RT Troubles

Lessons Learned & Open Questions

Gratian Crișan
RT-Summit 2017

gratian.crisan@ni.com, gratian@gmail.com
Context

- National Instruments
  - Makes hardware & software for test, measurement and control
- Real-Time OS group
  - Using PREEMPT_RT for 6+ years
  - ARM and Intel x86_64 architectures
  - Embedded CPU + FPGA products
  - OpenEmbedded / Yocto – based distribution
- Disclaimers
  - Work by multiple people
  - Intentionally picked problems with ugly hacks
  - Some data might be stale by now
Agenda

- problem_space();
- do {
  - rt_trouble_area();
  - discussion();
} while (topics && time);
Problem space

- Controller feedback process
- Input output loop frequency (latency)

10 kHz (100 µS)
- PREEMPT_RT
- 2-core ARMv7
- 12-core x86_64

100 kHz (10 µS)
- FPGAs

RT troubles

[1] our current use cases
[2] in general wake-up latency accounts for majority of control loop latency
RT Trouble #1
How bumping an Ethernet cable can ruin your day RT and some TPM troubles
Symptoms

- CPU appears stalled in a MMIO read instruction for hundreds of µS
- Timer interrupts get delivered late even though the interrupts are enabled
- Initially discovered in e1000 / e1000e network drivers\textsuperscript{[1]}
  - by (accidentally) bumping into an Ethernet cable during a cyclictest run
- Recently found in TPM driver\textsuperscript{[2]}
  - by (intentionally) accessing the TPM chip while running cyclictest

\textsuperscript{[1]} drivers/net/ethernet/intel/e1000e/*
\textsuperscript{[2]} drivers/char/tpm/tpm_tis.c
<table>
<thead>
<tr>
<th>CPU</th>
<th>Time Stamp</th>
<th>Task</th>
<th>PID</th>
<th>Latency</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>6081</td>
<td>37702.513077</td>
<td>ntpm</td>
<td>2564</td>
<td>...</td>
<td>sys_exit_open</td>
</tr>
<tr>
<td>6082</td>
<td>37702.513612</td>
<td>d.0</td>
<td>2564</td>
<td>...</td>
<td>page_fault_user</td>
</tr>
<tr>
<td>6083</td>
<td>37702.513615</td>
<td>d.0</td>
<td>2564</td>
<td>...</td>
<td>mmu_page_alloc</td>
</tr>
<tr>
<td>6084</td>
<td>37702.513532</td>
<td>d.0</td>
<td>2564</td>
<td>...</td>
<td>sys_enter</td>
</tr>
<tr>
<td>6085</td>
<td>37702.513534</td>
<td>d.0</td>
<td>2564</td>
<td>...</td>
<td>sys_entry_write</td>
</tr>
<tr>
<td>6086</td>
<td>37702.513399</td>
<td>d.0</td>
<td>2564</td>
<td>...</td>
<td>local_timer_entry</td>
</tr>
<tr>
<td>6087</td>
<td>37702.517109</td>
<td>ntpm</td>
<td>2564</td>
<td>d.1</td>
<td>hrtimer_cancel</td>
</tr>
<tr>
<td>6088</td>
<td>37702.517111</td>
<td>d.0</td>
<td>2564</td>
<td>d.1</td>
<td>hrtimer_expire_entry</td>
</tr>
<tr>
<td>6089</td>
<td>37702.517116</td>
<td>d.0</td>
<td>2564</td>
<td>d.1</td>
<td>sched_waking</td>
</tr>
<tr>
<td>6090</td>
<td>37702.517125</td>
<td>d.8h20</td>
<td>2564</td>
<td>d.1</td>
<td>hrtimer_expire_exit</td>
</tr>
<tr>
<td>6091</td>
<td>37702.517126</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.1</td>
<td>write_msr</td>
</tr>
<tr>
<td>6092</td>
<td>37702.517127</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.1</td>
<td>write_msr</td>
</tr>
<tr>
<td>6093</td>
<td>37702.517128</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.1</td>
<td>local_timer_exit</td>
</tr>
<tr>
<td>6094</td>
<td>37702.517130</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.1</td>
<td>rcu_utilization</td>
</tr>
<tr>
<td>6095</td>
<td>37702.517131</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.1</td>
<td>rcu_utilization</td>
</tr>
<tr>
<td>6096</td>
<td>37702.517134</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sched_stat_runtime</td>
</tr>
<tr>
<td>6097</td>
<td>37702.517139</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sched_switch</td>
</tr>
<tr>
<td>6098</td>
<td>37702.517140</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>tlb_flush</td>
</tr>
<tr>
<td>6099</td>
<td>37702.517142</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>x86_ipl_regs_deactivated</td>
</tr>
<tr>
<td>6100</td>
<td>37702.517143</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>x86_ipl_regs_deactived</td>
</tr>
<tr>
<td>6101</td>
<td>37702.517144</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>x86_ipl_regs_activate</td>
</tr>
<tr>
<td>6102</td>
<td>37702.517144</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>x86_ipl_regs_activate</td>
</tr>
<tr>
<td>6103</td>
<td>37702.517147</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sys_exit</td>
</tr>
<tr>
<td>6104</td>
<td>37702.517148</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sys_exit_clock_nanosleep</td>
</tr>
<tr>
<td>6105</td>
<td>37702.517169</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sys_entry</td>
</tr>
<tr>
<td>6106</td>
<td>37702.517170</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>sys_entry_write</td>
</tr>
<tr>
<td>6107</td>
<td>37702.517176</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>print</td>
</tr>
<tr>
<td>6108</td>
<td>37702.517178</td>
<td>d.8h0</td>
<td>2564</td>
<td>d.20</td>
<td>svs_exit</td>
</tr>
</tbody>
</table>
Cyclic test histogram with TPM load

