LINUTRONIX
LINUX FOR INDUSTRY

State of the PREEMPT_RT

Real-time Linux Summit 2019

Sebastian A. Siewior

Linutronix GmbH

October 31, 2019

LINUTRONIX
LINUX FOR INDUSTRY

LINUTRONIX
LINUX FOR INDUSTRY

Currently maintained version

- 5.2-RT development
- 4.19-RT Steven Rostedt 2020-12
- 4.14-RT Tom Zanussi 2024-01
- 4.9-RT Clark Williams 2023-01
- 4.4-RT Daniel Wagner 2022-02
- Documentation
  `https://wiki.linuxfoundation.org/realtime/start`
  `https://wiki.linuxfoundation.org/realtime/preempt_rt_versions`

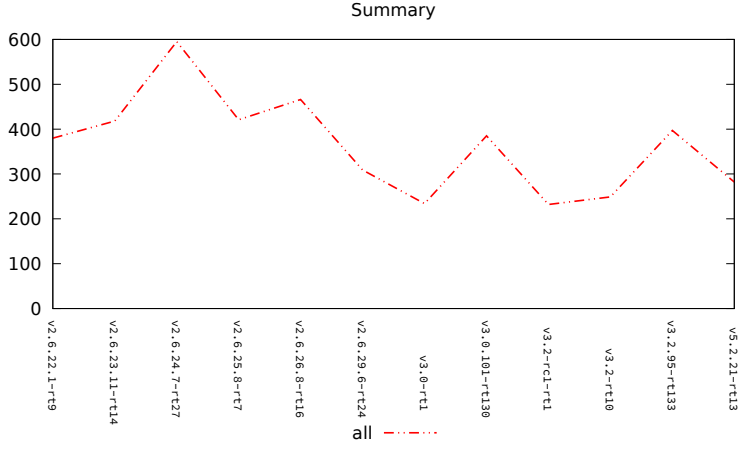
LINUTRONIX
LINUX FOR INDUSTRY

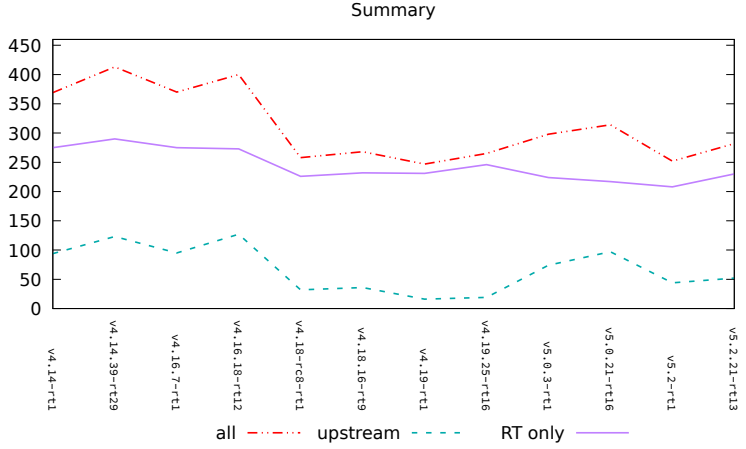
Pieces from the -RT queue

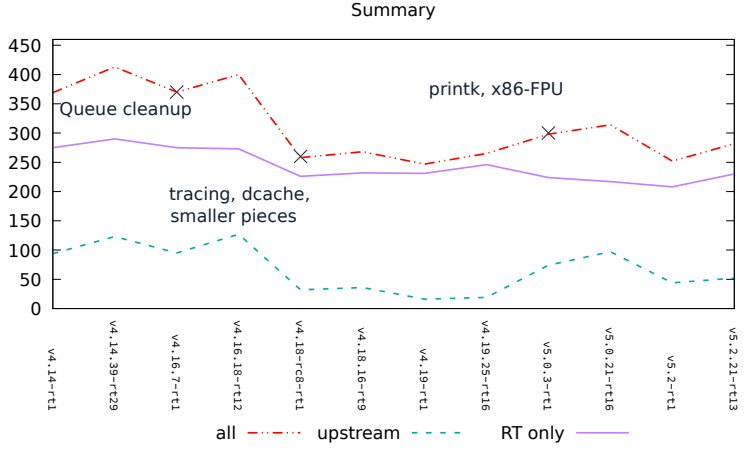
- v2.6.18-rc1: rtmutex, pi-futex, lockdep, generic irq-chip.
- v2.6.21-rc1: generic clockevents, highres timers, NO_HZ.
- v2.6.25-rc1: preemptible RCU.
- v2.6.27-rc1: tracing, ftrace.
- v2.6.30-rc1: adaptive spinning for mutex.

