IRQs: the Hard, the Soft, the Threaded and the Preemptible

Alison Chaiken

http://she-devel.com
alison@she-devel.com

Embedded Linux Conference Europe
Oct 11, 2016

Example code
Agenda

- Why do IRQs exist?
- About kinds of hard IRQs
- About softirqs and tasklets
- Differences in IRQ handling between RT and non-RT kernels
- Studying IRQ behavior via kprobes, event tracing, mpstat and eBPF
- Detailed example: when does NAPI take over for eth IRQs?
Sample questions to be answered

- What's all stuff in /proc/interrupts anyway?
- What are IPIs and NMIs?
- Why are atomic operations expensive?
- Why are differences between mainline and RT for softirqs?
- What is 'current' task while in interrupt context?
- When do we switch from individual hard IRQ processing to NAPI?
Interrupt handling: a brief pictorial summary

Top half: the hard IRQ

Bottom half: the soft IRQ
Why do we need interrupts at all?

- IRQs allow devices to notify the kernel that they require maintenance.
- Alternatives include
  - polling (servicing devices at a pre-configured interval);
  - traditional IPC to user-space drivers.
- Even a single-threaded RTOS or a bootloader needs a system timer.
Interrupts in Das U-boot

- For ARM, minimal IRQ support:
  - clear exceptions and reset timer (e.g., arch/arm/lib/interrupts_64.c or arch/arm/cpu/armv8/exceptions.S)

- For x86, interrupts are serviced via a stack-push followed by a jump (arch/x86/cpu/interrupts.c)
  - PCI has full-service interrupt handling (arch/x86/cpu/irq.c)
Interrupts in RTOS: Xenomai/ADEOS IPIPE

Figure 2: Adeos’ interrupt pipe.

From Adeos website, covered by GFDL
Zoology of IRQs

- Hard versus soft versus tasklets
- Level- vs. edge-triggered
- Local vs. global
- System vs. device
- Maskable vs. non-maskable
- Shared or not
- Multiple interrupt controllers per SOC

'cat /proc/interrupts' or 'mpstat -A'
ARM IPIs, from arch/arm/kernel/smp.c

void handle_IPI(int ipinr, struct pt_regs *regs) {
    switch (ipinr) {
        case IPI_TIMER:
            tick_receive_broadcast();
        case IPI_RESCHEDULE:
            scheduler_ipi();
        case IPI_CALL_FUNC:
            generic_smp_call_function_interrupt();
        case IPI_CPU_STOP:
            ipi_cpu_stop(cpu);
        case IPI_IRQ_WORK:
            irq_work_run();
        case IPI_COMPLETION:
            ipi_complete(cpu);
    }
}

Handlers are in kernel/sched/core.c

$ # cat /proc/interrupts
What is an NMI?

- A 'non-maskable' interrupt related to:
  - HW problem: parity error, bus error, watchdog timer expiration . . .
  - also used by perf

```c
/* non-maskable interrupt control */
#define NMICR_NMIF 0x0001 /* NMI pin interrupt flag */
#define NMICR_WDIF 0x0002 /* watchdog timer overflow */
#define NMICR_ABUSERR 0x0008 /* async bus error flag */
```

From arch/arm/mn10300/include/asm/intctl-reg.h
x86's Infamous System Management Interrupt

- SMI jumps out of kernel into System Management Mode
  - controlled by System Management Engine (Skochinsky)
- Identified as security vulnerability by Invisible Things Lab
- Traceable via hw_lat detector (sort of)

[RFC][PATCH 1/3] tracing: Added hardware latency tracer, Aug 4
From: "Steven Rostedt (Red Hat)" <rostedt@goodmis.org>
The hardware latency tracer has been in the PREEMPT_RT patch for some time. It is used to detect possible SMIs or any other hardware interruptions that the kernel is unaware of. Note, NMIs may also be detected, but that may be good to note as well.
ARM's **Fast Interrupt reQuest**

- An NMI with optimized handling due to dedicated registers.

- Underutilized by Linux drivers.

- Serves as the basis for Android's `fiq_debugger`. 

IRQ 'Domains' Correspond to Different INTC's

CONFIG_IRQ_DOMAIN_DEBUG:

This option will show the mapping relationship between hardware irq numbers and Linux irq numbers. The mapping is exposed via debugfs in the file "irq_domain_mapping".
Example: i.MX6 General Power Controller

Unmasked IRQs can wakeup sleeping power domains.
Threaded IRQs in RT kernel

ps axl | grep irq

with both RT and non-RT kernels.

