Measuring Responsiveness of Linux Kernel on Embedded Systems

2010. 4. 12.

YungJoon Jung, Donghyouk Lim, Chaedeok Lim
Embedded SW Research Department

한국전자통신연구원
Electronics and Telecommunications Research Institute

Contents

- Introduction
  - RT system characteristic
  - Needs on RT responsiveness measurement
  - Considerations for our measurement method
- Related measurement methods
- Our measurement method
  - Measurement interval
  - Measurement mechanism
  - Implementation
  - Measurement result
- Comparison
- Supporting measurement tool
  - Visualization system for measurement
- Future works

INTRODUCTION

Introduction

❑ Real-time System Characteristics

- System responsiveness vs. overall system performance
  - There is trade-off relationship between system responsiveness and system performance
- Real-time system should guarantee two things
  - Timeliness
    - Not quick response, but predictable
  - Should guarantee correct job execution
- Real-time system has two types
  - Hard
  - Soft

Introduction

- Applicable areas
  - Traditional industry
    - Military System, Avionics, Nuclear Power Plant, etc.
  - Consumer electronics industry
    - Cellular Phone, Portable Media Player, Digital Camera, Digital TV, etc.
- In general, real-time systems have almost used traditional RTOS
- Recently, many trials to adapt embedded Linux to many systems due to cost and convenience

Introduction

Improvement responsiveness of Linux

- Before Linux kernel 2.6
  - Sub Kernel approach mainly was used
    - Linux kernel runs as an application on real-time OS
  - Linux kernel modification approach
    - Several features start to enhance(i.e. preemptible kernel, lock-break, etc)
- After Linux kernel 2.6
  - Linux kernel modification approach has been mainly improved
  - Many rt features has been matured
    - O(1) scheduler, voluntary preemption, preemptible kernel, complete kernel preemption (by Ingo Molnar), etc.

Introduction – Needs on measurement Method

- Real-time features have improved significantly
  - Many people are interested in RT features
  - It's time to apply RT features on your system
- People starts to wonder how much level of RT responsiveness can be supported
  - This measurement needs have been raised
    - Customers want to know the criteria of RT performance
    - Developers want to know whether their developing system meets RT requirement or not
  - So far, people have been less interested in measurement method than rt patch improvement

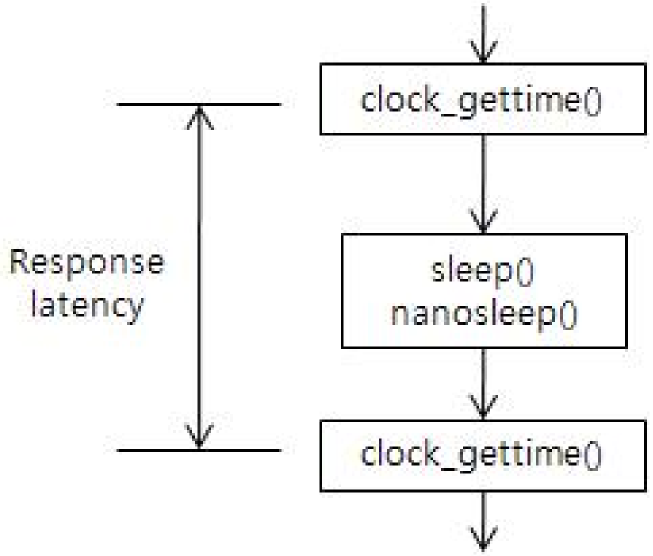
Introduction – Considerations for our method

- We think, RT performance measurement should have these requirements
  - Measurement interval should be defined
  - Measurement accuracy should be provided
  - Hardware dependency should be described
- We suggest a RT responsiveness measurement method for embedded Linux systems
- We have plans to share an open project page (sourceforge and rt-wiki)
  - We want to share our and other people's experiences on various systems

RELATED MEASURMENT METHODS

Related Measurement Methods (1/2)

