

Interrupt Entry Latency Behavior Analysis of Linux 2.4 vs 2.6

SangBae Lee

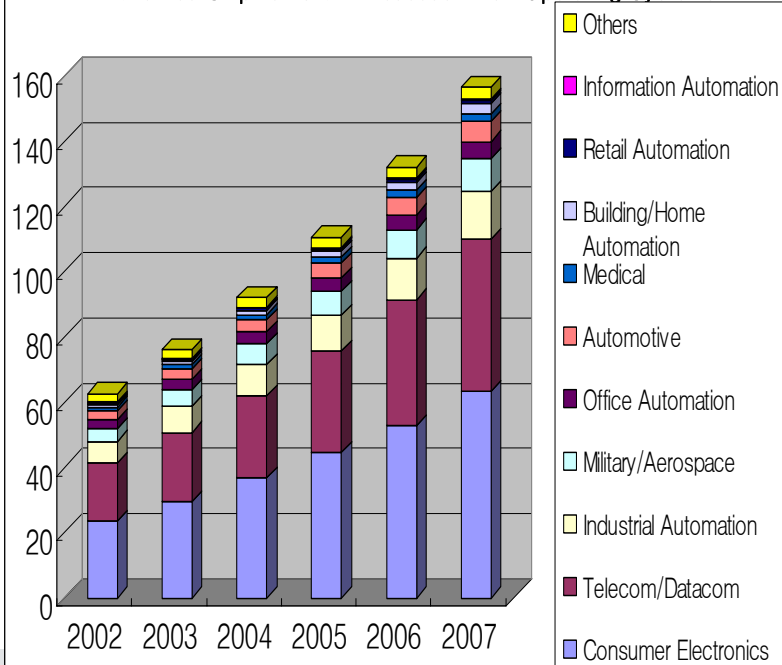
Software Laboratories (CTO)

Samsung Electronics Co.Ltd

Preface (1/2)



Worldwide Shipments of Embedded Linux Operating Systems



Preface (2/2)

◆ 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).

MV Linux Kernel Mode & Real Time 특성

◆ MontaVista Kernel Configuration for Real Time Feature

Preemption Mode	Target System	.config option	MV Support	
			Pro 3.1	Pro 4.0
No Forced Preemption	Server	CONFIG_PREEMPT_NONE	O	O
Voluntary Kernel Preemption	Desktop	CONFIG_PREEMPT_VOLUNTARY	X	O
Preemptible Kernel	Low Latency Desktop	CONFIG_PREEMPT_DESKTOP	O	O
Complete Preemption	Real Time	CONFIG_PREEMPT_RT	X	O

◆ 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.

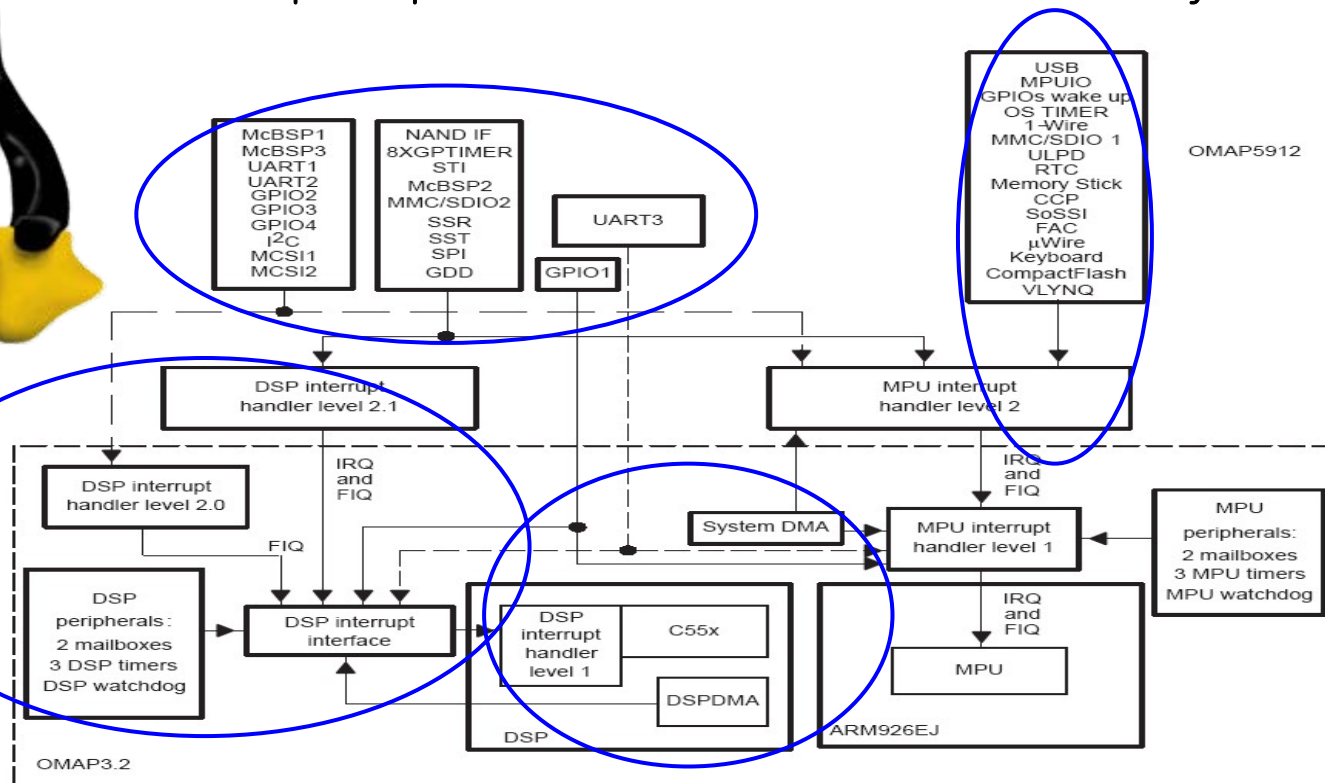
◆ Soft real-time system:

- system with soft real-time requirements

- Recently, most of embedded processor have 32~200 interrupt sources (OMAP5912 – ARM926) in one core

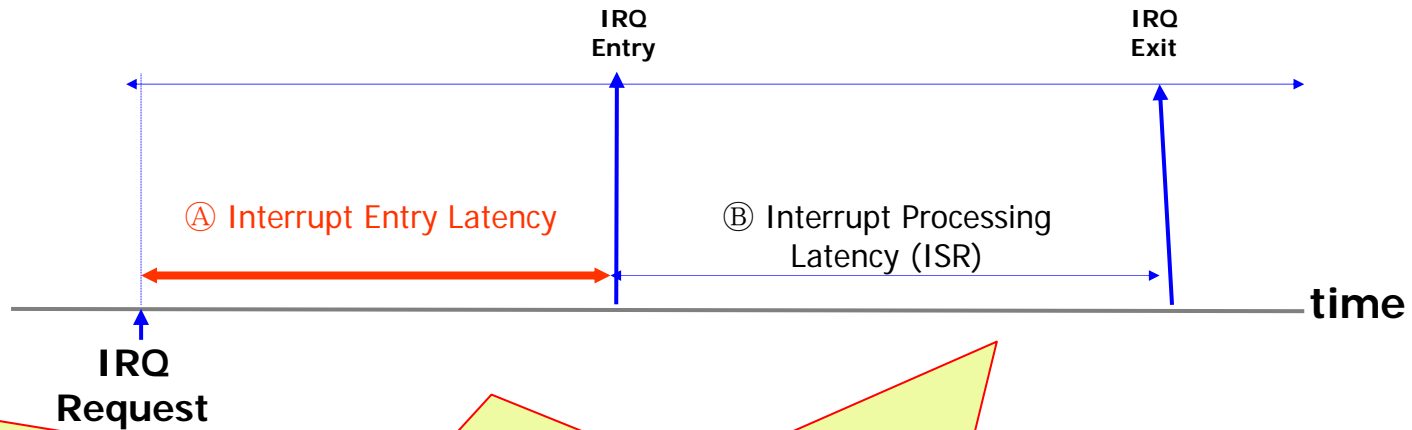


Is all interrupt request from devices handled correctly?



