Interrupt Entry Latency Behavior
Analysis of Linux 2.4 vs 2.6

SangBae Lee
Software Laboratories (CTO)
Samsung Electronics Co.Ltd
Preface (1/2)
◆ What is Real-Time?
  – “A real time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred.”
  
  – Donald Gillies

  – “Real time in operating systems:
  The ability of the operating system to provide a required level of service in a bounded response time.”
  
  – POSIX Standard 1003.1

◆ What is Real-Time System?
  – Real-time System이란, 어떤 event가 발생했을 때, 이것을 어떤 정해진 시간(Deadline) 이내에 처리하는 것을 보장하는 system이라고 할 수 있다. 즉, event에 대한 반응성이 빠르고, 중요한 event가 덜 중요한 event보다 먼저 수행되며, event를 놓치는 일이 결코 일어나서는 안 되는 system을 말한다. 이러한 것을 완벽하게 보장(놓치는 경우 error)하는 system을 hard-real-time이라고 하고, event를 가끔 놓쳐도 크게 문제되지 않는(error로 취급되지 않음) system을 soft-real-time이라고 한다.
MontaVista Kernel Configuration for Real Time Feature

<table>
<thead>
<tr>
<th>Preemption Mode</th>
<th>Target System</th>
<th>.config option</th>
<th>MV Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Forced Preemption</td>
<td>Server</td>
<td>CONFIG_PREEMPT_NONE</td>
<td>Pro 3.1: O</td>
</tr>
<tr>
<td>Voluntary Kernel Preemption</td>
<td>Desktop</td>
<td>CONFIG_PREEMPT_VOLUNTARY</td>
<td>Pro 3.1: X</td>
</tr>
<tr>
<td>Preemptible Kernel</td>
<td>Low Latency Desktop</td>
<td>CONFIG_PREEMPT_DESKTOP</td>
<td>Pro 3.1: O</td>
</tr>
<tr>
<td>Complete Preemption</td>
<td>Real Time</td>
<td>CONFIG_PREEMPT_RT</td>
<td>Pro 3.1: X</td>
</tr>
</tbody>
</table>

General Concept of Real-Time

- H/W (Device)의 IRQ(interrupt) Response Time
- Task(Thread/Process) Preemption Time
  - Scheduling time, Context Switching Time
- Task(Thread/Process) Priority & priority inversion
- Fast IPC (Inter Process/Task Communication)
- System Call Response Time & System Call Processing Behavior
Terminology

- **Hard deadline requirement:**
  - missing the deadline is considered an error.

- **Hard real-time system:**
  - system with hard real-time requirements.

- **Interrupt latency:**
  - time passed between interrupt occurrence and activation of interrupt handler.

- **Interrupt masking:**
  - making certain interrupts invisible to the software.

- **Interrupt response time (worst-case):**
  - (worst-case) time passed between interrupt occurrence and either completion of interrupt service routine (ISR) or wake up of dependent task.

- **Jitter – absolute:**
  - deviation of the occurrence of an event (e.g. completion of frame) from expected occurrence.

- **Jitter – relative:**
  - deviation of the interval between two successive occurrences of an event (e.g. completion of frame) from expected interval.
Terminology

- **Preemption:**
  - a running thread or process can be temporarily suspended. The state of the thread or process (including e.g., program counter, and register values) is saved. Until the thread is resumed, it remains runnable (active, ready). When the process or thread is later resumed, the saved state is restored.

- **Real-time requirement:**
  - a requirement on the completion time of a response, generally measured relative to the event that triggered the response.

- **Real-time system:**
  - system with one or more real-time requirements.

- **Response time (worst-case):**
  - (worst-case) time passed between event occurrence and completion of the response to that event. The event may be an interrupt. The response typically involves an interrupt handler and one or more synchronized tasks.

