# Unblocking the softirq lock for PREEMPT_RT

## Linux Plumbers Conference

Sebastian A. Siewior

Linutronix GmbH

November 15, 2023

## Softirq on PREEMPT_RT vs vanilla

- Preemptible. Context switch is possible.
- Runs after the threaded handler.
- No piggyback after hardirq.
- Everything from hardirq goes to ksoftirqd.
- Due to preemption *local_bh_disable()* is a per-CPU lock.

## Results of the lock

- Ressources depending on BH locking are protected. 👍
- Long–running forced threaded interrupts block other forced threaded interrupts. 👎
- Timer, Tasklets, ...block forced threaded interrupts. 👎
- Best we can do is a PI–boost. The need–resched condition is never observed. 👎

# Trace force-threaded interrupts preempted

```
irq/40-eno0-2034 D...2  681 softirq_raise: vec=3 [action=NET_RX]
irq/40-eno0-2034 ..s.2  681 softirq_entry: vec=3 [action=NET_RX]
irq/40-eno0-2034 d.H.3  690 irq_handler_entry: irq=35
irq/40-eno0-2034 dNH33  692 sched_wakeup: irq/35-ahci prio=44
irq/40-eno0-2034 d.s23  694 sched_switch: prio=49 R+->irq/35-ahci prio=44

irq/35-ahci-837  d..31  696 sched_pi_setprio: irq/40-eno0 prio 49 -> 44
irq/35-ahci-837  d..21  699 sched_switch: prio=44 D->irq/40-eno0 prio=44

irq/40-eno0-2034 d.s34  715 sched_wakeup: iperf3 prio=120
irq/40-eno0-2034 d..21  736 sched_switch: prio=49 R+->irq/35-ahci prio=44

irq/35-ahci-837  D..13  740 softirq_raise: vec=4 [action=BLOCK]
irq/35-ahci-837  ..s.2  740 softirq_entry: vec=4 [action=BLOCK]
```

# But why exactly the lock

- A FPGA interrupt handler doing only wake up of userland.
- A CAN driver needs to inject packet.
- AHCI driver needs to do IO.
- local_bh_disable() and enable() plus the lock:
  - Preserve the raise softirq and invoke semantic.
  - Protect per-CPU ressources. Mostly.

## How to get rid of the lock?

- Identify the per-CPU ressources. Add a lock.
- The working PoC
  - local_lock_nested_bh() (followed preempt_disable_nested()).
  - lockdep check for BH. CPU migration must be off (due to disabled BH).
  - A per-CPU lock only on RT.
  - The raise and invoke semantic of softirqs is the same.
  - The macros in_serving_softirq(), softirq_count() work unchanged.
  - guard notation.

# Example struct napi_alloc_cache

```
@@ -264,6 +264,7 @@ static void *page_frag_alloc_1k(struct
 struct napi_alloc_cache {
+        local_lock_t bh_lock;
         struct page_frag_cache page;
         struct page_frag_1k page_small;
         unsigned int skb_count;
@@ -295,6 +298,7 @@ void *__napi_alloc_frag_align(unsigned int
         struct napi_alloc_cache *nc = this_cpu_ptr(&napi_alloc_cache);

         fragsz = SKB_DATA_ALIGN(fragsz);
+        guard(local_lock_nested_bh)(&napi_alloc_cache.bh_lock);

         return page_frag_alloc_align(&nc->page, fragsz, GFP_ATOMIC,
     align_mask);
 }
```

# Example struct napi_alloc_cache

```
@@ -725,9 +730,11 @@ struct sk_buff *__netdev_alloc_skb(struct net_device
                pfmemalloc = nc->pfmemalloc;
        } else {
                local_bh_disable();
-               nc = this_cpu_ptr(&napi_alloc_cache.page);
-               data = page_frag_alloc(nc, len, gfp_mask);
-               pfmemalloc = nc->pfmemalloc;
+               scoped_guard(local_lock_nested_bh, &napi_alloc_cache.bh_lock) {
+                       nc = this_cpu_ptr(&napi_alloc_cache.page);
+                       data = page_frag_alloc(nc, len, gfp_mask);
+                       pfmemalloc = nc->pfmemalloc;
+               }
                local_bh_enable();
        }
```

