Task Scheduling for Multicore Embedded Devices

2013. 02. 22.

Gap-Joo Na
(funkygap@etri.re.kr)
Contents

I. Background
II. History of Linux Scheduler
III. Completely Fair Scheduler
IV. Case Study (DWRR)
V. Experiments
VI. Conclusion and Future Work
Background

What is multicore??
1. Multicore trends
2. New Architectures
3. Software Support
Multicore Trends on Embedded Devices

- Multicore based Application Processor market is expanding

(Billion Dollars)

Global Server/Desktop CPU and Application Processor Market Forecast

About 49% growth each year
AP surpass CPU by 2015
Multicore based embedded products have been come rapidly

- Ex) Quad-core CPU: Exynos 4412, Snapdragon S4, Tegra 4, etc
nVidia Tegra3, ‘Variable SMP’:  
- A Multi-Core CPU Architecture for Low Power and High Performance

<table>
<thead>
<tr>
<th></th>
<th>Power optimized Companion CPU Core</th>
<th>Performance optimized main CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Cortex A9</td>
<td>Cortex A9</td>
</tr>
<tr>
<td>Process Technology</td>
<td>Low Power (LP)</td>
<td>General/Fast (G).</td>
</tr>
<tr>
<td>Operating Frequency Range</td>
<td>0 MHz to 500 MHz</td>
<td>0 MHz to Max GHz</td>
</tr>
</tbody>
</table>

Tegra3 Architecture

Power-Performance gain curve of vSMP technology
**arm big.LITTLE solution:**

- Cortex-A15 and Cortex-A7 are ISA identical
- Cortex-A15 achieves high performance
- Cortex-A7 is highly energy efficient
- Samsung, Exynos 5410: 4 big cores (Cortex-A15) + 4 small cores (Cortex-A7)

![big.LITTLE Architecture](image)

![Cortex-A15-Cortex-A7 DVFS Curves](image)
Software Issues in Multicore

Insufficient adaptation for multicore due to high complexity of multicore software development

- Useful programing libraries and models are needed for multicore
  - Ex) OpenMP, OpenCL, ...

- More software development tools are needed for multicore

- Continuously, enhanced OS features are needed for multicore
  - Load balancing issue, Cache affinity, ...

New Software Issues in Multicore

- Traditional kernel technique
- Energy efficient SW technique
- Heterogeneous SW technique
- Virtual SW technique
History of Linux Scheduler

1. Before v2.6
2. After v2.6
3. Current Scheduler
Scheduler Before Kernel V2.6

Version 1.2
- User circular queue & Minimal design
- Round-Robin scheduling policy
  - Ring type runqueue for runnable task

Version 2.2
- Scheduling class supporting
  - Real-Time, Non Real-Time Task Class
- Including SMP(Symmetric Multiprocessing) support

Version 2.4
- Lack of scalability
- Weak for real-time systems
- Single runqueue supporting
  - Throughput oriented design
  - $O(N)$ complexity: the time it takes to schedule a task is a function of the number of tasks in the system
The early 2.6 Scheduler

**O(1) Scheduler**
- **O(1) complexity supporting**: using bitmap operation
- **Dual runqueues**
  - Active run queue
  - Expired run queue
- **Much more scalable**
- **Incorporated interactivity metrics**
  - Numerous heuristics (I/O, processor bound)

**Problems of O(1)**
- **Slow response time**
  - Frequent time slice allocation
- **Throughput fall**
  - Excessive switching overhead
- **None fair condition**
  - Nice 0 (100ms), Nice 1 (95ms) => 5%
  - Nice 18 (10ms), Nice 19 (5ms) => 50%
The main idea is to maintain balance (fairness) in providing processor time to tasks. To determine the balance, the CFS maintains the amount of time provided to a given task in what’s called the ‘virtual time’. The CFS maintains a time-ordered red-black tree. - Self balancing - \( O(\log n) \) time complexity

- SMP affinity
- Basic load balancing
- Priorities and CFS
- CFS group scheduling (after 2.6.24)

Example of a red-black tree
CFS (Completely Fair Scheduler)

1. Load Balancing
2. Limitations
3. Requirement of multicore embedded devices
Load Balancing of CFS (1)

1. Start Load Balancing in CFS
2. Task, Exec, Wakeup
3. Idle Runqueue
4. Periodic Checking Algorithm (check and find busiest runqueue)
Load Balancing of CFS (2)

