Profiling Linux Operations for Performance and Troubleshooting

by Tanel Põder

https://tanelpoder.com/
@tanelpoder
About me

• **Tanel Põder**
  • I’m a database performance geek (23 years)
  • Before that an Unix/Linux geek, (27 years)
  • Oracle, Hadoop, Spark, cloud databases 😊
  • Focused on performance & troubleshooting

• **Inventing & hacking stuff, consulting, training**
  • Co-author of the Expert Oracle Exadata book
  • Co-founder & technical advisor at Gluent
  • 2 patents in data virtualization space
  • Working on a secret project ;-

• Blog: tanelpoder.com
• Twitter: twitter.com/TanelPoder
• Questions: tanel@tanelpoder.com
Agenda

1. A short intro to Linux **task state sampling** method
2. Demos
3. More Demos
4. Always on profiling of production systems
Preferring low-tech tools for high-tech problems

• Why?

  • I do ad-hoc troubleshooting for different customers
  • No time to *engineer* a solution, the problem is already happening
  • Troubleshooting across a variety of servers, distros, installations
  • Old Linux distro/kernel versions
  • No permission to change anything (including enabling kernel tracing)
  • Sometimes no root access

• Idea: Ultra-low footprint tools that get the most out of already enabled Linux instrumentation
  • `/proc` filesystem!

Low tech tools aren't always "deep" enough or precise enough, but they are quick & easy to try out
System-level metrics & thread state analysis

Let's sample the threads!
Application thread state analysis tools

- Classic Linux tools
  - `ps`
  - `top` -> (`htop`, `atop`, `nmon`, ...)

- Custom /proc sampling tools
  - `0x.tools pSnapper`
  - `0x.tools xcapture`
  - `grep . /proc/*/stat`

- Linux (kernel) tracing tools
  - `perf top`, `perf record`, `perf probe`
  - `strace`
  - `SystemTap`, `eBPF/bpftrace`

- Application level tools
  - JVM attach + profile
  - Python attach + profile

These tools also sample, snapshot /proc files
Proc sampling complements, not replaces other tools

Proc sampling complements, not replaces other tools
### Listing processes & threads

$ ps -o pid,ppid,tid,thcount,comm -p 1994

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>TID</th>
<th>THCNT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>1883</td>
<td>1994</td>
<td>157</td>
<td>java</td>
</tr>
</tbody>
</table>

$ ps -o pid,ppid,tid,thcount,comm -L -p 1994 | head

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>TID</th>
<th>THCNT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>1883</td>
<td>1994</td>
<td>157</td>
<td>java</td>
</tr>
<tr>
<td>1994</td>
<td>1883</td>
<td>2008</td>
<td>157</td>
<td>java</td>
</tr>
<tr>
<td>1994</td>
<td>1883</td>
<td>2011</td>
<td>157</td>
<td>java</td>
</tr>
<tr>
<td>1994</td>
<td>1883</td>
<td>2014</td>
<td>157</td>
<td>java</td>
</tr>
</tbody>
</table>

...  

$ ps -eLf | wc -l

1162

$ ls -ld /proc/[0-9]* | wc -l

804

$ ls -ld /proc/[0-9]*/task/* | wc -l

1161

---

**Multi-threaded JVM process**

**List each thread individually**

**Thread group leader thread PID == TID**

**All threads are visible in /proc**

**Non-leader threads are listed in task subdirectories**
Task states

• Every thread (task) has a "current state" flag
  • Updated by kernel functions just before they call `schedule()`
  • Visible in `/proc/PID/stat` & `/proc/PID/status`

```bash
$ man ps
```

**TASK STATES**

```
D  uninterruptible sleep (usually IO)
R  running or runnable (on run queue)
S  interruptible sleep (waiting for an event to complete)
T  stopped by job control signal
t  stopped by debugger during the tracing
W  paging (not valid since the 2.6.xx kernel)
X  dead (should never be seen)
Z  defunct ("zombie") process, terminated but not reaped by its parent
```

*R = Running + Runnable*

Runnable = waiting for scheduler, ready to run on CPU runqueue

*We’ll talk about the D state soon*
Task states - examples

- **ps -o s** reads state from /proc/PID/stat

```bash
$ ps -eo s,comm | sort | uniq -c | sort -nbr | head
27 S sshd
15 S bash
15 I bioset
13 I kdmflush
8 S postmaster
8 S nfsd
8 I xfs-reclaim/dm-
8 I xfs-eofblocks/d
6 S httpd
4 S sleep
```

```bash
$ ps -eo s | sort | uniq -c | sort -nbr
486 S
352 I
2 Z
1 R
```

L – see all threads!

