

## Re: Sleep() issue

---

*Source:*

<http://www.tech-archive.net/Archive/WinXP/microsoft.public.windowsxp.embedded/2007-04/msg00273.html>

---

- *From:* "KM" <[konstmor@xxxxxxxxxxxxxxxxxxxx](mailto:konstmor@xxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Sun, 22 Apr 2007 16:31:32 -0700
- 

Senen,

Did you mean 35 Mb or 350 Mb? The latter sounds like you've got really heavy MinLogon image. In fact, if network is not required, you can get a working Minlogon image with CMD Shell within 20–30 Mb.

Just in case, fire up a tool like DiskMon (sysinternals.com) and check if there is any disk level activities going on in your system.

Since you are running the thread in time critical priority I wouldn't expect the 10–15 jitter from the Sleep(1) call unless there are other time critical threads are running in the system at the same time. Although it would be hard to believe so on Minlogon image.

Anyway, you may want to explore the system clock interval with NtQueryTimerResolution undocumented API exported by NTDLL.dll. You can also change the interval with NtSetTimerResolution function. More info about the functions you will find on Mark's page here: <http://www.microsoft.com/technet/sysinternals/information/HighResolutionTimers.mspx>. Try the ClockRes to see what the system tick is set to on your device: <http://www.microsoft.com/technet/sysinternals/utilities/ClockRes.mspx>. It is typically around 15.625ms on x86 PC.

Or, even easier if you don't want to mess with this code in your app, use the TimerResolution app from <http://users.tpg.com.au/lucash>. Really nice app but prohibits the free commercial use so you, if this approach taken, will end up calling the NtQueryTimerResolution in your own code anyway.

You may also want to take a look at the Multimedia timers. Specifically, by querying to timeGetDevCaps API ([http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/win32\\_timegetdevcaps.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/win32_timegetdevcaps.asp)) you can determine the supported minimum timer resolution. Calling the timeBeginPeriod API ([http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/win32\\_timebeginperiod.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/win32_timebeginperiod.asp)) you can set the timer resolution to its minimum and thus increase the accuracy of the sleep interval.

Re: Sleep() issue

I also hope that you setting `THREAD_PRIORITY_TIME_CRITICAL` priority level within `REALTIME_PRIORITY_CLASS` priority class (if not, please take a look at `SetPriorityClass` API <http://msdn2.microsoft.com/en-us/library/ms686219.aspx>). Also, disable system thread priority boost for your thread – `SetThreadPriorityBoost` (<http://msdn2.microsoft.com/en-us/library/ms686280.aspx>) and `SetProcessPriorityBoost` functions – to make it more deterministic.

--  
=====  
Regards,  
KM

KM,

Thanks for the advises and for your time.  
In the code `get_scans_transferred()` returns immediately. I guess this function of the driver only reads from the DMA controller the amount of data transferred to the user buffer.  
In the suggested test code, I believe the time between `time3` and `time4` will be 15ms.  
My image is already Minilogon based and very light (about 350Mb).

I was thinking that the problem could be the low level of activity in the XPe system. Due to the DMA capability of the A/D card, the CPU load is almost 0%. Then if there is no interrupts the kernel will be awaked only by the system tick (usually 10 or 15ms).  
In my XP Pro system there is probably much more activity then the kernel can check and reschedule more often the threads in the ready queue.

I will try your suggestion to use the `QueryPerformanceCounter` and `SetTimer` functions. I have never heard about these functions before.

Regards  
Senen

"KM" <konstmor@xxxxxxxxxxxxxxxxxxxx> a écrit dans le message de news: e%23FDDFJhHHA.392@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Senen,

Well, I suppose you don't have the source code of the driver you are referring to, right? Otherwise it would definitely make sense to fix the actual source of the problem and implement proper synchronization between the driver and the reading app.

Anyway, from the algorithmic code you showed below I'd say the difference in timing is rather in how long it takes to process the data (`process_the_data` call) rather than in the `Sleep` call. I was actually

## Re: Sleep() issue

hoping you show us the code where you do the time measuring.

Try to run the following code snippet on your hardware and see if you get different results on XP and XPe:

```
while (1) {
time1=GetTickCount()
count=get_scans_transferred()
time2=GetTickCount()
while (count > buffer_size) { // buffer_size = room for 4ms of data
process_the_data(buffer_size)
count -= buffer_size
}
time3=GetTickCount()
sleep(1)
time4=GetTickCount()
print (all timeN)
}
```

If you could provide us with the timings you see there in the output of the above code, it would help us to find out what's really wrong there.

Depending on the timing there I'd also try to eliminate some pieces – from the Sleep call to the process\_the\_data call.

Also, GetTickCount is not actually an accurate call with the jitter of several ms (but good for starter). As I already mentioned it is better to use something like QueryPerformanceCounter or more reliable multimedia timers.

