Soft-interruptible softirqs
(or per vector masking)

Frederic Weisbecker <frederic@kernel.org>
Suse
Current state of Softirqs

- Design hasn’t much evolved since early days (I guess)
- Masking: all in one (local_bh_disable(), spin_lock_bh(), ...)
- Execution: vector can’t interrupt another
What (I think) RT wants

• Prevent from softirq delaying latency sensitive tasks

• Ditto with softirq disabled sections (lots of them)

• How
  • Preempt softirqs

  • Soft-interrupt softirqs (eg: HRTIMER vector interrupting NET_RX)

  • Use of priority inheritance
What (I believe) mainline wants

- Some softirq vectors can eat a lot of CPUs, eg: NET_RX, NET_TX, TASKLET, TASKLET_HI, maybe also BLOCK

- Neither starve the rest of the world nor the softirq vectors

- Balance interrupt processing VS threaded processing
  - Interrupts can’t eat the CPU for too long/too often
  - But still need quick handling
Hack

```c
#define SOFTIRQ_NOW_MASK ((1 << HI_SOFTIRQ) | (1 << TASKLET_SOFTIRQ))
static bool ksoftirqd_running(unsigned long pending)
{
    struct task_struct *tsk = __this_cpu_read(ksoftirqd);

    if (pending & SOFTIRQ_NOW_MASK)
        return false;
    return tsk && (tsk->state == TASK_RUNNING) &&
            !__kthread_should_park(tsk);
}
```
RT solution

- Threaded softirqs
- Sleeping spinlocks
- Per-vector ksoftirqd can preempt each other (locking rules concurrency)
- Priority inheritance
RT solution

• Performance issue
  • Threading
  • Sleeping spinlocks

• What if a softirq accesses per CPU values that are only also accessed by other vectors?
  • Shouldn’t need locking in mainline
  • Would need locking with RT code...
Proposed solution to help RT and mainline

- [PATCH 00/32] softirq: Per vector masking v2
  (Article about it: https://lwn.net/Articles/779738/ )

- Allows softirq disabled sections to be soft-interruptible

- local_bh_disable() / spin_lock_bh() variants can mask with per vector granularity
Proposed solution to help RT and mainline

bh = local_bh_disable_mask(BIT(TASKLET_SOFTIRQ));
    bh2 = spin_lock_bh_mask(lock, BIT(NET_RX_SOFTIRQ));
    local_bh_disable();
    [...]  
    local_bh_enable();
    spin_unlock_bh_mask(lock, bh2);
    local_bh_enable_mask(bh));


Partial solution

- Doesn’t solve entirely for RT
  - Vectors can’t be preempted by higher priority task
  - Priority always in the outermost vector

- Doesn’t really solve for mainline
  - Still need a ksoftirqd pending mask to clean up the SOFTIRQ_NOW hack (current work in progress)
Long term work

- Convert all `local_bh_disable()` / `spin_lock_bh()` / `write_lock_bh()` to use appropriate per vector mask.

- `$ git grep spin_lock_bh\( | wc -l 3829
  $ git grep local_bh_disable\( | wc -l 224
  $ git grep write_lock_bh\( | wc -l 299
  $ git grep read_lock_bh\( | wc -l 286

- Lockdep provides runtime mask information but still...
Long term work

- Doesn’t make softirq vectors interruptible by other vectors
- Need to reuse spin_lock_bh_mask() and enable other vectors
Question

• Is it worth proceeding with this patchset?

• If so, how to sell upstream?