"s" is an alias for "state"

Show only R & D states

```bash
$ ps -eLo state,user,comm | grep "^[RD]" \ | sort | uniq -c | sort -nbr
64 R tanel java
24 D tanel java
13 R mysql mysqld
2 R tanel sysbench
2 D mysql mysqld
1 R tanel ps
1 R oracle java
```
Task state sampling vs. vmstat

$ nice stress -c 32
stress: info: [28802] dispatching hogs: 32 cpu, 0 io, 0 vm, 0 hdd

$ ps -eo state,user,comm | grep "^R" | uniq -c | sort -n
32 R tanel stress
1 R tanel ps

$ ps -eo state,user,comm | grep "^R" | uniq -c | sort -n
32 R tanel stress
1 R tanel ps
1 R tanel grep

$ vmstat 3
procs ------------memory---------- ---swap-- -----io---- -system-- -------cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
35 0 162560 26177012 276 61798720 0 0 67 45 2 0 1 0 98 0 0
32 0 162560 26177112 276 61798724 0 0 53 56 32 2666 1218 100 0 0 0 0
32 0 162560 26177484 276 61798724 0 0 21 13 32276 1203 100 0 0 0 0

$ dstat -vr
---procs--- ------memory-usage----- ---paging-- -dsk/total- ---system-- ----total-cpu-usage---- --io/total-
 run blk new| used buff cach free| in out | read writ| int csw |usr sys idl wai hiq siq| read writ
 0.0 0 10| 105G 276k 57.9G 25.0G| 32B 462B| 46M 2895k| 2002 6740 | 1 0 98 0 0 0| 282 116
33 0 0.7| 105G 276k 57.9G 25.0G| 0 0 | 85k 67k| 32k 1256 |100 0 0 0 0 0|5.33 3.67
33 0 21| 105G 276k 57.9G 25.0G| 0 0 | 93k 524k| 32k 1716 |100 0 0 0 0 0|7.33 48.0
32 0 1.0| 105G 276k 57.9G 25.0G| 0 0 | 0 0 | 32k 1235 |100 0 0 0 0 0|0 0
## Scheduler off-CPU reasons

- **Scheduler reasons for taking threads off CPU:**

<table>
<thead>
<tr>
<th>Reason</th>
<th>Thread State</th>
</tr>
</thead>
<tbody>
<tr>
<td>System CPU shortage, <strong>Runnable</strong> thread out of time-slice/credit</td>
<td>R</td>
</tr>
<tr>
<td>- Or a higher priority process runnable</td>
<td></td>
</tr>
<tr>
<td>Blocking I/O: within a system call (disk I/O, NFS RPC reply, lock wait)</td>
<td>D</td>
</tr>
<tr>
<td>Blocking I/O: without a system call (hard page fault)</td>
<td></td>
</tr>
<tr>
<td>Blocking I/O: syscall against a pipe, network socket, io_getevents</td>
<td>S</td>
</tr>
<tr>
<td>Voluntary sleep: nanosleep, semtimedop, lock get</td>
<td></td>
</tr>
<tr>
<td>Suspended with: kill -STOP, -TSTP signal</td>
<td>T, t</td>
</tr>
<tr>
<td>Suspended with: ptrace() by another process</td>
<td></td>
</tr>
<tr>
<td>Other:</td>
<td></td>
</tr>
<tr>
<td>- Linux Audit backlog, etc...</td>
<td></td>
</tr>
</tbody>
</table>

© Tanel Poder
tanelpoder.com
Task state *Disk sleep – uninterruptible* is not only for disk waits!

### Code Example

```
/*
 * get a read lock on the semaphore
 */

void __sched __down_read(struct rw_semaphore *sem) {
    struct rwsem_waiter waiter;
    struct task_struct *tsk;

    spin_lock_irq(&sem->wait_lock);
    if (sem->activity >= 0 && list_empty(&sem->wait_list)) {
        /* granted */
        sem->activity++;
        spin_unlock_irq(&sem->wait_lock);
        goto out;
    }

    tsk = current;
    set_task_state(tsk, TASK_UNINTERRUPTIBLE);
}
```

```
/* set up my own style of waitqueue */
waiter.task = tsk;
waiter.flags = RWSEM_WAITING_FOR_READ;
get_task_struct(tsk);
list_add_tail(&waiter.list, &sem->wait_list);

/* we don't need to touch the semaphore struct anymore */
spin_unlock_irq(&sem->wait_lock);

/* wait to be given the lock */
for (;;) {
    if (!waiter.task)
        break;
    schedule();
    set_task_state(tsk, TASK_UNINTERRUPTIBLE);
}

tsk->state = TASK_RUNNING;
out:
    ;
}
```

Threads waiting for kernel rw-spinlocks will show up with state "D - disk wait" !!!

Demos
0x.tools Linux Process Snapper

- A free, open source /proc file system sampling tool
  - Current: Thread state sampling (currently available)
  - Planned: Kernel counter snapshotting & deltas (CPU, IO, memory, scheduling latency etc)
  - Planned: Application profiling frontend

  - [https://tanelpoder.com/psnapper](https://tanelpoder.com/psnapper)

- Implementation
  - Python script (currently Python 2.6+)
  - Works with 2.6.18+ kernels (maybe older too)
  - Passive profiling - reads /proc files
  - Does not require installation
  - Basic usage does **not** require root access
    - Especially if sampling processes under your username
  - Some usage requires root access on newer kernels (wchan, kstack)
Linux Process Snapper

More info:

