How to Surprise by being a Linux Performance "know-it-all"
Agenda

- Tools are your swiss army knife
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat
Agenda

- Tools are your swiss army knife
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat
Everything was nice and easy up to now, → be Ready for Take-Off
Agenda

- Your swiss army knife for the complex cases
  - **Pidstat** – per process statistics
  - **Slabtop** – kernel memory pool consumption
  - **Lsof** – check file flags of open files
  - **Blktrace** – low level disk I/O analysis
  - **Hyptop** – cross guest cpu consumption monitor
  - **Iptraf** - network traffic monitor
  - **Dstat** – very configurable live system overview
  - **Irqstats** – check irq amount and cpu distribution
  - **Smem** – per process/per mapping memory overview
  - **Jinsight** – Java method call stack analysis
  - **Htop** – top on steroids
  - **Strace** – system call statistics
  - **Ltrace** – library call statistics
  - **Kernel tracepoints** – get in-depth timing inside the kernel
  - **Vmstat** – virtual memory statistics
  - **Sysstat** – full system overview
  - **Iostat** – I/O related statistics
  - **Dasdstat** – disk statistics
  - **scsi statistics** – disk statistics
  - **Perf** – hw counters, tracepoint based evaluations, profiling to find hotspots
  - **Valgrind** – in depth memory/cache analysis and leak detection
  - **Java Health Center** – high level java overview and monitoring
  - **Java Garbage Collection and Memory visualizer** – in depth gc analysis
  - **Netstat** – network statistics and overview
  - **Socket Statistics** – extended socket statistics
  - **top / ps** – process overview
  - **Icastats / Iszcrypt** – check usage of crypto hw support
  - **Lsluns / multipath** – check multipath setup
  - **Lsqeth** – check hw checksumming and buffer count
  - **Ethtool** – check offloading functions
  - ** Collectl** – full system monitoring
  - **Ftrace** – kernel function tracing
  - **Lttng** – complex latency tracing infrastructure (no s390 support yet)
  - **Ziomon** – Analyze FCP setup and I/O
  - **Systemtap** – another kernel tracing infrastructure
  - **Wireshark / Tcpdump** – analyze network traffic in depth
Agenda

- Your (little) swiss army knife for the complex cases
  - Pidstat
  - Slabtop
  - Smem
  - Valgrind
  - Lsof
  - Blktrace
  - Hyptop
  - Iptraf

60 min ?otto: 8 of 0x28
Non-legal Disclaimer

- This is an introduction and cheat sheet
  - Know what is out there
  - What could be useful in which case
  - How could I debug even further

- These descriptions are not full explanations
  - Most tools could get at least 1-2 presentations on their own
  - Don't start using them without reading howtos / man pages

- This is not about monitoring
  - Some tools used to start performance analysis CAN be monitors, but that's not part of the presentation
General thoughts on performance tools

- Things that are always to consider
  - Monitoring can impact the system
  - Most data gathering averages over a certain period of time
    → this flattens peaks
  - Start with defining the problem
    - which parameter(s) from the application/system indicates the problem
    - which range is considered as bad, what is considered as good
  - monitor the good case and save the results
    - comparisons when a problem occurs can save days and weeks

- Staged approach saves a lot of work
  - Try to use general tools to isolate the area of the issue
  - Create theories and try to quickly verify/falsify them
  - Use advanced tools to debug the identified area
## Orientation - where to go

<table>
<thead>
<tr>
<th>Tool</th>
<th>1st overview</th>
<th>CPU cons.</th>
<th>latencies</th>
<th>Hot spots</th>
<th>Disk I/O</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>top / ps</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sysstat</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>vmstat</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iostat</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dasdstat</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>scsistat</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>netstat / ss</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>htop / dstat / pidstat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irqstats</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>strace / ltrace</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>hytop</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perf</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jinsight</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Health Center</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMVC</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blktrace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsof</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>valgrind</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>smem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>slabtop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iptraf</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>tracepoints</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
PIDSTAT

- Characteristics: Easy to use extended per process statistics
- Objective: Identify processes with peak activity
- Usage: `pidstat [-w|-r|-d]`
- Package: RHEL: sysstat SLES: sysstat

- Shows
  - `-w` context switching activity and if it was voluntary
  - `-r` memory statistics, especially minor/major faults per process
  - `-d` disk throughput per process

- Hints
  - Also useful if run as background log due to its low overhead
    - Good extension to sadc in systems running different applications/services
  - `-p <pid>` can be useful to track activity of a specific process
Pidstat examples

12:46:18 PM       PID   cswch/s nvcswhch/s  Command
12:46:18 PM         3      2.39      0.00  smbd
12:46:18 PM         4      0.04      0.00  sshd
12:46:18 PM      1073    123.42    180.18  Xorg

Voluntarily / Involuntary

12:47:51 PM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
12:47:51 PM       985      0.06      0.00   15328   3948   0.10  smbd
12:47:51 PM       992      0.04      0.00    5592   2152   0.05  sshd
12:47:51 PM      1073    526.41      0.00 1044240 321512   7.89  Xorg