Completely Fair Scheduler

- Load of runqueue:

\[ L_k = \sum_{\tau_i \in S_k} W(\tau_i) \]

- Amount of load to be moved:

\[ L_{imbalance} = \min \left( \min \left( L_{busiest}, L_{avg} \right), L_{avg} - L_k \right) \]

- CFS does not move any task if the following condition holds:

\[ L_{imbalance} < \min_{\tau_i \in S_{busiest}} \left( \frac{W(\tau_i)}{2} \right) \]
Weight-based Algorithm

- Fail to achieve fairness in multicore

Weight of R1 (runqueue of core1) : 1024
Weight of R2 (runqueue of core2) : 335 * 4 = 1340
Average of runque Load : 1182

\[ L_k = \sum_{\tau_i \in S_k} W(\tau_i) \]

\[ L_{imb} = \min(\min(L_{busiest}, L_{avg}), L_{avg} - L_k) \]

\[ L_{imb} = \min(\min(1340, 1182), 1182 - 1024) \]

= 158

But, 158 < 335/2

Load Balancing will not be performed

T1 weight = (T2~T5) weight X 3
Run Time of T1 = Run Time of (T2~T5) X 4

=> Fairness will be broken.
Multicore and CFS

Multicore Scheduler
- Load Balancing
  - The most effective distribution is to have equal amounts of each core
  - Global fairness is most important
- Caches of Processors
  - CPU-affinity should be considered
  - Cache effectiveness vs. Global fairness

Embedded Devices
- I/O intensive processing
- Small number of tasks
- Foreground vs. Background Task
- Interactive task (touch screen GUI)
- Energy efficient
- Web-based application

It’s time to rethink the previous task scheduler for multicore embedded devices
Case Study (DWRR)

1. Introduction
2. Basic Concept
3. Operation
4. Weak Points
Main Goal: Enhances Global Task Fairness based on Multicore


Key Idea:

- Manages task fairness every round

RQ: runqueue
EQ: Expired RQ
All queue: red-black Tree
Basic Concept

Local Fairness

- Round: the shortest time period during which every thread in the system completes at least one of its round slice

- Round Slice: \( w \times B \) (\( w \): thread’s weight, \( B \): system-wide constant)

Global Fairness

- Round Balancing
  - It allows threads to go through the same number of rounds in any time interval.
  - Whenever a CPU finishes round balancing to move over threads from other CPUs before advancing to the next round
Operation of DWRR

Time 0

CPU 0
Round 0

A

B

CPU 1
Round 0

C

Time 1

CPU 0
Round 0

(½) A

(½) B

CPU 1
Round 0

C

Time 1.5

CPU 0
Round 0

A

CPU 1
Round 0

C

B

Possibility or Riskiness

- DWRR can always guarantee higher fairness among tasks
- But, DWRR may suffer from poor interactivity due to the existence of two runqueues originated from O(1) scheduler
- Frequent task migration may cause migration overhead
- DWRR has several practical implementation issues
Experiments (CFS vs. DWRR)

1. Test Environment
2. Fairness Test
3. Scheduler Benchmark
4. CPU Intensive Workload
5. Database Workload
6. JavaScript Benchmark
**Test Environment**

**H/W and S/W**
- **Target Board**: OdroidQ (hardkernel)
  - Exynos 4412 ARM Cotex-A9 Quad Core
  - Linaro Ubuntu 12.04
  - Kernel version 3.0.41
  - CFS (sched_min_granularity = 0.75ms, sched_latency = 6ms, sched_nr_latency = 8 )
  - DWRR (round slice = 25msec)