Interrupt Entry Latency

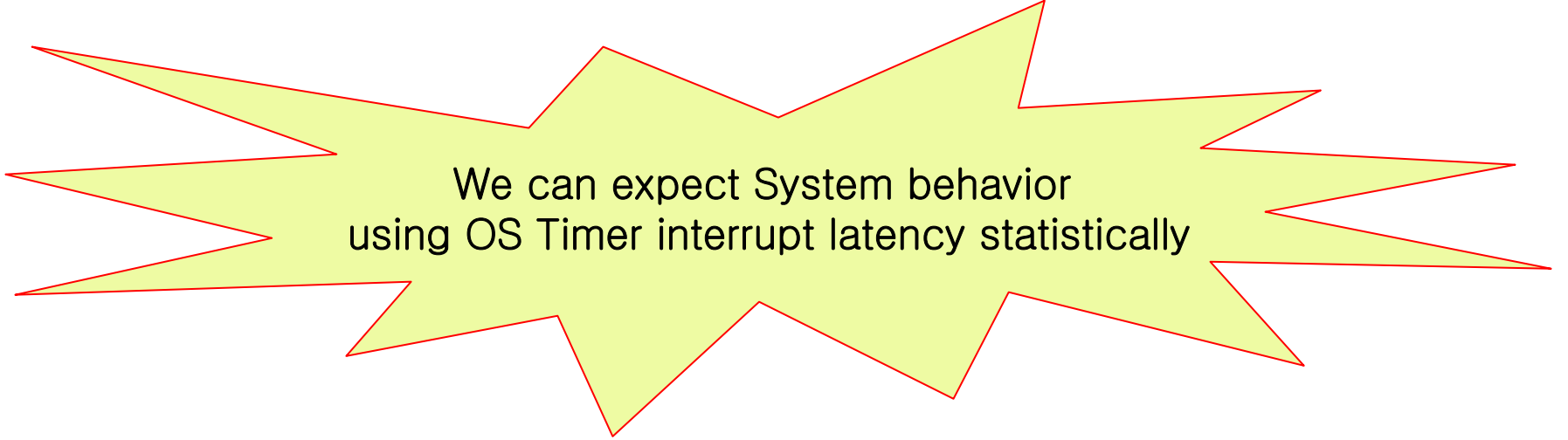
- ◆ Interrupt Entry Latency : from IRQ request of Device to IRQ entry of kernel



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)



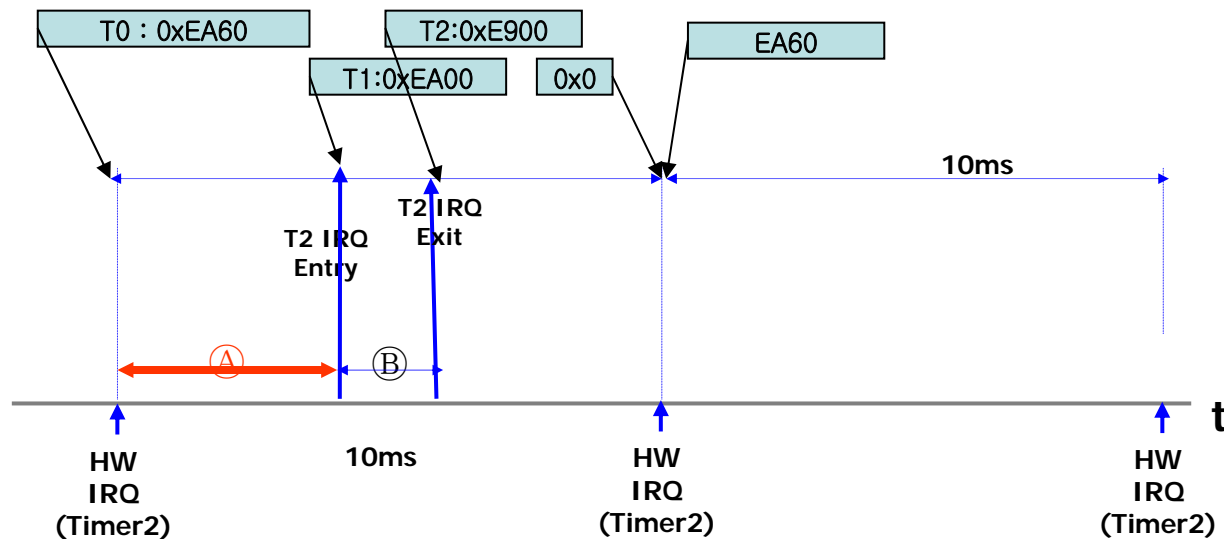
We can expect System behavior
using OS Timer interrupt latency statistically

