EECS 750: Advanced Operating Systems

01/29 /2014

Heechul Yun
Administrative

• Next summary assignment
  – due by 11:59 p.m., Thursday
  – Include “[EECS750]” and no attachment please.

• Lecture slides are available on the course website

• Student presentation starts on Feb 3. (Monday)

• Project group notification due on Feb 3. (Monday)
Recap: Completely Fair Scheduler (CFS)

• Each task maintains its virtual time
  – \( V_i = E_i \times \frac{1}{w_i} \), where \( E \) is executed time, \( w \) is a weight

• Pick the task with the **smallest virtual time**
  – Tasks are sorted according to their virtual times
  – Managed by a red-black tree, \( O(\log N) \)

• Guarantee **fairness** and **bounded latency**
  – No complex heuristics for interactivity.
Recap: BVT

• Key ideas
  – Weighted fair sharing using virtual time
    • \( \approx \) Linux CFS
  – Prioritizing real-time tasks by borrowing future virtual time (\textit{warpBack} mechanism)

• Project idea
  – Extending Linux CFS to include BVT’s \textit{warpBack} mechanism
Today

- Topic: Multicore CPU scheduling

- Linux CPU scheduling
  - SMP, load balancing

- Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin, PPoPP '09
Symmetric Multiprocessing (SMP)

- All CPUs are equal
- Memory is shared
- A single OS, a task can run on any CPU
Asymmetric Multiprocessing (AMP)

- All CPUs are NOT equal
- Memory is NOT shared
- Multiple OSes, A task can NOT run on any CPU
Uniform Memory Access (UMA)

• All CPUs have the **same memory access cost**
Non-Uniform Memory Access (NUMA)

- All CPUs still can access the entire memory space
- But, the access cost is different depending on the access location
Global Scheduling

OS

RunQueue

 HW
CPU1  CPU2  CPU3  CPU4

tasks
Partitioned Scheduling

- Linux’s basic design. Why?
Load balancing

• Goal
  – “Don’t let someone do nothing while others are busy working”
  – “Equally” distribute “loads” to all CPUs
Load Balancing in Linux

• Move tasks to equalize loads of all cores

• When to check?
  – Task fork, clone, exec, wake up
  – A runqueue becomes empty
  – Periodically

• When to balance?
  – When the imbalance is “high” enough
Load Balancing in Linux: Example
Load Balancing in Linux: Example
Load Balancing in Linux: Example 2

- What will happen?
Load Balancing in Linux: Example 2

- Not globally fair
Load Balancing in Linux

• Further complications
  – SMT (hyper threading)
  – Shared cache vs. private cache
  – NUMA

• Linux defines “high” enough imbalance differently on a case-by-case basis
Today’s Paper

• Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin, PPoPP '09

• Guarantees global fairness
Distributed Weighted Round Robin

• Goal: guarantees task-level global fairness

• Local fairness
  – Fair between the tasks on a single cpu

• Global fairness
  – Fair between the all tasks on all CPUs
  – CFS: no, DWRR=yes
Distributed Weighted Round Robin

• Algorithm
  – For each round, each task can execute $w \times B$ time
    • $w$: weight, $B$: round time unit
  – Once a task consumes its round slice ($w \times B$), it is pushed to the ‘expired queue’
  – Once all tasks in a core are in the ‘expired queue’, pull tasks from other ‘busy’ cores
  – If there’s no busy cores in the current round, then move to next round (highest round)
Distributed Weighted Round Robin

Assume A, B, and C have weight one and round slice of one time unit.
Time 0: A and B start on CPU 0, C on CPU 1.
Time 1 (left): A and B each finish half a round and remain in round-active.
  C finishes one round and moves to round-expired.
Time 1 (right): CPU 1 performs round balancing and moves B over.
Time 1.5 (left): A and B both finish one round and move to round-expired.
  Time 1.5 (right): Both CPU 0 and 1 have nothing to do for round balancing.
  So they switch round-active and round-expired, and advance to next round.
Comparison

<table>
<thead>
<tr>
<th>PID</th>
<th>PR</th>
<th>NI</th>
<th>%CPU</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3195</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.58</td>
<td>while1</td>
</tr>
<tr>
<td>3196</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.58</td>
<td>while1</td>
</tr>
<tr>
<td>3197</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.56</td>
<td>while1</td>
</tr>
<tr>
<td>3198</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.56</td>
<td>while1</td>
</tr>
<tr>
<td>3199</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.56</td>
<td>while1</td>
</tr>
<tr>
<td>3200</td>
<td>25</td>
<td>0</td>
<td>100</td>
<td>0:07.55</td>
<td>while1</td>
</tr>
<tr>
<td>3201</td>
<td>25</td>
<td>0</td>
<td>50</td>
<td>0:03.84</td>
<td>while1</td>
</tr>
<tr>
<td>3202</td>
<td>25</td>
<td>0</td>
<td>50</td>
<td>0:03.70</td>
<td>while1</td>
</tr>
<tr>
<td>3203</td>
<td>25</td>
<td>0</td>
<td>50</td>
<td>0:03.81</td>
<td>while1</td>
</tr>
<tr>
<td>3204</td>
<td>25</td>
<td>0</td>
<td>50</td>
<td>0:03.72</td>
<td>while1</td>
</tr>
</tbody>
</table>

(a) Linux 2.6.22.15.