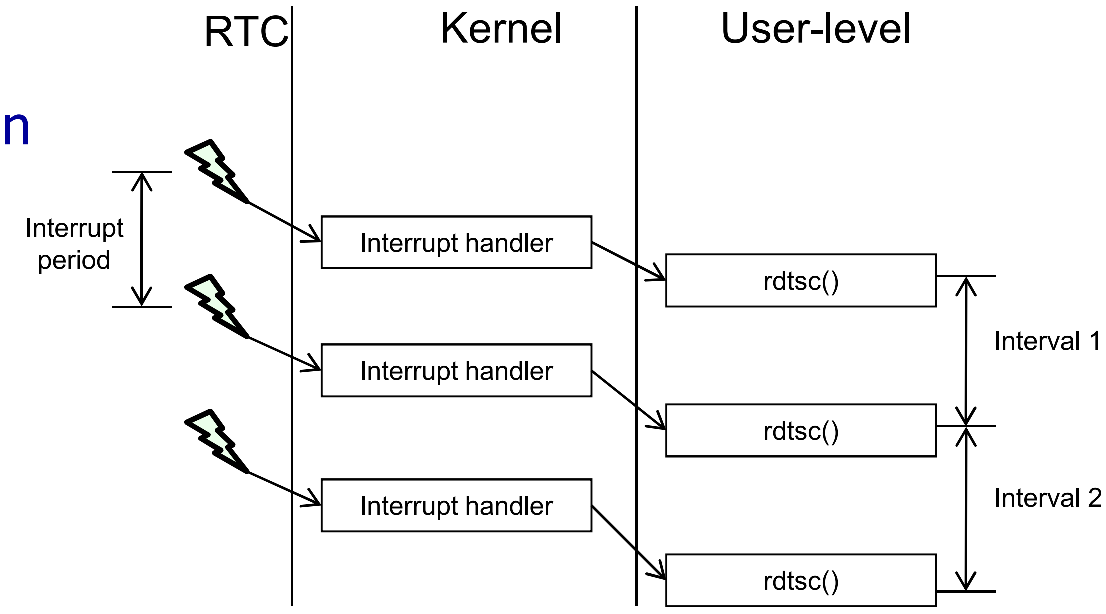
- Cyclictest
  - Was developed by tglx
    - Mostly used in community
  - Measures the delay of sleeping API such as sleep() and nanosleep()
  - Uses a high-resolution timer, if available
  - Otherwise, uses a posix timer
  - A smaller delay means higher responsiveness.



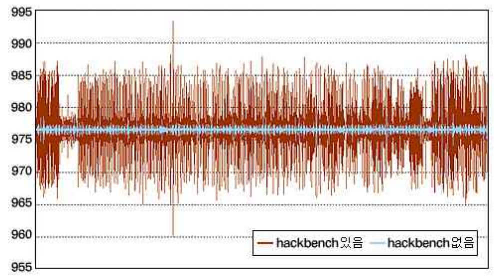
Related Measurement Methods (2/2)

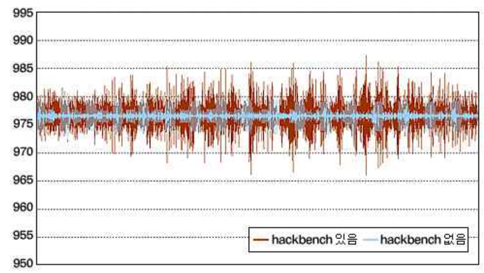
❑ Realfeel

- Was developed by Mark Hahn
- Uses periodic interrupt of a real-time clock(RTC)
- Measures jitters between interrupt period and task invocation period
- Ideally, interrupt period and task invocation interval are same.
- The long interval of user-level task invocation means low responsiveness.