Faults per process

12:49:18 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
12:49:18 PM       330      0.00      1.15      0.00  sshd
12:49:18 PM      2899      4.35      0.09      0.04  notes2
12:49:18 PM      3045     23.43      0.01      0.00  audacious2

How much KB disk I/O per process
Slabtop

- Characteristics: live profiling of kernel memory pools
- Objective: Analyze kernel memory consumption
- Usage: `slabtop`
- Package: RHEL: procps SLES: procps

- Shows
  - Active / Total object number/size
  - Objects per Slab
  - Object Name and Size
  - Objects per Slab

- Hints
  - -o is one time output e.g. to gather debug data
  - Despite slab/slob/slub in kernel its always slabtop
Slabtop - example

Active / Total Objects (% used) : 2436408 / 2522983 (96.6%)
Active / Total Slabs (% used) : 57999 / 57999 (100.0%)
Active / Total Caches (% used) : 75 / 93 (80.6%)
Active / Total Size (% used) : 793128.19K / 806103.80K (98.4%)
Minimum / Average / Maximum Object : 0.01K / 0.32K / 8.00K

<table>
<thead>
<tr>
<th>OBJS</th>
<th>ACTIVE</th>
<th>USE</th>
<th>OBJ SIZE</th>
<th>SLABS</th>
<th>OBJ/SLAB</th>
<th>CACHE SIZE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>578172</td>
<td>578172</td>
<td>100%</td>
<td>0.19K</td>
<td>13766</td>
<td>42</td>
<td>110128K</td>
<td>dentry</td>
</tr>
<tr>
<td>458316</td>
<td>458316</td>
<td>100%</td>
<td>0.11K</td>
<td>12731</td>
<td>36</td>
<td>50924K</td>
<td>sysfs_dir_cache</td>
</tr>
<tr>
<td>368784</td>
<td>368784</td>
<td>100%</td>
<td>0.61K</td>
<td>7092</td>
<td>52</td>
<td>226944K</td>
<td>proc_inode_cache</td>
</tr>
<tr>
<td>113685</td>
<td>113685</td>
<td>100%</td>
<td>0.10K</td>
<td>2915</td>
<td>39</td>
<td>11660K</td>
<td>buffer_head</td>
</tr>
<tr>
<td>113448</td>
<td>113448</td>
<td>100%</td>
<td>0.55K</td>
<td>1956</td>
<td>58</td>
<td>62592K</td>
<td>inode_cache</td>
</tr>
<tr>
<td>111872</td>
<td>44251</td>
<td>39%</td>
<td>0.06K</td>
<td>1748</td>
<td>64</td>
<td>6992K</td>
<td>kmalloc-64</td>
</tr>
<tr>
<td>54688</td>
<td>50382</td>
<td>92%</td>
<td>0.25K</td>
<td>1709</td>
<td>32</td>
<td>13672K</td>
<td>kmalloc-256</td>
</tr>
<tr>
<td>40272</td>
<td>40239</td>
<td>99%</td>
<td>4.00K</td>
<td>5034</td>
<td>8</td>
<td>161088K</td>
<td>kmalloc-4096</td>
</tr>
<tr>
<td>39882</td>
<td>39882</td>
<td>100%</td>
<td>0.04K</td>
<td>391</td>
<td>102</td>
<td>1564K</td>
<td>ksm_stable_node</td>
</tr>
<tr>
<td>38505</td>
<td>36966</td>
<td>96%</td>
<td>0.62K</td>
<td>755</td>
<td>51</td>
<td>24160K</td>
<td>shmem_inode_cache</td>
</tr>
<tr>
<td>37674</td>
<td>37674</td>
<td>100%</td>
<td>0.41K</td>
<td>966</td>
<td>39</td>
<td>15456K</td>
<td>dm_rq_target_io</td>
</tr>
</tbody>
</table>

- How is kernel memory managed by the slab allocator used
  - Named memory pools or Generic kmalloc pools
  - Active/total objects and their size
  - growth/shrinks of caches due to workload adaption
smem

- Characteristics: Memory usage details per process/mapping
- Objective: Where is userspace memory really used
- Usage: `smem -tk -c "pid user command swap vss uss pss rss"`
- `smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"

- Package: RHEL: n/a SLES: n/a WWW http://www.selenic.com/smem/
- Shows
  - Pid, user, Command or Mapping, Count, Pid
  - Memory usage in categories vss, uss, rss, pss and swap

- Hints
  - Has visual output (pie charts) and filtering options as well
  - No support for huge pages or transparent huge pages (kernel interface missing)
smem – process overview

smem -tk -c "pid user command swap vssuss pss rss"