~400 µS added latency
Cyclic test histogram with TPM load

hackbench max ~56 µS

~400 µS added latency
Why this happens

- CPU can write exponentially faster than the I/O device can sink
- Writes are buffered between the CPU and I/O
- Generally not an issue if the number of writes is small
- A MMIO read has to wait for every write to drain resulting in hundreds of μS latency spikes
e1000/e1000e hack[1]

--- a/drivers/net/ethernet/intel/e1000e/mac.c
+++ b/drivers/net/ethernet/intel/e1000e/mac.c
@@ -353,6 +353,7 @@ void e1000e_update_mc_addr_list_generic(struct e1000_hw
  
  /* replace the entire MTA table */
  for (i = hw->mac.mta_reg_count - 1; i >= 0; i--)
+     E1000_WRITE_REG_ARRAY(hw, E1000_MTA, i,
       hw->mac.mta_shadow[i]);
     E1000_WR_DELAY();
     ele_flush();

+#ifdef CONFIG_E1000_DELAY
+#define E1000_WR_DELAY() usleep_range(50, 100)
+%else ...
--- a/drivers/char/tpm/tpm_tis.c
+++ b/drivers/char/tpm/tpm_tis.c
@@ -103,7 +128,7 @@ static int tpm_tcg_write_bytes(struct tpm_tis_data
     struct tpm_tis_tcg_phy *phy = to_tpm_tis_tcg_phy(data);
     while (len--)
-    iowrite8(*value++, phy->iobase + addr);
+    tpm_tis_iowrite8(*value++, phy->iobase, addr);
     return 0;
+static inline void tpm_tis_iowrite8(u8 b, void __iomem *iobase,…)
+{
+    iowrite8(b, iobase + addr);
+    tpm_tis_flush(iobase);
+}

#ifdef CONFIG_PREEMPT_RT_FULL
    ioread8(iobase + TPM_ACCESS(0));
#endif
Discussion

- Preface: we know this is a hardware problem that might not have a good software solution.

- Is it possible / likely on other archs?

- Other drivers you know of that have this problem?
Discussion

• Can we detect this access pattern (at runtime)?

• Any way to track I/O buffer states?
Discussion

- Other ways to throttle MMIO stores?
- Is there a more general solution possible?
  - Adding a load (with exceptions) in iowriteN()/writeX() macros for PREEMPT_RT?[1]

RT Trouble #2
Concurrent hrtimer expirations from low priority threads
RT Trouble #2 – Thank You!

