SMP/Linux Real-time Analysis & Enhancements

Jim Huang (黃敬群) <jserv.tw@gmail.com>
Real-Time System

- Deadline

- Hard Real-time
  - Meet deadline Deterministically
  - Guaranteed worst case

- Soft Real-time
  - Best Effort
Real-time System

- Critical Task
  - Type
    - Periodic Task
    - Non Periodic Task
  - Latency
    - Preemption
    - Interrupt
    - Critical Section
    - Others
Real-time Task in Linux

- Schedule Policy
  - SCHED_FIFO
  - SCHED_RR
Real-time Priority of Linux

- Nice value: -20 ~ +19 (19 is lowest)
- Real-time Priority: 0 ~ 99
  - higher value with higher priority
Real-time Task

• Periodic Real-time Task
  – Precise timer is needed

• Non periodic Real-time task
  – Triggered by Interrupt
Periodic real-time task

- Periodic Real-time task

e.g.

```
+-----+-----+-----+-----+-----+
| task| task| task| task| task|
| 5ms | 5ms | 5ms | 5ms | 5ms |
+-----+-----+-----+-----+-----+
```
Linux Periodic Timer

• Tick in Linux
  – jiffies
  – HZ
    • Number of tick of one second
    • Configurable
  – Timer Resolution
Periodic Timer Resolution

Request sleep time (ms) vs Real sleep time (ms)

HZ=100
Period = 10ms
Periodic Timer Resolution

Request sleep time (ms)

HZ=1000
Period = 1ms
Periodic Timer Issue

• High resolution
  – High overhead
  – Max HZ value : 1000
    • Min time granularity: 1ms

• Bound to jiffies
High Resolution Timer

- hrtimer

- Time Source
  - Hardware clock event
  - All timer events are One shot events
  - Recent expired timer will be triggered

- e.g. Tick with hrtimer support
  - tick_sched_timer
  - Periodic timer event by one shot hrtimer event
  - Current tick event will register next tick event
Periodic Real-time Task

- Periodic Real-time Task with hrtimer support
  - User Space
    - `clock_nanosleep`
    - `clock source: CLOCK_MONOTONIC`
  - Kernel Space
    - `hrtimer_init`
    - Triggered by Clock Event
    - One shot event
Periodic Real-time Task

```
const int NSEC_IN_SEC = 10000000001, INTERVAL = 50000001;
clock_gettime(CLOCK_MONOTONIC, &timeout);

While (1) {
    task_work(&some_data);
    if (timeout.tv_nsec >= NSEC_IN_SEC) {
        timeout.tv_nsec -= NSEC_IN_SEC;
        timeout.tv_nsec = NSEC_IN_SEC;
        timeout.tv_nsec++;
    }
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &timeout, NULL);
}
```
Real-time Task with hrtimer

- All tasks are triggered by Interrupt
Latency in Linux
Latency

- Time interval from event trigger till handler task react this event
  - Event: Interrupt
  - Handler: Real-time Task
From Interrupt to Received

Critical section with interrupts disabled

HW Exception

“Top Half” / ISR

Signal/Wakeup

Exit from IRQ

Reschedule

Context Switch

Interrupt Received in User/Thread Context

Resource Conflicts

Something else is executing (probably another ISR)

E.g. locks (xtime lock could be one example?)

Locks, RCUs, etc.

Softirqs, RCUs

Priority inversion/conflict

Cache misses, etc.
Latency results from...

- Preemption
- Critical Section
- Interrupt
Preemption

- Re-schedule when high priority task is ready
- Increase responsibility
- Decrease throughput
Preemption in Linux

- Preempt configurations
  - NONE, Voluntary, Basic RT, RT_FULL
Toward complete preemption

- Most important aspects of Real-time
  - Controlling latency by allowing kernel to be preemptible everywhere

Non real-time tailored kernel:
- Not so important code
- Critical sections
- Interrupt handler wakes up Proc B

Real-time tailored kernel:
- Not so important code
- Preemptible kernel code

Urgent interrupt:
- Unbound latency
- Much reduced Interrupt latency
Non-Preemptive

- CONFIG_PREEMPT_NONE

- Preemption is not allowed in **Kernel Mode**

- Preemption could happen upon returning to user space
Non-Preemptive Issue

- Latency of Non-Preemptive configuration

![Diagram showing the latency issue in a non-preemptive configuration.](image-url)
Preemption Point

- CONFIG_PREEMPT_VOLUNTARY
- Insert *explicit* preemption point in Kernel
  - might_sleep

- Kernel can be preempted only at preemption point
Preemptible Kernel

- CONFIG_PREEMPT
- *Implicit* preemption in Kernel

- preempt_count
  - Member of thread_info
  - Preemption could happen when preempt_count == 0
Preemptible Kernel

- Preemption could happen when
  - return to user mode
  - return from irq handler
- Kernel is preemptible with timer interrupt (RR)
Full Preemptive

- CONFIG_PREEMPT_RT_BASE / CONFIG_PREEMPT_RT_FULL
  - Difference appears in the interrupt context