- **Soft deadline:**
  - missing deadlines is sometimes acceptable. Compared to hard deadlines, where there is no reason to consider the value of a late result, the value of a late result for a soft deadline is of interest. The value of the result may, for instance, decrease linearly after the deadline.

- **Soft real-time requirement:**
  - soft deadline, or average-case response time requirement. Note that hard and soft real-time requirements are orthogonal to the temporal granularity that is required. Meeting a soft requirement in the microsecond domain may be more difficult than meeting a hard requirement in the milliseconds domain.

- **Soft real-time system:**
  - system with soft real-time requirements
Recently, most of embedded processor have 32~200 interrupt sources (OMAP5912 – ARM926) in one core. Is all interrupt request from devices handled correctly?
Interrupt Entry Latency

- **Interrupt Entry Latency**: from IRQ request of Device to IRQ entry of kernel

![Diagram of Interrupt Entry Latency and Interrupt Processing Latency (ISR)]

*But, kernel don’t know the exact time that any device requests specific IRQ request. So, we cannot measure interrupt Entry Latency Without additional hardware (ex. Jtag or oscilloscope)*
Using OS Timer to know the time of Device IRQ request

- There is only one device which we can measure interrupt entry latency: OS Timer (osk5912 : Timer2)
- Usually, OS timer use down-counter (but, on MIPS, it is up-counter)
- Down-counter is cleared from user-defined values per every period
- And, OS timer has a lowest priority than other devices.

So, from OS timer behavior, we can expect an Interrupt behavior of any system (interrupt entry latency behavior of other devices)

We can expect System behavior using OS Timer interrupt latency statistically
How to measure Interrupt Entry Latency

**OMAP5912 Example**

- OS Timer : 6MHz clock ➔ 100Hz ➔ 0xEA60 (down-counter initial value)
  - T(A) = (T0−T1) * (10000ns / 0xEA60)
    - Interrupt Latency : IRQ Entry Time : T(A) ➔ Ⓐ
  - T(B) = (T1−T2) * (10000ns / 0xEA60)
    - Interrupt Handler Duration : IRQ Handler Duration Time : T(B) ➔ Ⓑ
Because of higher priority request (ethernet interrupt request), OS timer interrupt was delayed.
**Practical Test Environments**

- **Target**: OSK5912 (192Mhz)
  - Original montavista patch
  - Using NFS root filesystem
  - Stress tool: netperf, SYS_syscall(getpid)
  - Measurement tool: LTT (Linux Trace Toolkit)

<table>
<thead>
<tr>
<th>Kernel Configuration</th>
<th>No stress</th>
<th>Ethernet IRQ Stress</th>
<th>IPC Stress</th>
<th>System Call Stress</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MV pro 4.0 (2.6.10)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Desktop</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Low latency Desktop</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Real time</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td><strong>MV pro 3.1 (2.4.20)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preemption ON</td>
<td>O</td>
<td>O</td>
<td>X*</td>
<td>X*</td>
</tr>
<tr>
<td>Preemption OFF</td>
<td>O</td>
<td>O</td>
<td>X*</td>
<td>X*</td>
</tr>
</tbody>
</table>

*X* : Test 중 LTT Data Lost로 인해 불가능

---

12 / 7
Practical Test: using netperf

- GPIO INT (IRQ 14) will be requested heavily by Netperf
Interrupt Entry Latency Measurement using LTT

- A comparison of Linux 2.4 vs 2.6 Interrupt latency
  - Interrupt entry latency of Linux 2.6 is worse than Linux 2.4

What makes the performance difference between Linux 2.6 and 2.4?
Linux 2.4 is at least 2-times faster than Linux 2.6.
# Summary of LTT test result

