

Re: [PATCH 0/5] clocksource patches

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-05/msg01393.html>

- *From:* john stultz <johnstul@xxxxxxxxxxx>
 - *Date:* Fri, 05 May 2006 19:04:12 -0700
-

On Thu, 2006-04-27 at 22:33 +0200, Roman Zippel wrote:

I really don't understand your problem with a clocksource specific `nsec_offset()`, the author already has to provide most of the basic parameters, it's not that difficult to put them together and for the really lazy we can provide a:

```
my_clock_nsec_offset(struct clocksource *cs)
{
return generic_nsec_offset(cs, my_get_cycles(), MASK, SHIFT);
}
```

this is `_very_` simple and if some of the parameters are constant you already saved a few cycles.

We already have this, its the `per-arch_gettimeofday()`.

My issue here is that by pushing this functionality off into the clocksource driver, not only do you complicate the driver, you also are implicitly binding the driver to a form of tick based timekeeping (question: from what point is the `nsec_offset` measuring from?). Also, this has implications w/ NTP, as the `nsec_offset()` function now must do its own scaling internally.

I think this sort of `get_nsec_offset()` interface is needed, but it doesn't belong in the clocksource driver, because that fouls the abstraction.

Instead, why don't we have two implementations of `__get_nsec_offset()`? One which uses the clean clocksource abstraction, and one that can call your `arch_get_nsec_offset()`?

Here's what I'm proposing for generic code:

```
do_get(ns)timeofday():
xtime + __get_nsec_offset()
```

```
update_wall_time():
```

Re: [PATCH 0/5] clocksource patches

```
now = clocksource_read() /* ← that's jiffies in tick-clock case */
while (now - last > interval_cycles):
last += interval_cycles
xtime += interval_nsecs
error += interval_ntp - interval_nsecs
ntp_adjust(...)
...

#ifdef CONTINUOUS_CLOCK
__get_nsec_offset():
now = clock_read
return ((now - last)*mult)>>shift
#else /*TICK_CLOCK*/
__get_nsec_offset():
return arch_get_nsec_offset()
#endif
```

How's that sound?

John, we have to find some compromise here, but I think sacrificing everything to driver simplicity is IMO a huge mistake.

We're coming up on the one year mark for this discussion. Every pass I've taken your feedback, worked to understand it (which, i admit can take me awhile :), and implemented parts of your ideas into the code. I really do appreciate this cycle, so I hope you don't find me thick-headed or stubborn. I believe I've been happy to compromise every step of the way.

I do want to find a solution that we both like, but I am feeling some fatigue and we need to make some progress. The existing tick based assumptions in mainline is blocking the ability to sanely implement both bug fixes and new features.

A "simple" driver gives you a lot of flexibility on the generic side, but you remove this flexibility from the driver side, i.e. if a driver doesn't fit into this cycle model (e.g. tick based drivers) it has to do extra work to provide this abstraction.

I'm not trying to fit tick based clocks into the continuous model. I'm trying to allow the currently jiffies based timekeeping code to possibly use something else, cleaning up a lot of code in the process.

Re: [PATCH 0/5] clocksource patches

Re: [PATCH 0/5] clocksource patches

A good abstraction should concentrate on the `_common_` properties and I don't think that the continuous cycle model is a good general abstraction for all types of clocks. Tick based clocks are already more complex due to the extra work needed to emulate the cycle counter. Some clocks may already provide a nsec value (e.g. virtual clocks in a virtualized environment), where your generic nsec calculation is a complete waste of time. A common property of all clocks is that we want a nsec value from them, so why not simply ask the clock for some kind of nsec value and provide the clock driver with the necessary library routines to convert the cycle value to a nsec value, where you actually have the necessary information to create efficient code. As long as you try to pull the cycle model into the generic model it will seriously suck from a performance perspective, as you separate the cycle value from the information how to deal with it efficiently.

For features like robust timekeeping in the face of lost ticks (needed for virtualization, and realtime), as well as high-res timers and dynamic/no-idle ticks, we **NEED** a continuous clock.

I made a quick audit of the arches to see what the breakdown was for continuous vs tick clocks:

Continuous:

i386,x86-64, ia64, powerpc, ppc, sparc, alpha, mips, parisc, s390, xtensa, and arm (pxa, sa1100, plat-omap)

Tick:

arm (other), cris, m32, m68k, sh

xtime/no intertick resolution:

frv, h8300, v850

I'll admit, the continuous cycle model doesn't fit tick based clocks, but we shouldn't limit the abstraction to the lowest common denominator. By providing what I suggested above (w/ the two `__get_nsec_offset()` implementations), we reduce code for both models, and allow progress for systems w/ continuous clocks in a generic fashion.

I'm also still interested in opinions about different possibilities to organize the clocksource infrastructure (although I'd more appreciate pro/con arguments than outright rejections). One possibility here would be

Re: [PATCH 0/5] clocksource patches

to also shorten the function names a bit (clocksource_ is a lot to type :), cs_... or clock_... would IMO work too.

I *much* prefer the clarity of clocksource over the wear and tear typing it might take on my fingers.

What is the special meaning of "clocksource" vs e.g. just "clock"?

"clock" is already overloaded. We have the CLOCK_MONOTONIC/CLOCK_REALTIME clocks, we have the RTC clocks... Its a cycle counter we're using to accumulate time, thus clocksource seemed understandable and unique.

I also kept it separate from the do_timer() callback and simply created a new callback function, which archs can use. This makes it less likely to break any existing setup and even allows to coexists both methods until archs have been completely converted.

One of the reasons I did the significant rework of the TOD patches was so that we could generically introduce the clocksource abstraction first (using jiffies) for all arches. I feel this provides a much smoother transition to the generic timekeeping code (and greatly reduces the amount of CONFIG_GENERIC_TIME specific code), so I'm not sure I understand why you want to go back to separate do_timer() functions.

Apropos code size: I checked the generated code size of timer.o (default i386 config), before it was 3248 bytes, with your patches it goes to 5907 bytes and by disabling CONFIG_GENERIC_TIME it still is 4554. With my patches it's 4767/4347 bytes (removing the old time code saves another 315 bytes).

This isn't really a fair comparison (yet atleast), as your patches don't appear to handle suspend/resume correctly. Nor did your patches even boot on my laptop. :(

I'll admit that my code could use more low-level optimization, and I welcome patches against my code to improve it. My code has already

Re: [PATCH 0/5] clocksource patches

Re: [PATCH 0/5] clocksource patches

gotten a good amount of testing in both -mm and -rt, so I know it works.
Lets make it fast too, but just in steps.

In my version I was very careful to keep the common interrupt paths as short as possible, so that most of this code is not even executed most of the time. You can't just "hope" that gcc does the work for you (especially with 64bit math), you actually have to check the code and you would have the noticed the bloated code it generates.

Good point. I just sent a patch to Andrew fixing the assumption that gcc could avoid a few of the mults. And I'm working on more patches to reduce the text size as well.

Once again, I do appreciate the feedback.

thanks
-john

-

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