- psn -h
- psn --list
- https://0x.tools

```
$ psn
Process Snapper sampling cmdline, stat for 5 seconds...
finished sampling

=== Active Threads =======================================================
samples | avg_threads | cmdline    | state
-----------------------------------
316 | 9.58 | fio     | Disk (Uninterruptible)
212 | 6.42 | fio     | Running (ON CPU)
33  | 1.00 | python  | Running (ON CPU)
30  | 0.91 |         | Running (ON CPU)
3  | 0.09 |         | Disk (Uninterruptible)
2  | 0.06 | /usr/bin/perl | Running (ON CPU)
1  | 0.03 | ora_vktd_LINPRD | Running (ON CPU)
1  | 0.03 | top     | Running (ON CPU)
```

```
$ psn -p 18286 -G syscall,filename
Linux Process Snapper v0.14 by Tanel Poder [https://tp.dev/psnapper]
Sampling /proc/stat, syscall for 5 seconds... finished.

=== Active Threads =======================================================
samples | avg_threads | comm | state              | syscall         | filename
-----------------------------------
79 | 0.79 | (dd) | Disk (Uninterruptible) | write | /backup/tanel/test (stdout)
7 | 0.07 | (dd) | Disk (Uninterruptible) | [running] |
5 | 0.05 | (dd) | Running (ON CPU) | write | /backup/tanel/test (stdout)
4 | 0.04 | (dd) | Disk (Uninterruptible) | read | /reco/fio/mmapfile.0.0 (stdin)
3 | 0.03 | (dd) | Running (ON CPU) | [running] |
2 | 0.02 | (dd) | Running (ON CPU) | read | /reco/fio/mmapfile.0.0 (stdin)
```
Linux Process Snapper

$ sudo psn -G syscall, wchan -r -p "sync|kworker"

Linux Process Snapper v0.11 by Tanel Poder [https://tp.dev/psnapper]
Sampling /proc/stat, syscall, wchan for 5 seconds... finished.

---

<table>
<thead>
<tr>
<th>samples</th>
<th>avg_threads</th>
<th>comm</th>
<th>state</th>
<th>syscall</th>
<th>wchan</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.00</td>
<td>(sync)</td>
<td>Disk (Uninterruptible)</td>
<td>sync</td>
<td>wb_wait_for_completion</td>
</tr>
<tr>
<td>98</td>
<td>0.98</td>
<td>(kworker/u66:0)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>wait_barrier</td>
</tr>
<tr>
<td>82</td>
<td>0.82</td>
<td>(md10_resync)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>raise_barrier</td>
</tr>
<tr>
<td>15</td>
<td>0.15</td>
<td>(md10_resync)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>md_do_sync</td>
</tr>
<tr>
<td>3</td>
<td>0.03</td>
<td>(kworker/29:2)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>rpm_resume</td>
</tr>
<tr>
<td>3</td>
<td>0.03</td>
<td>(md10_resync)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>raid10_sync_request</td>
</tr>
<tr>
<td>2</td>
<td>0.02</td>
<td>(kworker/1:0)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>hub_event</td>
</tr>
<tr>
<td>2</td>
<td>0.02</td>
<td>(kworker/29:2)</td>
<td>Disk (Uninterruptible)</td>
<td>read</td>
<td>msleep</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>(kworker/20:1H)</td>
<td>Running (ON CPU)</td>
<td>read</td>
<td>worker_thread</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>(kworker/30:0)</td>
<td>Running (ON CPU)</td>
<td>[userland]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>(kworker/6:0)</td>
<td>Running (ON CPU)</td>
<td>[userland]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>(kworker/u66:0)</td>
<td>Running (ON CPU)</td>
<td>[userland]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>(kworker/u66:0)</td>
<td>Running (ON CPU)</td>
<td>read</td>
<td>wait_barrier</td>
</tr>
</tbody>
</table>
Always-on profiling of production systems?

• **0x.tools**
  - [https://0x.tools](https://0x.tools)
  - [https://twitter.com/0xtools](https://twitter.com/0xtools)
  - Open Source (GPLv3)
  - Low-footprint & low-overhead (no large dependencies)

  • **xcapture** – samples `/proc` states like pSnapper
  • **run_xcpu.sh** – uses **perf** for on-CPU stack sampling at 1 Hz

• Always-on low-frequency sampling of on-CPU & thread sleep samples
  • xcapture outputs hourly `.csv` files ("query" with anything)
  • perf logs can be used just with perf report `-i xcpu.20201201100000`
Thank you!

- **Blog, Tools, Videos, Articles**
  - [https://tanelpoder.com/categories/linux](https://tanelpoder.com/categories/linux)
  - [https://tanelpoder.com/videos](https://tanelpoder.com/videos)
  - [https://0x.tools](https://0x.tools)

- **Events, Hacking Sessions, Online Training**
  - [https://tanelpoder.com/events/](https://tanelpoder.com/events/)

- **Contact**
  - Blog: [tanelpoder.com](http://tanelpoder.com)
  - Twitter: [twitter.com/TanelPoder](http://twitter.com/TanelPoder)
  - Questions: [tanel@tanelpoder.com](mailto:tanel@tanelpoder.com)