# Example softnet_data: xmit.recursion_lock

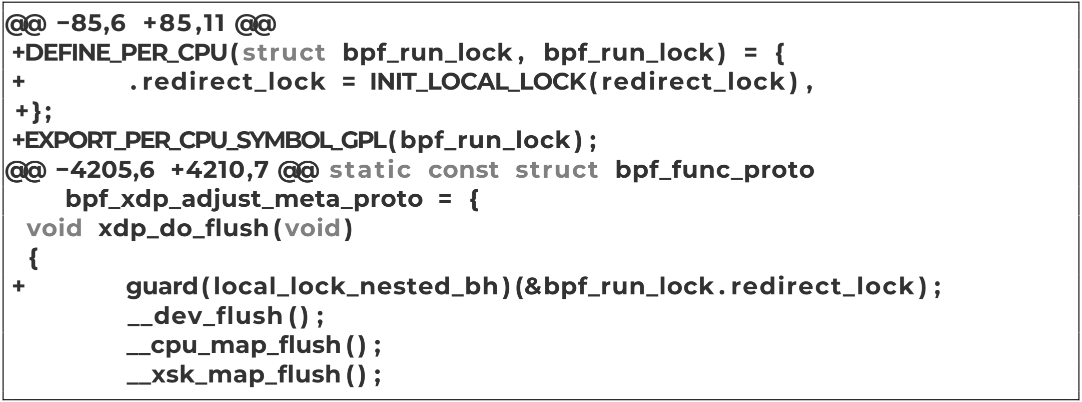
```
@@ -3208,6 +3208,10 @@ struct softnet_data {
 #endif
        /* written and read only by owning cpu: */
        struct {
+#ifdef CONFIG_PREEMPT_RT
+               struct task_struct *recursion_owner;
+               local_lock_t recursion_lock;
+#endif
               u16 recursion;
               u8  more;
 #ifdef CONFIG_NET_EGRESS
```

# Example softnet_data: xmit.recursion_lock

```
@@ -3272,6 +3276,27 @@ static inline bool dev_xmit_recursion(void)
                                 XMIT_RECURSION_LIMIT);
 }

+#ifdef CONFIG_PREEMPT_RT
+
+static inline void dev_xmit_recursion_inc(void)
+{
+	if (__this_cpu_read(softnet_data.xmit.recursion_owner) != current) {
+		local_lock_nested_bh(&softnet_data.xmit.recursion_lock);
+		__this_cpu_write(softnet_data.xmit.recursion_owner, current);
+	}
+	__this_cpu_inc(softnet_data.xmit.recursion);
+}
```

# Example BPF, net/core/filter.c



```
@@ -85,6 +85,11 @@
+DEFINE_PER_CPU(struct bpf_run_lock, bpf_run_lock) = {
+        .redirect_lock = INIT_LOCAL_LOCK(redirect_lock),
+};
+EXPORT_PER_CPU_SYMBOL_GPL(bpf_run_lock);
@@ -4205,6 +4210,7 @@ static const struct bpf_func_proto
    bpf_xdp_adjust_meta_proto = {
 void xdp_do_flush(void)
 {
+        guard(local_lock_nested_bh)(&bpf_run_lock.redirect_lock);
         __dev_flush();
         __cpu_map_flush();
         __xsk_map_flush();
```

# Example BPF supports redirect

```
--- a/drivers/net/ethernet/amazon/ena/ena_netdev.c
+++ b/drivers/net/ethernet/amazon/ena/ena_netdev.c
@@ -385,6 +385,7 @@ static int ena_xdp_execute(struct ena_ring *rx_ring,
     struct xdp_buff *xdp)
         if (!xdp_prog)
                 goto out;

+        guard(local_lock_nested_bh)(&bpf_run_lock.redirect_lock);
         verdict = bpf_prog_run_xdp(xdp_prog, xdp);

         switch (verdict) {
```

