

## Re: fixed time slices?

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2007-06/msg00239.htm>

---

- *From:* "Jan Bruns" <[testzugang\\_janbruns@xxxxxxxx](mailto:testzugang_janbruns@xxxxxxxx)>
  - *Date:* Mon, 25 Jun 2007 18:59:25 +0200
- 

"m" <m@xxx>:

"Jan Bruns":

What I meant with reliability was, if I can count on that the minimum time a sleep(n) might take is really  $(n+m+k)$  ms, where m is the least valid value passed to timeBeginPeriod(m) and k is some overhead value less than m.

No; assuming no clock skew, the minimum duration of Sleep(n) is  $(n + (n \% m) + k)$

By  $(n \% m)$  yu mean the remainder of  $n/m$ ?

That's clear. But it also says:

| If a higher-priority thread becomes available to run, the system  
| ceases to execute the lower-priority thread (without allowing it  
| to finish using its time slice), and assigns a full time slice  
| to the higher-priority thread.

The key here is that a thread becomes ready to run. A thread can only become ready to run, and be scheduled on a CPU, when the scheduler is running and decides that it is ready to run. The scheduler only runs in response to a timer interrupt or a call to a wait function (maybe other kernel entry points also), so the precision of pre-emption timing is always bounded by the system timer frequency (by the case, on a UP machine, where

Re: fixed time slices?

the running thread executes all user mode code for its entire time slice).

But the scheduler would also have the chance to let the hardware generate a timer-IRQ within whatever delay is desired. So it could put the CPU on a ready low-priority thread for as long as a higher priority thread is known to be unrunnable.

In old MS-DOS days, the precision of the programmable interval timer (an extra chip for this purpose?) AFAIR was better than 1/10MHz. At least it was possible to sync it to a screen's scanline ( $60\text{Hz} * 480\text{scanlines} = 28800\text{Hz}$  -> timer resolution better than  $35e-6$  s.

Maybe I should have said a noticeable performance cost to increasing the timer frequency.

Not on my computer. I've tried this using a CPU-intensive working process, and another high priority process, that effectively does nothing but sleep(2) (measured).

The working process slowed down about 2%. 500 sleep instructions, each resulting in about 4 CPU-task switches -> 2000 Task-switches per second might really eat 2% of CPU, when the intermediate scheduler-task also takes some overhead. That's fully in the range I expected.

```
PROCEDURE bute_force_sleep(ms : dword);
```

This depends on how frequently you would need to use bute\_force\_sleep.

It's near to always\*somewhat\_lesser.

Given that you are planning a standalone application, I doubt you will have a problem either way. As long as you know that you don't play nice in the sandbox, you can do whatever you like ;)

;(o;

Gruss

Jan Bruns

.

Re: fixed time slices?