How to measure Interrupt Entry Latency

◆ OMAP5912 Example

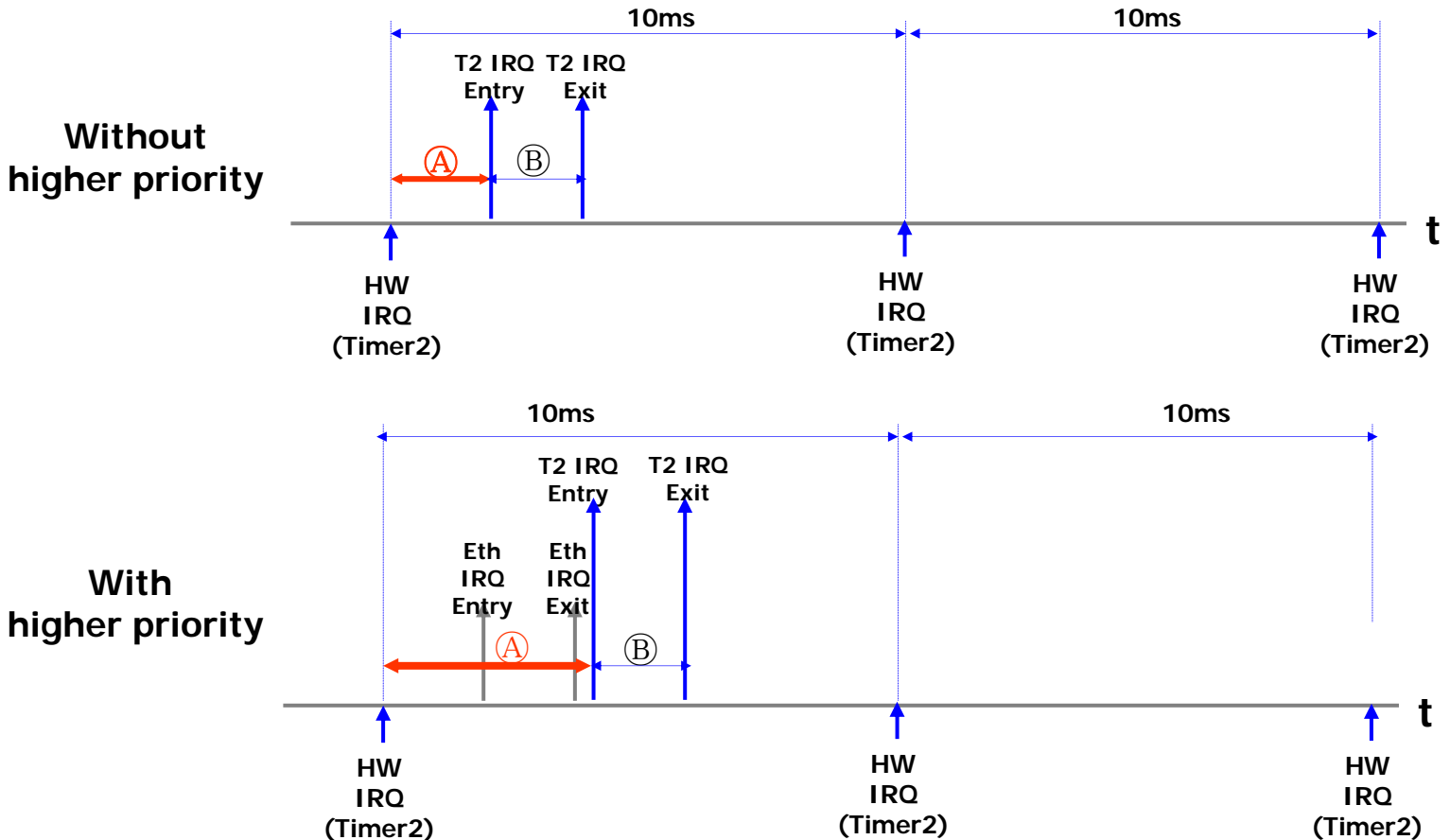
– OS Timer : 6MHz clock \rightarrow 100Hz \rightarrow 0xEA60 (down-counter initial value)

- $T(A) = (T0 - T1) * (10000ns / 0xEA60)$
 ☑ Interrupt Latency : IRQ Entry Time : $T(A) \rightarrow \textcircled{A}$
- $T(B) = (T1 - T2) * (10000ns / 0xEA60)$
 ☑ Interrupt Handler Duration : IRQ Handler Duration Time : $T(B) \rightarrow \textcircled{B}$



Delayed handling example of OS timer IRQ

- Because of higher priority request (ethernet interrupt request), OS timer interrupt was delayed



Practical Test Environments

◆ Target : OSK5912 (192Mhz)

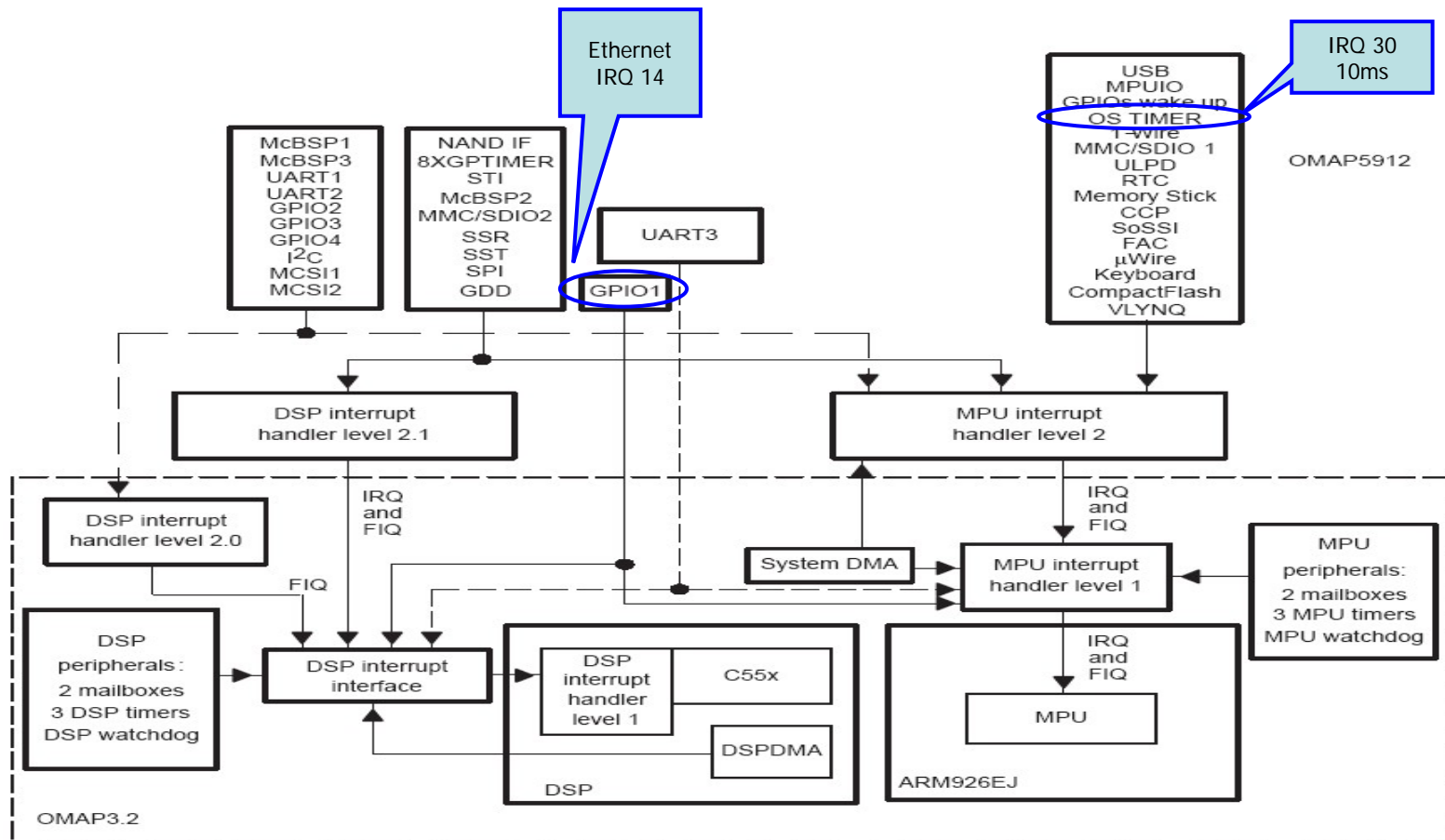
- Original montavista patch
- Using NFS root filesystem
- Stress tool : netperf, SYS_syscall(getpid)
- Measurement tool : LTT (Linux Trace Toolkit)

	Kernel Configuration	No stress	Ethernet IRQ Stress	IPC Stress	System Call Stress
MV pro 4.0 (2.6.10)	Server	O	O	O	O
	Desktop	O	O	O	O
	Low latency Desktop	O	O	O	O
	Real time	O	O	O	O
MV pro 3.1 (2.4.20)	Preemption ON	O	O	X*	X*
	Preemption OFF	O	O	X*	X*

