

Re: Multimedia Timer

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2008-02/msg00524.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Tue, 12 Feb 2008 12:36:39 -0500
-

First, as already observed, WIndows is **not** a real-time system, and 20ms events are dangerously close to what it can handle.

See below...

On Tue, 12 Feb 2008 01:44:54 -0800 (PST), clinisbut <clinisbut@xxxxxxxxxx> wrote:

On Feb 11, 10:33 pm, malachy.mo...@xxxxxxxxxx wrote:

On Feb 11, 11:46 am, clinisbut <clinis...@xxxxxxxxxx> wrote:

I need to execute some code every 20ms. This code is just send some data through serial port, so I think it's not heavy.

At first I was using SetTimer function and OnTimer event, but with no good results: seems that not sending every 20ms, instead is sending 3 or so consecutive frames at same time.

<< snip >>

Does the recipient of the serial port data need to receive one "chunk" of data every 20 ms, without fail? In other words, is it a requirement that your program send exactly one single "chunk" of serial port data at each and every single interval of 20 ms?

If so, then you probably need to re-think your design and your use of the Windows OS, while repeating the mantra "Windows is not a RTOS, Windows is not a RTOS"

In fairness, you might be using WinCE, which in fact *_is_* a RTOS, but

Re: Multimedia Timer

somehow your post leads me to believe otherwise.

Malachy

Maybe I should post some code to find the "bottleneck".

```
1- Timer_SendData just calls a function Send_UART()
2- Send_UART does a pMySerial_UIThread->PostThreadMessage( WRITE,
buffer, 0 );
```

But PostThreadMessage to a thread that is not high priority can be delayed up to hundreds of milliseconds; even if you make this thread priority 15, you will be preempted by file system threads and other kernel threads, and will still have potential delays. Note on Windows Server, the timeslices on a server can be well over 100ms (to favor services over interaction), and this means that system threads can preempt you significantly.

Bottom line is that you have **NO** guarantees that an interval at the level of 20ms can **ever** be met. Hard-real-time at this level is not a suitable problem domain to write a Windows application for. There is a RTOS you can buy that is Windows-compatible, see www.ardence.com ; I've never used it, it is expensive (both for development, and for binary distribution license), but I was told, **many** years ago, that they were guaranteeing from interrupt-to-running-in-user-DLL times < 200us, and that number is probably much smaller today with the machines being so much faster. Some people who have used it have given me favorable reports on it, but all I know is what I've been told by their folks at trade shows and what I've read on their Web site.

3- WRITE handler message sends through serial port:

```
unsigned char* buffer = (unsigned char*)wParam;
BOOL ok = ::WriteFile( hComm, buffer, lParam, &bytesWritten,
&ovWriter );
if( !ok )
{
    DWORD err = ::GetLastError();
    if( err!=ERROR_IO_PENDING ) //Error grave
    {
        PostQuitMessage(0);
        return;
    }
}
```

```
HANDLE waiters[2];
waiters[0] = ShutdownEvent;
waiters[1] = WriteEvent;
DWORD reason = ::WaitForMultipleObjects( 2, waiters, FALSE,
INFINITE );
```

Note that once the I/O completes, it can be tens to hundreds of milliseconds before this

Re: Multimedia Timer

thread can run (all the WFSO/WFMO do is say to the scheduler, "OK, this thread is now runnable", but you are still at the mercy of the scheduler, and kernel threads are going to be able to preempt you arbitrarily)

```
switch( razon )
{
case WAIT_OBJECT_0:
err = ::GetLastError();
delete buffer;
PostQuitMessage(0);
return;

case WAIT_OBJECT_0 + 1:
ok = ::GetOverlappedResult( hComm, &ovWriter, &bytesWritten,
TRUE );
if( !ok )
{ PostQuitMessage(0);
return;
}
break;

default:
PostQuitMessage(0);
return;
}
delete buffer;
```

And answering some questions:

Are you sure that your operation is not taking longer than 20ms?

MMm not really sure...

Are you pausing the timer during your operation?

Nop, should I?

Does the recipient of the serial port data need to receive one "chunk" of data every 20 ms, without fail? In other words, is it a requirement that your program send exactly one single "chunk" of serial port data at each and every single interval of 20 ms?

Not exactly, look:

The device is recollecting data and storing it into a buffer.

To request this data, my app sends a frame to the device.

Re: Multimedia Timer

If my app takes too long to send this request, I can lose data because the device can overwrite data not requested if the buffer gets full. That's because I need those 20ms delay between two requests.

This is not quite what you said. What you are now saying is that the *minimum* time is 20ms, to give the device time to accumulate data. You have not said what the *maximum* time is. But a serial port device that cannot withstand a 500ms delay is not going to be really reliable under Windows. With a PCI interface, the constants drop down tremendously, but you've got issues there if the data rate is too high (e.g., 4MB/sec is a bit tense at kernel level).

I would write a specification that said any serial device that could lose data would have to buffer 500ms worth of data. But there are alternatives, but you need to change the approach.

The alternative is that you always have a pending ReadFile, perhaps several queued up asynchronously (my serial port "schema" allows this, but what I would do there is issue a lot of ReadFile requests and handle their completion in an I/O Completion Port). Key here is that the complete-request-and-switch-to-next-readfile operation in the kernel is typically done as

```
IoStartNextPacket(device, ...);
```

```
IoCompleteRequest(device, IRP);
```

which means the *actual* delay in the kernel is well under 10us to swap to the next buffer in sequence (I did this for one device I cannot talk about, where we got the inter-packet timing jitter < 100us, consistently, and typically ~80us). Since you have lots of pending ReadFiles, the issue there is how to get the response to the WriteFile up. In this case, I would *not* use the separate writer-thread approach, but do the WriteFile directly in the timer handler. Even if the reader threads are slow, there are enough queued ReadFile requests (hEvent of the OVERLAPPED structure is NULL, relying on the IOCP for notification, and never doing a WFSO/WFMO at all).

To get the jitter < 100us, by the way, we had to queue 50 ReadFile requests for the particular high-speed device I was interfacing to (40 didn't quite do it). I used a single thread listening via GetQueuedCompletionStatus so that requests were processed in FIFO order; if FIFO is not critical, or you can reassemble the packets in order (put the sequence number in the OVERLAPPED structure extension, for example...) then you can have multiple threads waiting for a response. In a multicore CPU, you can use SetThreadAffinity to limit this thread to N-1 processors so GUIs stay alive, and then you can boost the thread priority of this thread up to 15 (this may not be necessary).

You can see an example of this in my Async Explorer, where I push out a bunch of ReadFile requests and receive them out-of-order as the disk actually completes them in non-FIFO order.

joe

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxx

Web: <http://www.flounder.com>

MVP Tips: http://www.flounder.com/mvp_tips.htm

.