

Re: nanosleep returning early

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/current/2004-07/1742.html>

From: Bruce Evans (*bde_at_zeta.org.au*)

Date: 07/22/04

Date: Fri, 23 Jul 2004 00:28:52 +1000 (EST)

To: John Birrell <jb@cimlogic.com.au>

On Thu, 22 Jul 2004, John Birrell wrote:

> *On Wed, Jul 21, 2004 at 11:01:20PM +1000, Bruce Evans wrote:*
> > *On Wed, 21 Jul 2004, Brian Fundakowski Feldman wrote:*
> > > *On Wed, Jul 21, 2004 at 06:13:10PM +1000, John Birrell wrote:*
> > > >
> > > > *Today I increased HZ in a current kernel to 1000 when adding dummynet.*
> > > > *Now I find that nanosleep regularly comes back a little early.*
> > > > *Can anyone explain why?*
> >
> > *The most obvious bug is that nanosleep() uses the low-accuracy interface*
> > *getnanouptime(). I can't see why the the problem is more obvious with*
> > *large HZ or why it affects short sleeps. From kern_time.c 1.170:*
> > ...
> > *% getnanouptime(&ts);*
> >
> > *This may lag the actual (up)time by 1/HZ seconds.*

The inaccuracy of the "get" time functions is actually `tc_tick/HZ` seconds, where `tc_tick` is 1 for small values of HZ but is scaled to give an inaccuracy of 1 mS for large values of HZ or can be adjusted using `sysctl` to give almost any inaccuracy that you want. But this makes no difference until HZ exceeds 1000 or you use the `sysctl`.

> *For any NTP clock adjustment to have an effect, it must happen between*
> *the time when my (crude) test gets the time and when the kernel calls*
> *getnanouptime to fill ts. All the rest of the code in nanosleep1()*
> *refers back to the value of ts. Any errors involved in the tv and hz*
> *conversions would just cause the code to go around the loop and tsleep*
> *more than once.*

`ts` is the uptime, so there is another window (larger) for NTP adjustments to have an affect: most of the sleep time. If `ts` were perfectly accurate or larger then the correct value, then the sleep interval (measured as the difference between uptimes) is sure to be large enough. However, the sleep interval measured as a difference between real times reported by `gettimeofday()` may be smaller because NTP adjusts the clocks differently,

which it only does for stepping the time.

- > *So, does increasing HZ expose the lower accuracy of getnanouptime() and is that what I'm seeing?*

I still don't know the reason. Unfortunately, I deleted your original mail so I can't run the test program in it easily.

- > *If so, shouldn't the time interface provided by the kernel to userland at least be consistently inaccurate? Or put another way, why should nanosleep be any less accurate than gettimeofday?*

I think it can't be consistent if the time is stepped (i.e., set using `settimeofday()` or `clock_settime(CLOCK_REALTIME)`). The real time is the best known approximation to the, er, real time, but most other times are for intervals. Having the most accurate times possible for intervals is inconsistent with having the most accurate times possible for real times if the real time is stepped.

- > *Bruce, have you seen this document: <<http://www.dragonflybsd.org/docs/nanosleep/>>?*
- > *I'm not looking for a critique here. The document talks about the sleep overruns in various operating systems. There is a patch that was applied to DragonFly which applies to the FreeBSD code base too.*

I just looked. It doesn't apply to FreeBSD. Adding 1 to the tick count in `tvtohz()` is necessary in many places in FreeBSD (e.g., for `itimers`). It happens to be just an optimization in `nanosleep1()` for the reasons you mentioned above (going around again fixes any underestimate of the tick count, and overestimating by 1 avoids going around again in most cases).

It's normal for simple test programs to take 2-epsilon ticks extra for `nanosleep()`, because they get in sync with clock interrupts. Returns from `nanosleep()` normally occur epsilon seconds after a clock interrupt. Then the kernel must wait until the next clock interrupt before returning from the next `nanosleep()` even if the sleep interval is 1 nsec (unless the kernel optimizes this operation and pessimizes others by busy-waiting or checking on other interrupts...). This gives a minimal sleep of 1-epsilon ticks. The FreeBSD implementation adds an extra tick to optimize the calculation of the minimal number of ticks. Without this, it would have used code like that in `nanosleep1()` after every sleep and `itimer` interrupt to check that the interval really has expired (underestimate the tick count a little to ensure not wasting an extra tick, at a cost of going around once more in the usual case).

I believe `dragonflybsd` has high resolution timeouts which make the issues different. 'tick' could be something like 10^{*6} and tick counts in microseconds would be so large that off by 1 errors in them would be too small to measure.

freebsd-current: Re: nanosleep returning early

Bruce

freebsd-current@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-current>

To unsubscribe, send any mail to "freebsd-current-unsubscribe@freebsd.org"