<table>
<thead>
<tr>
<th>PID</th>
<th>User</th>
<th>Command</th>
<th>Swap</th>
<th>VSS</th>
<th>USS</th>
<th>PSS</th>
<th>RSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1860</td>
<td>root</td>
<td>/sbin/agetty -s sclp_line0</td>
<td>0</td>
<td>2.1M</td>
<td>92.0K</td>
<td>143.0K</td>
<td>656.0K</td>
</tr>
<tr>
<td>1861</td>
<td>root</td>
<td>/sbin/agetty -s ttysclp0 11</td>
<td>0</td>
<td>2.1M</td>
<td>92.0K</td>
<td>143.0K</td>
<td>656.0K</td>
</tr>
<tr>
<td>493</td>
<td>root</td>
<td>/usr/sbin/atd -f</td>
<td>0</td>
<td>2.5M</td>
<td>172.0K</td>
<td>235.0K</td>
<td>912.0K</td>
</tr>
<tr>
<td>1882</td>
<td>root</td>
<td>/sbin/udevd</td>
<td>0</td>
<td>2.8M</td>
<td>128.0K</td>
<td>267.0K</td>
<td>764.0K</td>
</tr>
<tr>
<td>1843</td>
<td>root</td>
<td>/usr/sbin/crond -n</td>
<td>0</td>
<td>3.4M</td>
<td>628.0K</td>
<td>693.0K</td>
<td>1.4M</td>
</tr>
<tr>
<td>514</td>
<td>root</td>
<td>/bin/dbus-daemon --system -</td>
<td>0</td>
<td>3.2M</td>
<td>700.0K</td>
<td>771.0K</td>
<td>1.5M</td>
</tr>
<tr>
<td>524</td>
<td>root</td>
<td>/sbin/rsyslogd -n -c 5</td>
<td>0</td>
<td>219.7M</td>
<td>992.0K</td>
<td>1.1M</td>
<td>1.9M</td>
</tr>
<tr>
<td>2171</td>
<td>root</td>
<td>./hhhtest</td>
<td>0</td>
<td>5.7G</td>
<td>1.0M</td>
<td>1.2M</td>
<td>3.2M</td>
</tr>
<tr>
<td>1906</td>
<td>root</td>
<td>-bash</td>
<td>0</td>
<td>103.8M</td>
<td>1.4M</td>
<td>1.5M</td>
<td>2.1M</td>
</tr>
<tr>
<td>2196</td>
<td>root</td>
<td>./hhhtest</td>
<td>0</td>
<td>6.2G</td>
<td>2.0M</td>
<td>2.2M</td>
<td>3.9M</td>
</tr>
<tr>
<td>1884</td>
<td>root</td>
<td>sshd: root@pts/0</td>
<td>0</td>
<td>13.4M</td>
<td>1.4M</td>
<td>2.4M</td>
<td>4.2M</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>/sbin/init</td>
<td>0</td>
<td>5.8M</td>
<td>2.9M</td>
<td>3.0M</td>
<td>3.9M</td>
</tr>
<tr>
<td>2203</td>
<td>root</td>
<td>/usr/bin/python /usr/bin/sm</td>
<td>0</td>
<td>109.5M</td>
<td>6.1M</td>
<td>6.2M</td>
<td>6.9M</td>
</tr>
</tbody>
</table>

- How much of a process is:
  - Swap - Swapped out
  - VSS - Virtually allocated
  - USS - Really unique
  - RSS - Resident
  - PSS - Resident accounting a proportional part of shared memory
smem – mappings overview

```
smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"
```

<table>
<thead>
<tr>
<th>Map</th>
<th>Count</th>
<th>PIDs</th>
<th>Swap</th>
<th>VSS</th>
<th>USS</th>
<th>RSS</th>
<th>PSS</th>
<th>AVGRSS</th>
<th>AVGPPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[stack:531]</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8.0M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[vdso]</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>200.0K</td>
<td>0</td>
<td>132.0K</td>
<td>0</td>
<td>5.0K</td>
<td>0</td>
</tr>
<tr>
<td>/dev/zero</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2.5M</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
</tr>
<tr>
<td>/usr/lib64/sasl2/libsasldb.so.2.0.23</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>28.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
</tr>
<tr>
<td>/bin/dbus-daemon</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>404.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
</tr>
<tr>
<td>/usr/sbin/sshd</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>1.2M</td>
<td>248.0K</td>
<td>728.0K</td>
<td>488.0K</td>
<td>364.0K</td>
<td>244.0K</td>
</tr>
<tr>
<td>/bin/systemd</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>768.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
</tr>
<tr>
<td>/bin/bash</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1.0M</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
</tr>
<tr>
<td>[stack]</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>4.1M</td>
<td>908.0K</td>
<td>976.0K</td>
<td>918.0K</td>
<td>39.0K</td>
<td>36.0K</td>
</tr>
<tr>
<td>/lib64/libc-2.14.1.so</td>
<td>75</td>
<td>25</td>
<td>0</td>
<td>40.8M</td>
<td>440.0K</td>
<td>9.3M</td>
<td>1.2M</td>
<td>382.0K</td>
<td>48.0K</td>
</tr>
<tr>
<td>/lib64/libcrypto.so.1.0.0j</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>7.0M</td>
<td>572.0K</td>
<td>2.0M</td>
<td>1.3M</td>
<td>501.0K</td>
<td>321.0K</td>
</tr>
<tr>
<td>[heap]</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>8.3M</td>
<td>6.4M</td>
<td>6.9M</td>
<td>6.6M</td>
<td>444.0K</td>
<td>422.0K</td>
</tr>
<tr>
<td>&lt;anonymous&gt;</td>
<td>241</td>
<td>25</td>
<td>0</td>
<td>55.7G</td>
<td>20.6M</td>
<td>36.2M</td>
<td>22.3M</td>
<td>1.4M</td>
<td>913.0K</td>
</tr>
</tbody>
</table>