Handling IRQs as kernel threads in RT allows priority and CPU affinity to be managed individually.

Mainline kernels have some threaded IRQs in kernel/irq/manage.c:

static irqreturn_t irq_forced_thread_fn(struct irq_desc *desc, struct irqaction *action)
{
    ret = action->thread_fn(action->irq, action->dev_id);
    irq_finalize_oneshot(desc, action);
}

Why are atomic operations more expensive?

`arch/arm/include/asm/atomic.h`:
static inline void atomic_##op(int i, atomic_t *v) {
raw_local_irq_save(flags);
    v->counter c_op i;
raw_local_irq_restore(flags);
}

`include/linux/irqflags.h`:
#define raw_local_irq_save(flags)  
do {  flags = arch_local_irq_save();  } while (0)

`arch/arm/include/asm/atomic.h`:
/* Save the current interrupt enable state & disable IRQs */
static inline unsigned long arch_local_irq_save(void) {
    ... }

Introduction to softirqs

Tasklet interface  Raised by devices  Kernel housekeeping

In kernel/softirq.c:

```c
const char * const softirq_to_name[NR_SOFTIRQS] = {
    "HI", "TIMER", "NET_TX", "NET_RX", "BLOCK", "BLOCK_IOPOLL",
    "TASKLET", "SCHED", "HRTIMER", "RCU"
};
```

In ksoftirqd, softirqs are serviced in the listed order.
What are tasklets?

- Tasklets are one kind of softirq.
- Tasklets perform deferred work started by IRQs but not handled by other softirqs.
- Examples: crypto, USB, DMA.
- More latency-sensitive drivers (sound, PCI) are part of tasklet_hi_vec.
- Number of softirqs is capped; any driver can create a tasklet.
- `tasklet_hi_schedule()` or `tasklet_schedule` are called directly by ISR.
```bash
[alison@sid ~]$ sudo mpstat -l SCPU

Linux **4.1.0-rt17**+ (sid) 05/29/2016 **_x86_64_**(4 CPU)

<table>
<thead>
<tr>
<th>CPU</th>
<th>HI/s</th>
<th>TIMER/s</th>
<th>NET_TX/s</th>
<th>NET_RX/s</th>
<th>BLOCK/s</th>
<th>TASKLET/s</th>
<th>SCHED/s</th>
<th>HRTIMER/s</th>
<th>RCU/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.03</td>
<td>249.84</td>
<td>0.00</td>
<td>0.11</td>
<td>19.96</td>
<td>0.43</td>
<td>238.75</td>
<td>0.68</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>249.81</td>
<td>0.38</td>
<td>1.00</td>
<td>38.25</td>
<td>1.98</td>
<td>236.69</td>
<td>0.53</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.02</td>
<td>249.72</td>
<td>0.19</td>
<td>0.11</td>
<td>53.34</td>
<td>3.83</td>
<td>233.94</td>
<td>1.44</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.59</td>
<td>249.72</td>
<td>0.01</td>
<td>2.05</td>
<td>19.34</td>
<td>2.63</td>
<td>234.04</td>
<td>1.72</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Linux **4.6.0+** (sid) 05/29/2016 **_x86_64_**(4 CPU)

<table>
<thead>
<tr>
<th>CPU</th>
<th>HI/s</th>
<th>TIMER/s</th>
<th>NET_TX/s</th>
<th>NET_RX/s</th>
<th>BLOCK/s</th>
<th>TASKLET/s</th>
<th>SCHED/s</th>
<th>HRTIMER/s</th>
<th>RCU/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.26</td>
<td>16.13</td>
<td>0.20</td>
<td>0.33</td>
<td>40.90</td>
<td>0.73</td>
<td>9.18</td>
<td>0.00</td>
<td>19.04</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>9.45</td>
<td>0.00</td>
<td>1.31</td>
<td>14.38</td>
<td>0.61</td>
<td>7.85</td>
<td>0.00</td>
<td>17.88</td>
</tr>
<tr>
<td>2</td>
<td>0.01</td>
<td>15.38</td>
<td>0.00</td>
<td>0.20</td>
<td>0.08</td>
<td>0.29</td>
<td>13.21</td>
<td>0.00</td>
<td>16.24</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>9.77</td>
<td>0.00</td>
<td>0.05</td>
<td>0.15</td>
<td>0.00</td>
<td>8.50</td>
<td>0.00</td>
<td>15.32</td>
</tr>
</tbody>
</table>

Linux **4.1.18-rt17-00028-g8da2a20** (vpc23) 06/04/16 **_armv7l_**(2 CPU)

<table>
<thead>
<tr>
<th>CPU</th>
<th>HI/s</th>
<th>TIMER/s</th>
<th>NET_TX/s</th>
<th>NET_RX/s</th>
<th>BLOCK/s</th>
<th>TASKLET/s</th>
<th>SCHED/s</th>
<th>HRTIMER/s</th>
<th>RCU/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>999.72</td>
<td>0.18</td>
<td>9.54</td>
<td>0.00</td>
<td>89.29</td>
<td>191.69</td>
<td>261.06</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>999.35</td>
<td>0.00</td>
<td>16.81</td>
<td>0.00</td>
<td>15.13</td>
<td>126.75</td>
<td>260.89</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Linux **4.7.0** (nitrogen6x) 07/31/16 **_armv7l_**(4 CPU)

<table>
<thead>
<tr>
<th>CPU</th>
<th>HI/s</th>
<th>TIMER/s</th>
<th>NET_TX/s</th>
<th>NET_RX/s</th>
<th>BLOCK/s</th>
<th>TASKLET/s</th>
<th>SCHED/s</th>
<th>HRTIMER/s</th>
<th>RCU/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>2.84</td>
<td>0.50</td>
<td>40.69</td>
<td>0.00</td>
<td>0.38</td>
<td>2.78</td>
<td>0.00</td>
<td>3.03</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>89.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.64</td>
<td>0.00</td>
<td>46.22</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>16.59</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.23</td>
<td>0.00</td>
<td>3.05</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>10.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.25</td>
<td>0.00</td>
<td>1.45</td>
</tr>
</tbody>
</table>
```
Two paths by which softirqs run

