Evaluation of Real-time Property in Embedded Linux

LinuxCon Japan 2014

Hiraku Toyooka, Hitachi
<hiraku.toyooka.gu@hitachi.com>
• Hiraku Toyooka
  • Software engineer at Hitachi

• Working on operating systems
  • Linux (mainly) for industrial control systems
    • Server, Embedded
    • Tracing
    • Real-time virtualization (partitioning)
Agenda

- What is a real-time system?
- Real-time features in Linux
- Tuning points for evaluation
- Performance measurement
What is a real-time system?
What is a real-time system?

- Real-time system must always handle events within deadlines
  - Real-time doesn’t mean fast
  - Worst case performance is important
  - Deterministic behavior is required

Time

Cycle #1

Non-RT system

RT system

Deadline Miss!

Cycle #2

Cycle #3

Done

Done

Done

Done
• Robot control (inverted pendulum)
Real-time features in Linux
Linux and real-time

- Linux is originally designed to be a general purpose OS (GPOS)
  - High functionality
  - Average or peak performance is usually important for GPOS
  - Latency varies widely
Linux and real-time

• Current mainline Linux has a variety of real-time features
  • Certain level of deterministic behavior by appropriate configurations and tuning

• PREEMPT_RT patchset reduce maximum latency much more
  • Many real-time features in mainline are derived from PREEMPT_RT
(Some) Real-time features

• Mainline
  • Fixed-priority scheduling
  • Kernel preemption
  • High resolution timer (hrtimer)
  • IRQ thread
  • Others

• Out of tree
  • PREEMPT_RT patchset
Fixed-priority scheduling

- Linux has three types of schedulers
  - Completely Fair Scheduler (CFS)
    - Policy: SCHED_OTHER (Default), _BATCH, _IDLE
    - Task has dynamic-priority based on time slice (non-deterministic)
  - Real-time scheduler
    - Policy: SCHED_FIFO, _RR
    - Task has fixed-priority (deterministic)
  - Deadline scheduler
    - Policy: SCHED_DEADLINE
    - Merged in 3.14
Fixed-priority scheduling (cont.)

- There are per-cpu runqueues in Linux
- RT tasks are always dispatched prior to CFS tasks

Priority (for user)

<table>
<thead>
<tr>
<th>99</th>
<th>98</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT tasks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFS tasks</td>
</tr>
</tbody>
</table>

- TASK_RUNNING (READY)
- head
- task
- tail

- This task has highest priority in this CPU

CFS runqueue (red-black tree)
Kernel preemption

• Without preemption support: when higher priority tasks wake up, they are delayed until current task exits from a syscall or yields explicitly

• CONFIG_PREEMPT
  • Enables forcible context-switch even in kernel mode
  • Though spin-locks disable preemption
  • Improves response latency of high priority tasks
High resolution timer (hrtimer)

• Some real-time systems start processing triggered by periodic timer events
  • But timer_list subsystem has only tick(jiffies) resolution
    • Tick intervals of 1~10ms are too long
    • Tick-independent software timer is necessary

• CONFIG_HIGH_RES_TIMERS
  • Enables software timer mechanism which supports nanosecond resolution
  • Userspace programs using nanosleep, itimers or posix-timers can benefit from this feature
IRQ thread

- Move a part of the interrupt handler code to a kernel thread
  - reduce interrupt-disabled section
  - prioritize interrupt handlers

- CONFIG_IRQ_FORCED_THREADING
  - Force threading of all interrupt handlers
    - except handlers marked IRQF_NO_THREAD
  - Enabled by boot parameter “threadirqs”
# ps |grep irq

3 root 0:00 [ksoftirqd/0]
64 root 0:00 [irq/12-edma]
65 root 0:00 [irq/14-edma_err]
203 root 0:00 [irq/70-omap_i2c]
921 root 0:00 [irq/30-omap_i2c]
956 root 0:00 [irq/18-musb-hdr]
959 root 0:00 [irq/19-musb-hdr]
967 root 0:00 [irq/75-rtc0]
968 root 0:00 [irq/76-rtc0]
977 root 0:00 [irq/64-mmc0]
978 root 0:00 [irq/166-mmc0]
988 root 0:00 [irq/77-wkup_m3]
989 root 0:00 [irq/78-wkup_m3]
990 root 0:00 [irq/120-smartre]
991 root 0:00 [irq/121-smartre]
1006 root 0:00 [irq/72-OMAP UAR]
Others

- PI (Priority Inheritance) mutex
- Preemptive RCU
- RT group scheduling
- RT throttling
- Full tickless (no_hz)
- etc…
PREEMPT_RT patchset

- provides more deterministic behavior
  - It consists of lots of patches lowering maximum latency
    - There are 318 patches in 3.14.2-rt3

- CONFIG_PREEMPT_RT_FULL
  - allows “full preemption”
    - nearly all of the kernel can be preempted
    - replaces spin-locks with mutexes (Sleeping spin-lock)
    - many other improvements
Tuning points for evaluation
Tuning points for evaluation

- Clocksource resolution
- Task priority
- Task switching cost
- Page faults
- Multi-core
- Lock debugging
Clocksource resolution