Jitter = Interval n – Interrupt period

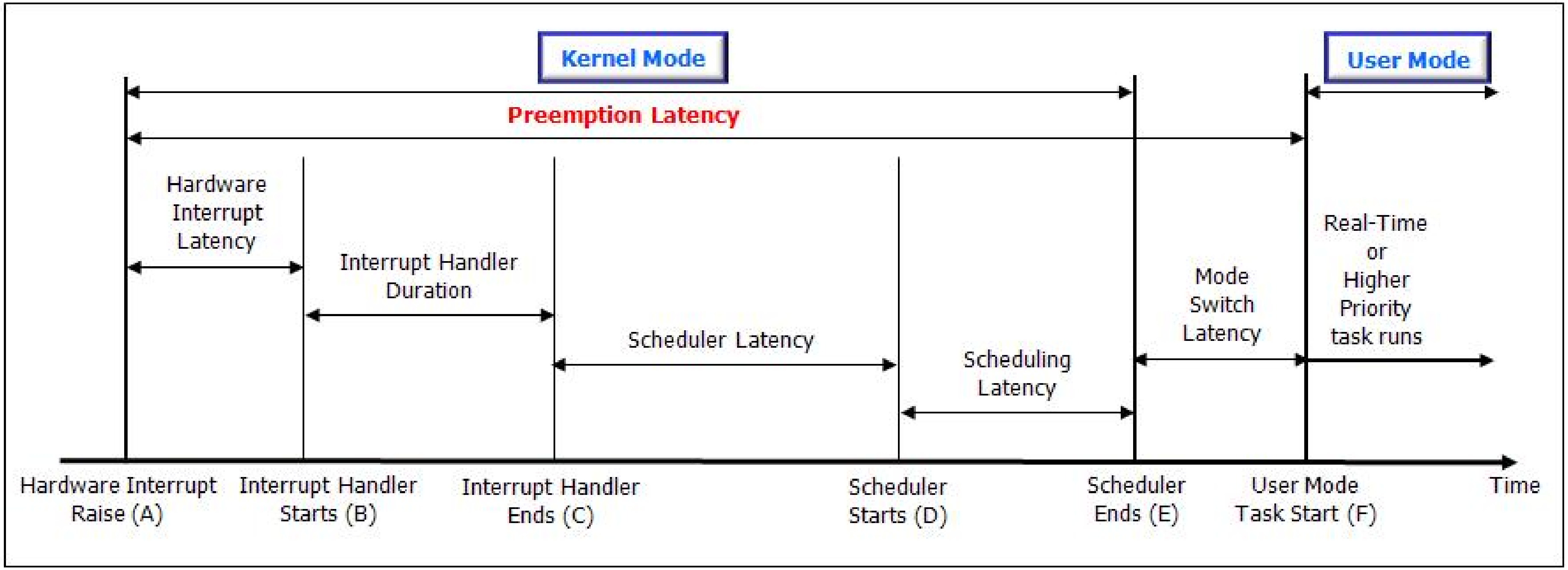




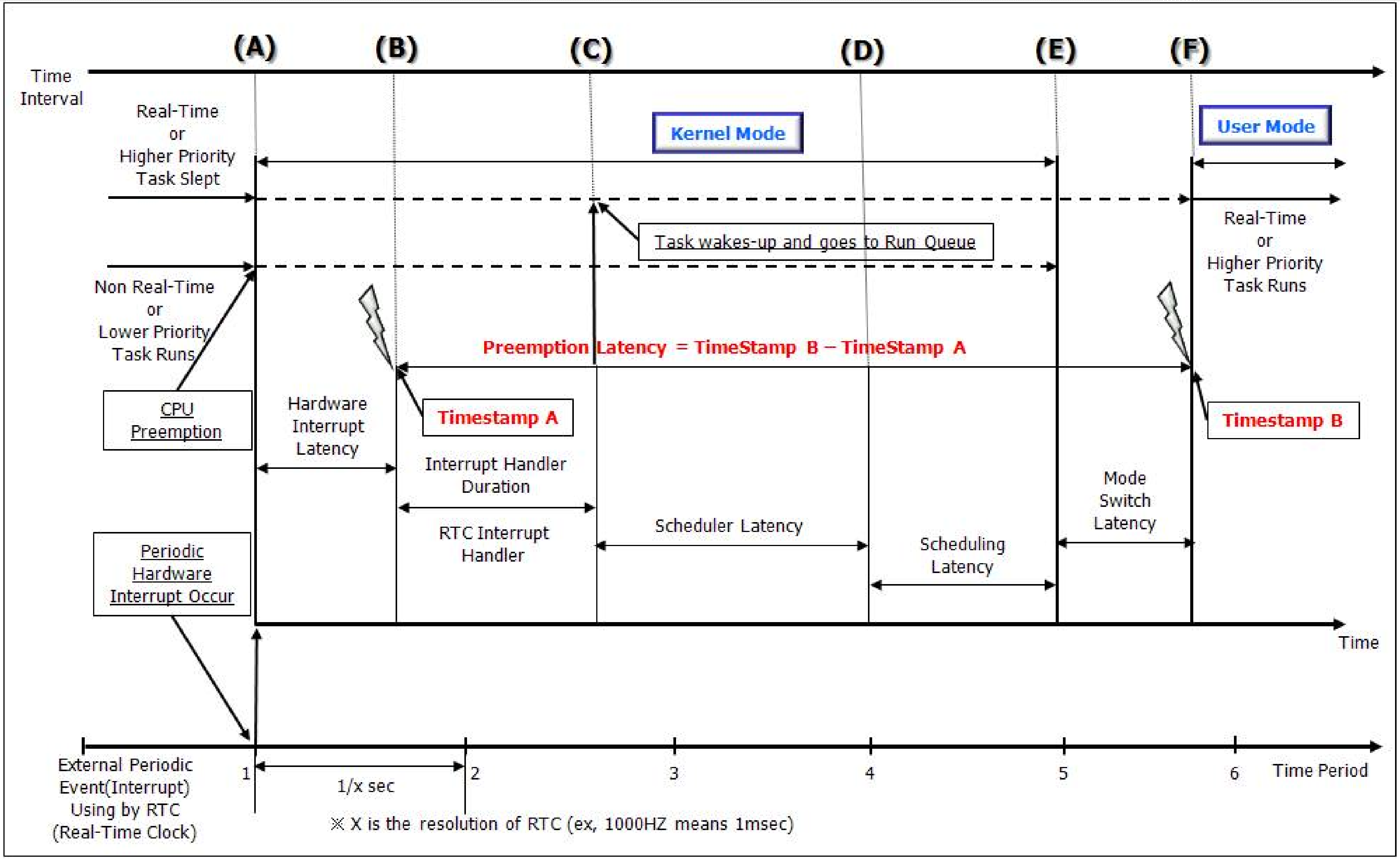
OUR MEASUREMENT METHOD

Measurement interval definition

- Our definition of measurement intervals
  - Timeline from hardware interrupt to user task invocation
