Improvement of Scheduling Granularity for Deadline Scheduler

Yoshitake Kobayashi
Advanced Software Technology Group
Corporate Software Engineering Center
TOSHIBA CORPORATION
Outline

- Motivation
- Deadline scheduler
- SCHED_DEADLINE and its evaluation
- Budget management
- Conclusion
Motivation

- We would like to use Linux on control systems
- Real-time is one of the most critical topic

Problem statement
- Need to evaluate deeply to meet the deadline
- CPU resource used too much by higher priority tasks

EDF scheduler
Definition of deadline

- **Wakeup time**: The time of an event occurred (Ex. Timer interrupt) and target task’s state changed to runnable.
- **Event response time**: Interrupt latency
- **Deadline**: The time for a target task must finish
Earliest Deadline First scheduling (EDF)

- The earliest deadline task has the highest priority
- Task’s priority is dynamically changed and managed
  - SCHED_FIFO is static priority management
- Theoretically the total CPU usage by all tasks is up to 100%
  - Includes the kernel overhead
  - If usage of CPU by all tasks is less than 100%, all tasks meet the deadline

Reference
An example of EDF Scheduling

- Task1: budget 1ms  period 8ms
- Task2: budget 2ms  period 5ms
- Task3: budget 4ms  period 10ms

CPU usage = 0.925% < 100%
Rate-Monotonic Scheduling (RMS)

- One of the popular scheduling algorithm for RTOS
- Assumptions for task behavior
  - NO resource sharing such as hardware, a queue, or any kind of semaphore
  - Deterministic deadlines are exactly equal to periods
  - Static priorities (the task with the highest static priority that is runnable immediately preempts all other tasks)
  - Static priorities assigned according to the rate monotonic conventions (tasks with shorter periods/deadlines are given higher priorities)
  - Context switch times and other thread operations are free and have no impact on the model
- CPU utilization
  - \( n \): number of periodic tasks, \( T_i \): Release period, \( C_i \): Computation time
  \[
  U = \sum_{i=0}^{n} \frac{C_i}{T_i} \leq n(\sqrt{n} - 1) \rightarrow \ln 2 \approx 0.69
  \]
  - CPU utilization depends on the combination of periodic tasks and it is possible to meet the deadline even the CPU utilization is around 80%
- Reference
Compared with the RMS scheduling

- Task1: budget 1ms  period 4ms
- Task2: budget 2ms  period 6ms
- Task3: budget 3ms  period 8ms

CPU usage = 0.958%

---

RMS

EDF

TOSHIBA Confidential
### Comparison of deadline algorithms

<table>
<thead>
<tr>
<th></th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS</td>
<td>Easier to implement</td>
<td>Evaluation for scheduling possibility is required to meet the deadline</td>
</tr>
<tr>
<td>EDF</td>
<td>No evaluation for scheduling possibility is required to meet the deadline</td>
<td>Difficult to implement</td>
</tr>
</tbody>
</table>
SCHED_DEADLINE

  - Implements the EDF scheduling algorithm
  - Posted to LKML by Dario Faggioli and Jury Lelli
  - Latest version is V6 (2012/10/24)

Key features of SCHED_DEADLINE

- Temporal isolation
  - The temporal behavior of each task (i.e., its ability to meet its deadlines) is not affected by the behavior of any other task in the system. All CPU managed as budget.
  - Each task is characterized by the following aspects:
    - Budget: sched_runtime
    - Period: sched_period, equal to its deadline
Build SCHED_DEADLINE (linux kernel)

- Get rt-deadline from the following place (for V3)
  - git clone git://gitorious.org/rt-deadline

- Kernel configuration
  - CONFIG_EXPERIMENTAL = y
  - CONFIG_CGROUPS = y
  - CONFIG_CGROUP_SCHED = n
  - CONFIG_HIGH_RES_TIMERS = y
  - CONFIG_PREEMPT = y
  - CONFIG_HZ_1000 = y

- Note: For V6
  - git clone git://github.com/jlelli/sched-deadline.git
Overview of SCHED_DEADLINE