- Hard IRQ
  - raises softirq
  - local_bh_enable()
    - do_current_softirqs() (RT)
    - or
      - __do_softirq()

- system management thread
  - raises softirq
  - run_ksoftirqd()
    - __do_softirq();
    - exhausts timeslice?

Related demo and sample code
Case 0: Run softirqs at exit of a hard IRQ

**RT (4.6.2-rt5)**

```
local_bh_enable();
__local_bh_enable();
do_current_softirqs();
while (current->softirqs_raised) {
  i = __ffs(current->softirqs_raised);
do_single_softirq(i);
}
handle_softirq();
```

*Run softirqs raised in the current context.*

**non-RT (4.6.2)**

```
local_bh_enable();
do_softirq();
__do_softirq();
handle_pending_softirqs();
while ((softirq_bit = ffs(pending))) {
  handle_softirq();
}
```

*Run all pending softirqs up to MAX_IRQ_RESTART.*
Case 1: Scheduler runs the rest from ksoftirqd

RT (4.6.2-rt5)
run_ksoftirqd();
do_current_softirqs()
[ where current == ksoftirqd ]

h = softirq_vec;
while ((softirq_bit = ffs(pending)))
{
    h += softirq_bit - 1;
    h->action(h);
}

non-RT (4.6.2)
run_ksoftirqd();
do_softirq();
__do_softirq();
Two ways of entering softirq handler

4.7.-rc1:
[11661.191187] [<ffffffffa0236c36>] ? e1000e_poll+0x126/0xa70 [e1000e]
[11661.191197] [<ffffffff81d4d16e>] ? net_rx_action+0x52e/0xcd0
[11661.191206] [<ffffffff82123a4c>] ? __do_softirq+0x15c/0x5ce
[11661.191215] [<ffffffff811274f3>] ? irq_exit+0xa3/0xd0
[11661.191222] [<ffffffff821235c2>] ? do_IRQ+0x62/0x110
[11661.191230] [<ffffffff82121782>] ? common_interrupt+0x82/0x82