Concurrent hrtimer expirations from low priority threads
Symptoms

- Multiple timed sleeps or timeouts coming from SCHED_OTHER threads can stack up to large latencies for RT threads
- It is not just clock_nanosleep()
  - lots of other things that use hrtimers e.g. futexes (with timeouts)
high priority thread wakeup

~15 µS

more timer expirations

another timer irq

sched tick ~40 µS

~15 µS

... more timer expirations

149µS
Pathological test

* Configurable number of SCHED_OTHER threads doing

```c
while (!g_stop) {
    t.tv_sec = 0;
    interval = (rand() * 1000000LL) / RAND_MAX;
    t.tv_nsec = interval;
    clock_nanosleep(CLOCK_MONOTONIC, 0, &t, NULL);
}
```
cyclic test wakeup

~15 µS

~15 µS

...
• Intel(R) Atom(TM) E3825 @ 1.33GHz dual core
• kernel: 4.11.12-rt14
• cyclcietest -m -S -p 98 -i 237 –H 200
• three day run
• two devices:
  1. hackbench -g25 -l 1000000000
     • running in process mode with 25 groups using 40 file descriptors each
       (== 1000 tasks)
  2. 1000 threads running timer stress
• ARMv7 (v7l) @ 667MHz dual core
• kernel: 4.11.12-rt14
• cyclictest -m -S -p 98 -i 737 –H 500
  • three day run
• two devices:
  1. hackbench -l 1000000000 -f 25
     • running in process mode with 10 groups using 50 file descriptors each (== 500 tasks)
  2. 500 threads running timer stress

hackbench max 103 µS
hrtimer stress max 261 µS
Hack\textsuperscript{[1]} that (kind of) worked before v4.11.12-rt13

```c
void hrtimer_init_sleeper(struct hrtimer_sleeper *sl,  
                          struct task_struct *task)
{
    sl->timer.function = hrtimer_wakeup;
-   sl->timer.irqsafe = 1;
+   sl->timer.irqsafe = rt_task(task);
    sl->task = task;
}
```

\textsuperscript{[1]} https://marc.info/?l=linux-rt-users&m=148354667204563&w=2
time/hrtimer: Use softirq based wakeups for non-RT threads

Normal wake ups (like clock_nanosleep()) which are performed by normal users can easily lead to 2ms latency spikes if (enough) hrtimer wakeups are synchronized. This patch moves all hrtimers wakeups to the softirq queue unless the caller has a RT priority.

Reported-by: Gratian Crisan <gratian.crisan@ni.com>
Signed-off-by: Sebastian Andrzej Siewior <bigeasy@linutronix.de>

• Intel (R) Atom (TM) E3825 @ 1.33GHz
• kernel: 4.11.12-rt14 + patch
• cyclitest -m -S -p 98 -i 237 –H 200
• 1000 threads, hrtimer stress load
• ARMv7 (v7l) @ 667MHz dual core
• kernel: 4.11.12-rt14 + patch
• cyclicstest -m -S -p 98 -i 237 –H 200
• 500 threads, hrtimer stress load

hrtimer stress, w/o patch max 261 µS
with patch max 90 µS
Lessons learned

- Report problems early
- Upgrade to the latest linux-rt-devel branch as soon as possible
- Don’t go on vacation before your RT-Summit presentation
RT Trouble #3

Lack of priority inheritance support in the glibc pthread library
libpthread priority inheritance support

- **With** priority inheritance support:
  - pthread_mutex_*
  - \{ FUTEX_LOCK_PI/UNLOCK_PI \}

- **Without** priority inheritance support:
  - pthread_rwlock_* internal lock
  - sem_*
  - pthread_spin_*
  - pthread_cond_* internal lock
  - \{ FUTEX_WAIT/WAKE, FUTEX_WAIT_BITSET/WAKE, user-space spinning \}
  - \{ see next slide \}
**Bug 11588 - pthread condvars are not priority inheritance aware**

- **Status**: NEW
- **Alias**: None
- **Product**: glibc
- **Component**: nptl (show other bugs)
- **Version**: 2.12
- **Importance**: P2 enhancement
- **Target Milestone**: ---
- **Assignee**: Not yet assigned to anyone
- **Reported**: 2010-05-11 18:45 UTC by Darren Hart
- **Modified**: 2017-08-31 20:44 UTC (History)
- **CC List**: 16 users (show)
- **See Also**:
  - **Host**:
  - **Target**:
  - **Build**:  
- **Last reconfirmed**:  
- **Flags**: fweimer: security-
Current state glibc bug #11588

Torvald Riegel  2017-01-11 11:50:41 UTC

The new condition variable implementation is now committed upstream. It should be the base for any improvement suggestions from now on.