**Architecture**

<table>
<thead>
<tr>
<th>JTAG</th>
<th>CPU Platform</th>
<th>USB 2.0 OTG</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL/OSC</td>
<td>Cortex-A9 32KB 1/D-NEON</td>
<td>USB 2.0 HSIC</td>
</tr>
<tr>
<td>DMA</td>
<td>Cortex-A9 32KB 1/D-NEON</td>
<td>eMMC 4.4 8bit</td>
</tr>
<tr>
<td>TIMER</td>
<td>Cortex-A9 32KB 1/D-NEON</td>
<td>SD Card 4bit</td>
</tr>
<tr>
<td>PWM/ADC</td>
<td>1MB L2-Cache + VFPv3</td>
<td>SDIO 4bit</td>
</tr>
<tr>
<td>EBI</td>
<td>DMC + LPDDR2 RAM 8Gbit (PoP)</td>
<td>SPI x3</td>
</tr>
<tr>
<td>LCD RGB</td>
<td>Multimedia</td>
<td>I2C x 8</td>
</tr>
<tr>
<td>HDMI</td>
<td>Multi-400 MP VG/3D GPU x 4</td>
<td>UART x 4</td>
</tr>
<tr>
<td>MIPI DSI/CSI</td>
<td>2D Graphics</td>
<td>I2S x 3</td>
</tr>
</tbody>
</table>

Arm Quad Core Architecture

Target System
Global Fairness:

- Test Method
  - Creates and runs 5 threads on 4 multicores
  - Measures average utilization of each cores and calculates standard deviation

CFS (3.0.15)

<table>
<thead>
<tr>
<th>PID USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>5512 root</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>99.0</td>
<td>0.0</td>
<td>4:59.27 test_while</td>
</tr>
<tr>
<td>5517 root</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>96.0</td>
<td>0.0</td>
<td>4:07.00 test_while</td>
</tr>
<tr>
<td>5519 root</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>76.0</td>
<td>0.0</td>
<td>5:07.92 test_while</td>
</tr>
<tr>
<td>5521 root</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>74.0</td>
<td>0.0</td>
<td>5:55.35 test_while</td>
</tr>
<tr>
<td>5522 root</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>72.0</td>
<td>0.0</td>
<td>4:17.00 test_while</td>
</tr>
<tr>
<td>4160 root</td>
<td>-99</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>1.0</td>
<td>0.0</td>
<td>0:04.35 dmd_dpc</td>
</tr>
<tr>
<td>5522 root</td>
<td>20</td>
<td>0</td>
<td>2150</td>
<td>984</td>
<td>700</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:18.95 top</td>
</tr>
<tr>
<td>3962 root</td>
<td>20</td>
<td>0</td>
<td>5676</td>
<td>2358</td>
<td>1785</td>
<td>S</td>
<td>0.3</td>
<td>0.0</td>
<td>0:00.14 modem-manager</td>
</tr>
<tr>
<td>4150 root</td>
<td>-98</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.02 dmd watchdog</td>
</tr>
</tbody>
</table>

CFS with DWRR (3.0.15)

<table>
<thead>
<tr>
<th>PID USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>14212 linaro</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>50.0</td>
<td>0.0</td>
<td>4:52.31 test_while</td>
</tr>
<tr>
<td>14215 linaro</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>79.0</td>
<td>0.0</td>
<td>4:56.06 test_while</td>
</tr>
<tr>
<td>14219 linaro</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>79.0</td>
<td>0.0</td>
<td>4:53.91 test_while</td>
</tr>
<tr>
<td>14211 linaro</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>79.0</td>
<td>0.0</td>
<td>4:59.32 test_while</td>
</tr>
<tr>
<td>14914 linaro</td>
<td>20</td>
<td>0</td>
<td>1168</td>
<td>252</td>
<td>184</td>
<td>R</td>
<td>79.0</td>
<td>0.0</td>
<td>4:55.01 test_while</td>
</tr>
<tr>
<td>5146 root</td>
<td>20</td>
<td>0</td>
<td>2960</td>
<td>1184</td>
<td>312</td>
<td>S</td>
<td>0.1</td>
<td>0.0</td>
<td>0:08.86 udevd</td>
</tr>
<tr>
<td>4720 colord</td>
<td>20</td>
<td>0</td>
<td>50148</td>
<td>9244</td>
<td>6940</td>
<td>S</td>
<td>0.1</td>
<td>0.0</td>
<td>0:10.17 colord</td>
</tr>
<tr>
<td>4740 linaro</td>
<td>20</td>
<td>0</td>
<td>216m</td>
<td>50m</td>
<td>25m</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:08.59 unity-2d-shell</td>
</tr>
<tr>
<td>14718 root</td>
<td>20</td>
<td>0</td>
<td>2160</td>
<td>980</td>
<td>700</td>
<td>R</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.31 top</td>
</tr>
</tbody>
</table>

