Interrupt service Routines

Introduction

- A typical declaration of an ISR

  static irqreturn_t intr_handler(int irq, void *dev_id, struct pt_regs *regs)

  - irq: the IRQ line it is servicing
  - dev_id: a generic pointer to the same dev_id given to request_irq()
  - regs: processor registers prior to servicing the interrupt
- Return value
  - IRQ_NONE: ISR detects an interrupt for which its device was not the originator
  - IRQ_HANDLED: Otherwise
- At a minimum, most ISRs need to provide acks to the device that they received the interrupt
- When a line is shared by multiple ISRs, kernel invokes sequentially each registered handler
  - A HW device should have a status register its ISR can check

Why Bottom Half?

- IH (top halves) have following properties (requirements)
  - IH (top half) need to run as quickly as possible
  - IH runs with some (or all) interrupt levels disabled
  - IH are often time-critical and they deal with HW
  - IH do not run in process context and cannot block

- No hard and fast rules exist about what work to perform where
  - Research work needed

- Bottom halves are to defer work later
  - “Later” is often simply “not now”
  - Often, bottom halves run immediately after interrupt returns
  - They run with all interrupts enabled

ISR Design Considerations

1. Critical actions
   - ✓ must be executed immediately following an interrupt.
   - ✓ Irq's disabled while execution.

2. Noncritical actions:
   - ✓ should be performed as quickly as possible but with enabled interrupts (they may therefore be interrupted by other system events).

3. Deferrable actions:
   - ✓ not important and need not be implemented in the interrupt handler.
   - ✓ These actions can be delayed until kernel has nothing better to do.

Linux Bottom Halves

- Multiple mechanisms are available for implementing a bottom half
  - softirq, tasklet, work queues
- softirq: (available since 2.3)
  - A set of 32 statically defined bottom halves that can run simultaneously on any processor
    - Even 2 of the same type can run concurrently
  - Used when performance is critical
  - Must be registered statically at compile-time
- tasklet: (available since 2.3)
  - Are built on top of softirqs
  - Two different tasklets can run simultaneously on different processors
    - But 2 of the same type cannot run simultaneously
  - Used most of the time for its ease and flexibility
  - Code can dynamically register tasklets
- work queues: (available since 2.5)
  - Queueing work to later be performed in process context

Softirqs

- Softirqs are rarely used
  - can concurrently run
  - Statically allocated at compile-time
- Related code: kernel/softirq.c

```
struct softirq_action
{
    void   (*action)(struct softirq_action *);   // function to run
    void   *data;                                // data to pass to function
};
static struct softirq_action softirq_vec[32];
```

- In 2.6.30 kernel, only 9 softirqs are used

```
enum
{
    HI_SOFTIRQ=0,          TIMER_SOFTIRQ,
    NET_TX_SOFTIRQ,        NET_RX_SOFTIRQ,
    BLOCK_SOFTIRQ,         TASKLET_SOFTIRQ
};
```