X* : Test 중 LTT Data Lost로 인해 분석 불가

Practical Test : using netperf

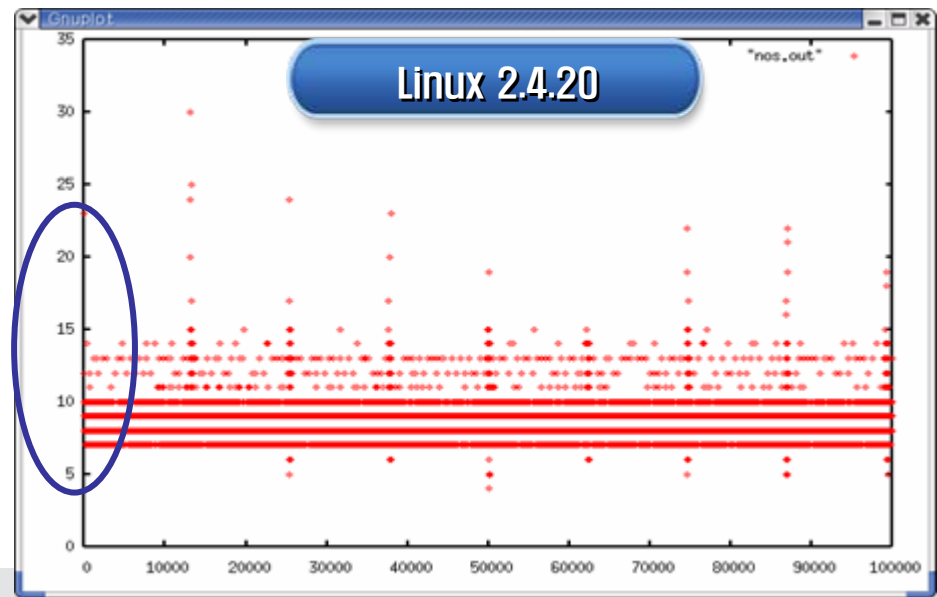
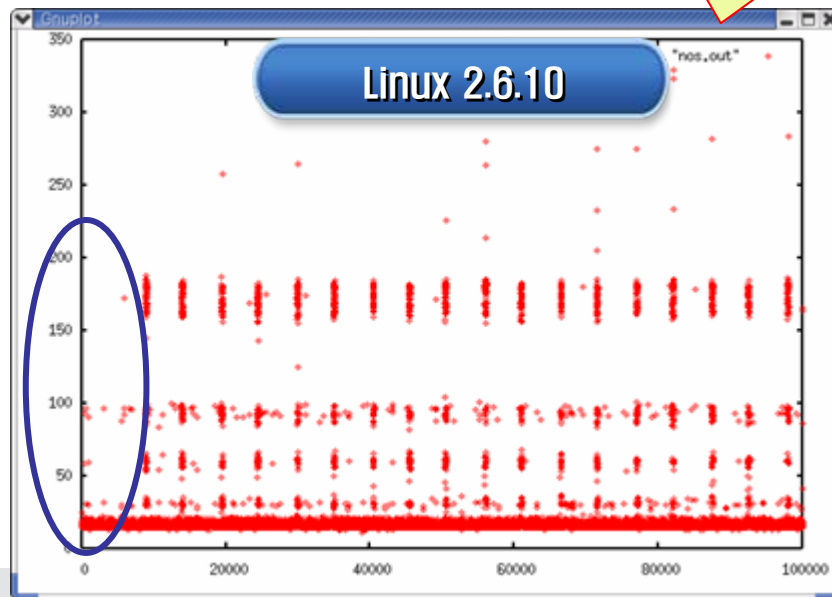
- ◆ GPIO INT(IRQ 14) will be requested heavily by Netperf



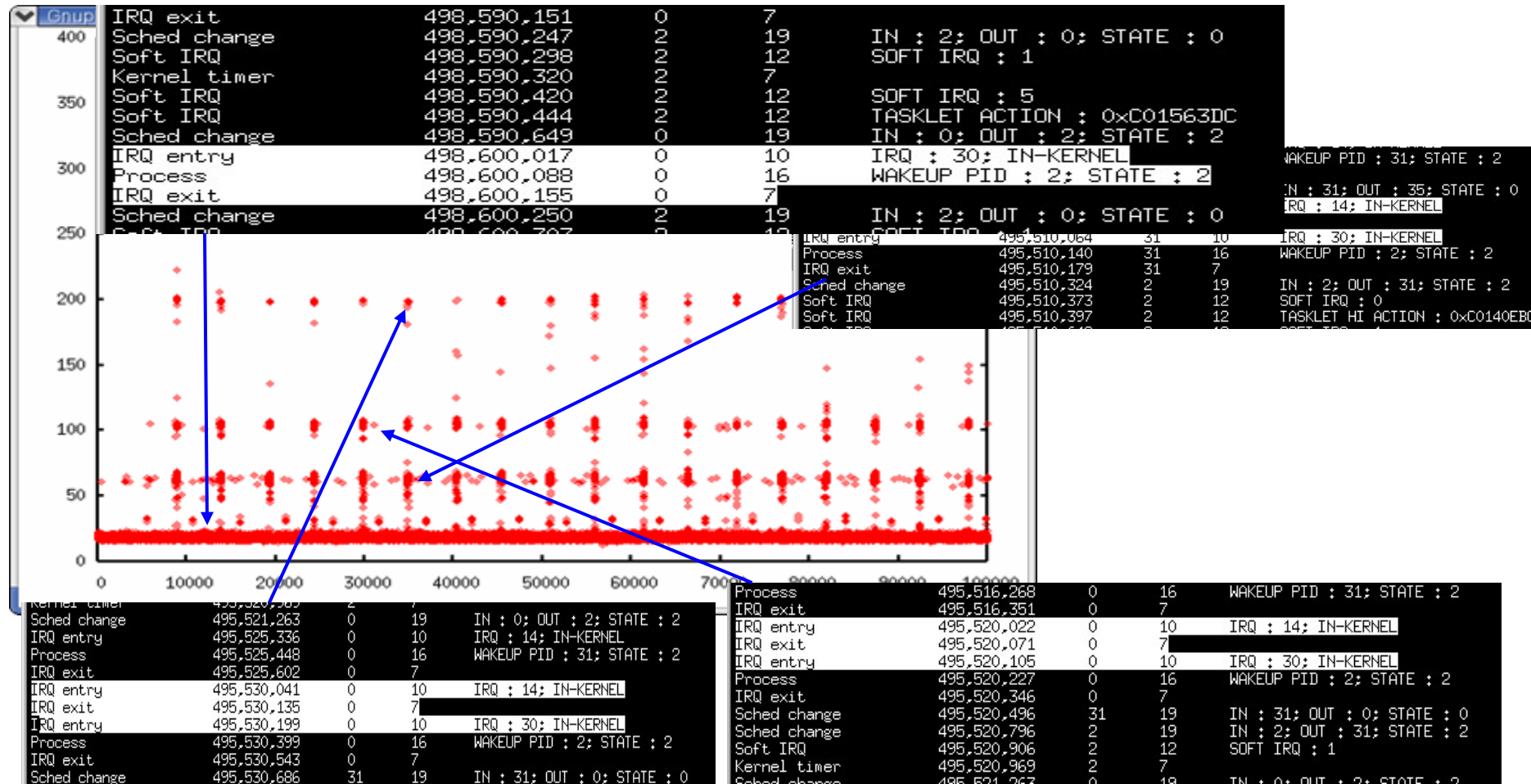
Interrupt Entry Latency Measurement using LTT

- ◆ 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.



