Subject: asynchronous timers
Posted by markb77 on Thu, 14 Aug 2014 10:40:01 GMT
View Forum Message <> Reply to Message

Has anyone put to use the new "asynchronous timer" that was introduced in version 8.3?

I was just reading the help section for it and noticed that this timer can "interrupt pro code" when it fires. hmmm. Does this offer some possibilities for pseudo-"multithreaded" operation? For example, one process could be off doing a heavy calculation, and a separate gui process could periodically fire a timer and update it's display?

thoughts?

thanks
Mark

Subject: Re: asynchronous timers
Posted by dg86 on Thu, 14 Aug 2014 15:16:13 GMT
View Forum Message <> Reply to Message

On Thursday, August 14, 2014 6:40:01 AM UTC-4, superchromix wrote:
> Has anyone put to use the new "asynchronous timer" that was introduced in version 8.3?
>
>
>
> I was just reading the help section for it and noticed that this timer can "interrupt pro code" when it fires. hmmm. Does this offer some possibilities for pseudo-"multithreaded" operation? For example, one process could be off doing a heavy calculation, and a separate gui process could periodically fire a timer and update it's display?
>
>
>
> thoughts?
>
>
>
> thanks
>
> Mark

I've used the asynchronous timers to update a video interface for a microscope controller: you click on the video image to make things move under the microscope. IDL's timers do indeed interrupt IDL code, but do not interrupt calls to the compiled libraries underlying IDL code. For instance, a timer won't interrupt a long-running FFT, but instead will take action once the FFT is complete and control has returned to IDL. The same is true of spawn'ed commands and calls to external libraries.

The set time on a timer thus is the minimum time before the callback routine will execute. The actual time depends on external factors.

It appears that timer events can pile up, and there does not appear to be any way to look at the interrupt queue or to modify it. It is possible to clear the queue altogether, but that isn't always desirable. I may be missing something here, so other folks' insights would be helpful.

TTFN,

David

Subject: Re: asynchronous timers
Posted by Jim Pendleton on Fri, 15 Aug 2014 03:30:36 GMT
View Forum Message <> Reply to Message

On Thursday, August 14, 2014 9:16:13 AM UTC-6, David Grier wrote:
> On Thursday, August 14, 2014 6:40:01 AM UTC-4, superchromix wrote:
>
>> Has anyone put to use the new "asynchronous timer" that was introduced in version 8.3?
>
>>
>
>>
>
>>
>
>> I was just reading the help section for it and noticed that this timer can "interrupt pro code" when it fires. hmmm. Does this offer some possibilities for pseudo-"multithreaded" operation? For example, one process could be off doing a heavy calculation, and a separate gui process could periodically fire a timer and update it's display?
>
>>
>
>>
>
>>
>
>> thoughts?
>
>>
>
>>
>
>>
>
>> thanks
>

>>
>
>> Mark
>
>
>
> I've used the asynchronous timers to update a video interface for a microscope controller: you click on the video image to make things move under the microscope. IDL's timers do indeed interrupt IDL code, but do not interrupt calls to the compiled libraries underlying IDL code. For instance, a timer won't interrupt a long-running FFT, but instead will take action once the FFT is complete and control has returned to IDL. The same is true of spawn'ed commands and calls to external libraries.
>
>
>
> The set time on a timer thus is the minimum time before the callback routine will execute. The actual time depends on external factors.
>
>
>
> It appears that timer events can pile up, and there does not appear to be any way to look at the
>
> interrupt queue or to modify it. It is possible to clear the queue altogether, but that isn't always
>
> desirable. I may be missing something here, so other folks' insights would be helpful.
>
>
>
> TTFN,
>
>
>
> David

Doug E. might have some comments on this. IDL 8.4 will have some modified behavior with respect to timers, relative to 8.3. There will be more defined rules about when they will be allowed to interrupt the interpreter.

Jim P.

Subject: Re: asynchronous timers
Posted by Doug on Fri, 15 Aug 2014 22:51:23 GMT
View Forum Message <> Reply to Message

All,

For IDL 8.4, we have done a couple of things to timers:

* They no longer fire in the middle of system callbacks. For example,
they won't interrupt widget event handlers and object cleanup methods.
This is better since there are fewer nasty surprises.

* For when there still are nasty surprises, there will be the "block"
and "unblock" methods to enable the programmer to specify when they
don't want code to be interrupted. This is useful if currently
executing code is writing/reading to data that a timer will read/write.
  Place "block" and "unblock" around code that shouldn't be interrupted.