1. root | 20 | 0 | 3112 | 1624 | 980 | S | 0.0 | 0.0 | 0:06.27 init |
2. root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 kthread |
3. root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 kernel/s |
6. root | 0 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 migration/0 |
7. root | 0 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 migration/0 |
8. root | 0 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 kworker/0 |
Scheduler Performance

Scheduling Latency Benchmark: Sysbench (test: threads)

- When a scheduler has a large number of threads competing for some set of mutexes

- Command:
  - `sysbench -num-threads=32 -test=threads -thread-yields=100 0 -thread-locks=8 run`

<table>
<thead>
<tr>
<th>scheduler</th>
<th>CFS (sched_granularity = 0.75)</th>
<th>CFS (sched_granularity = 0.5)</th>
<th>CFS (sched_granularity = 0.25)</th>
<th>DWRR (round_slice unit = 0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>12.8319s</td>
<td>13.0980s</td>
<td>21.3573s</td>
<td>7.4515s</td>
</tr>
<tr>
<td>Total Number of Events</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>Total time taken by event execution</td>
<td>410.0162</td>
<td>418.2351</td>
<td>682.6435</td>
<td>237.6006</td>
</tr>
<tr>
<td>Threads Fairness</td>
<td>Events (avg/stddev) : 312.5000/9.67</td>
<td>Events (avg/stddev) : 312.5000/10.60</td>
<td>Events (avg/stddev) : 312.5000/6.98</td>
<td>Events (avg/stddev) : 312.5000/41.63</td>
</tr>
<tr>
<td></td>
<td>Execution time (avg/stddev) : 12.8130/0.01</td>
<td>Execution time (avg/stddev) : 13.0698/0.01</td>
<td>Execution time (avg/stddev) : 21.3326/0.01</td>
<td>Execution time (avg/stddev) : 7.4250/0.01</td>
</tr>
</tbody>
</table>
**CPU Intensive Test**

### Video Codec Processing

- Mplayer benchmark nosound ao null vo null robot_720p.mp4

- Running time: 150s

<table>
<thead>
<tr>
<th>scheduler</th>
<th>CFS (sched_granularity = 0.75)</th>
<th>CFS (sched_granularity = 0.5)</th>
<th>CFS (sched_granularity = 0.25)</th>
<th>DWRR (round_slice unit = 0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BenchmarkS</td>
<td>Video Codec: 38.011s Video Out: 0.016s Audio: 0s Sys: 1.138s Total: 39.165s</td>
<td>Video Codec: 39.562s Video Out: 0.017s Audio: 0s Sys: 2.381s Total: 41.960s</td>
<td>Video Codec: 40.206s Video Out: 0.017s Audio: 0s Sys: 6.104s Total: 46.327s</td>
<td>Video Codec: 26.580s Video Out: 0.014s Audio: 0s Sys: 1.024s Total: 27.617s</td>
</tr>
<tr>
<td>Benchmark%</td>
<td>Video Codec: 97.0533% Video Out: 0.0409% Sys: 2.9057% Total: 100%</td>
<td>Video Codec: 94.2858% Video Out: 0.0403% Sys: 5.6739% Total: 100%</td>
<td>Video Codec: 86.7869% Video Out: 0.0362% Sys: 13.1769% Total: 100%</td>
<td>Video Codec: 96.2441% Video Out: 0.0489% Sys: 3.7070% Total: 100%</td>
</tr>
</tbody>
</table>
### Real database workload (Online Transaction Process)

- **Benchmark**: Sysbench, Database: Mysql
  
  ```
  sysbench --test=oltp --mysql-user=sbtest --mysql-password=sbtest --mysql-table-engine=myisam --oltp-table-size=1000000 --mysql-socket=/var/run/mysqld/mysqld.sock prepare
  
  sysbench --test=oltp --mysql-user=sbtest --mysql-password=sbtest --oltp-table-size=1000000 --mysql-socket=/var/run/mysqld/mysqld.sock --max-requests=100000 --oltp-read-only --num-threads=16 run
  ```

