

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:* "Cyril" <Cyril@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 28 Apr 2005 02:59:16 -0700
-

> 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?
>
Hi,

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).

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

Re: DirectShow base classes and the possible deadlocks

(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.

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). Moreover, the FillBuffer shouldn't be using any event to communicate with other thread (nor OnThreadCreate, OnThreadDestroy). If the lock would have been 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)

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".

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
```

while it should have been :

Re: DirectShow base classes and the possible deadlocks

Re: DirectShow base classes and the possible deadlocks

```
// 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(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. (In that case, the Inactive method could have blocked waiting for FilterStateLock hold in FillBuffer working thread, then FillBuffer would have released the FilterStateLock, allowing Inactive to advance (and take FilterStateLock), but as soon as FillBuffer is called again, it is blocked waiting for the CS, and the worker thread couldn't continue, nor send the Stop command reply to the Inactive method => deadlock).

I hope the next version will correct those pesky bugs, and be more consistant on the locking order, and documentation.

Sincerely,
Cyril

• *References:*

- ◆ *DirectShow base classes and the possible deadlocks*
 ◇ *From:* Cyril
- ◆ *Re: DirectShow base classes and the possible deadlocks*
 ◇ *From:* Thore Karlsen [MVP DX]

- Prev by Date: *Re: DirectShow base classes and the possible deadlocks*
- Next by Date: *Enumerating capture devices excluding VFW*
- Previous by thread: *Re: DirectShow base classes and the possible deadlocks*
- Next by thread: *Record from composite input!*
- Index(es):
 - ◆ *Date*
 - ◆ *Thread*