How to support PI for condvars has also been discussed at the Linux Real-Time Summit 2016: https://wiki.linuxfoundation.org/realtime/events/rt-summit2016/schedule

So far, there is no known solution for how to achieve PI support given the current constraints we have (eg, available futex operations, POSIX requirements, ...).
Austin Group defect #609

<table>
<thead>
<tr>
<th>ID</th>
<th>Category</th>
<th>Severity</th>
<th>Type</th>
<th>Date Submitted</th>
<th>Last Update</th>
</tr>
</thead>
</table>

**Reporter**: mmihaylov  | **View Status**: public

**Assigned To**: ajosey  | **Resolution**: Open

**Priority**: normal

**Status**: Under Review

**Name**: Mihail Mihaylov

**Organization**:  

**User Reference**:  

**Section**: pthread_cond_broadcast, pthread_cond_signal

**Page Number**: 1043

**Line Number**: 33043 - 33046

**Interp Status**: ---

**Final Accepted Text**:  

**Summary**: 0000609: It is not clear what threads are considered blocked with respect to a call to pthread_cond_signal() or pthread_cond_broadcast()
Discussion

- Do you know of work underway / progress since last year?
- Alternative libraries out there for RT friendly locking?
- Any RT friendly data structures library (e.g. circular buffers, FIFOs, etc.)
RT Trouble #4
Managing IRQ priorities and IRQ priority inversions
Big disclaimers

- We know the following patches are not appropriate for upstream.
- The need for them arises from our inexperience at the time and a usability problem with mapping an IRQ to its corresponding thread PID.
- We are looking for best practice ideas.
What is the best way to set IRQ priorities?

* Currently carrying[^1]:

```
irq: Add priority support to /proc/irq/...
```

This patch allows configuring priority for different irq threads through the `/proc/irq/` system (much same as the existing mechanism to configure the core affinity for irqs).

Signed-off-by: Sankara S Muthukrishnan <sankara.m@ni.com>
Signed-off-by: Julia Cartwright <julia.cartwright@ni.com>

[^1]: [https://github.com/ni/linux/commit/5ff3a76173659863b6c5bda9ecf094d4621ccba7](https://github.com/ni/linux/commit/5ff3a76173659863b6c5bda9ecf094d4621ccba7)
What is the best way to set priorities on new IRQs?

- Currently carrying[1][2]:

[RFC][PATCH] fs/proc: add poll()ing support to /proc/interrupts

Implement polling on procfs' "interrupts" file which observes changes to IRQ action handlers. The poll fires each time an action handler is registered or unregistered.

This change enables daemons to watch for changes and apply certain system policies relating to IRQ processing. For example, modify execution priority of dedicated IRQ tasks after they're created.

Signed-off-by: Haris Okanovic <haris.okanovic@ni.com>
Signed-off-by: Ovidiu-Adrian Vancea <ovidiu.vancea@ni.com>
Signed-off-by: Brad Mouring <brad.mouring@ni.com>

---

[1] https://github.com/ni/linux/commit/8e2f148e2f4762cc2b6490ec01d5d31bc440bcf8
IRQ priority inversions

Example

- **Context**
  - Watchdog functionality implemented in a CPLD connected to a I\(^2\)C bus
  - It can be configured to fire an interrupt (as opposed to a straight reset)

- **Behavior**
  - High priority watchdog interrupt fires
  - To acknowledge the interrupt slow I\(^2\)C transfers need to happen
  - I\(^2\)C interrupt thread has low priority
  - Some unrelated mid-priority thread preempts the I\(^2\)C interrupt
Discussion

- Ongoing work on avoiding IRQ priority inversions?
- Is there a more general solution to the priority inversion problem with completion objects?
Extra stuff
Tip #1

Check config options after a kernel upgrade
config **CPU_SW_DOMAIN_PAN**[1]
bool "Enable use of CPU domains to implement privileged no-access"
depends on MMU && !ARM_LPAE
default y

help
Increase kernel security by ensuring that normal kernel accesses are unable to access userspace addresses. This can help prevent use-after-free bugs becoming an exploitable privilege escalation by ensuring that magic values (such as LIST_POISON) will always fault when dereferenced.