Using Softirqs

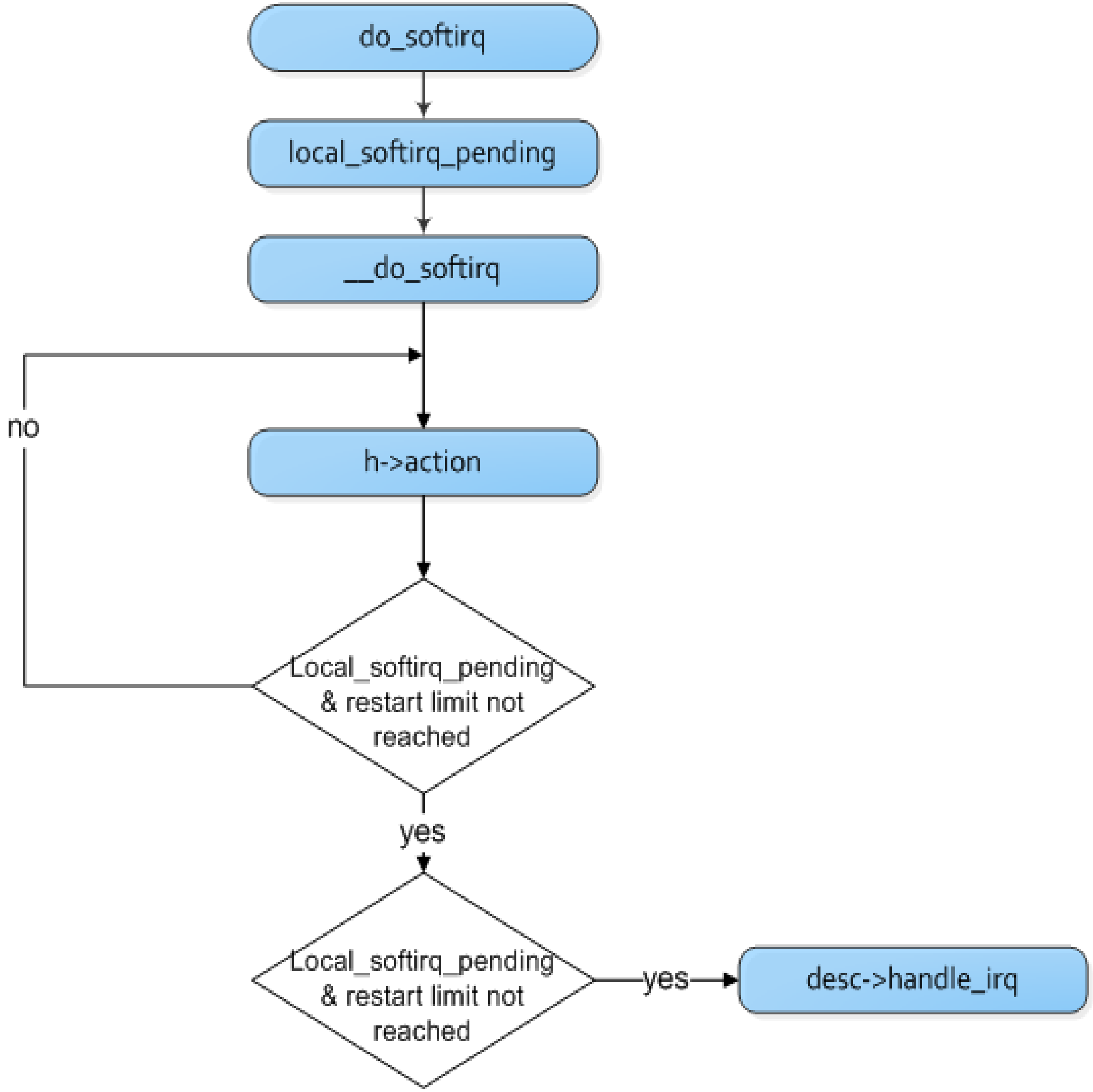
- Index assignment:
  - Before using softirqs, you must declare its index at compile time via an enum
  - Softirqs with lower numerical priority execute first
- Register handler:
  - Softirq handler is registered at run-time via open_softirq()

```
void open_softirq(int nr, void (*action)(struct softirq_action*), void *data)
{
    softirq_vec[nr].data = data;
    softirq_vec[nr].action = action;
}
```

Using Softirq (2/2)

- Raising softirq
  - Call: raise_softirq(NEX_TX_SOFTIRQ), for example
  - Softirqs are often raised from within interrupt handlers
  - When done processing interrupts, kernel invokes do_softirq()
- Sofirqs run with interrupt enabled and cannot sleep

do_softirq



do_softirq
local_softirq_pending
__do_softirq
no
h->action
Local_softirq_pending & restart limit not reached
yes
Local_softirq_pending & restart limit not reached
yes
desc->handle_irq

ksoftirqd

- Most commonly, kernel processes softirqs on return from handling an interrupt
  - In interrupt context
- However, softirqs may be raised at very high rates
  - Sometimes, they reactivate themselves
  - It may lead to starvation of user programs
- Kernel solution
  - When softirqs grow excessively, kernel wakes up a family of kernel threads
  - One thread per processor, named ksoftirqd/n
  - static int ksoftirqd(void * __bind_cpu) [code]

Tasklet Properties

- Guaranteed to run once
  - Once scheduled, a tasklet is guaranteed to be executed once after that
  - An already scheduled but not yet executed tasklet can be rescheduled, but will be executed only once
  - Once a tasklet starts running, it can be rescheduled to run again later
  - Tasklet is strictly serialized (no nesting)
  - Different tasklets can run simultaneously on different CPUs

