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
