LINUTRONIX
LINUX FOR INDUSTRY

Following mainline with RT_PREEMPT

Linux Plumbers 2014

Sebastian A. Siewior

Linutronix GmbH

October 16, 2014

the usual approach

- take new Linux kernel release
- grab the 300 patches from last -RT release
- apply them one by one
- fix up rejects
- try to compile

why the reject?

- code simply moved, reorganized
- different logic?
- open coded locking
- you know the code, you did a dozen times, you fat finger a patches
- hopefully the typo is noticed a few patches later

once the queue applied, does it boot?

- something does not work, disable it ”temporary“
  - netconsole
  - preempt lazy on x86_64
- lucky when you see it directly
- less lucky when RCU stalls
- even less lucky when there is no output

what can go wrong?

- -RT changes large parts of the infrastructure
- nobody cares about -RT
- mostly performance optimization leads to funny code
- the core changes frequently

so what does the -RT patch do in principle?

- keep code interruptible as possible
  - spin_lock_irq() does not disable interrupts.
  - interrupts are threaded
  - remaining IRQ-off regions should be very short
  - break large preempt_disable() regions

where is it done?

- everywhere!
- arch/ changes are almost none
- but then we have sched, mm, timer, rtmutex, ...
- lightweight implementation of wait queue (swait)
- driver related changes are usually small but fun to find

and now the fallout

- awesome if the debug infrastructure is already working
- lockdep for spinlock, mutex, semaphore
- might_sleep() catches some other where the assumption of vanilla vs -RT change.
- take this

```
spin_lock_irq(lock);
```

vs

```
local_irq_disable();
spin_lock(lock);
```

it is all about different assumptions

- have you noticed cpu_chill()?
- lets say you spin_try_lock() until you get the lock
- the lock is taken, drop everything try again
- or yield() until it is ready
- ”works“ on SMP Systems
- on -RT we have a busy loop

one example of that

```
@@ -157,7 +157,7 @@ static struct dentry *get_next_positive_
                        parent = p->d_parent;
                        if (!spin_trylock(&parent->d_lock)) {
                                spin_unlock(&p->d_lock);
 -                              cpu_relax();
 +                              cpu_chill();
                                goto relock;
                        }
                        spin_unlock(&p->d_lock);
```

different assumptions, take two

- vanilla, spin_lock() means also preempt_disable()
- on -RT we do migrate_disable() instead
- now we have code that relies on preempt_disable()
- welcome the per-CPU BKL
- no annotation what it protects (no protection scope)

per-CPU BKL

- every break out of preempt_disable(), local_irq_disable() has to manually analyzed
- lets hope once upstream changes that code that our patch does not apply anymore
- per-CPU variables are similar
- the BKL design is bad, not really easy to maintain
- if you forgot about lock_kernel() just look at tty_lock()

local lock example

```
+static DEFINE_LOCAL_IRQ_LOCK(rotate_lock);
 @@ -440,11 +444,11 @@ void rotate_reclaimable_page(struct page
                 unsigned long flags;

                 page_cache_get(page);
 -               local_irq_save(flags);
 +               local_lock_irqsave(rotate_lock, flags);
                 pvec = &__get_cpu_var(lru_rotate_pvecs);
                 if (!pagevec_add(pvec, page))
                         pagevec_move_tail(pvec);
 -               local_irq_restore(flags);
 +               local_unlock_irqrestore(rotate_lock, flags);
         }
  }
```

what is keeping up really?

- no fun doing the same thing every six months
- good things came out of -RT
- pin-point problems in -RT, good design for mainline
- one typo and you wonder why RCU stalls :)
- luck is when you notice the typo 30 patches later
- hope that the old patch works and another way of doing is not needed

Thank you for your attention

Contact

Linutronix GmbH
Sebastian A. Siewior

Auf dem Berg 3
88690 Uhldingen
Germany

eMail bigeasy@linutronix.de