As for pseudo-multithreading, one really doesn't get any benefit since
there's still only one IDL interpreter. For that you'll need another
IDL, of course.

Cheers,
Doug

Subject: Re: asynchronous timers
Posted by markb77 on Sat, 16 Aug 2014 08:28:16 GMT
View Forum Message <> Reply to Message

On Saturday, August 16, 2014 12:51:23 AM UTC+2, Doug wrote:
> All,
>
>
>
> For IDL 8.4, we have done a couple of things to timers:
>
>
>
> * They no longer fire in the middle of system callbacks. For example,
>
> they won't interrupt widget event handlers and object cleanup methods.
>
> This is better since there are fewer nasty surprises.
>
>
>
> * For when there still are nasty surprises, there will be the "block"
>
> and "unblock" methods to enable the programmer to specify when they
>
> don't want code to be interrupted. This is useful if currently
>
> executing code is writing/reading to data that a timer will read/write.
>

> Place "block" and "unblock" around code that shouldn't be interrupted.
>
>
>
> As for pseudo-multithreading, one really doesn't get any benefit since
>
> there's still only one IDL interpreter. For that you'll need another
>
> IDL, of course.
>
>
>
> Cheers,
>
> Doug

Thanks for the info. When you say "they won't interrupt widget event handlers", what do you mean exactly? Suppose I have a GUI application, which is event driven. Essentially ALL routines are handling events in some sense. Does this mean the asynchronous timer wouldn't fire at all in my application? Or do you mean only that the timer wouldn't interrupt the "top level" event handler? (ie the procedure that is specified when starting Xmanager?)

best,
Mark

Subject: Re: asynchronous timers
Posted by Doug on Mon, 18 Aug 2014 16:27:51 GMT
View Forum Message <> Reply to Message

On 8/16/14, 2:28 AM, superchromix wrote:
> On Saturday, August 16, 2014 12:51:23 AM UTC+2, Doug wrote:
>> All,
>>
>>
>>
>> For IDL 8.4, we have done a couple of things to timers:
>>
>>
>>
>> * They no longer fire in the middle of system callbacks. For example,
>>
>> they won't interrupt widget event handlers and object cleanup methods.
>>
>> This is better since there are fewer nasty surprises.
>>
>>

>>
>> * For when there still are nasty surprises, there will be the "block"
>>
>> and "unblock" methods to enable the programmer to specify when they
>>
>> don't want code to be interrupted. This is useful if currently
>>
>> executing code is writing/reading to data that a timer will read/write.
>>
>> Place "block" and "unblock" around code that shouldn't be interrupted.
>>
>>
>>
>> As for pseudo-multithreading, one really doesn't get any benefit since
>>
>> there's still only one IDL interpreter. For that you'll need another
>>
>> IDL, of course.
>>
>>
>>
>> Cheers,
>>
>> Doug
>
>
> Thanks for the info. When you say "they won't interrupt widget event handlers",
what do you mean exactly? Suppose I have a GUI application, which is
event driven.
Essentially ALL routines are handling events in some sense. Does this
mean the
asynchronous timer wouldn't fire at all in my application? Or do you
mean only
that the timer wouldn't interrupt the "top level" event handler? (ie
the procedure
that is specified when starting Xmanager?)
>
> best,
> Mark
>

Hi Mark,

Asynchronous timers can have their callbacks invoked at two points:

(1) While IDL is idle (waiting for the next IDL statement or native
event (which often get translated into an IDL event))
(2) While IDL is executing PRO code (with the exceptions mentioned

previously)

The first one is what widget-based timers were limited to. The new timers still fire there as well, so if one has an event-driven widget program the new timer's callbacks will be invoked between widget events.

If a timer pops while IDL is busy processing a callback, invocation of the timer's callback is postponed until IDL is able to handle it.

Cheers,
Doug

Subject: Re: asynchronous timers
Posted by Doug on Tue, 19 Aug 2014 18:12:14 GMT
View Forum Message <> Reply to Message

On 8/19/14, 2:59 AM, superchromix wrote:
> On Monday, August 18, 2014 6:27:51 PM UTC+2, Doug wrote:
>> On 8/16/14, 2:28 AM, superchromix wrote:
>>
>>> On Saturday, August 16, 2014 12:51:23 AM UTC+2, Doug wrote:
>>
>>>> All,
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>> For IDL 8.4, we have done a couple of things to timers:
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>> * They no longer fire in the middle of system callbacks. For example,
>>
>>>>
>>
>>>> they won't interrupt widget event handlers and object cleanup methods.
>>
>>>>
>>
>>>> This is better since there are fewer nasty surprises.

