Tuning SCHED_ULE on FreeBSD

George Neville-Neil

gnn@neville-neil.com

May 7, 2009
Outline

- BSD Scheduler History
- SCHED_ULE
- Tuning Hooks
- Testing Methodology
- Effects
BSD Scheduler History

- BSD written for uni-processor machines
- No SMP
- No HTT
- No multicores
- Up through FreeBSD 5 only modified not wholesale rewritten
Why SCHED_ULE?

- SMP and multi-core
- SMP is NOT multi-core
- Cache effects
Why keep SCHED_BSD?

- One size does not fit all
- There are still uniprocessors
- Embedded systems
- A baseline to compare against
Scheduler Responsibilities and Goals

- Arbitrate amongst competing processes
- Adhere to the will of the administrator
- Stay out of the way
Why tune the scheduler?

- Can change overall performance of the system
- Favor one type of job over another
- Not all workloads are interactive
Don’t Panic

- The scheduler is one of the most important components of the kernel
- You (probably) cannot destroy your system via scheduler tuning
- Proceed with caution
- *Measure*, modify, *measure*, modify
- All of the tunables can simply turned off if they cause trouble
Interactivity Tunables

- **name**: Name of scheduler, ULE or 4BSD
- **interact**: Interactivity score threshold
- **slice**: Time slice for timeshare threads (100ms)
SCHED_ULE Tuning Hooks

- **steal_thresh**: Minimum load on a remote CPU before we’ll steal work.
- **steal_idle**: Attempt to steal idle work from other CPUs before this CPU goes idle.
- **steal_htt**: Steals work from another core on idle.
Stealing

- Stealing in SCHED_ULE can be virtuous
- Cores can steal work from each other
- It is a way of balancing work in an SMP/multi-core system
SCHED_ULE Tuning Hooks

balance  Enable the long term load balancer.

balance_interval  Average frequency in stathz ticks to run the long term load balancer (below).

affinity  Number of ticks to keep a thread from changing CPU.
SCHED_ULE Tuning Hooks

idlespinthresh  Threshold before idle spinning can occur

idlespins  Number of times the idle thread will spin waiting for new work

static_boost  Assign static priorities to sleeping threads

preempt_threshold  Minimum priority for preemption, lower priorities are more likely to be picked.
Testing Methodology

- We introduce a dummy load on the system
- Read data from another process
- Do some math in a loop
- Should have few or no voluntary context switches
- Wish to reduce involuntary context switches
Context Switching

- Changing the process which is executing on a core
  - **Voluntary** Process takes an action that blocks or calls `sched_yield()`
  - **Involuntary with Preemption** On exiting a critical section or interrupt service routine a process may be pre-empted.
  - **Involuntary without Preemption**
The output of top(1)

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>VCSW</th>
<th>IVCSW</th>
<th>READ</th>
<th>WRITE</th>
<th>FAULT</th>
<th>TOTAL</th>
<th>PERCENT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1019</td>
<td>gnn</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy2</td>
</tr>
<tr>
<td>982</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>tcsh</td>
</tr>
<tr>
<td>1015</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy1</td>
</tr>
<tr>
<td>1011</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>usdlogd</td>
</tr>
</tbody>
</table>
Tuning Tests

- Turn off balancing
- Change the time slice
- Test system has eight cores total
- Each test was run for 15 minutes while observing top.
Turn off Balancing

- The CPU balancer runs every 133 ticks
- In a system that is being hand tuned why run the balancer?
- What’s the effect of turning off the balancer?
Testing

With Balancing

---

last pid: 1023; load averages: 0.96, 0.53, 0.25 up 0+00:08:21 14:40:28
100 processes: 10 running, 58 sleeping, 32 waiting
CPU: 12.5% user, 0.0% nice, 0.0% system, 0.0% interrupt, 87.5% idle
Mem: 17M Active, 9848K Inact, 106M Wired, 68K Cache, 16M Buf, 7785M Free
Swap: 8192M Total, 8192M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>VCSW</th>
<th>IVCSW</th>
<th>READ</th>
<th>WRITE</th>
<th>FAULT</th>
<th>TOTAL</th>
<th>PERCENT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1019</td>
<td>gnn</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy2</td>
</tr>
<tr>
<td>982</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>tcsh</td>
</tr>
<tr>
<td>1015</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy1</td>
</tr>
<tr>
<td>1011</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>usdlogd</td>
</tr>
</tbody>
</table>

---
**Without Balancing**

Last PID: 1024; Load averages: 0.98, 0.61, 0.30 up 0+00:09:21 14:41:28

100 processes: 10 running, 58 sleeping, 32 waiting

CPU: 12.4% user, 0.0% nice, 0.1% system, 0.0% interrupt, 87.5% idle

Mem: 17M Active, 9852K Inact, 106M Wired, 68K Cache, 16M Buf, 7785M Free

Swap: 8192M Total, 8192M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>VCSW</th>
<th>IVCSW</th>
<th>READ</th>
<th>WRITE</th>
<th>FAULT</th>
<th>TOTAL</th>
<th>PERCENT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1019</td>
<td>gnn</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy2</td>
</tr>
<tr>
<td>982</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>tcsh</td>
</tr>
<tr>
<td>1015</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>dummy1</td>
</tr>
<tr>
<td>1011</td>
<td>gnn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00%</td>
<td>usdlogd</td>
</tr>
</tbody>
</table>
Balancing Results

- A slight increase in load average (0.96 to 0.99)
- The load average remains slightly higher
- The number of involuntary context switches does not change
Time Slice

- The default time slice is 13 ticks
- Increase the time slice to 64, 128, and 256 ticks
- At each level run for 15 minutes
Time Slice Evaluation
How long does a switch take?

- A set of scheduler stats are available
- Need to build the kernel with SCHED_STATS
- Locally added calls to rdtsc to mi_switch
- Store the difference between these values on each switch
- Crude but effective
- Reading the sysctl every 3 seconds
Switch Timing Results

![Graph showing timing results for different slices and CPU sets.]

- **Slice 256**
- **Slice 13 (Default)**
- **Using cpuset**

**Y-axis:** Cycles
**X-axis:** Time (Samples)
Scheduler Statistics

- **preempt**: Pre-emptions anywhere in the system.
- **owe preempt**: Were in a critical section and should have been pre-empted.
- **turnstile**: Switches due to mutex contention.
- **sleepq**: Switches due to sleep.
- **relinquish**: Called a yield function.
- **needresched**: Pre-emption of user processes on exit from the kernel.
Turning All This Off

- Sometimes you *know* what must be done
- Assigning processes to cores is also possible
- See `cpuset(4)` man page
- See also Brooks Davis’ presentation
Concluding Remarks

Further Reading

- /usr/src/sys/kern/sched_ule.c
- /usr/src/sys/kern/sched_switch.c
- “ULE: A Modern Scheduler for FreeBSD”, by Jeff Roberson
- “The Design and Implementation of the FreeBSD Operating System”, by McKusick and Neville-Neil
Questions?