- Goal: Preempt Everywhere except
  - Preempt disable
  - Interrupt disable

- Reduce non-preemptible cases in kernel
  - spin_lock
  - Interrupt
Latency Issue

- Preemption
- Critical Section
- Interrupt
Spinlock

- Task will be busy waiting until it acquires the lock

- Spinlock in Preemptible Linux (CONFIG_PREEMPT)
  - Uniprocessor
    - preempt_disable
  - SMP
    - preempt_disable
    - Lock acquire, busy waiting
Spin lock in Full Preemptive

- Preemptible spin_lock
  - rtmutex
  - Avoid priority inversion
    - Priority Inheritance (PI) protocol
Priority Inversion

1. Low Priority task takes mutex
2. High Priority task preempts low priority task
3. Hi Priority task block on mutex
4. Medium Priority task preempts low priority task and high priority task

Task Priority

Time
Priority Inversion

- High priority task is preempted by medium priority task
- Unbound waiting for high priority task
Priority Inheritance

- Acquires a lock
- Tries to get the lock
- Waits
- Executes with the priority of the waiting process
- Replaces the lock
- Acquires the lock
- Preempted

Priority

Time
Latency Issue

- Preemption
- Critical Section
- Interrupt
Task triggered by IRQ

- Task Sleep
- Interrupt latency
- Interrupt handler
  - Makes the task runnable
- Scheduler
  - Schedule latency
  - Schedule duration
- Task wakeup
- Process context
- Interrupt context
Task interrupted by IRQ

Interrupt Preempt Latency Unbound??

Task Run

Interrupt handler

Interrupt handler

...
Interrupt Context

- In original Kernel
  - HardIRQ
  - SoftIRQ
  - Highest priority in system
Original Linux Kernel

- Interrupt Context
- SoftIRQs
- Scheduling Points
- User Context
- User Space
- Interrupt Handlers
- Regular tasks
- Process Thread
- Kernel Thread
- Hi-prio tasks
- Network Stack
- Timers
- Regular tasks
Original Linux Kernel

Priority of interrupt context is always higher than others

Interrupt Context

Interrupt Handlers

SoftIRQs

Hi prio
Network Stack
Timers
...
Regular

Scheduling Points

User Context

User Space

Kernel Stack

Kernel Space

Process Thread

Kernel Thread

Scheduling Points
Non-determined Issue

- Interrupt context can always preempt others

- Interrupt as an external event
  - Interrupt number of a time interval is non-determinated
  - Nature of interrupt, can not be avoided

- Behavior of interrupt handler is not well defined
  - Non-determined interrupt handler
  - Threaded IRQ
IRQ in PREEMPT_RT

• Threaded IRQ
  – IRQ handler is actually a kernel thread by default
  – Hard IRQ handler only wakes up IRQ handler thread
    • Behavior of hard IRQ handler is well defined
    – Original IRQ handler (No Delayed) is reserved by
      IRQF_NO_THREAD flag.

• Remove softirq
  – ksoftirqd as a normal kernel thread, handles all softirqs
PREEMPT_RT

NODELAY Interrupt Handlers

Kernel Threads

Kernel Space

User Space

Process Thread

Scheduling Points

Network Stack

Timers

Tasklets
Experiments on ARM

- NXP i.MX6Q SABRE Board
  - Cortex-A9 x 4; 1 GHz
- Linux kernel 4.1+
實驗方式

- 在 user space 中使用高精準度的沉睡系統呼叫進行沉睡，並在不同的工作情境以及不同的 Kernel Timer 測量實際沉睡的時間。
- 測量方式：使用 clock_gettime 獲取沉睡前以及沉睡後的時間
  
  ```c
  clock_gettime(CLOCK_MONOTONIC, &time1);
  next_time = timel;
  next_time.tv_nsec += (sleep_us * 1000);
  tsnorm(&next_time);
  clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &next_time, NULL);
  clock_gettime(CLOCK_MONOTONIC, &time2);
  ```
- 沉睡方式：以 clock_nanosleep 沉睡
- 工作情境：分為無週邊通訊負載以及有大量的週邊負載兩種情境，負載產生方式為
  - 每秒印出 /proc/interrupts 的內容來產生 serial 負載
  - 週期為 1ms 的 ping 網路通訊來產生網路負載
Practical Issues in PREEMPT_RT
Tool for Observation

- Linux Kernel Tracing Tool
- Event tracing
  - Tracing kernel events
    
    ```
    mount -t debugfs debugfs /sys/kernel/debug
    ```
  - Event: irq_handler_entry, irq_handler_exit, sched:*  
    (`/sys/kernel/debug/tracing/events/`)    
    ```
    trace-cmd record -e irq_handler_entry \  
    -e irq_handler_exit \  
    -e sched:*  
    ```
trace-cmd-1061 [000] 142.334403: irq_handler_entry: irq=29
name=critical_irq
trace-cmd-1061 [000] 142.334437: sched_wakeup:
critical_task:1056 [9] success=1 CPU:000
trace-cmd-1061 [000] 142.334456: irq_handler_exit: irq=29
  ret=handled
trace-cmd-1061 [000] 142.334480: sched_wakeup:
ksoftirqd/0:3 [98] success=1 CPU:000
trace-cmd-1061 [000] 142.334505: sched_stat_runtime:
  comm=trace-cmd pid=1061 runtime=201500 [ns] vruntime=4720432487 [ns]
trace-cmd-1061 [000] 142.334526: sched_switch:
  trace-cmd:1061 [120] R ==> critical_task:1056 [9]
Trace Log Format