EDF task

User

Set deadline and budget

System call

sched_prama_ex

task_struct

sched_dl_entity

dl_deadline

dl_runtime

sched_calss

Kernel

dl_sched_class

enqueue_task_dl

dequeue_task_dl

set_curr_task_dl

task_tick_dl

sysctl_sched_dl_runtime

sysctl_sched_dl_period

procf

Set parameters for SCHED_DEADLINE

sysctl

r b_node

dl_runtime

dl_deadline

dl_timer
Overview of SCHED_DEADLINE

EDF task

Set deadline and budget

System call

User

Set parameters for SCHED_DEADLINE

sched_prama_ex

procfs

sysctl_sched_dl_runtime

sysctl_sched_dl_period

Kernel

task_struct

sched_dl_entity

sched_class

dl_deadline

dl_runtime

dl_timer

r b_node

dl_runtime

dl_deadline

dl_timer

enqueue_task_dl

dequeue_task_dl
	
task_tick_dl
	sched_calss

setCurrTask_dl

dl_sched_class

Setting CPU utilization for a EDF task

- There are 2 ways
  - Parameters can be defined via procfs
    - CPU utilization for rt(SCHED_FIFO and SCHED_RR) and dl(SCHED_DEADLINE) should be under 100%
  - Parameters for EDF scheduler
    - /proc/sys/kernel/sched_dl_period_us
    - /proc/sys/kernel/sched_dl_runtime_us
  - When a task requires more than above limit, the task cannot submit to run.
  - An example setting (rt: 50%, dl:50%)
    - # echo 500000 > /proc/sys/kernel/sched_rt_runtime_us
    - # echo 100000 > /proc/sys/kernel/sched_dl_period_us
    - # echo 50000 > /proc/sys/kernel/sched_dl_runtime_us
Overview of SCHED_DEADLINE

EDF task

Set deadline and runtime → System call

sched_prama_ex

task_struct

sched_dl_entity
dl_deadline rb_node
dl_runtime dl_timer

Kernel

dl_sched_class
enqueue_task_dl
dequeue_task_dl

User

Set parameters for SCHED_DEADLINE → procs

syscall_sched_dl_runtime
syscall_sched_dl_period

TOSHIBA Confidential
Run a EDF task

- **Schedtool**
  - # schedtool -E -t 10000:100000 -a 0 -e ./yes
  - Options
    - **-E**: a task runs on **SCHED_DEADLINE**
    - **-t**: <execution time> and <period> in micro seconds
    - **-a**: Affinity mask
    - **-e**: command

- **System call**
  - sched_setscheduler_ex()
Budget management for EDF tasks

EDF task

Set deadline and runtime

System call

User

Set parameters for SCHED_DEADLINE

sysctl_sched_dl_runtime

sysctl_sched_dl_period

procfs

task_struct

sched_dl_entity

dl_deadline

dl_runtime

dl_timer

sched_class

Kernel

dl_scheduled_class

enqueue_task_dl

dequeue_task_dl

task_tick_dl

set_curr_task_dl
Budget management for EDF tasks

- Each task on SCHED_DEADLINE has budget which allows it to use CPU

- Budget management
  - Refill budget: `dl_timer` (high resolution timer)
  - Use budget: `task_tick_dl` (tick based)
Evaluation (Period:100ms, Budget: 50ms)
Evaluation (Period: 100ms, Budget: 10ms)
Evaluation

- Task T1: budget 1ms  period 4ms
- Task T2: budget 2ms  period 6ms
- Task T3: budget 3ms  period 8ms

Compare with page 6
Evaluation (Period: 1ms, Budget: 0.5ms)

TOSHIBA Confidential
Budget management for EDF tasks

- Each task on SCHED_DEADLINE has budget which allows it to use CPU
- Budget management
  - Refill budget: dl_timer (high resolution timer)
  - Use budget: task_tick_dl (tick based)
- An Issue
  - Difficult to keep task's budget if the budget has micro seconds granularity

**Diagram:**

1. **Step 1: Refill budget**
   - 1.5ms
   - `dl_timer`

2. **Step 2: Use budget**
   - Task execution
   - 1ms
   - 2ms
   - `task_tick_dl`
   - 1ms

3. **Time**
   - Wakeup
   - Period
   - Wakeup
   - Period
Support for micro seconds granularity

**Overview**

- When a task’s budget is less than 1ms, set HRTICK for the rest of budget

---

**Step 1: Refill budget**

- 3.2ms
- Wakeup
- Period
- Task execution
- Set HRTICK here

**Step 2: Use budget**

- 1ms
- 2ms
- 3ms
- Wakeup
Advantage and Disadvantage

- Advantage
  - Easy to support high resolution budget

- Disadvantage
  - Increase overhead
Conclusion

- **SCHED_DEADLINE** is useful for real time systems
- **An Enhancement for budget management**
  - Support fine grained budget such as 100 micro seconds
  - HRTICK is needed to support fine grained budget
Thank you