• Getting and updating clock time depend on clocksource
  • gettimeofday(2), clock_gettime(2)
• This means that time measurement is affected by clocksource resolution
  • Confirmation is necessary before measurement

# dmesg | grep clock
[0.000000] OMAP clocksource: GPTIMER1 at 24000000 Hz
[0.000000] sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every 178956ms
Task priority

- We need to use RT policy and priorities for deterministic scheduling
  - sched_setscheduler(2) can change those scheduling attributes of specified tasks
  - In pthread library, new thread’s scheduling attributes can be set by
    pthread_attr_setschedparam(3)
  - Do not forget to call pthread_attr_setinheritsched(3) with PTHREAD_EXPLICIT_SCHED
Task switching

- Process switching cost is significantly larger than thread switching
  - Process switching needs to flush TLB
  - If your real-time application is composed of lots of processes, process switching measurement is necessary
Page faults

- Memory allocation to userspace is usually only virtual address space allocation
  - Initial memory access causes page fault, and this produces more latency
  - Page-out to swap area also causes page faults

- To prevent page faults, mlockall(2) is used
  - All memory-mapped pages remain on physical memory after mlockall(2)
In multi-core SoC, tasks can move from local core to remote cores
- This migration causes additional latency
- Tasks can be fixed to a specific core by cpuset cgroup

Influence of unrelated interrupt handling
- It’s effective to fix real-time tasks to cores which doesn’t handle unrelated IRQs
  - after setting IRQ affinity.
Lock debugging features

- Lock debugging features add much more latency
- Disabling them is effective
Performance measurement
• Interrupt response time
  • Time period between expiry of a hardware timer and scheduling of a userspace task
  • measured by cyclictest (in rt-tests)
  • cyclicetest -a 0 -p 99 -m -n -l 100000 -q --histogram=500
    • -a: set CPU affinity to CPU0
    • -p 99: set priority to 99
    • -m: use mlockall
    • -n: use clock_nanosleep to measure time
    • -l 100000: number of trials
    • -q: don’t print anything while testing
    • --histogram=50: print histogram 0-49us after the test
• Task switching time
  • measured by own program (though ptsematetest exists…)
  • Cause thread switching by POSIX semaphore 100k times

Time

(1) Task A high-prio
Runtime on CPU

(2) Task A high-prio
Running

(3) Scheduler selects B
Sleep

(4) Task B low-prio
Sleeping

Ready (on runqueue)

Time

(1) Task A high-prio
Running on CPU

(2) Task A high-prio
Running

(3) Task B low-prio
Running

(4) Task A high-prio
Running

Wake up

Task B low-prio

Task A high-prio

Task B low-prio

Task A high-prio

Task A high-prio

Task B low-prio

Task A high-prio

Task B low-prio

Task A high-prio

Task B low-prio

Task A high-prio
### Evaluation environment

<table>
<thead>
<tr>
<th>Board</th>
<th>BeagleBone Black</th>
<th>Altima Helio board</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>TI Sitara AM3359</td>
<td>Altera Cyclone V</td>
</tr>
<tr>
<td>CPU</td>
<td>ARM Cortex-A8 (1GHz)</td>
<td>ARM Cortex-A9 (800MHz x2)</td>
</tr>
<tr>
<td>L1 cache</td>
<td>32KB / 32KB</td>
<td>32KB / 32KB</td>
</tr>
<tr>
<td>L2 cache</td>
<td>256KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Main memory</td>
<td>512MB</td>
<td>1GB</td>
</tr>
<tr>
<td>Linux version</td>
<td>3.14.0</td>
<td>3.14.0 [1]</td>
</tr>
<tr>
<td>PREEMPT_RT version</td>
<td>3.14.0-rt1</td>
<td>3.14.0-rt1</td>
</tr>
</tbody>
</table>


**Diagram:**
- Measurement programs
- iperf
- Linux
- Target board
- Network I/O load (UDP 100Mbps)
- iperf
- Linux
- PC
Comparison

• Three types of kernels
  • “No preempt”
  • “Preempt”
  • “Preempt RT”

<table>
<thead>
<tr>
<th>Settings</th>
<th>“No Preempt”</th>
<th>“Preempt”</th>
<th>“Preempt RT”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preemption</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Forced IRQ threading</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Full preemption (RT patch)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Interrupt response time (BeagleBoneBlack)

BeagleBoneBlack

Max latency
- No Preempt: 293us
- Preempt: 88us
- Preempt RT: 55us
Interrupt response time (Helio)

Max latency
- No Preempt: 248us
- Preempt: 108us
- Preempt RT: 30us
Task switching time (BeagleBoneBlack)

Max latency
- No Preempt: 208us
- Preempt: 69us
- Preempt RT: 75us
Task switching time (Helio)

Max latency

- No Preempt: 219us
- Preempt: 43us
- Preempt RT: 42us
Conclusion

- Current mainline Linux has lots of real-time features
- PREEMPT_RT patchset provides more deterministic behavior
- We can reduce maximum latency by using these features with appropriate configuration and tuning
Trademarks

- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.