Interrupts, Spin Locks, and Preemption

Interrupts

- Process context vs. Interrupt context
  - system calls run in process context – can sleep
  - interrupt handlers run in interrupt context – cannot sleep
- Interrupt handler
  - single interrupt will not nest, so hander need not be reentrant
    - but handler can be interrupted by a different interrupt
  - only time-critical stuff in handlers
    - push rest to bottom half
  - all handlers share one interrupt stack per processor

Spin Locks

- `spin_lock()` / `spin_unlock()`
  - must not lose CPU while holding a spin lock
    - other threads will wait for the lock for a long time
  - spin_lock() prevents kernel preemption by ++preempt_count
    - in uniprocessor, that's all spin_lock() does
  - must NOT call any function that can potentially sleep
    - ex) kmalloc, copy_from_user
  - hardware interrupt is ok unless the interrupt handler may try to lock this spin lock
    - spin lock not recursive: same thread locking twice will deadlock
  - keep the critical section as small as possible
- `spin_lock_irqsave()` / `spin_unlock_irqrestore()`
  - disable all interrupts on local CPU, lock, unlock, restore interrupts to how it was before
  - need to use this version if the lock is something that an interrupt handler may try to acquire
  - no need to worry about interrupts on other CPUs – spin lock will work normally

    - again, no need to spin in uniproc – just ++preempt_count & disable irq
- `spin_lock_irq()` / `spin_unlock_irq()`
    - disable & enable irq assuming it was disabled to begin with
    - should not be used in most cases

Preemption

If `TIF_NEED_RESCHED` is set, preemption occurs by calling `schedule()` in the following cases:

1. Returning to user space:
    - from a system call
    - from an interrupt handler
2. Returning to kernel from an interrupt handler, only if `preempt_count` is zero
3. `preempt_count` just became zero – right after spin_unlock(), for example
4. Thread running in kernel mode calls schedule() itself – blocking syscall, for example

Last updated: 2016–03–31