Tasklet Data Structure

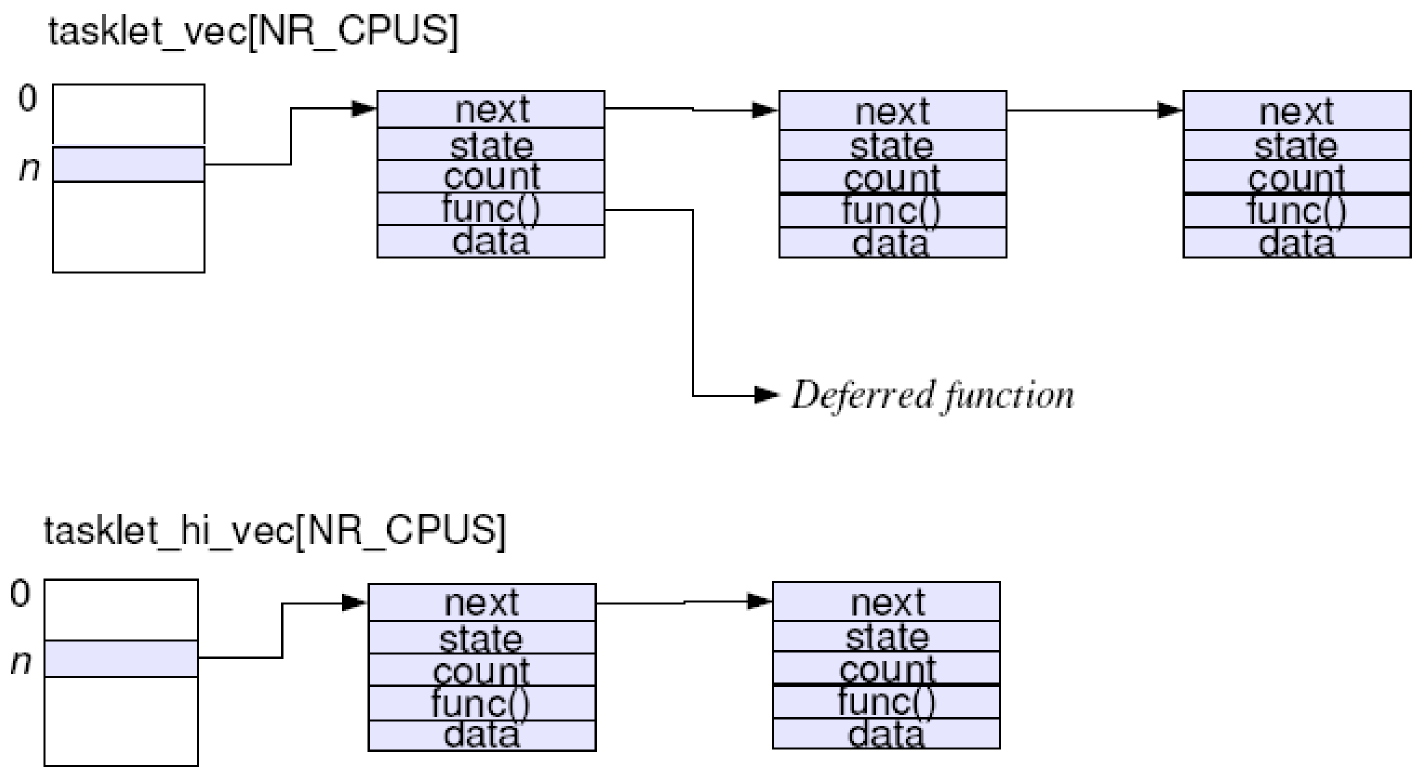
- Deferred function and the argument

- Data Structure: in include/linux/interrupt.h

```
struct tasklet_struct
{
        struct tasklet_struct *next;
        unsigned long state;
        atomic_t count;
        void (*func)(unsigned long);
        unsigned long data;
};
```

- Two tasklet lists per CPU (one higher priority)

Tasklet Lists



Declaring Tasklet

- Define a function that takes one argument
  - void function(unsigned long data)
- DECLARE_TASKLET(name,function,data)
  - Declares a tasklet (of type struct tasklet_struct) with the given name, function, and (unsigned long) data value (as the argument when function is later called).
- DECLARE_TASKLET_DISABLED(name, function,data)
  - Declares a tasklet but with initial state "disabled" -- it can be scheduled but will not be executed until enabled at some future time

Using Tasklet

- To schedule a tasklet to run soon:

  void tasklet_schedule(struct tasklet_struct *t)
  void tasklet_hi_schedule(struct tasklet_struct *t)

  – Both functions: add to the beginning of curresponding tasklet list, and raise softirq

- Disable or enable a tasklet:

  void tasklet_disable(struct tasklet_struct *t);
  void tasklet_enable(struct tasklet_struct *t);

  – Disabled tasklet can be scheduled but won't run -- will remain in the tasklet list until enabled again