### How much of a mapping is:
- Swap - Swapped out
- VSS - Virtually allocated
- USS - Really unique
- RSS - Resident
- PSS - Resident accounting a proportional part of shared memory
- Averages as there can be multiple mappers
Example of a memory distribution Visualization (many options)

But before thinking of monitoring be aware that the proc/#pid/smaps interface is an expensive one
Valgrind

- Characteristics: in-depth memory analysis
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: valgrind [program]
- Package: RHEL: valgrind SLES: valgrind

- Shows
  - Memory leaks
  - Cache profiling
  - Heap profiling

- Hints
  - Runs on binaries, therefore easy to use
  - Debug Info not required but makes output more useful
Valgrind Overview

- Technology is based on a JIT (Just-in-Time Compiler)
- Intermediate language allows debugging instrumentation

Diagram:

- Binary
  - Translation into IR
  - Instrumentation
  - Translation to machine code

- New binary

- System call wrapper

- System call interface

- Libraries
  - Replace some of the library calls by using a preload library

- Valgrind
  - Translation into IR
  - Instrumentation

- Kernel
Valgrind – sample output of “memcheck”

# valgrind buggy_program
==2799== Memcheck, a memory error detector
==2799== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2799== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2799== Command: buggy_program
==2799==
==2799== HEAP SUMMARY:
==2799==    in use at exit: 200 bytes in 2 blocks
==2799==    total heap usage: 2 allocs, 0 frees, 200 bytes allocated
==2799==
==2799== LEAK SUMMARY:
==2799==      definitely lost: 100 bytes in 1 blocks
==2799==    indirectly lost: 0 bytes in 0 blocks
==2799==      possibly lost: 0 bytes in 0 blocks
==2799==    still reachable: 100 bytes in 1 blocks
==2799==    suppressed: 0 bytes in 0 blocks
==2799== Rerun with --leak-check=full to see details of leaked memory

- Important parameters:
  -- --leak-check=full
  -- --track-origins=yes
Valgrind - Tools

- Several tools
  - Memcheck (default): detects memory and data flow problems
  - CacheGrind: cache profiling
  - Massif: heap profiling
  - Helgrind: thread debugging
  - DRD: thread debugging
  - None: no debugging (for valgrind JIT testing)
  - Callgrind: codeflow and profiling

- Tool can be selected with –tool=xxx
- System z support since version 3.7 (SLES-11-SP2)
- Backports into 3.6 (SLES-10-SP4, RHEL6-U1)
Valgrind - Good to know

- No need to recompile, but
  - Better results with debug info
  - Gcc option -O0 might result in more findings (the compiler might hide some errors)
  - Gcc option -fno-built-in might result in more findings

- --trace-children=yes will also debug child processes

- Setuid programs might cause trouble
  - Valgrind is the process container (→ no setuid)
  - Possible solution: remove setuid and start as the right user, check documentation for other ways

- The program will be slower
  - 5-30 times slower for memcheck
lsof

- Characteristics: list of open files plus extra details
- Objective: which process accesses which file in which mode
- Usage: `lsof +fg`
- Package: RHEL: lsof SLES: lsof

- Shows
  - List of files including sockets, directories, pipes
  - User, Command, Pid, Size, Device
  - File Type and File Flags

- Hints
  - `+fg` reports file flags which can provide a good cross check opportunity
### lsof - example

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PID</th>
<th>TID</th>
<th>USER</th>
<th>FD</th>
<th>TYPE</th>
<th>FILE-FLAG</th>
<th>DEVICE</th>
<th>SIZE/OFF</th>
<th>NODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>mem</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>165000</td>
<td>881893</td>
<td>/usr/lib64/ld-2.16.so</td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>0r</td>
<td>CHR</td>
<td>LG</td>
<td>1,3</td>
<td>0t0</td>
<td>2051</td>
<td>/dev/null</td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>1u</td>
<td>unix</td>
<td>RW 0x00000001f1ba02000</td>
<td>0t0</td>
<td>106645 socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>2u</td>
<td>unix</td>
<td>RW 0x00000001f1ba02000</td>
<td>0t0</td>
<td>106645 socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>4r</td>
<td>a_inode</td>
<td>0x80000</td>
<td>0,9</td>
<td>0</td>
<td>6675 inotify</td>
<td></td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>5u</td>
<td>unix</td>
<td>RW,0x80000 0x00000001f5d3ad000</td>
<td>0t0</td>
<td>68545 socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>cwd</td>
<td>DIR</td>
<td></td>
<td>94,1</td>
<td>4096</td>
<td>16321</td>
<td>/root</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>rtd</td>
<td>DIR</td>
<td></td>
<td>94,1</td>
<td>4096</td>
<td>2</td>
<td>/</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>txt</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>70568</td>
<td>1053994</td>
<td>/usr/bin/dd</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>mem</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>165000</td>
<td>881893</td>
<td>/usr/lib64/ld-2.16.so</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>0r</td>
<td>CHR</td>
<td>LG</td>
<td>1,9</td>
<td>0t0</td>
<td>2055</td>
<td>/dev/urandom</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>1w</td>
<td>REG</td>
<td>W,DIR,LG</td>
<td>94,1</td>
<td>5103616</td>
<td>16423</td>
<td>/root/test</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>2u</td>
<td>CHR</td>
<td>RW,LG</td>
<td>136,2</td>
<td>0t0</td>
<td>5</td>
<td>/dev/pts/2</td>
</tr>
</tbody>
</table>

