base classes and the possible deadlocks***
- Next by thread: ***Re: DirectShow base classes and the possible deadlocks***
- Index(es):
  - ◆ ***Date***
  - ◆ ***Thread***