<table>
<thead>
<tr>
<th>Kernel Configuration</th>
<th>Best Case</th>
<th>95% samples</th>
<th>Worst Case</th>
<th>비고</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV pro 4.0 (2.6.10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>&lt; 30us</td>
<td>&lt;200us</td>
<td>&lt;350us</td>
<td></td>
</tr>
<tr>
<td>Desktop</td>
<td>&lt; 30us</td>
<td>&lt;250us</td>
<td>&lt;500us</td>
<td></td>
</tr>
<tr>
<td>Low latency Desktop</td>
<td>&lt; 30us</td>
<td>&lt;320us</td>
<td>&lt;700us</td>
<td></td>
</tr>
<tr>
<td>Real time</td>
<td>&lt; 30us</td>
<td>&lt;200us</td>
<td>&lt;400us</td>
<td></td>
</tr>
<tr>
<td>Preemption ON</td>
<td>&lt; 10us</td>
<td>&lt;15us</td>
<td>&lt;40us</td>
<td></td>
</tr>
<tr>
<td>Preemption OFF</td>
<td>&lt; 10us</td>
<td>&lt;15us</td>
<td>&lt;40us</td>
<td></td>
</tr>
<tr>
<td>MV pro 3.1 (2.4.20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Source Analysis (2/4) : details

MM Pro 3.1 `asm_do_IRQ()` sequence

1. `asm_do_IRQ()`
2. `do_IRQ()`
3. `_do_IRQ()`
4. `action->handler()`
5. `do_softirq()`
6. `in_interrupt()`
7. `h->action(h)`
8. `wakeup_softirq()`
9. `ksoftirqd_task()`
10. `wake_up_process()`
asmlinkage notrace void asm_do_IRQ(unsigned int irq, struct pt_regs *regs) {
    struct irqdesc *desc = irq_desc + irq;
    ltt_ev_irq_entry(irq, !(user_mode(regs)));

    /*
     * Some hardware gives randomly wrong interrupts. Rather
     * than crashing, do something sensible.
     */
    if (irq >= NR_IRQS)
        desc = &bad_irq_desc;

    interrupt_overhead_start();
    irq_enter();
    spin_lock(&irq_controller_lock);
    desc->handle(irq, desc, regs);

    /*
     * Now re-run any pending interrupts.
     */
    if (!list_empty(&irq_pending))
        do_pending_irqs(regs);

    spin_unlock(&irq_controller_lock);
    irq_exit();
    latency_check();
    ltt_ev_irq_exit();}
asm linkage void asm_do_IRQ(int irq, struct pt_regs *regs)
{
    irq = fixup_irq(irq);

#define CONFIG_LATENCY
interrupt_overhead_start();
#undef CONFIG_LATENCY

TRACE_IRQ_ENTRY(irq, !(user_mode(regs)));
preempt_lock_start(99);

/*
 * Some hardware gives randomly wrong interrupts. Rather
 * than crashing, do something sensible.
 */
if (irq < NR_IRQS) {
    int cpu = smp_processor_id();
    irq_enter(cpu, irq);
    spin_lock(&irq_controller_lock);
    do_IRQ(irq, regs);

    /*
     * Now re-run any pending interrupts.
     */
    if (!list_empty(&irq_pending))
        do_pending_irqs(regs);

    spin_unlock(&irq_controller_lock);
    irq_exit(cpu, irq);

    if (softirq_pending(cpu))
        do_softirq();
    TRACE_IRQ_EXIT();
    preempt_lock_stop();
    return;
}