# Example BPF driver, no redirect

```
--- a/drivers/net/ethernet/cavium/thunder/nicvf_main.c
+++ b/drivers/net/ethernet/cavium/thunder/nicvf_main.c
@@ -554,7 +554,8 @@ static inline bool nicvf_xdp_rx(struct nicvf *nic,
    xdp_prepare_buff(&xdp, hard_start, data - hard_start, len, false);
    orig_data = xdp.data;

-   action = bpf_prog_run_xdp(prog, &xdp);
+   scoped_guard(local_lock_nested_bh, &bpf_run_lock.redirect_lock)
+           action = bpf_prog_run_xdp(prog, &xdp);

    len = xdp.data_end - xdp.data;
    /* Check if XDP program has changed headers */
```

# Example BPF driver, move REDIRECT bits, cpsw_priv.c

```
          return CPSW_XDP_PASS;

- act = bpf_prog_run_xdp(prog, xdp);
- /* XDP prog might have changed packet data and boundaries */
- *len = xdp->data_end - xdp->data;
+ scoped_guard(local_lock_nested_bh, &bpf_run_lock.redirect_lock) {
+         act = bpf_prog_run_xdp(prog, xdp);
+         *len = xdp->data_end - xdp->data;
+         if (act == XDP_REDIRECT) {
+                 if (xdp_do_redirect(ndev, xdp, prog))
+                         goto drop;
+         }
+ }

        switch (act) {
        case XDP_PASS:
```

# Trace force-threaded interrupts preempted, patched

```
irq/38-eno0-2006 D...1 032 softirq_raise: vec=3 [action=NET_RX]
irq/38-eno0-2006 ..s.1 032 softirq_entry: vec=3 [action=NET_RX]
irq/38-eno0-2006 d.H.1 033 irq_handler_entry: irq=35 name=ahci
irq/38-eno0-2006 dNH31 034 sched_wakeup: irq/35-ahci prio=44
irq/38-eno0-2006 d.s21 035 sched_switch: prio=49 R+->irq/35-ahci prio=44

irq/35-ahci-842  D..12 038 softirq_raise: vec=4 [action=BLOCK]
irq/35-ahci-842  ..s.1 039 softirq_entry: vec=4 [action=BLOCK]
irq/35-ahci-842  d.s32 041 sched_wakeup: grep prio=120
irq/35-ahci-842  ..s.1 042 softirq_exit: vec=4 [action=BLOCK]
irq/35-ahci-842  d..2. 043 sched_switch: prio=44 S->irq/38-eno0 prio=49

irq/38-eno0-2006 ..s.1 044 softirq_exit: vec=3 [action=NET_RX]
irq/38-eno0-2006 d..2. 051 sched_switch: prio=49 S->swapper/2 prio=120
```

# Need to look at tw_timer_handler, allocation

```
 struct inet_timewait_sock *inet_twsk_alloc(const struct sock *sk,
{
  struct inet_timewait_sock *tw;
...

  tw = kmem_cache_alloc(sk->sk_prot_creator->twsk_prot->twsk_slab,
                              GFP_ATOMIC);
  if (tw) {
...

          timer_setup(&tw->tw_timer, tw_timer_handler, TIMER_PINNED);
 void tcp_time_wait(struct sock *sk, int state, int timeo)
 {
...
    tw = inet_twsk_alloc(sk, &net->ipv4.tcp_death_row, state);
    if (tw) {
...
        /* tw_timer is pinned, so we need to make sure BH are disabled
         * in following section, otherwise timer handler could run before
         * we complete the initialization.
         */
        local_bh_disable();
        inet_twsk_schedule(tw, timeo);
        /* Linkage updates.
         * Note that access to tw after this point is illegal.
         */
        inet_twsk_hashdance(tw, sk, net->ipv4.tcp_death_row.hashinfo);
        local_bh_enable();
```

## The page_pool is probably safe

- Acquires memory in softirq.
- Returns memory in softirq.
- If preempted by another NAPI instance then it ends up in slowpath.

# Thank you for your attention

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

`<bigeasy@linutronix.de>`