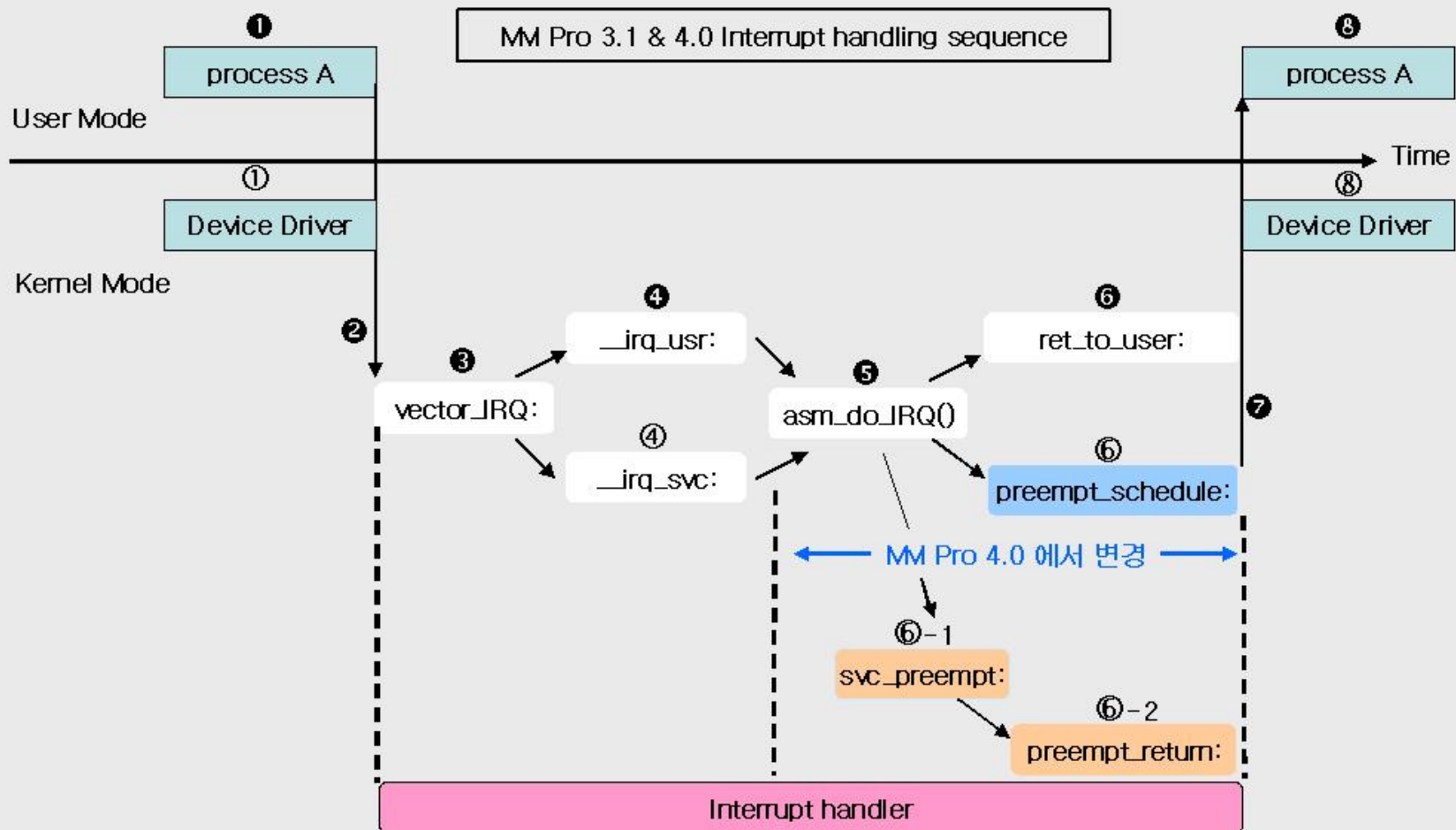
Detailed analysis of IRQ Entry Latency Pattern



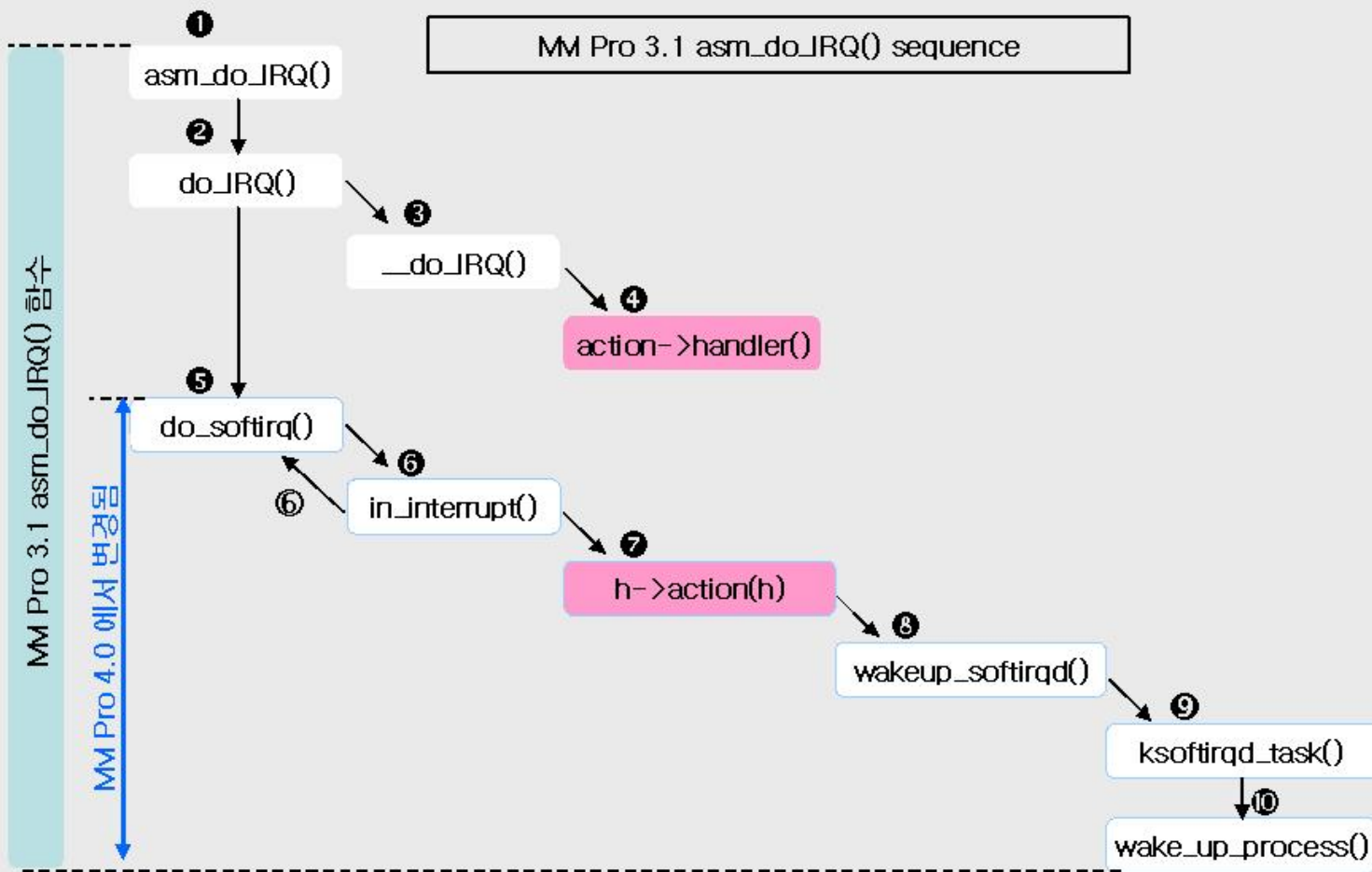
Summary of LTT test result

	Kernel Configuration	Best Case	95% samples	Worst Case	비고
MV pro 4.0 (2.6.10)	Server	< 30us	<200us	<350us	
	Desktop	< 30us	<250us	<500us	
	Low latency Desktop	< 30us	<320us	<700us	
	Real time	< 30us	<200us	<400us	
MV pro 3.1 (2.4.20)	Preemption ON	< 10us	<15us	<40us	
	Preemption OFF	< 10us	<15us	<40us	