4.6.2-rt5:
[ 6937.393805] [<ffffffffa0478d36>] ? e1000e_poll+0x126/0xa70 [e1000e]
[ 6937.393808] [<ffffffff818c778b>] ? check_preemption_disabled+0xab/0x240
[ 6937.393815] [<ffffffff81d54ebe>] ? net_rx_action+0x53e/0xc90
[ 6937.393824] [<ffffffff81132a98>] ? do_current_softirqs+0x488/0xc30
[ 6937.393831] [<ffffffff81132615>] ? do_current_softirqs+0x5/0xc30
[ 6937.393836] [<ffffffff81133332>] ? __local_bh_enable+0xf2/0x1a0
[ 6937.393840] [<ffffffff81223c91>] ? irq_forced_thread_fn+0x91/0x140
[ 6937.393845] [<ffffffff81223570>] ? irq_thread+0x170/0x310
[ 6937.393848] [<ffffffff81223c00>] ? irq_finalize_oneshot.part.6+0x4f0/0x4f0
[ 6937.393853] [<ffffffff81223d40>] ? irq_forced_thread_fn+0x140/0x140
[ 6937.393857] [<ffffffff81223400>] ? irq_thread_check_affinity+0xa0/0xa0
[ 6937.393862] [<ffffffff8117782b>] ? kthread+0x12b/0x1b0

kick off soft IRQ

hard IRQ
Summary of softirq execution paths

Case 0: Behavior of local_bh_enable() differs significantly between RT and mainline kernel.

Case 1: Behavior of ksoftirqd itself is mostly the same (note discussion of ktimersoftd below).
What is 'current'?

include/asm-generic/current.h:
#define get_current() (current_thread_info()->task)
#define current get_current()

arch/arm/include/asm/thread_info.h:
static inline struct thread_info *current_thread_info(void)
{ return (struct thread_info *) (current_stack_pointer & ~(THREAD_SIZE - 1));
}

arch/x86/include/asm/thread_info.h:
static inline struct thread_info *current_thread_info(void)
{ return (struct thread_info *)(current_top_of_stack() - THREAD_SIZE);}

In do_current_softirqs(), current is the threaded IRQ task.
What is 'current'? part 2

arch/arm/include/asm/thread_info.h:
/*
 * how to get the current stack pointer in C
 */
register unsigned long current_stack_pointer asm("sp");

arch/x86/include/asm/thread_info.h:
static inline unsigned long current_stack_pointer(void) {
    unsigned long sp;
#ifdef CONFIG_X86_64
    asm("mov %%rsp,%0" : "=g" (sp));
#else
    asm("mov %%esp,%0" : "=g" (sp));
#endif
    return sp;
}
Q.: When do system-management softirqs get to run?
Introducing systemd-irqd!!†

†As suggested by Dave Anders
Do timers, scheduler, RCU ever run as part of do_current_softirqs?

Examples:

--every jiffy,
    raise_softirq_irqoff(HRTIMER_SOFTIRQ);

-- scheduler_ipi() for NOHZ calls
    raise_softirq_irqoff(SCHED_SOFTIRQ);

-- rcu_bh_qs() calls
    raise_softirq(RCU_SOFTIRQ);

These softirqs then run when ksoftirqd is current.
**Demo: kprobe on do_current_softirqs() for RT kernel**

- At Github
- Counts calls to do_current_softirqs() from ksoftirqd and from a hard IRQ context.
- Tested on 4.4.4-rt11 with Boundary Devices' Nitrogen i.MX6.

Output showing what task of 'current_thread' is:

```
[ 52.841425] task->comm is ksoftirqd/1
[ 70.051424] task->comm is ksoftirqd/1
[ 70.171421] task->comm is ksoftirqd/1
[ 105.981424] task->comm is irq/43-2188000.
[ 165.260476] task->comm is ksoftirqd/1
[ 165.261406] task->comm is irq/43-2188000.
[ 225.321529] task->comm is irq/43-2188000.
```
Softirqs can be pre-empted with PREEMPT_RT

include/linux/sched.h:

    struct task_struct {
    #ifdef CONFIG_PREEMPT_RT_BASE
        struct rcu_head put_rcu;
        int softirq_nestcnt;
        unsigned int softirqs_raised;
    #endif
    
    #endif
    }


How IRQ masking works

arch/arm/include/asm/irqflags.h:
#define arch_local_irq_enable arch_local_irq_enable
static inline void arch_local_irq_enable(void)
{   asm volatile(
       "cpsie i
        :: "memory", "cc"); }

arch/arm64/include/asm/irqflags.h:
static inline void arch_local_irq_enable(void)
{   asm volatile(
       "msr      daifclr, #2    // arch_local_irq_enable"
        :: "memory"); }

arch/x86/include/asm/irqflags.h:
static inline notrace void arch_local_irq_enable(void)
{   native_irq_enable(); }
static inline void native_irq_enable(void)
{   asm volatile("sti": ::"memory"); }
RT-Linux headache: 'softirq starvation'

- Timer, scheduler and RCU softirqs may not get to run.
- Events that are triggered by timer interrupt won't happen.
- RCU will report a stall.
- *Example:* main event loop in userspace did not run due to missed timer ticks.