<table>
<thead>
<tr>
<th>scheduler</th>
<th>CFS (sched_granularity = 0.75)</th>
<th>CFS (sched_granularity = 0.5)</th>
<th>CFS (sched_granularity = 0.25)</th>
<th>DWRR (round_slice unit = 0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Performed</td>
<td>Read: 1400644</td>
<td>Read: 1400560</td>
<td>Read: 1400616</td>
<td>Read: 1400896</td>
</tr>
<tr>
<td></td>
<td>Write: 0</td>
<td>Write: 0</td>
<td>Write: 0</td>
<td>Write: 0</td>
</tr>
<tr>
<td></td>
<td>Other: 200092</td>
<td>Other: 200080</td>
<td>Other: 200088</td>
<td>Other: 200128</td>
</tr>
<tr>
<td></td>
<td>Total: 1600736</td>
<td>Total: 1600640</td>
<td>Total: 1600704</td>
<td>Total: 1601024</td>
</tr>
<tr>
<td>Transactions</td>
<td>100046</td>
<td>100040</td>
<td>100044</td>
<td>100064</td>
</tr>
<tr>
<td></td>
<td>(273.63 per sec.)</td>
<td>(322.05 per sec.)</td>
<td>(389.61 per sec.)</td>
<td>(353.96 per sec.)</td>
</tr>
</tbody>
</table>
# JavaScript Benchmark

**SunSpider JavaScript Benchmark** (http://www.webkit.org/perf/sunspider/sunspider.html)

<table>
<thead>
<tr>
<th>scheduler</th>
<th>CFS (sched_granularity = 0.75)</th>
<th>CFS (sched_granularity = 0.5)</th>
<th>CFS (sched_granularity = 0.25)</th>
<th>DWRR (round_slice unit = 0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1533.6ms +/- 0.8%</td>
<td>2289.1ms +/- 0.6%</td>
<td>2284.4ms +/- 0.6%</td>
<td>1533.2ms +/- 1.3%</td>
</tr>
<tr>
<td>3d</td>
<td>265.7ms +/- 1.5%</td>
<td>374.0ms +/- 1.5%</td>
<td>371.6ms +/- 1.8%</td>
<td>273.4ms +/- 2.9%</td>
</tr>
<tr>
<td>access</td>
<td>191.0ms +/- 2.8%</td>
<td>299.7ms +/- 2.8%</td>
<td>301.9ms +/- 4.0%</td>
<td>194.3ms +/- 4.8%</td>
</tr>
<tr>
<td>bitops</td>
<td>102.5ms +/- 2.3%</td>
<td>178.1ms +/- 3.3%</td>
<td>176.1ms +/- 2.5%</td>
<td>105.6ms +/- 7.0%</td>
</tr>
<tr>
<td>controlflow</td>
<td>14.5ms +/- 4.2%</td>
<td>23.1ms +/- 6.6%</td>
<td>22.2ms +/- 3.0%</td>
<td>14.4ms +/- 5.3%</td>
</tr>
<tr>
<td>cryoto</td>
<td>148.1ms +/- 6.8%</td>
<td>190.5ms +/- 1.6%</td>
<td>192.7ms +/- 2.4%</td>
<td>145.2ms +/- 2.7%</td>
</tr>
<tr>
<td>date</td>
<td>200.3ms +/- 6.7%</td>
<td>283.9ms +/- 2.7%</td>
<td>288.6ms +/- 2.0%</td>
<td>199.7ms +/- 4.6%</td>
</tr>
<tr>
<td>math</td>
<td>205.0ms +/- 1.3%</td>
<td>207.3ms +/- 2.2%</td>
<td>205.0ms +/- 1.3%</td>
<td>124.2ms +/- 4.6%</td>
</tr>
<tr>
<td>regexp</td>
<td>123.1ms +/- 1.1%</td>
<td>99.3ms +/- 2.0%</td>
<td>99.4s +/- 3.3%</td>
<td>65.2ms +/- 5.1%</td>
</tr>
<tr>
<td>string</td>
<td>417.3ms +/- 2.2%</td>
<td>633.2ms +/- 1.1%</td>
<td>626.9ms +/- 0.7%</td>
<td>411.2ms +/- 1.7%</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

- Multicore processors are becoming an integral part of embedded devices
- In Linux, CFS is the best scheduler until now
  - CFS performs load balancing depending on task’s weight
  - The weight-based algorithms fails to achieve global fairness in practical
- DWRR can be new trial to improve the multicore in terms of fairness
- Rethink the scheduler for multicore embedded devices.

Future Work

- Optimal load balancing algorithm
- Enhanced runqueue structure
- Per core scheduler policy
Q & A

Thank You