Source Analysis(1/4) :Linux 2.4 vs 2.6



Source Analysis (2/4) : details



Source analysis(3/4) : Linux2.6 asm_do_IRQ()

```
asmmlinkage 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();
}
```

Source Analysis(4/4) : Linux 2.4 asm_do_IRQ()

```
asmlinkage void asm_do_IRQ(int irq, struct pt_regs *regs)
{
    irq = fixup_irq(irq);

#ifdef CONFIG_ILATENCY
    interrupt_overhead_start();
#endif
    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);

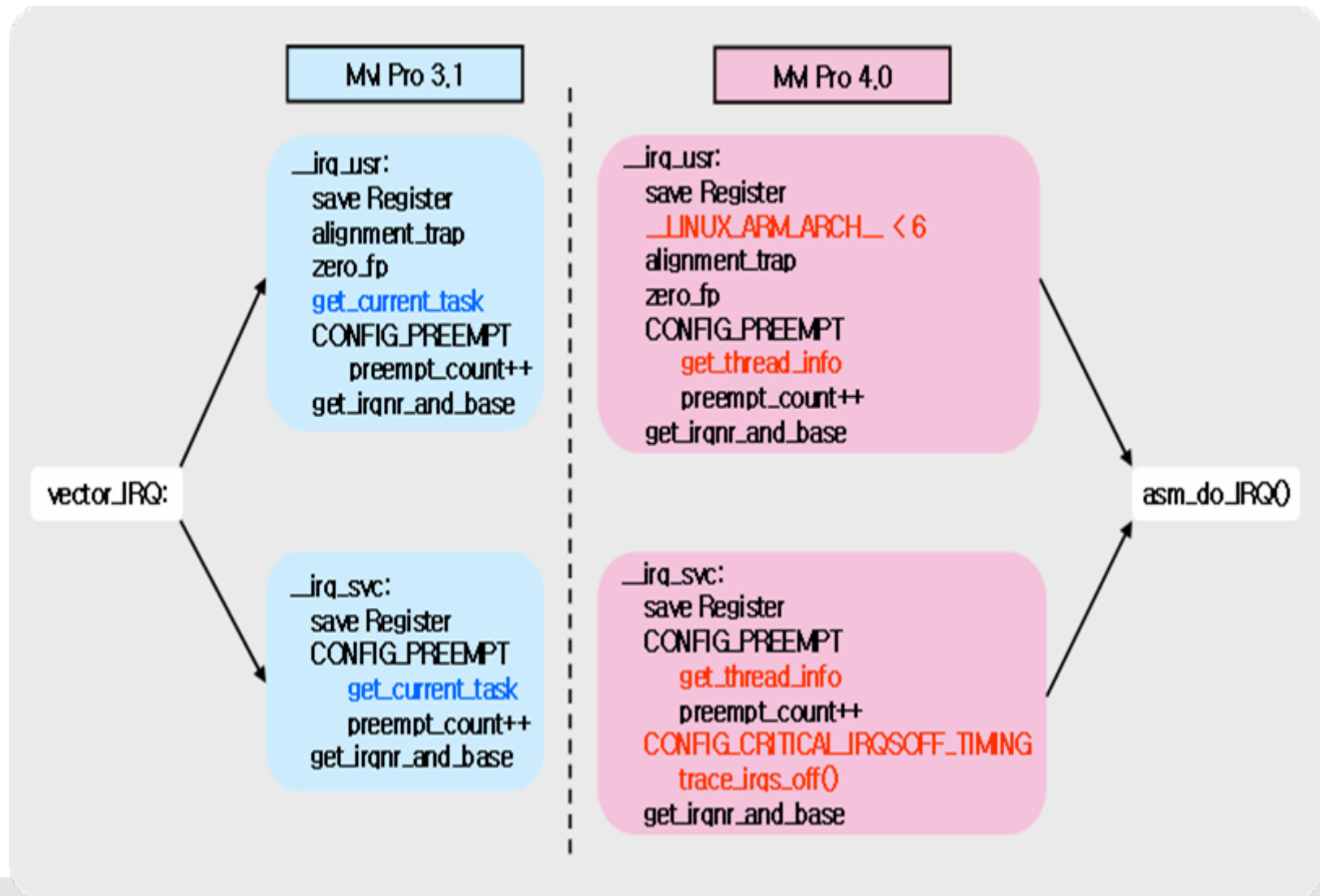
        /*
         * Now re-run any pending interrupts.
         */
        if (!list_empty(&irq_pending))
            do_pending_irqs(regs);

        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



What makes the difference between Linux 2.4 and 2.6

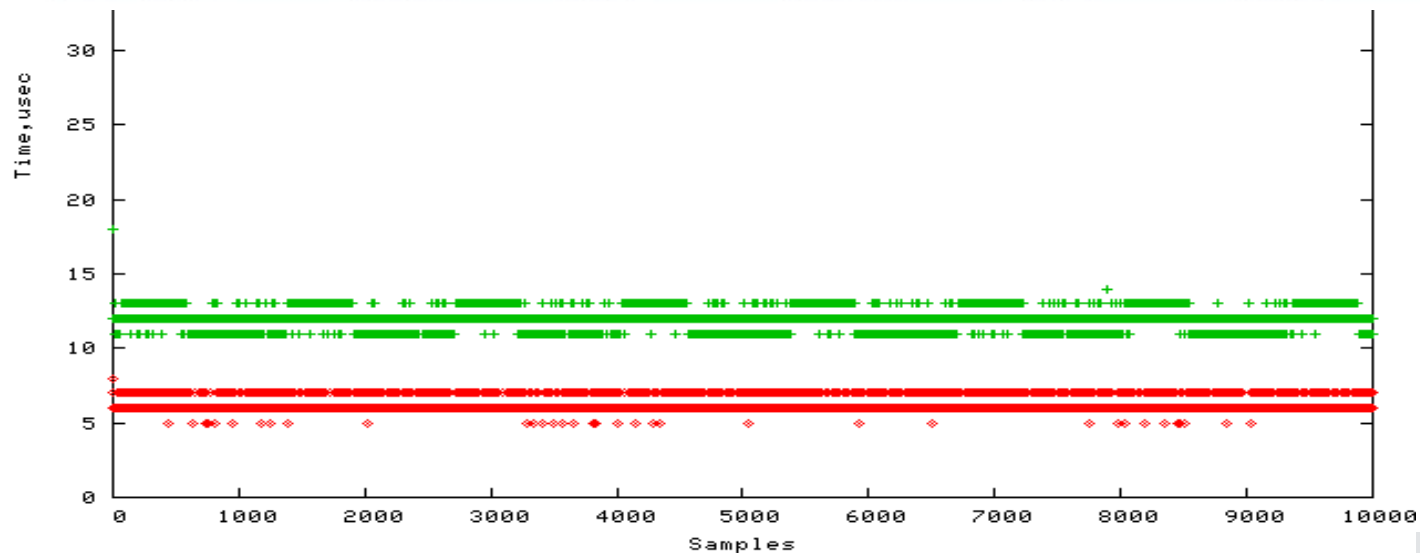
- ◆ We decided to verify the overhead of LTT (measurement tool)
 - 2.4 : /dev/trace
 - 2.6 : /mnt/relayfs

There is a big difference to save traced data

LTT Overhead Analysis

We need another way
OK, let's make a simple light-weight tracer
(Zoom-in Tracer)

	ltt_log_event()		ltt_log_std_formatted_event()		ltt_log_raw_event()	
	2.4.20-mvl31	2.6.10-mvl40	2.4.20-mvl31	2.6.10-mvl40	2.4.20-mvl31	2.6.10-mvl40
Mean	6.13	12.1	22.72	27.58	9.20	14.96
Min	5.00	11.00	21.00	25.00	8.00	13.00
Max	65.00	246.00	296.00	4441.00	99.00	242.00



ZI(Zoom In) Tracer

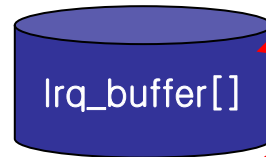
proc interface

/proc/zi/rawdata
/proc/zi/entry_time
/proc/zi/control
/proc/zi/duration

Arch/arm/kernel/irq.c