- `trace-cmd-1061 [000] 142.334403: irq_handler_entry: irq=21 name=critical_irq`

<table>
<thead>
<tr>
<th>Current Task</th>
<th>CPU#</th>
<th>Time Stamp</th>
<th>Event Name</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>trace-cmd-1061</code></td>
<td>[000]</td>
<td>142.334403</td>
<td><code>irq_handler_entry</code></td>
<td><code>Irq=21 name=critical_irq</code></td>
</tr>
</tbody>
</table>
Trace Log

**irq_handler_entry**  **irq_handler_exit**

*trace-cmd*

Interruption latency

Interrupt

**sched_wakeup***

Interrupt handler

**Makes the task runnable**

Scheduler

**sched_wakeup***: wakeup critical_task

**sched_wakeup**: wakeup ksoftirqd

**sched_switch**

Process context

Interrupt context
# Trace Log Visualization

- KernelShark

---

<table>
<thead>
<tr>
<th>#</th>
<th>CPU</th>
<th>Time Stamp</th>
<th>Task</th>
<th>PID</th>
<th>Latency</th>
<th>Event</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>729230</td>
<td>0</td>
<td>1639.355062</td>
<td>fork04</td>
<td>13507</td>
<td>dNh30</td>
<td>sched_wakeup</td>
<td>test_simple:2963 [9] success=1 CPU:000</td>
</tr>
<tr>
<td>729231</td>
<td>0</td>
<td>1639.355359</td>
<td>fork04</td>
<td>13507</td>
<td>d..30</td>
<td>sched_switch</td>
<td>fork04:13507 [120] R ==» test_simple:2963 [9]</td>
</tr>
<tr>
<td>729232</td>
<td>0</td>
<td>1639.355401</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h30</td>
<td>irq_handler_entry</td>
<td>irq=34 name=eth0</td>
</tr>
<tr>
<td>729233</td>
<td>0</td>
<td>1639.355412</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h30</td>
<td>irq_handler_exit</td>
<td>irq=34 ret=handled</td>
</tr>
<tr>
<td>729234</td>
<td>0</td>
<td>1639.355437</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h50</td>
<td>sched_wakeup</td>
<td>irq=34-eth0:1042 [49] success=1 CPU:000</td>
</tr>
<tr>
<td>729235</td>
<td>0</td>
<td>1639.355525</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h10</td>
<td>irq_handler_entry</td>
<td>irq=59 name=ohci_hcd:usb1</td>
</tr>
<tr>
<td>729236</td>
<td>0</td>
<td>1639.355541</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h10</td>
<td>irq_handler_exit</td>
<td>irq=59 ret=handled</td>
</tr>
<tr>
<td>729237</td>
<td>0</td>
<td>1639.355566</td>
<td>test_simple</td>
<td>2963</td>
<td>d.h30</td>
<td>sched_wakeup</td>
<td>irq=59-ohci_hcd:979 [49] success=1 CPU:000</td>
</tr>
<tr>
<td>729238</td>
<td>0</td>
<td>1639.355620</td>
<td>test_simple</td>
<td>2963</td>
<td>d..30</td>
<td>sched_switch</td>
<td>test_simple:2963 [9] D ==» irq/34-eth0:1042 [49]</td>
</tr>
<tr>
<td>729239</td>
<td>0</td>
<td>1639.356052</td>
<td>irq/34-eth0</td>
<td>1042</td>
<td>dNh30</td>
<td>sched_wakeup</td>
<td>test_simple:2963 [9] success=1 CPU:000</td>
</tr>
<tr>
<td>729240</td>
<td>0</td>
<td>1639.356123</td>
<td>irq/34-eth0</td>
<td>1042</td>
<td>d..30</td>
<td>sched_switch</td>
<td>irq/34-eth0:1042 [49] R ==» test_simple:2963 [9]</td>
</tr>
<tr>
<td>729241</td>
<td>0</td>
<td>1639.356209</td>
<td>test_simple</td>
<td>2963</td>
<td>d..30</td>
<td>sched_switch</td>
<td>test_simple:2963 [9] D ==» irq/34-eth0:1042 [49]</td>
</tr>
</tbody>
</table>
HRT Impact

- High Resolution Timer
  - Non threaded Interrupt
    - Non-determined IRQ handler
  - processor affinity overhead
  - Important and complex component of Linux Kernel
    - Tick timer
    - Timer for RR scheduler
    - nanosleep, clock_nanosleep
    - Futex, rtmux
    - Others

Can not be disabled

Many Unpredicted Timer Interrupts !!