hrtimers and beyond
-
transformation of the Linux time(r) system

Thomas Gleixner
Douglas Niehaus

OLS 2006
Original time(r) system

- Timekeeping
- Tick
  - Process acc.
  - Profiling
  - Jiffies
  - Timer wheel

Arch 1
- TOD: Clock source
- HW
- ISR: Clock event source
- HW

Arch 2
- TOD: Clock source
- HW
- ISR: Clock event source
- HW

Arch 3
- TOD: Clock source
- HW
- ISR: Clock event source
- HW
History

- double linked list sorted by expiry time
- UTIME (1996)
- timer wheel (1997)
- HRT (2001)
- hrtimers (2006)
Timer Wheel

- periodic tick necessary
- O(1) insertion / deletion
- recascading in bursts (can cause high latencies)
- higher tick frequencies don't scale due to long lasting timer callbacks and increased recascading
Cascading
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>100</th>
<th>250</th>
<th>1000</th>
<th>HZ</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>256</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>64</td>
<td>2560</td>
<td>1024</td>
<td>256</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>64</td>
<td>164</td>
<td>66</td>
<td>16</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>64</td>
<td>175</td>
<td>70</td>
<td>17</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>64</td>
<td>186</td>
<td>75</td>
<td>19</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>250</td>
<td>1000</td>
<td>Hz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>64</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>16</td>
<td>640</td>
<td>256</td>
<td>64</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>16</td>
<td>10240</td>
<td>4096</td>
<td>1024</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>16</td>
<td>164</td>
<td>66</td>
<td>16</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>16</td>
<td>44</td>
<td>17</td>
<td>4</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cascading

- array sizes have to be chosen carefully taking tick frequency into account
- rare (multiple) cascades increase latency
  - use cases have to be analysed to avoid problematic cascading
- separating timers with high accuracy requirement from coarse grained timeouts will relax the situation
timers vs. timeouts

**timers**
- precise event scheduling
- accurate
- likely to expire

**timeouts**
- report error conditions
- coarser grained
- likely to be removed before expiration
History of high resolution timers

- UTIME – KURT-Linux
  - University of Kansas
- HRT – fork of UTIME
  - Monta Vista
- Hrtimers
  - Linutronix
Why hrtimers?

- UTIME and HRT added a subjiffy field
  - Kept jiffy ticks by design to avoid broader kernel change impact
  - Modes: on top of the timer wheel or separate high-resolution event list
- HRT moved high resolution timers into a separate list one tick before expiry
  - Suffered from timer wheel latencies
hrtimers

- timers inserted into a red-black tree sorted by expiration time
- separate queue for each base clock, which allowed simplifying POSIX timers
- base code is still tick driven (softirq is called in the timer softirq context)
- time values are kept in new data type ktime_t (using nanosecond base)
ktime_t

- optimizable data type for both 32 and 64 bit machines
- plain nanosecond value on 64 bit CPU
- (seconds, nanoseconds) pair on 32 bit CPUs with field order allowing (depending on the endianess) 64 bit add, subtract, compare operations.
hrtimer users

• nanosleep
• itimer
• POSIX timers
• timed futex operations
how to get high resolution timers?

- solve the tick (jiffy) dependency of timekeeping
- create a generic framework for next event interrupt programming
- replace the periodic tick interrupt by timers under hrtimers
Timekeeping

• Make use of John Stultz's Generic Time of Day framework
  - architecture independent
  - generic framework replaces duplicated architecture code
  - better decoupling from tick
clockevents

- Generic infrastructure to distribute timer related events
  - architecture independent
  - generic framework replaces duplicated architecture code
  - allows quality based selection of clock event hardware
hrtimers + GTOD + clockevents
tick emulation

- Use a per-CPU hrtimer to emulate tick
  - update jiffies and NTP adjustments
  - per-CPU calls
- process accounting and profiling
- Allows high resolution timers and/or dynamic ticks
**hrtimers + GTOD + clockevents + tick emulation**

![Diagram of hrtimers, GTOD, clockevents, and tick emulation with architectural diagrams for Arch 1, Arch 2, and Arch 3.](image-url)
## High Resolution Performance

```plaintext
clock_nanosleep(ABS_TIME)
interval: 10ms
10000 loops
no load

<table>
<thead>
<tr>
<th>Kernel</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.16</td>
<td>24</td>
<td>4042</td>
<td>1989</td>
<td>µs</td>
</tr>
<tr>
<td>2.6.16-hrt</td>
<td>12</td>
<td>94</td>
<td>20</td>
<td>µs</td>
</tr>
<tr>
<td>2.6.16-rt</td>
<td>6</td>
<td>40</td>
<td>10</td>
<td>µs</td>
</tr>
</tbody>
</table>
```
high resolution performance

clock_nanosleep(ABS_TIME)
interval 10ms
10000 loops
100% load

<table>
<thead>
<tr>
<th>Kernel</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.16</td>
<td>55</td>
<td>4280</td>
<td>2198</td>
<td>µs</td>
</tr>
<tr>
<td>2.6.16-hrt</td>
<td>11</td>
<td>458</td>
<td>55</td>
<td>µs</td>
</tr>
<tr>
<td>2.6.16-rt</td>
<td>16</td>
<td>55</td>
<td>20</td>
<td>µs</td>
</tr>
</tbody>
</table>
dynamic tick idle behaviour

- timer interrupts reduced to ~1 per second.
  - instrumentation to identify the timer (ab)users to improve the idle sleep length
timer wheel batching

- run the timer wheel at a lower frequency than the scheduler tick by skipping timer wheel processing for a user space configurable number of ticks
- improves interactivity
things to be done

- get it merged (target is 2.6.19)
- support more architectures (prototypes for ARM and PPC available)
- tighter integration into power management
Conclusions

• significant changes are necessary but the benefit is significant increases in:
  – architecture independent code
  – ease of using wide range of time keeping and timer event hardware
  – increased resolution for scheduled events when desired