  - What we want to measure = “Preemption Latency”

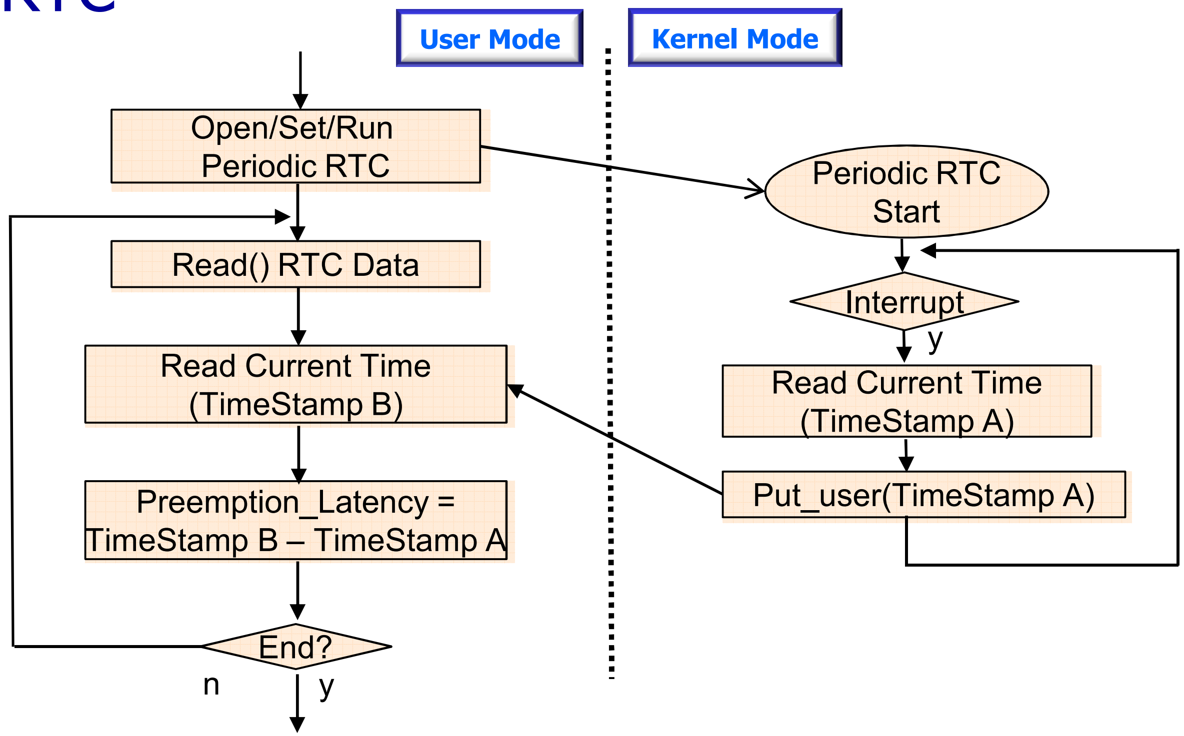


Detail measurement interval and situation



Skeleton for Measurement Implementation

- Measure "Preemption Latency"
- Uses period interrupt of RTC
- Executes while loop and measure latency