- You can filter that per application or per file
  - Fd holds fdnumber, type, characteristic and lock information
    - File descriptors can help to read strace/ltrace output
  - Flags can be good to confirm e.g. direct IO, async IO
  - Size (e.g. mem) or offset (fds), name, ...
BLKTRACE

- Characteristics: High detail info of the block device layer actions
- Objective: Understand whats going with your I/O in the kernel and devices
- Usage: blktrace -d [device(s)]
- Then: blkparse -st [commontracefilepart]
- Package: RHEL: blktrace SLES: blktrace
- Shows
  - Events like merging, request creation, I/O submission, I/O completion, ...
  - Timestamps and disk offsets for each event
  - Associated task and executing CPU
  - Application and CPU summaries

- Hints
  - Filter masks allow lower overhead if only specific events are of interest
  - Has an integrated client/server mode to stream data away
    - Avoids extra disk I/O on a system with disk I/O issues
Blktrace – when is it useful

- Often it's easy to identify that I/O is slow, but
  → Where?
  → Because of what?

- Blocktrace allows to
  - Analyze Disk I/O characteristics like sizes and offsets
    • Maybe your I/O is split in a layer below
  - Analyze the timing with details about all involved Linux layers
    • Often useful to decide if HW or SW causes stalls
  - Summaries per CPU / application can identify imbalances
Blktrace - events

Common:
A -- remap For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what.
Q -- queued This notes intent to queue i/o at the given location. No real requests exists yet.
G -- get request To send any type of request to a block device, a struct request container must be allocated first.
I -- inserted A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time.
D -- issued A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver.
C -- complete A previously issued request has been completed. The output will detail the sector and size of that request, as well as the success or failure of it.

Plugging & Merges:
P -- plug When i/o is queued to a previously empty block device queue, Linux will plug the queue in anticipation of future I/Os being added before this data is needed.
U -- unplug Some request data already queued in the device, start sending requests to the driver. This may happen automatically if a timeout period has passed (see next entry) or if a number of requests have been added to the queue.
Recent kernels associate the queue with the submitting task and unplug also on a context switch.
T -- unplug due to timer If nobody requests the i/o that was queued after plugging the queue, Linux will automatically unplug it after a defined period has passed.
M -- back merge A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together.
F -- front merge Same as the back merge, except this i/o ends where a previously inserted requests starts.

Special:
B -- bounced The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU.
S -- sleep No available request structures were available, so the issuer has to wait for one to be freed.
X -- split On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid/dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o.
Blktrace - events

Common:
A -- remap For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what.
Q -- queued This notes intent to queue i/o at the given location. No real requests exist yet.
G -- get request To send any type of request to a block device, a struct request container must be allocated first.
I -- inserted A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time.
D -- issued A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver.
C -- complete A previously issued request has been completed. The output will detail the sector and size of that request, as well as the success or failure of it.

Plugging & Merges:
P -- plug When i/o is queued to a previously empty block device queue, Linux will plug the device in anticipation of future I/Os being added before this data is needed.
U -- unplug Some request data already passed (see next entry) or if a number of requests have been added to the queue. Recent kernels associate the queue with the submitting task, and unplug also on a context switch.
T -- unplug due to timer If nobody requested the i/o that was queued after plugging the queue, Linux will automatically unplug it after a defined period has passed.
M -- back merge A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together.
F -- front merge Same as the back merge, except this i/o ends where a previously inserted request starts.

Special:
B -- bounced The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU.
S -- sleep No available request structures were available, so the issuer has to wait for one to be freed.
X -- split On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid or dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o.

Good as documentation, but hard to understand/remember
Block device layer – events (simplified)

- **App / A / X**
  - Need to **G**enerate a new request

- **Q**
  - **N**
  - **mergeable**
    - **Y**
    - **M / F**
      - Merge with an existing request
  - **G**
  - **I**
    - **P**
      - **U**
        - **Unplug on upper limit (stream) or Time reached / submitting task ctx switch**
      - **D**
        - **Dispatch from block device layer to device driver**
      - **C**
        - Time from **Dispatch to Complete**
    - **Plug queue and wait a bit if following requests can be merged**

Add device driver info like dasdstat and scsi sysfs statistics to fill this gap and gain a full circle latency insight.
### Example Case

- The snippet shows a lot of 4k requests (8x512 byte sectors)
  - We expected the I/O to be 32k
- Each one is dispatched separately (no merges)
  - This caused unnecessary overhead and slow I/O