<table>
<thead>
<tr>
<th>PID</th>
<th>PR</th>
<th>NI</th>
<th>%CPU</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3218</td>
<td>25</td>
<td>0</td>
<td>81</td>
<td>0:05.24</td>
<td>while1</td>
</tr>
<tr>
<td>3211</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.24</td>
<td>while1</td>
</tr>
<tr>
<td>3215</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.24</td>
<td>while1</td>
</tr>
<tr>
<td>3214</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
<tr>
<td>3216</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
<tr>
<td>3217</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.23</td>
<td>while1</td>
</tr>
<tr>
<td>3219</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
<tr>
<td>3210</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
<tr>
<td>3212</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
<tr>
<td>3213</td>
<td>25</td>
<td>0</td>
<td>80</td>
<td>0:05.22</td>
<td>while1</td>
</tr>
</tbody>
</table>

(b) 2.6.22.15 with DWRR.

<table>
<thead>
<tr>
<th>PID</th>
<th>PR</th>
<th>NI</th>
<th>%CPU</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3983</td>
<td>20</td>
<td>0</td>
<td>90</td>
<td>0:07.90</td>
<td>while1</td>
</tr>
<tr>
<td>3985</td>
<td>20</td>
<td>0</td>
<td>85</td>
<td>0:07.60</td>
<td>while1</td>
</tr>
<tr>
<td>3980</td>
<td>20</td>
<td>0</td>
<td>82</td>
<td>0:07.44</td>
<td>while1</td>
</tr>
<tr>
<td>3988</td>
<td>20</td>
<td>0</td>
<td>79</td>
<td>0:07.00</td>
<td>while1</td>
</tr>
<tr>
<td>3987</td>
<td>20</td>
<td>0</td>
<td>79</td>
<td>0:07.54</td>
<td>while1</td>
</tr>
<tr>
<td>3979</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.38</td>
<td>while1</td>
</tr>
<tr>
<td>3981</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.10</td>
<td>while1</td>
</tr>
<tr>
<td>3984</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:06.86</td>
<td>while1</td>
</tr>
<tr>
<td>3986</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.70</td>
<td>while1</td>
</tr>
<tr>
<td>3982</td>
<td>20</td>
<td>0</td>
<td>73</td>
<td>0:07.08</td>
<td>while1</td>
</tr>
</tbody>
</table>

(c) Linux 2.6.24.

<table>
<thead>
<tr>
<th>PID</th>
<th>PR</th>
<th>NI</th>
<th>%CPU</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3983</td>
<td>20</td>
<td>0</td>
<td>90</td>
<td>0:07.90</td>
<td>while1</td>
</tr>
<tr>
<td>3985</td>
<td>20</td>
<td>0</td>
<td>85</td>
<td>0:07.60</td>
<td>while1</td>
</tr>
<tr>
<td>3980</td>
<td>20</td>
<td>0</td>
<td>82</td>
<td>0:07.44</td>
<td>while1</td>
</tr>
<tr>
<td>3988</td>
<td>20</td>
<td>0</td>
<td>79</td>
<td>0:07.00</td>
<td>while1</td>
</tr>
<tr>
<td>3987</td>
<td>20</td>
<td>0</td>
<td>79</td>
<td>0:07.54</td>
<td>while1</td>
</tr>
<tr>
<td>3979</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.38</td>
<td>while1</td>
</tr>
<tr>
<td>3981</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.10</td>
<td>while1</td>
</tr>
<tr>
<td>3984</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:06.86</td>
<td>while1</td>
</tr>
<tr>
<td>3986</td>
<td>20</td>
<td>0</td>
<td>78</td>
<td>0:07.70</td>
<td>while1</td>
</tr>
<tr>
<td>3982</td>
<td>20</td>
<td>0</td>
<td>73</td>
<td>0:07.08</td>
<td>while1</td>
</tr>
</tbody>
</table>

(d) 2.6.24 with DWRR.

Figure 4: Snapshots of `top` for 10 threads on 8 CPUs.
Migration Cost

Figure 7: Migration cost for different working set sizes.

- Shared cache vs. private cache
Summary

• Multicore scheduling and load balancing in Linux

• Linux doesn’t guarantee global fairness

• DWRR guarantees global fairness
Discussion

• Is migration cost really negligible?

• Does it really matter to guarantee CPU time fairness?

• Load balancing in heterogeneous multicore core processors?
  – Big cores + small cores: e.g., ARM’s bigLITTLE