

## Re: [PATCH 1/6] new timeofday core subsystem for -mm (v.B3)

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-06/4746.html>

---

**From:** Roman Zippel ([zippel\\_at\\_linux-m68k.org](mailto:zippel_at_linux-m68k.org))

**Date:** 06/21/05

Date: Tue, 21 Jun 2005 00:05:34 +0200 (CEST)

To: john stultz <[johnstul@us.ibm.com](mailto:johnstul@us.ibm.com)>

Hi,

On Mon, 20 Jun 2005, john stultz wrote:

> > *I see lots of new u64 variables. I'm especially interested how this code*  
> > *scales down to small and slow machines, where such a precision is absolute*  
> > *overkill. How do these patches change current and possibly common time*  
> > *operations?*

>

>

> *Hey Roman,*

> *That's a good issue to bring up. With regards to the timeofday*  
> *infrastructure, there are two performance concerns (though let me know*  
> *if I'm forgetting something):*

You don't really answer the core question, why do you change everything to nanoseconds and 64bit values?

> *On smaller systems, timer interrupt processing is a concern, with the*  
> *shift to HZ=1000, we got a number of complaints from folks w/ old 486s*  
> *where time would drift due lost ticks. This would happen when something*  
> *(usually IDE in PIO mode) would disable interrupts and they would miss a*  
> *ton of timer interrupts. Also the impact of running the timekeeping code*  
> *10x more frequently was seen in a number of cases.*

>

> *With the new infrastructure, timekeeping is all done via a soft-timer*  
> *outside of interrupt context. In fact, the timekeeping soft-timer is*  
> *setup to run every 50ms instead of every ms. This should help overall*  
> *performance on slower systems using high HZ values.*

With -mm you can now choose the HZ value, so that's not really the problem anymore. A lot of archs even never changed to a higher HZ value. So now I still like to know how does the complexity change compared to the old code?

Linux-Kernel: Re: [PATCH 1/6] new timeofday core subsystem for -mm (v.B3)

- > *As for gettimeofday() syscall performance, I one had some numbers, but I*
- > *would need to re-create them. I'll see if I can grab a slower box and*
- > *give you some hard numbers. The gettimeofday() path is fairly*
- > *streamlined and should be pretty straight forward in the patch (see*
- > *kernel/timeofday.c), so let me know if you have specific concerns.*
- >
- > *There will probably be a bit of a drop, but I have some ideas for*
- > *cacheing a precomputed timeval in the timekeeping soft-timer if its a*
- > *serious issue.*

Well, AFAICT on slower machines (older and embedded stuff) it's a serious issue. The current code calculates the timeval with some simple 32bit calculations. Your code introduces the nsec step, which means several 64bit calculations and suddenly the overhead explodes on some machines.

As m68k maintainer I see no reason to ever switch to your new code, which might leave you with the dilemma having to maintain two versions of the timer code. What reason could I have to switch to the new timer code?

I had no problems with a little more overhead, like a *\_few\_* more 64bit operations per second (and preferably add/shifts), but I'm not really enthusiastic about the new code. Why don't you keep the main part 32 bits (or long)? What's wrong with using timeval or timespec?

I like the concept of the a time source in your patch. m68k already uses a number of timer related callbacks into machine specific code. If I could replace that with a timer driver, I'd be really happy. OTOH if it requires several expensive conversion between different time formats, I rather keep the current code.

bye, Roman

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>