<table>
<thead>
<tr>
<th>Maj/Min</th>
<th>CPU</th>
<th>Seq-nr</th>
<th>sec.nsec</th>
<th>pid</th>
<th>Action</th>
<th>RWBS</th>
<th>sect + size</th>
<th>map source / task</th>
</tr>
</thead>
<tbody>
<tr>
<td>94,4</td>
<td>27</td>
<td>21</td>
<td>0.059363692</td>
<td>18994</td>
<td>A</td>
<td>R 20472832 + 8 &lt;- (94,5) 20472640</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>22</td>
<td>0.059364630</td>
<td>18994</td>
<td>Q</td>
<td>R 20472832 + 8 [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>23</td>
<td>0.059365286</td>
<td>18994</td>
<td>G</td>
<td>R 20472832 + 8 [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>24</td>
<td>0.059365598</td>
<td>18994</td>
<td>I</td>
<td>R 20472832 + 8 ( 312) [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>25</td>
<td>0.059366255</td>
<td>18994</td>
<td>D</td>
<td>R 20472832 + 8 ( 657) [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>26</td>
<td>0.059370223</td>
<td>18994</td>
<td>A</td>
<td>R 20472840 + 8 &lt;- (94,5) 20472648</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>27</td>
<td>0.059370442</td>
<td>18994</td>
<td>Q</td>
<td>R 20472840 + 8 [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>28</td>
<td>0.059370880</td>
<td>18994</td>
<td>G</td>
<td>R 20472840 + 8 [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>29</td>
<td>0.059371067</td>
<td>18994</td>
<td>I</td>
<td>R 20472840 + 8 ( 187) [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94,4</td>
<td>27</td>
<td>30</td>
<td>0.059371473</td>
<td>18994</td>
<td>D</td>
<td>R 20472840 + 8 ( 406) [qemu-kvm]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**blktrace**

- **Example Case**
  - Analysis turned out that the I/O was from the swap code
    - Same offsets were written by kswapd
  - A recent code change there disabled the ability to merge I/O
  - The summary below shows the difference after a fix

<table>
<thead>
<tr>
<th></th>
<th>Total initially</th>
<th></th>
<th>Total after Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads Queued:</td>
<td>560,888, 2,243MiB</td>
<td>Writes Queued:</td>
<td>226,242, 904,968KiB</td>
</tr>
<tr>
<td>Read Dispatches:</td>
<td>544,701, 2,243MiB</td>
<td>Write Dispatches:</td>
<td>159,318, 904,968KiB</td>
</tr>
<tr>
<td>Reads Requeued:</td>
<td>0</td>
<td>Writes Requeued:</td>
<td>0</td>
</tr>
<tr>
<td>Reads Completed:</td>
<td>544,716, 2,243MiB</td>
<td>Writes Completed:</td>
<td>159,321, 904,980KiB</td>
</tr>
<tr>
<td>Read Merges:</td>
<td>16,187, 64,748KiB</td>
<td>Write Merges:</td>
<td>61,744, 246,976KiB</td>
</tr>
<tr>
<td>IO unplugs:</td>
<td>149,614</td>
<td>Timer unplugs:</td>
<td>2,940</td>
</tr>
</tbody>
</table>

|                      | 734,315, 2,937MiB | Writes Queued:       | 300,188, 1,200MiB |
| Read Dispatches:     | 214,972, 2,937MiB | Write Dispatches:    | 215,176, 1,200MiB |
| Reads Requeued:      | 0               | Writes Requeued:     | 0               |
| Reads Completed:     | 214,971, 2,937MiB | Writes Completed:    | 215,177, 1,200MiB |
| Read Merges:         | 519,343, 2,077MiB | Write Merges:        | 73,325, 293,300KiB |
| IO unplugs:          | 337,130         | Timer unplugs:       | 11,184          |
Hyptop

- Characteristics: Easy to use Guest/LPAR overview
- Objective: Check CPU and overhead statistics of your and sibling images
- Usage: `hyptop`
- Package: RHEL: s390utils-base SLES: s390-tools

- Shows
  - CPU load & Management overhead
  - Memory usage (only under zVM)
  - Can show image overview or single image details

- Hints
  - Good “first view” tool for linux admins that want to look “out of their linux”
  - Requirements:
    - For z/VM the Guest needs Class B
    - For LPAR “Global performance data control” checkbox in HMC
Hyptop

Why are exactly 4 CPUs used in all 6 CPU guests

All these do not fully utilize their 2 CPUs

No peaks in service guests

LPAR images would see other LPARs

memuse = resident

service guest weights
IPTRAFF

- Characteristics: Live information on network devices / connections
- Objective: Filter and format network statistics
- Usage: iptraf
- Package: RHEL: iptraf SLES: iptraf

- Shows
  - Details per Connection / Interface
  - Statistical breakdown of ports / packet sizes
  - LAN station monitor

- Hints
  - Can be used for background logging as well
    - Use SIGUSR1 and logrotate to handle the growing amount of data
  - Knowledge of packet sizes important for the right tuning
iptraf

- Questions that usually can be addressed
  - Connection behavior overview
  - Do you have peaks in your workload characteristic
  - Who does your host really communicate with

- Comparison to wireshark
  - Not as powerful, but much easier and faster to use
  - Lower overhead and no sniffing needed (often prohibited)
Cachestat