Reference: “Understanding a Real-Time System” by Rostedt, slides and video
(partial) solution: ktimersoftd

Author: Sebastian Andrzej Siewior <bigeasy@linutronix.de>
Date:   Wed Jan 20 16:34:17 2016 +0100
softirq: split timer softirqs out of ksoftirqd

The softirqd runs in -RT with SCHED_FIFO (prio 1) and deals mostly with
timer wakeup which can not happen in hardirq context. The prio has been
risen from the normal SCHED_OTHER so the timer wakeup does not happen
too late.

With enough networking load it is possible that the system never goes idle
and schedules ksoftirqd and everything else with a higher priority. One of the
tasks left behind is one of RCU's threads and so we see stalls and eventually
run out of memory. This patch moves the TIMER and HRTIMER softirqs out
of the `ksoftirqd` thread into its own `ktimersoftd`. The former can now run
SCHED_OTHER (same as mainline) and the latter at SCHED_FIFO due to
the wakeups. [ . . . ]
ftrace produces a copious amount of output
Investigating IRQs with eBPF: IOvisor and bcc

- BCC - Tools for BPF-based Linux analysis
- BCC tools/ and examples/ illustrate simple interfaces to kprobes and uprobes.
- Documentation is outstanding.
- BCC tools are a convenient way to study low-frequency events dynamically.
- Based on insertion of snippets into running kernel using Clang Rewriter JIT.
eBPF, IOvisor and IRQs: limitations

- JIT compiler for eBPF is currently available for the x86-64, arm64, and s390 architectures.
- No stack traces unless CONFIG_FRAME_POINTER=y
- Requires recent versions of kernel, LLVM and Clang
- bcc/src/cc/export/helpers.h:
  ```c
  #ifdef __powerpc__
  [. . .]
  #elif defined(__x86_64__)
  [. . .]
  #else
  #error "bcc does not support this platform yet"
  #endif
  ```
**bcc tip**

- The kernel source must be present on the host where the probe runs.
- `/lib/modules/$(uname -r)/build/include/generated` must exist.
- To switch between kernel branches and continue quickly using bcc:
  - run `mrproper; make config; make`
  - `make` need only to populate `include/generated` in kernel source before bcc again becomes available.
  - `make headers_install` as non-root user
Get latest version of clang by compiling from source
(or from Debian Sid)

$ git clone http://llvm.org/git/llvm.git
$ cd llvm/tools
$ git clone --depth 1 http://llvm.org/git/clang.git
$ cd ..; mkdir build; cd build
$ cmake .. -DLLVM_TARGETS_TO_BUILD="BPF;X86"
$ make -j $(getconf _NPROCESSORS_ONLN)

from samples/bpf/README.rst
Example: NAPI: changing the bottom half
Quick NAPI refresher

The problem:

“High-speed networking can create thousands of interrupts per second, all of which tell the system something it already knew: it has lots of packets to process.”

The solution:

“Interrupt mitigation . . . NAPI allows drivers to run with (some) interrupts disabled during times of high traffic, with a corresponding decrease in system load.”

The implementation:

Poll the driver and drop packets without processing in the NIC if the polling frequency necessitates.
Example: i.MX6 FEC RGMII NAPI turn-on

static irqreturn_t fec_enet_interrupt(int irq, void *dev_id)
[ . . . ]
    if ((fep->work_tx || fep->work_rx) && fep->link) {
        if (napi_schedule_prep(&fep->napi)) {
            /* Disable the NAPI interrupts */
            writel(FEC_ENET_MII, fep->hwp + FEC_IMASK);
            __napi_schedule(&fep->napi);
        }
    }
}
Example: i.MX6 FEC RGMII NAPI turn-off

static int fec_enet_rx_napi(struct napi_struct *napi, int budget){

[ . . . ]

    pkts = fec_enet_rx(ndev, budget);

    if (pkts < budget) {
        napi_complete(napi);
        writel(FEC_DEFAULT_IMASK, fep->hwp + FEC_IMASK);
    }