Pieces from the -RT queue

- v2.6.33-rc1: raw_spinlock_t, arch_spinlock, PICK_OP().
- v2.6.39-rc1: threaded interrupts.
- v3.10-rc1: NO_HZ_FULL
- v4.8-rc1: non cascading timer wheel
- v4.16-rc1: softirq based hrtimers


Summary
600
500
400
300
200
100
0
v2.6.22.1-rt9
v2.6.23.11-rt14
v2.6.24.7-rt27
v2.6.25.8-rt7
v2.6.26.8-rt16
v2.6.29.6-rt24
v3.0-rt1
v3.0.101-rt130
v3.2-rc1-rt1
v3.2-rt10
v3.2.95-rt133
v5.2.21-rt13
all


Summary
450
400
350
300
250
200
150
100
50
0
v4.14-rt1
v4.14.39-rt29
v4.16.7-rt1
v4.16.18-rt12
v4.18-rc8-rt1
v4.18.16-rt9
v4.19-rt1
v4.19.25-rt16
v5.0.3-rt1
v5.0.21-rt16
v5.2-rt1
v5.2.21-rt13
all
upstream
RT only


Summary
450
400
350
300
250
200
150
100
50
0
Queue cleanup
printk, x86-FPU
tracing, dcache,
smaller pieces
v4.14-rt1
v4.14.39-rt29
v4.16.7-rt1
v4.16.18-rt12
v4.18-rc8-rt1
v4.18.16-rt9
v4.19-rt1
v4.19.25-rt16
v5.0.3-rt1
v5.0.21-rt16
v5.2-rt1
v5.2.21-rt13
all
upstream
RT only

Preparing

- Removed a lot of RT specifics function replacements
- With nort or rt suffix
- Usually older code, reasons were not valid
- It was still needed but could be changed
- Supporting the refcount_t efforts (atomic_...to refcount_...).

CPU chill

- Lock order A $\Rightarrow$ B.
- Reverse order B $\Rightarrow$ A will deadlock.
- Lock B, trylock A
- Never works if owner if A was preempted
- Using cpu_chill() as a workaround

Sequence counts

- Writer does write_seqcount_begin() + write_seqcount_end()
- The counter goes odd, even
- Reader does read_seqcount_begin() + read_seqcount_retry()
- Retry if counter changes or is odd.
- A reader with a high priority may preempt the writer

Different printk

- printk() supports printing from every context.
- Printing from IRQ-off regions requires different locks
- This would block interrupts while printing
- John's talk
  `https://www.linuxplumbersconf.org/event/4/contributions/290/attachments/276/463/lpc2019_jogness_printk.pdf`

Bringing RT pieces upstream

- Getting code upstream got difficult.
- Some accepted code and small rewrites.
- Others refused changes for OOT code.
- CONFIG_PREEMPT_RT switch in v5.3-rc1.

Canceling an active timer

- Schedule a hrtimer, timer fires.
- Cancel the timer while the callback is active
- Retry if callback is active.
- Can be preempted by caller.
- Now acquiring a lock if the callback is active

lock_lock()

- Per-CPU variables protected by preempt_disable().
- Sometimes by local_irq_save().
- On RT a per-CPU spinlock.
- On !RT the old behaviour.
- Code is left untouched once all sots are identified.
- Should this go upstream?

random example

- get_random_u64() access a per-CPU array. Protected by preempt_disable()
- Array contains random numbers.
- Once the array is consume, the array will be refilled.
- kmalloc() → new_slab() → get_random_u32()
- Reworked with a spinlock_t
- b7d5dc21072cd (”random: add a spinlock_t to struct batched_entropy”)

apparmor example

- Has per-CPU aa_buffers. Allocates by default 2 * PATH_MAX * 2.
- This mains 64KiB on a system with 4 CPUs or 1MiB with 64 CPUs.
- Accessing the buffer(s) is protected with preempt_disable().
- profile_transition() needs to invoke other function which uses those buffers.
- so allocate, copy, invoke, copy-back, free the additional memory.
- Use a global pool instead.
- df323337e507a (”apparmor: Use a memory pool instead per-CPU caches”)

BPF

- Someone suggested to turn BPF off for the beginning .
- More discussion and information how things followed.
- The local-locks mechanism might be work.
- Then it ended.
- We are investigating what might work.

Softirq rework

- Initially one thread for each softirq.
- After a rework softirq is invoked after the interrupt tread.
- Different softirq vectors may interrupts one another.
- Two interrupt threads may not ”partly” run tasklet softirq.
- But tasklet and NET_RX and NET_TX may run in ”parallel”.
- ...requires additional locks for per-CPU data.
- Giving priority to different vectors is not possible.

Softirq rework

- Frederick attempted to mimic this.
- local_bh_disable() disables all 10 softirqs.
- Introduce local_bh_disable_mask(BIT(NET_RX_SOFTIRQ));
- Softirqs can interrupt one another.
- The locks have to be acquired carefully.

Softirq rework

- local_bh_disable() acts a BKL.
- It is not obvious what it protects
- In v5.0-RT cycle it received a local_lock().
- Behaves like a BKL on -RT, too.
- A forced threaded interrupt needs to disable softirqs.
- May lead to additional latency.

Softirq rework

A report on the mailing list:

- request_irq() for special thread with a high priority.
- Busy ATA interrupt. Is running for a long time.
- Forced threaded handler run within a local_bh_disable() section.
- The higher priority handler will block on this local_bh_disable().

What needs to be done

- Synchronisation via seqcount.
- Locking in mm, cgroup, workqueue.
- Simple waitqueues.
- Start with X86-64.

LINUTRONIX
LINUX FOR INDUSTRY

Thank you for your attention

**Special thanks to the Linux Foundation
for supporting our efforts to
bring PREEMPT_RT mainline.**

<sebastian.siewior@linutronix.de>