- Characteristics: Simple per page views of caching
- Objective: Detect what parts of a file are in page cache
- Usage: Write – or search for example code
- Package: n/a (pure code around the mincore system call)

- Shows
  - How much of a file is in cache

- Hints
  - We are now going from unsupported to non existent packages
  - Still the insight can be so useful, it is good to know
**Cachestat usage**

```
./cachestat -v ../Music/mysong.flac
```

pages in cache: 445/12626 (3.5%) [filesize=50501.0K, pagesize=4K]

cache map:

```
0: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
32: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|

[...]
384: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
416: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
448: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

[...]
12576: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
12608: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
```

- Here I show how much of a file is in cache while playing a song
  - You'll see readahead here
  - You'll also see the last block is almost always read in this case
# Orientation - where to go

<table>
<thead>
<tr>
<th>Tool</th>
<th>1st overview</th>
<th>CPU cons.</th>
<th>latencies</th>
<th>Hot spots</th>
<th>Disk I/O</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>top / ps</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sysstat</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmstat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iostat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>dasdstat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>scsistat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>netstat / ss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>htop / dstat / pidstat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irqstats</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strace / ltrace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>hytop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>perf</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jinsight</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Health Center</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMVC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>blktrace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>lsof</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>valgrind</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>smem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>slabtop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>iptraf</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tracepoints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis

This is like "Heisenbergs uncertainty principle". The more time you put into preparation, the less time you'll need to solve issues. They fundamentally are never both huge, what do you prefer?
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis

This is like “Heisenbergs uncertainty principle”
The more time you put into preparation, the less time you'll need to solve issues
They fundamentally are never both huge, What do you prefer?

.. combined with Murphy: there is always a bug
That means with enough preparation you'll surely get a bug that no one can fix
Ultimate Swiss Army knife

- The one you should always have → IBM System z Enterprise
Appendix Preview covering even more tools

- Further complex tools
  - Dstat – very configurable live system overview
  - Iqrsstats – check irq amount and cpu distribution
  - Java Health Center – high level java overview and monitoring
  - Java Garbage Collection and Memory visualizer – in depth gc analysis
  - Jinsight – Java method call stack analysis
  - Perf – hw counters, tracepoint based evaluations, profiling to find hotspots
  - Htop – top on steroids
  - Strace – system call statistics
  - Ltrace – library call statistics
  - Kernel tracepoints – get in-depth timing inside the kernel
  - Icastats / Iszcrypt – check usage of crypto hw support
  - Lsluns / multipath – check multipath setup
  - Ethtool – check offloading functions
  - Ziomon – analyze FCP setup and I/O
  - Systemtap – another kernel tracing infrastructure
  - Wireshark / tcpdump – analyze network traffic in depth

- Entry level Tools
  - Vmstat – virtual memory statistics
  - Sysstat – full system overview
  - Iostat – I/O related statistics
  - Dasdstat – disk statistics
  - scsi statistics – disk statistics
  - Netstat – network statistics and overview
  - Socket Statistics – extended socket statistics
  - top / ps – process overview
  - Lsqeth – check hw checksumming and buffer count

- Further tools - (no slides yet)
  - Collectl – full system monitoring
  - Ftrace – kernel function tracing
  - Lttng – complex latency tracing infrastructure (no s390 support yet)
Questions

- Further information is available at
  - Linux on System z – Tuning hints and tips
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/

Christian Ehrhardt  
Linux on System z  
Performance Evaluation

Research & Development  
Schöneicher Strasse 220  
71032 Böblingen, Germany

ehrhardt@de.ibm.com
STRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing system calls of a program
- Usage: `strace -p [pid of target program]`
- Package: RHEL: strace SLES: strace

- Shows
  - Identify kernel entries called more often or taking too long
    - Can be useful if you search for increased system time
  - Time in call (`-T`)
  - Relative timestamp (`-r`)

- Hints
  - The option “-c” allows medium overhead by just tracking counters and durations
# strace - example

To share to rate importance of system calls, we can use the `strace` command with options. Here's an example:

```
strace -cf -p 26802
```

This command attaches `strace` to process 26802, allowing us to monitor its calls. When you're ready to stop monitoring, you can interrupt the process to detach `strace`.

```
^Process 26802 detached
```

After the process is detached, we can see the summary of system calls with their time spent, average usecs per call, number of calls, and error counts. The table below illustrates this:

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>errors</th>
<th>syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.43</td>
<td>0.007430</td>
<td>17</td>
<td>450</td>
<td></td>
<td>read</td>
</tr>
<tr>
<td>24.33</td>
<td>0.003094</td>
<td>4</td>
<td>850</td>
<td>210</td>
<td>access</td>
</tr>
<tr>
<td>5.53</td>
<td>0.000703</td>
<td>4</td>
<td>190</td>
<td>10</td>
<td>open</td>
</tr>
<tr>
<td>4.16</td>
<td>0.000529</td>
<td>3</td>
<td>175</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>2.97</td>
<td>0.000377</td>
<td>2</td>
<td>180</td>
<td></td>
<td>munmap</td>
</tr>
<tr>
<td>1.95</td>
<td>0.000248</td>
<td>1</td>
<td>180</td>
<td></td>
<td>close</td>
</tr>
<tr>
<td>1.01</td>
<td>0.000128</td>
<td>1</td>
<td>180</td>
<td></td>
<td>mmap</td>
</tr>
<tr>
<td>0.69</td>
<td>0.000088</td>
<td>18</td>
<td>5</td>
<td></td>
<td>fdatasync</td>
</tr>
<tr>
<td>0.61</td>
<td>0.000078</td>
<td>0</td>
<td>180</td>
<td></td>
<td>fstat</td>
</tr>
<tr>
<td>0.13</td>
<td>0.000017</td>
<td>3</td>
<td>5</td>
<td></td>
<td>pause</td>
</tr>
</tbody>
</table>