```
asmlinkage void asm_do_IRQ()
{
    ZI_IRQ_ENTRY(irq);

    ZI_IRQ_EXIT(irq);
}
```



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) &&
        ((tv.tv_usec - trace_start.tv_usec) > 0) ) { /* stop tracer */
        zi_tracer_running = 0;
        printk(" trace done. duration is expired (%d)\n", zi_trace_duration);
        return 0;
    }

    .....
```

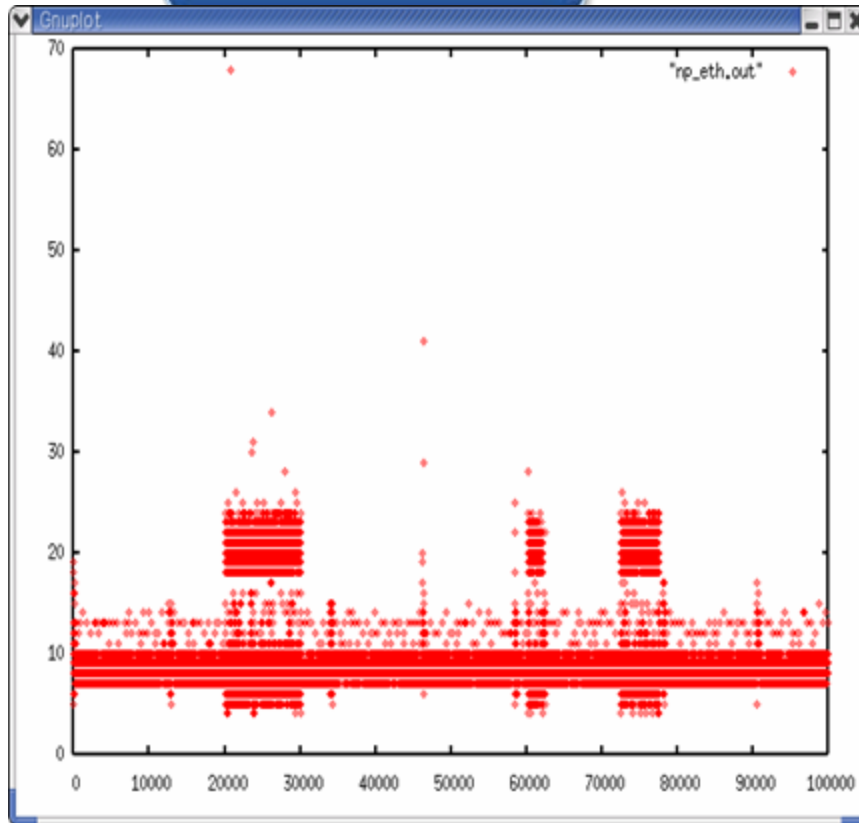
include/linux/zi_trace.h

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

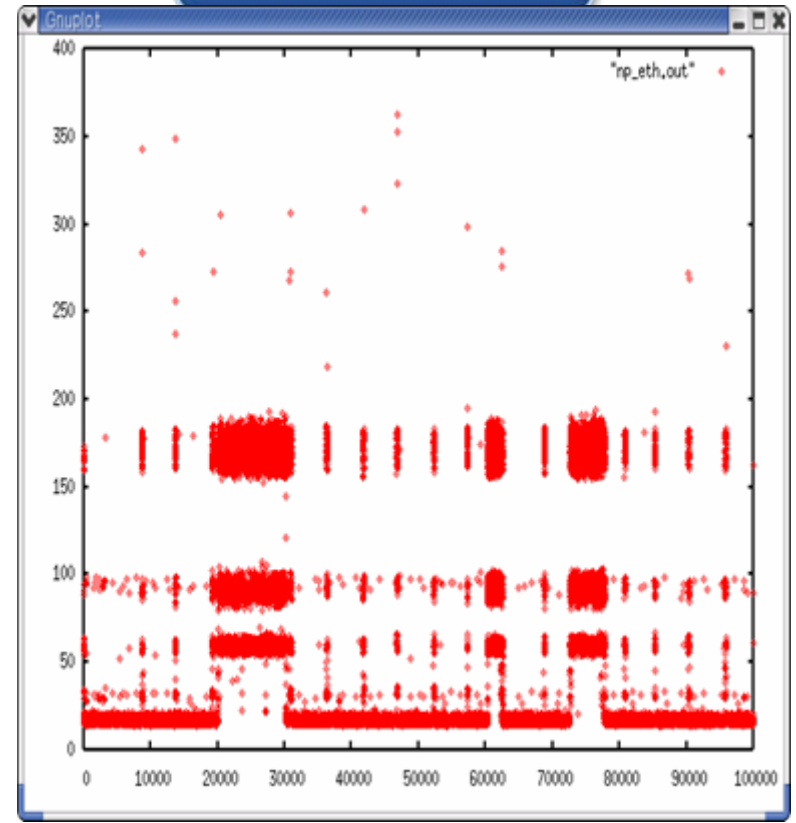
#define ZI_IRQ_EXIT(tracing_point) \
    zi_register_event(( ZI_GROUP_IRQ << 24 | ZI_TYPE_IRQ_EXIT), \
        tracing_point);
```


A Comparison – with LTT

Linux 2.4.20-LTT



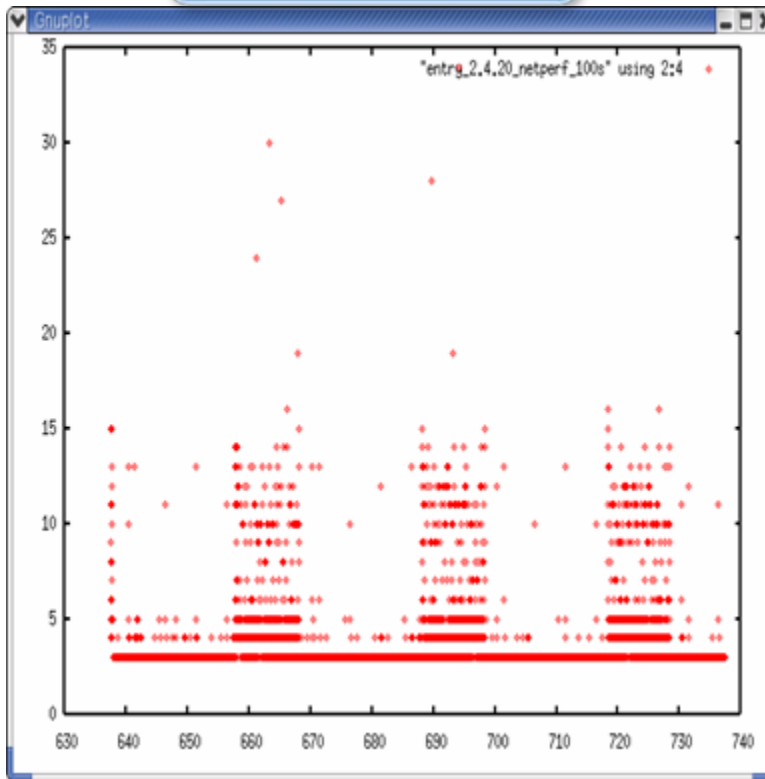
Linux 2.6.10-LTT



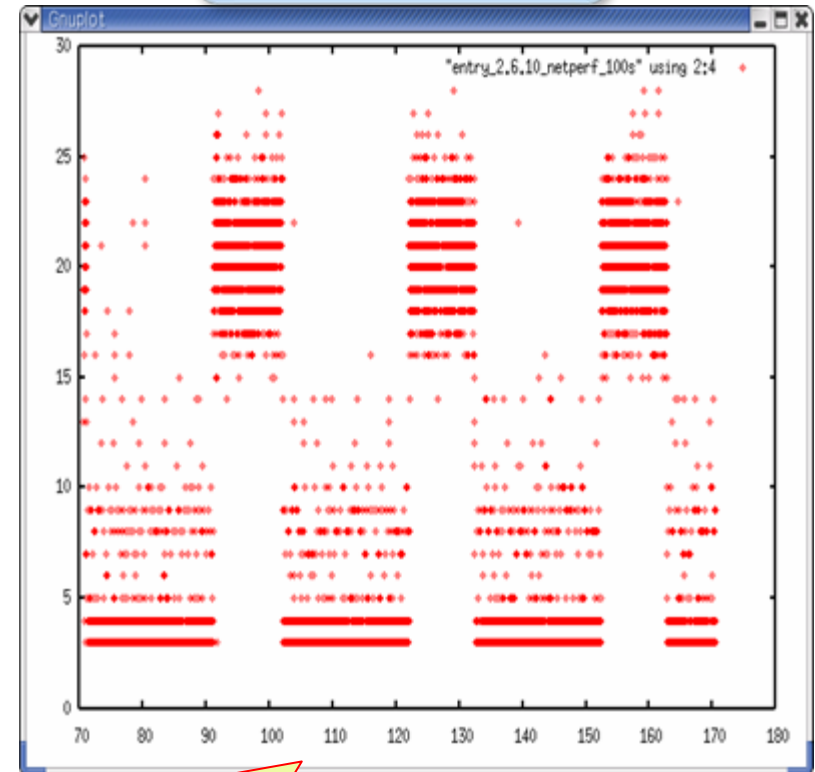
A comparison –with ZI

- ◆ minimum Interrupt Entry Latency is 3us~30us

Linux 2.4.20-ZI



Linux 2.6.10-ZI



Good !!

There is no big difference between Linux 2.6 and Linux 2.4