Control flow of our skeleton

What you need to measure responsiveness

- Real-time clock(RTC) must support a periodic interrupt
  - An interrupt source
  - Some RTCs don't support periodic interrupt
  - Test your RTC driver (ioctl() command)

- Timer or Clock counter
  - Processor clock counter
  - Timers included in your system
  - Check the resolution of timer or clock
  - Timestamps

Clock Counters in Processors

- What we found
  - How to get the clock count
    - Dedicated operation (x86)
    - Coprocessor register (ARM, MIPS)
  - Accessibility
    - Accessible in kernel and user mode (x86)
    - Configurable by special register (ARM11, MIPS32R2)
    - Only accessible in kernel mode (ARM9, xscale, MIPS)

Clock Counters in Processors

- Implementation
  - Use inline assembly
    - No special library or API
    - Use some dedicated operations to access special registers
  - Case 1 – x86
    - No access restriction
    - “rdtsc” operation provides clock count

```
__asm__ __volatile__("rdtsc"
       : "=A" (tsc));
```

Clock Counters in Processors

Implementation

- Case 2 – ARM (ARM11 and higher)
  - Special operation : “mcr”(move cp from reg), “mrc”
  - Access validation control : coprocessor 15, c15, c9 register

```
__asm__ __volatile__("mcr p15, 0, %0, c15, c9, 0"
              ::"r"(0x1));
```

  - Read clock counter : coprocessor 15, c15, c12 register

```
__asm__ __volatile__ ("mrc p15, 0, %0, c15, c12, 1"
              :"=r"(tsc_irq));
```

Clock Counters in Processors

- Implementation
  - Case 3 – MIPS (MIPS revision 2 and higher)
    - Special operation : "mtc0"(move to cp0), "rdhwr"(read hardware register)
    - Hardware register enable : coprocessor 0, register 7

```
__asm__ __volatile__ ("mtc0 %0, $7"
             ::"r"(0x40000000));
```

    - Read clock counter: hardware register 2, cycle counter

```
__asm__ __volatile__ ("rdhwr %0, $2"
             :"=r"(tsc_irq));
```

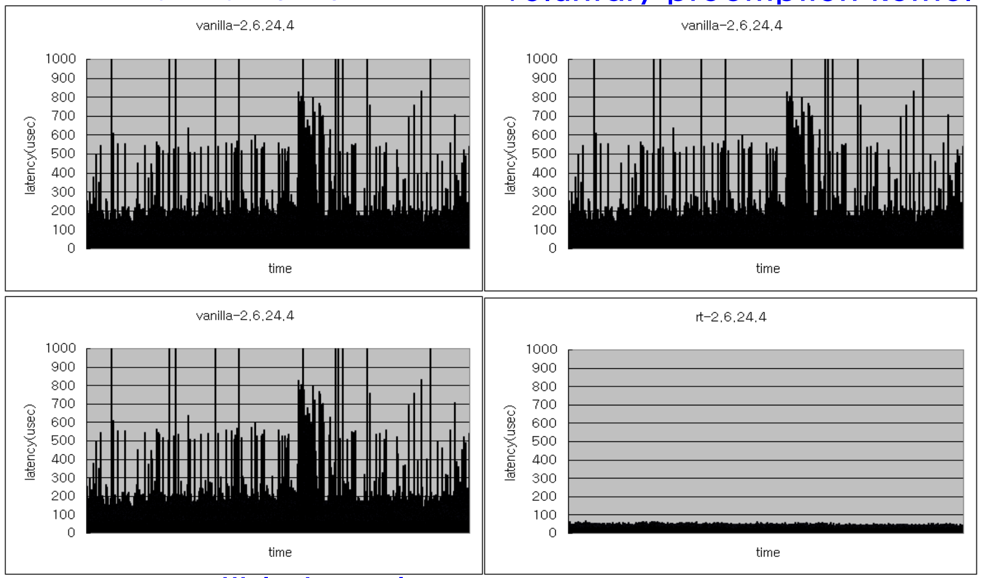
Kernel Patch for RT Measurement

- Modifying RTC driver - /drivers/rtc/
  - Access grant of clock counter
    - When rtc device is open
    - In case of x86, no special operation
    - In case of ARM, add access validation control code
    - In case of MIPS, add HWR access enable code
    - Add code to open() handler : rtc_dev_open()
  - Timestamp in kernel
    - When an interrupt handler is invoked
    - Find Interrupt handler code
    - Add code that read a value from clock counter
    - Return this timestamp with RTC value in Read() handler : rtc_dev_read()