}

netif_napi_add(ndev, &fep->napi, fec_enet_rx_napi, NAPI_POLL_WEIGHT);

    Interrupts are re-enabled when budget is not consumed.
Using existing tracepoints

- function_graph tracing causes a lot of overhead.
- How about napi_poll tracer in /sys/kernel/debug/events/napi?
  - Fires constantly with any network traffic.
  - Displays no obvious change in behavior when actual NAPI packet-handling path is triggered.
Investigation on ARM:

kprobe with 4.6.2-rt5;
ping-flood and simultaneously

while true;
do  scp /boot/vmlinuz-4.5.0  root@172.17.0.1:/tmp;
done
“In general, you can install a probe \textit{anywhere} in the kernel. In particular, you can probe interrupt handlers.”

Takeaway: \textbf{not} limited to existing tracepoints!
root@nitrogen6x:~# insmod 4.6.2/kp_raise_softirq_irqoff.ko
[ 1749.935955] Planted kprobe at 8012c1b4
[ 1749.936088] Internal error: Oops - undefined instruction: 0 [#1]
PREENOPT SMP ARM
[ 1749.936109] Modules linked in: kp_raise_softirq_irqoff(+)
[ 1749.936116] CPU: 0 PID: 0 Comm: swapper/0 Not tainted 4.6.2
[ 1749.936119] Hardware name: Freescale i.MX6 Quad/DualLite
[ 1749.936131] PC is at __raise_softirq_irqoff+0x0/0xf0
[ 1749.936144] LR is at __napi_schedule+0x5c/0x7c
[ 1749.936766] Kernel panic - not syncing: Fatal exception in interrupt
patch samples/kprobes/kprobe_example.c

/* For each probe you need to allocate a kprobe structure */
static struct kprobe kp = {
    .symbol_name= "__raise_softirq_irqoff_ksoft",
};

/* kprobe post_handler: called after the probed instruction is executed */
static void handler_post(struct kprobe *p, struct pt_regs *regs,unsigned long flags)
{
    unsigned id = smp_processor_id();
    /* change id to that where the eth IRQ is pinned */
    if (id == 0) {
        pr_info("Switched to ethernet NAPI.\n");
        pr_info("post_handler: p->addr = 0x%p, pc = 0x%lx,"
            " lr = 0x%lx, cpsr = 0x%lx\n",
            p->addr, regs->ARM_pc, regs->ARM_lr, regs->ARM_cpsr);
    }
}
Watching net_rx_action() switch to NAPI

alison@laptop:~# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- samples/kprobes/ modules

root@nitrogen6x:~# insmod samples/kprobes/kp_ksoft.ko

root@nitrogen6x:~# dmesg | tail
[ 6548.644584] Planted kprobe at 8003344

root@nitrogen6x:~# dmesg | grep post_handler

root@nitrogen6x:~#

. . . . . . . Start DOS attack . . . Wait 15 seconds . . .

root@nitrogen6x:~# dmesg | tail
[ 6548.644584] Planted kprobe at 80033440
[ 6617.858101] pre_handler: p->addr = 0x80033440, pc = 0x80033444, lr = 0x80605ff0, cpsr = 0x20070193
[ 6617.858104] Switched to ethernet NAPI.
Counting activation of two softirq execution paths

static struct kprobe kp = {
    .symbol_name= "do_current_softirqs",
};

if (raised == NET_RX_SOFTIRQ) {
    ti = current_thread_info();
    task = ti->task;
    if (chatty)
        pr_debug("task->comm is %s\n", task->comm);

    if (strstr(task->comm, "ksoftirq"))
        p->ksoftirqd_count++;
    if (strstr(task->comm, "irq/"))
        p->local_bh_enable_count++;
}

modprobe kp_do_current_softirqs chatty=1
The Much Easier Way:

BCC on x86_64 with 4.6.2-rt5 and Clang-3.8; ping-flood and simultaneously

while true;
do  scp /boot/vmlinuz-4.5.0 root@172.17.0.1:/tmp;
done
Catching the switch from Eth IRQs to NAPI on x86_64

root $ ./stackcount.py e1000_receive_skb
Tracing 1 functions for "e1000_receive_skb"
^C

e1000_receive_skb
 e1000e_poll
 net_rx_action
 do_current_softirqs
 run_ksoftirqd
 smpboot_thread_fn
 kthread
 ret_from_fork
 1

COUNTS

e1000_receive_skb
 e1000e_poll
 net_rx_action
 do_current_softirqs
 _local_bh_enable
 irq_forced_thread_fn
 irq_thread
 kthread
 ret_from_fork
 26469

NAPI polling: running from ksoftirqd, not from hard IRQ handler.