Real Target Analysis (DTV)

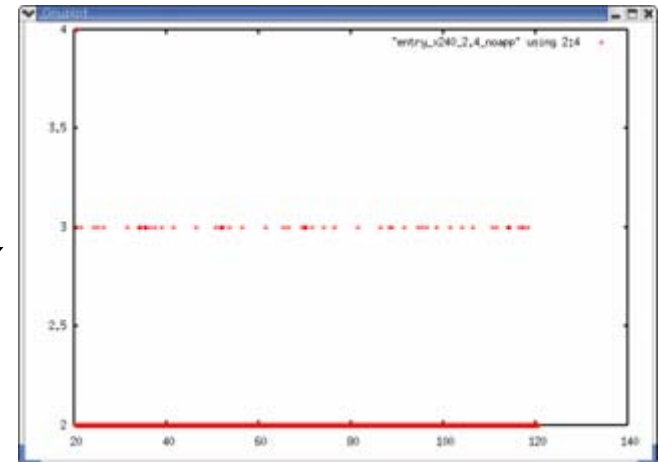
◆ DTV System Interrupt Entry Latency

– X240-MIPS (ATI)

- CPU frequency 264.06 MHz
- Calibrating delay loop... 263.78 BogoMIPS

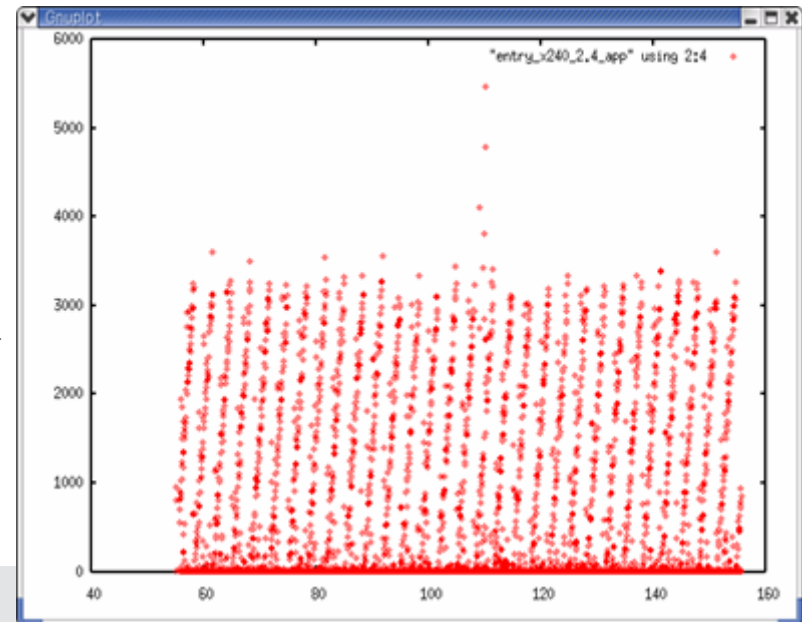
< after basic kernel booting >

```
/ # 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  atyx2200
10:         0      XILLEON Int  atyx2200
64:       36786      XILLEON Int  timer
65:      121572      XILLEON Int  atyx2200
99:       1791      XILLEON Int  serial
100:      27372      XILLEON Int  serial
101:      85245      XILLEON Int  atyx2200
106:      35067      XILLEON Int  eth0
113:       4412      XILLEON Int  atyx2200
114:       9785      XILLEON Int  atyx2200
115:         0      XILLEON Int  atyx2200
119:         0      XILLEON Int  atyx2200
120:         0      XILLEON Int  atyx2200
121:         0      XILLEON Int  atyx2200
126:         0      XILLEON Int  atyx2200
128:         0      XILLEON Int  atyx2200
```



Q & A

Thank you