Measurement Result

- Platform : Via EPIA(Nehemiah) 1GHz, 256Mbyte memory
- Kernel version : Linux 2.6.24.4
- Stress : ping (per 100 nano sec) from other machine, hackbench 20 (per 50sec)
- Test time : 10 hours

vanilla kernel voluntary preemption kernel

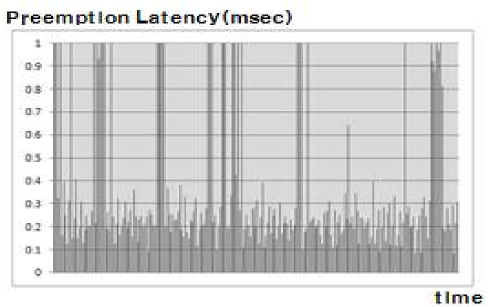


preemptible kernel real-time preemption kernel

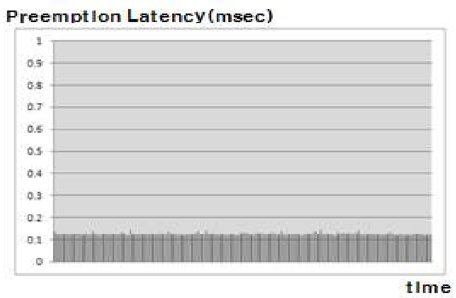
	Max latency time (usec)	Min latency time (usec)	Ave latency time (usec)
Vanilla	3888.57	3.96148	13.8279
Voluntary	3904.31	3.88947	11.2108
Preemptible	6792.74	4.75657	11.6637
Realtime-reempt	65.8249	9.36812	13.0913

Measurement Result

- Platform : SMDK6410, 256Mbyte memory
- Kernel version : Linux 2.6.21.5
- Stress : hackbench 20 (per 50sec)
- Test time : 1 hours



Vanilla Kernel



Real-Time Kernel

	Max latency time (usec)	Min latency time (usec)	Ave latency time (usec)
Vanilla	742	22	7633
Realtime-reempt	125	55	145

COMPARISON

Comparison with Other Methods

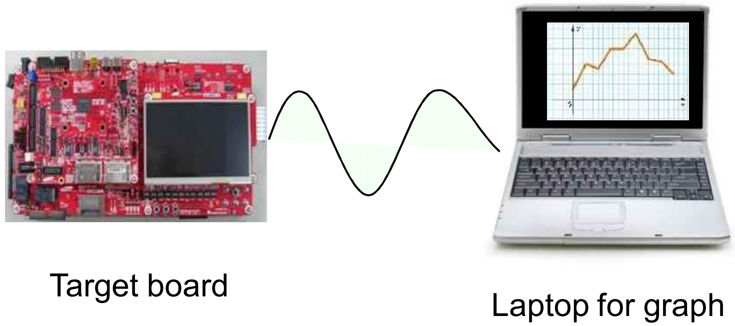
	Cyclictest	Realfeel	Our method
Measurement Interval	Scheduling Latency (delay)	Jitter	Preemption Latency
Interrupt Generation Method	No	Periodic programmed via /dev/rtc	Periodic programmed via /dev/rtc
Requirement	Sometimes needs HRT for measurement accuracy	/dev/rtc, RDTSC for x86	/dev/rtc, performance counter for each CPU architecture
Advantage	CPU architecture independent, but sometimes it requires high resolution timer for specific CPU architecture	Can be used easy and conveniently	Measure the Preemption Latency, Implementation is easy, Result data is intuitive
Disadvantage	Only measure Scheduling Latency	Only measures jitter and supports x86	Can be architecture dependent, but it supports the skeleton for steady and easy adaption, already supports x86 and arm

SUPPORTING MEASUREMENT TOOL

Visualization of Responsiveness Measurement

- Use "Livegraph" tool
  - Open source graph tool
  - Read data from a file and draw a graph
  - The refresh rate of a graph can be adjusted

- Measurement toolkit
  - Measurement data transferred via serial line
  - A program reads data and write them on a file
  - Livegraph will show you result



Future Work

- Open project
  - All of measurement programs will be open
  - Sourceforge.net page will be open
  - Please come and join our project

- Technical showcase in ELC2010
  - Demonstration – measurement on ARM processor
  - Please visit us and watch our result tomorrow night

THANK YOU