Normal behavior: packet handler runs immediately after eth IRQ, in its context.
Summary

- IRQ handling involves a 'hard', fast part or 'top half' and a 'soft', slower part or 'bottom half.'
- Hard IRQs include arch-dependent system features plus software-generated IPIs.
- Soft IRQs may run directly after the hard IRQ that raises them, or at a later time in ksoftirqd.
- Threaded, preemptible IRQs are a salient feature of RT Linux.
- The management of IRQs, as illustrated by NAPI's response to DOS, remains challenging.
- If you can use bcc and eBPF, you should be!
Acknowledgements

Thanks to Sebastian Siewor, Brenden Blanco, Brendan Gregg, Steven Rostedt and Dave Anders for advice and inspiration.
Useful Resources

- NAPI docs
- Documentation/kernel-per-CPU-kthreads
- Brendan Gregg's blog
- Tasklets and softirqs discussion at KLDP wiki
- #iovisor at OFTC IRC
- Alexei Starovoitov's 2015 LLVM Microconf slides
“Preemption Disabled Tracing

When interrupts are disabled, events from devices and timers and even inter-processor communication is disabled. But the kernel can keep interrupts enabled but disable preemption.”
### ARMv7 Core Registers

#### Application level view

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
<th>Hyp†</th>
<th>Supervisor</th>
<th>Abort</th>
<th>Undefined</th>
<th>Monitor‡</th>
<th>IRQ</th>
<th>FIQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>R1_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>R2_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>R3_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>R4_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>R5_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>R6_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>R7_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>R8_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R8_fiq</td>
</tr>
<tr>
<td>R9</td>
<td>R9_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R9_fiq</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>R10_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R10_fiq</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>R11_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R11_fiq</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R12_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R12_fiq</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>SP_usr</td>
<td>SP_hyp</td>
<td>SP_svc</td>
<td>SP_abt</td>
<td>SP_und</td>
<td>SP_mon</td>
<td>SP_irq</td>
<td>SP_fiq</td>
</tr>
<tr>
<td>LR</td>
<td>LR_usr</td>
<td></td>
<td>LR_svc</td>
<td>LR_abt</td>
<td>LR_und</td>
<td>LR_mon</td>
<td>LR_irq</td>
<td>LR_fiq</td>
</tr>
<tr>
<td>PC</td>
<td>PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### System level view

<table>
<thead>
<tr>
<th>APSR</th>
<th>CPSR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPSR_hyp</td>
</tr>
<tr>
<td></td>
<td>ELR_hyp</td>
</tr>
</tbody>
</table>
A.: Softirqs that don't run in context of hard IRQ run “on behalf of ksoftirqd”

static inline void ksoftirqd_set_sched_params(unsigned int cpu) {
    /* Take over all but timer pending softirqs when starting */
    local_irq_disable();
    current->softirqs_raised = local_softirq_pending() & ~TIMER_SOFTIRQS;
    local_irq_enable();
}

static struct smp_hotplug_thread softirq_threads = {
    .store       = &ksoftirqd,
    .setup       = ksoftirqd_set_sched_params,
    .thread_should_run = ksoftirqd_should_run,
    .thread_fn    = run_ksoftirqd,
    .thread_comm  = "ksoftirqd/%u",
};
Compare output to source with GDB

[alison@hildesheim linux-4.4.4 (trace_napi)]$ arm-linux-gnueabihf-gdb vmlinux
(gdb) p *__raise_softirq_irqoff_ksoft
$1 = {void (unsigned int)} 0x80033440 <__raise_softirq_irqoff_ksoft>
(gdb) l *(0x80605ff0)
0x80605ff0 is in net_rx_action (net/core/dev.c:4968).
4963        list_splice_tail(&repoll, &list);
4964        list_splice(&list, &sd->poll_list);
4965        if (!list_empty(&sd->poll_list))
4966              __raise_softirq_irqoff_ksoft(NET_RX_SOFTIRQ);
4967
4968        net_rps_action_and_irq_enable(sd);
4969    }