>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>> * For when there still are nasty surprises, there will be the "block"
>>
>>>>
>>
>>>> and "unblock" methods to enable the programmer to specify when they
>>
>>>>
>>
>>>> don't want code to be interrupted. This is useful if currently
>>
>>>>
>>
>>>> executing code is writing/reading to data that a timer will read/write.
>>
>>>>
>>
>>>>   Place "block" and "unblock" around code that shouldn't be interrupted.
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>> As for pseudo-multithreading, one really doesn't get any benefit since
>>
>>>>
>>
>>>> there's still only one IDL interpreter. For that you'll need another
>>
>>>>
>>
>>>> IDL, of course.
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>>
>>
>>>> Cheers,

>>
>>>>
>>
>>>> Doug
>>
>>>
>>
>>>
>>
>>> Thanks for the info. When you say "they won't interrupt widget event handlers",
>>
>> what do you mean exactly? Suppose I have a GUI application, which is
>>
>> event driven.
>>
>> Essentially ALL routines are handling events in some sense. Does this
>>
>> mean the
>>
>> asynchronous timer wouldn't fire at all in my application? Or do you
>>
>> mean only
>>
>> that the timer wouldn't interrupt the "top level" event handler? (ie
>>
>> the procedure
>>
>> that is specified when starting Xmanager?)
>>
>>>
>>
>>> best,
>>
>>> Mark
>>
>>>
>>
>>
>>
>> Hi Mark,
>>
>>
>>
>> Asynchronous timers can have their callbacks invoked at two points:
>>
>>
>>
>> (1) While IDL is idle (waiting for the next IDL statement or native

>>
>> event (which often get translated into an IDL event))
>>
>> (2) While IDL is executing PRO code (with the exceptions mentioned
>>
>> previously)
>>
>>
>>
>> The first one is what widget-based timers were limited to. The new
>>
>> timers still fire there as well, so if one has an event-driven widget
>>
>> program the new timer's callbacks will be invoked between widget events.
>>
>>
>>
>> If a timer pops while IDL is busy processing a callback, invocation of
>>
>> the timer's callback is postponed until IDL is able to handle it.
>>
>>
>>
>> Cheers,
>>
>> Doug
>
>
> hi Doug,
>
> I can't quite understand what you mean by "until IDL is able to handle it". Let's say the user clicks on a button labeled "Call very long PRO code".
The event is handled by the event handler, which directly calls a function named "Long running IDL PRO code". This function, written in IDL, takes a long time to execute. Meanwhile, the timer is ready to fire. Does the timer interrupt the function, "Long running IDL PRO code", which was called by the event handler? Technically, while "Long running IDL PRO code" is running, the handling of the event is not finished. However, to be useful, the timer should interrupt the PRO code.
>
> thanks,
> Mark
>

Hi Mark,

I think you have it right. In this scenario, the async timer will not fire until the event handler finishes. As long as execution is in the

"scope" of the handler, timers will not be processed. In a widget program, when the event handler finishes, IDL returns to a state where it's looking for something to do... process native events, process a new command or process popped async timers.

Widget-based timers are still valuable, and in such a scenario they might be a better choice. Though they also won't fire during an event handler (or any time PRO code is being executed), one can "pump" them by calling widget_event. The new async timer doesn't have this feature (yet!?!), but note that timer.fire(<id>) can be called anywhere, even from a widget event handler. The assumption is that explicit, programmatic invocations of the timer's callback are okay.

So there are pluses and minus to both types of timers. One thing async timers do give us is the ability to run in headless environments. For example, if you're writing ENVI Services Engine tasks that will run remotely on a headless server and you need timers, you'll want the new async timers.

Cheers,
Doug

Subject: Re: asynchronous timers
Posted by markb77 on Wed, 20 Aug 2014 15:59:16 GMT
View Forum Message <> Reply to Message

hi Doug,

Thanks for the info. If the timer did have the ability to interrupt "Long running PRO code", then I could imagine using it to periodically update the GUI, for example, rather than it appearing to be frozen while the long PRO code is running. I suppose this could get quite complicated though, and there are probably other solutions to avoiding the "frozen GUI" problem.

Mark