[1] Introduced in v4.3, commit a5e090acb54 ("ARM: software-based priviledged-no-access support"). Adds code in uaccess_*, save_regs, load_regs macros.
# Event 'cycles:ppp'

<table>
<thead>
<tr>
<th>#</th>
<th>Event</th>
<th>Baseline</th>
<th>Delta</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>20.03%</td>
<td>-7.06%</td>
<td>[kernel.kallsyms]</td>
<td>(k) __gettimeofday64</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>17.34%</td>
<td>-1.25%</td>
<td>libc-2.23.so</td>
<td>(.) __clock_gettime</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>15.93%</td>
<td>+3.05%</td>
<td>[kernel.kallsyms]</td>
<td>(k) vector_sw1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>8.88%</td>
<td>+10.38%</td>
<td>[kernel.kallsyms]</td>
<td>(k) sys_clock_gettime</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6.53%</td>
<td>-2.07%</td>
<td>[kernel.kallsyms]</td>
<td>(k) __copy_to_user_std</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>5.59%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) gt_counter_read</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>4.86%</td>
<td>-1.24%</td>
<td>[kernel.kallsyms]</td>
<td>(k) ret_fast_syscall</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>3.82%</td>
<td>-0.36%</td>
<td>simple_clock_gettime</td>
<td>(.) main</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>3.54%</td>
<td>+1.15%</td>
<td>[kernel.kallsyms]</td>
<td>(k) gt_clocksource_read</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>3.29%</td>
<td>-0.12%</td>
<td>[kernel.kallsyms]</td>
<td>(k) gettimeofday64</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>2.99%</td>
<td>+0.19%</td>
<td>posix_clock_gettime</td>
<td>(k) clock_gettime</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>2.94%</td>
<td>+0.44%</td>
<td>[kernel.kallsyms]</td>
<td>(k) clockid_to_kclock</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2.06%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) aeabi_llsr</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0.70%</td>
<td>-0.32%</td>
<td>[kernel.kallsyms]</td>
<td>(k) local_restart</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0.55%</td>
<td>-0.54%</td>
<td>simple_clock_gettime</td>
<td>(.) memsetplt</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0.01%</td>
<td></td>
<td>ld-2.23.so</td>
<td>(.) do_lookup_x</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) vma interval tree_remove</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>0.01%</td>
<td>-0.00%</td>
<td>[kernel.kallsyms]</td>
<td>(k) v7 flush icache_all</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) test_and_set_bit</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) strlen_user</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>0.09%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) __alloc_pages_nodemask</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>0.09%</td>
<td>-0.00%</td>
<td>[kernel.kallsyms]</td>
<td>(k) perf_event_exec</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>+2.99%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) gt_counter_read</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>+1.53%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) __aeabi_t sar</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>+0.68%</td>
<td></td>
<td>simple_clock_gettime</td>
<td>(.) __gmon_start@plt</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>+0.20%</td>
<td></td>
<td>simple_clock_gettime</td>
<td>(.) sched_setaffinity@plt</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>+0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) cpuset_node_allowed</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>+0.01%</td>
<td></td>
<td>ld-2.23.so</td>
<td>(.) dl_init_paths</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>+0.01%</td>
<td></td>
<td>ld-2.23.so</td>
<td>(.) dl_lookup_symbol_x</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>+0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) f ilemap_map_pages</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>+0.01%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) do page fault</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>+0.00%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) pin valid</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>+0.00%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) rt mutex_lock</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>+0.00%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) finish_task_switch</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>+0.00%</td>
<td></td>
<td>[kernel.kallsyms]</td>
<td>(k) _raw_spin_unlock_irq</td>
</tr>
</tbody>
</table>
Tip #2
Check your clock sources
Check your clock sources

- Issues encountered
  - TSC clock source gets disabled by the clock source watchdog due to acpi_pm rollover
  - Boot hang caused by left over test code in BIOS that sets the TSC_ADJUST register on core 0

- On upgrades and new hardware it helps to:
  - Check current clock source
    
    `/sys/devices/system/clocks/clocks/clocksource0/current_clocksource`
  - Compare timer expirations against external reference
  - Drive trace off external clock source (e.g. FPGA)
Tip #3
Run reboot tests
Run reboot tests

- Multiple issues discovered by running a simple reboot test
  - Hangs on boot
  - Ext4 data corruptions
  - NAND read disturbs
  - Ethernet link detection issues
  - futex race on exit
  - i915 crash on module load

- Suggestions
  - Simple test - calls reboot once the software stack is up
  - Hard reboots - data loss is OK, data corruption is not
  - (optional) Temperature controlled chamber