Another thing I'd do in your case is simplifying the image and minimizing its footprint. Less software components you run there – less troubles and better performance you get.

Sounds like your application and the driver are pretty lightweight when it comes to their dependencies. You may be able to test those even on a very minimum XPe Minlogon based image.

Stop/disable/remove all the services you don't need. Disable any system background activities from disk caching to background fragmentation to performance counters.

Remove most unnecessary drivers and interrupts. make sure your app code and data are not getting paged (either disable pagefile on the system or lock the user process working set).

Also, if possible, please move your data process code from user mode (it is currently running in the user mode, right?) to ring 0. In other words, create another driver that will process the data. That should minimize the path to the data from one driver to another. Although in general, application compilers have better optimizing options than DDK.

Also hope you test the same code under XPe and XP Pro on the same hardware. In general XPe is going to show you the same

## Re: Sleep() issue

performance as the XP Pro in driver areas but since you can dramatically reduce the system workload in XPe it may show better results than XP Pro for an app. This is however too generic statement since you can always tweak the performance and components for both.

The 4 to 15ms difference in timing doesn't really sound really scary for a non real time OS such as XP. Depending on your system workload it could be up to 50–100ms. However, running your reading thread with time critical priority should help a bit as well as all "optimizations" I mentioned above.

And, after all, if you still suspect the issue is due to the system work load that affects some system call latencies, please investigate it with Performance Tools.

--

=====

Regards,  
KM

The code is for a Daq board (A/D converter). Unfortunately the driver has no synchronization mechanism therefore I have to continuously scan the driver to check what amount of data was transfered to my buffer with DMA. According to the scan rate and refresh period of my system the needed amount of data is ready every 4ms. The code is very simple and runs in a time critical priority thread:

```
while (1) {
count=get_scans_transferred()
while (count > buffer_size) { // buffer_size = room for 4ms
of data
process_the_data(buffer_size)
count -= buffer_size
}
sleep(1)
}
```

This is a skeleton, the real code works dealing with the remaining data and all the indexes management.

The accuracy of the duration of the sleep is not critical, it's here mainly to avoid 100% of CPU load as with the driver synchronization functions. Under XP Pro it works fine, but in XPE the duration of the sleep is 15ms then we have the followin bahavior: After the scans corresponding to 4ms acquisition tha data is processed. The next call to function get\_scans\_transferred() is done only 15ms after, then the buffer has collected data for 4 cycles

## Re: Sleep() issue

(4x4ms = 16ms).

All the buffers are processed in the loop while (count>buffer\_size).

After processing the data a new value is sent to D/A converters and the output looks like this:

t=0:new value – t+15ms later: updated value followed by 3 or 4 updated values – t+30ms wait: updated\_value ...

The operation in XP Pro is like this:

t=0new value – t+4ms updated value – t+8ms updated value – t+12ms...

Again, the Sleep(1) can wait for 1, 2, 3 or 4ms it doesn't matter, but not more and not 15ms.

I don't need a real time OS because the 4ms period for the update is not critical, it can be 4ms most of the time and sometimes

5, 6, 7, 8 and even more.

No data is lost because of internal FIFO management and threads communication.

Thanks  
Senen

"KM" <konstmor@xxxxxxxxxxxxxxxxxxxx> a écrit dans le message de news:

OGP1HA7gHHA.4140@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Senen,

I must admit I couldn't get what exactly you are trying to do and what code you are testing.

Can you show up that piece of code you blame doesn't work the same way as in XP Pro?

Anyway, you probably realize that XP (XP Pro, XP) is not a real-time OS. That means any "timeout" there is not promised by OS and OS itself may not be deterministic. If you really wanted to get precise numbers there you'd probably go for a 3rd party solution that makes the XP(e) real-time (TenAsys's INtime, ex-Venturcom's RTX).

What you want to do to get more statistic data on the issues is as follows:

– bump up your "waiting" thread priority to highest or time-critical. Especially make sure to run the piece of code in high

## Re: Sleep() issue

priority where you measure the timing (the place where you calculate it took 15ms instead of 2ms).

– use high-resolution timers by utilizing QueryPerformanceCounter API.

The above suggestion may not make sense for your application at all since I had to guess about some things.

It would be more helpful if you could show us the code you are having troubles with.

--

=====

Regards,  
KM

Hi,

In my program I need to wait for a device for about 2ms. The accuracy of the wait is not critical.

It works under Windos XP with a simple Sleep(2).

Under XPE the same program will wait for 15ms instead of 2ms.

Any value for Sleep() lower than 15ms will produce the same wait of 15ms.

The use of SetWaitableTimer produces the same effect.

How can I get a delai of few milliseconds in XPE?

Thanks  
Senen

Re: Sleep() issue