irq_err_count += 1;
printk(KERN_ERR "IRQ: spurious interrupt %d\n", irq);
TRACE_IRQ_EXIT();
preempt_lock_stop();
return;
}
Source Analysis result: no difference

```
vecโทรIRG:

__irq_usr:
  save Register
  alignment_trap
  zero_fp
  get_current_task
  CONFIG_PREEMPT
    preempt_count++
    get_irqnr_and_base

asm_do_IRQ()

__irq_svc:
  save Register
  CONFIG_PREEMPT
    get_current_task
    preempt_count++
    get_irqnr_and_base
```

```
vecโทรIRG:

__irq_usr:
  save Register
  LINUX_ARMARCH_ < 6
  alignment_trap
  zero_fp
  CONFIG_PREEMPT
    get_thread_info
    preempt_count++
    get_irqnr_and_base
```

```
vecโทรIRG:

__irq_svc:
  save Register
  CONFIG_PREEMPT
    get_thread_info
    preempt_count++
    CONFIG_CRITICAL_IRQSOFF_TIMING
    trace_irqs_off()
    get_irqnr_and_base
```

```
vecโทรIRG:

__irq_usr:
  save Register
  zero_fp
  get_current_task
  CONFIG_PREEMPT
    preempt_count++
    get_irqnr_and_base
```

```
vecโทรIRG:

__irq_svc:
  save Register
  CONFIG_PREEMPT
    get_current_task
    preempt_count++
    get_irqnr_and_base
```
What makes the difference between Linux 2.4 and 2.6

- We decided to verify the overhead of LTT (measurement tool)
  - 2.4: /dev/trace
  - 2.6: /mnt/relayfs

There is a big difference to save traced data
We need another way
OK, let's make a simple light-weight tracer
(Zoom-in Tracer)
ZI (Zoom In) Tracer

Arch/arm/kernel/irq.c

asmlinkage void asm_do_IRQ() {
    ZI_IRQ_ENTRY(irq);
    ZI_IRQ_EXIT(irq);
}

include/linux/zi_trace.h

#define ZI_IRQ_ENTRY(tracing_point)
zi_register_event(((ZI_GROUP_IRQ << 24) | ZI_TYPE_IRQ_ENT), tracing_point);

#define ZI_IRQ_EXIT(tracing_point)
zi_register_event(((ZI_GROUP_IRQ << 24) | ZI_TYPE_IRQ_EXIT), tracing_point);

Kernel/zi_trace.c

int zi_register_event(unsigned int flag, int tracing_point) {
    unsigned int group_id;
    struct irq_event_desc *bufptr;
    int event_id;
    struct timeval tv;

    if(zi_tracer_running != 0xF628)
        return -EBUSY;

    /* check if duration is expired or not */
    do_gettimeofday(&tv);
    if((tv.tv_sec - trace_start.tv_sec) >= zi_trace_duration)
        zi_tracer_running = 0;
    printk(" trace done. duration is expired (%d)\n", zi_trace_duration);
    return 0;
}

.....................
A Comparison – with LTT

Linux 2.4.20–LTT

Linux 2.6.10–LTT
A comparison – with ZI

- minimum Interrupt Entry Latency is 3us~30us

There is no big difference between Linux 2.6 and Linux 2.4
Real Target Analysis (DTV)

- **DTV System Interrupt Entry Latency**
  - X240-MIPS (ATI)
    - CPU frequency 264.06 MHz
    - Calibrating delay loop... 263.78 BogoMIPS

<after basic kernel booting>

```
# cat /proc/interrupts
CPU0
  2:  0  XILLEON Int usb-ohci
  64: 6370  XILLEON Int timer
  100:  56  XILLEON Int serial
  106: 2616  XILLEON Int eth0
```

<after full application booting>

```
# cat /proc/interrupts
CPU0
  2:  0  XILLEON Int usb-ohci
  9:  0  XILLEON Int atx2200
  10:  0  XILLEON Int atx2200
  64: 36736  XILLEON Int timer
  65: 121572  XILLEON Int atx2200
  99: 1791  XILLEON Int serial
  100: 27372  XILLEON Int serial
  101: 85245  XILLEON Int atx2200
  106: 39057  XILLEON Int eth0
  115: 4412  XILLEON Int atx2200
  116: 9735  XILLEON Int atx2200
  118: 0  XILLEON Int atx2200
  120: 0  XILLEON Int atx2200
  121: 0  XILLEON Int atx2200
  122: 0  XILLEON Int atx2200
  123: 0  XILLEON Int atx2200
```
Q & A

Thank you