The total time spent on system calls is 0.012715 seconds, with 2415 total calls and 225 errors. This table helps us identify which system calls are taking up the most time and may need further investigation to determine if they are taking too long or if they are failing frequently.
LTRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing library calls of a program
- Usage: `ltrace -p [pid of target program]`
- Package: RHEL: ltrace SLES: ltrace

- Shows
  - Identify library calls that are too often or take too long
    - Good if you search for additional user time
    - Good if things changed after upgrading libs
  - Time in call (\(-T\))
  - Relative timestamp (\(-r\))

- Hints
  - The option "\(-c\)" allows medium overhead by just tracking counters and durations
  - The option -S allows to combine ltrace and strace
**ltrace - example**

```
1trace -cf -p 26822
% time   seconds  usecs/call   calls   function
------ ----------- ---------- --------- --------------------
98.33   46.765660     5845707         8 pause
0.94    0.445621          10     42669 strncmp
0.44    0.209839          25      8253 fgets
0.08    0.037737          11      3168 __isoc99_sscanf
0.07    0.031786          20      1530 access
0.04    0.016757          10      1611 strchr
0.03    0.016479          10      1530 snprintf
0.02    0.010467        1163         9 fdatasync
0.02    0.008899          27      324 fclose
0.02    0.007218          21      342 fopen
0.01    0.006239          19      315 write
0.00    0.000565          10        54 strncpy

------ ----------- ---------- --------- --------------------
100.00   47.560161                 59948 total
```
Strace / Ltrace – full trace

- Without -c both tools produce a full detail log
  - Via -f child processes can be traced as well
  - Extra options “-Tr” are useful to search for latencies
    follow time in call / relative timestamp
  - Useful to “read” what exactly goes on when

Example strace'ing a sadc data gatherer
0.000028 write(3, "\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000027 write(3, "\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000026 fdatasync(3)            = 0 <0.002673>
0.002688 pause()                = 0 <3.972935>
3.972957 --- SIGALRM (Alarm clock) @ 0 (0) ---
0.000051 rt_sigaction(SIGALRM, {0x8000314c, [ALRM], SA_RESTART}, 8) = 0 <0.000005>
0.000038 alarm(4)               = 0 <0.000005>
0.000031 sigreturn()            = ? (mask now []) <0.000005>
0.000024 stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0 <0.000007>
0.000034 open("/proc/uptime", O_RDONLY) = 4 <0.000009>
0.000024 fstat(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000005>
0.000029 mmap(NULL, 4096, PROT_READ, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x3fffd20a000 <0.000006>
0.000028 read(4, "11687.70 24836.04\n", 1024) = 18 <0.000010>
0.000027 close(4)               = 0 <0.000006>
0.000020 munmap(0x3fffd20a000, 4096) = 0 <0.000009>
DSTAT

- Characteristics: Live easy to use full system information
- Objective: Flexible set of statistics
- Usage: dstat -tv -aio -disk-util -n -net-packets -i -ipc
  -D total,[diskname] -top-io [...] [interval]
- Short: dstat -tinv
- Package: RHEL: dstat SLES: n/a WWW: http://dag.wieers.com/home-made/dstat/
- Shows
  - Throughput
  - Utilization
  - Summarized and per Device queue information
  - Much more ... it more or less combines several classic tools like iostat and vmstat

- Hints
  - Powerful plug-in concept
    - "--top-io" for example identifies the application causing the most I/Os
  - Colorization allows fast identification of deviations
Dstat – the limit is your screen width

similar to vmstat

similar to iostat
(also per device)

new in live tool
Perf

- Characteristics: Easy to use profiling and kernel tracing
- Objective: Get detailed information where & why CPU is consumed
- Usage: `perf` (to begin with)
- Package: RHEL: perf SLES: perf

- Shows
  - Sampling for CPU hotspots
    - Annotated source code along hotspots
  - CPU event counters
  - Further integrated non-sampling tools

- Hints
  - Without HW support only userspace can be reasonably profiled
  - “successor” of oprofile that is available with HW support (SLES11-SP2)
  - Perf HW support partially upstream, wait for next distribution releases
Perf

- What profiling can and what it can't
  + Search hotspots of CPU consumption worth to optimize
  + List functions according to their usage
  - Search where time is lost (I/O, Stalls)

- Perf is not just a sampling tool
  − Integrated tools to evaluate tracepoints like “perf sched”, “perf timechart”, …
    • Other than real “sampling” this can help to search for stalls
  − Counters provide even lower overhead and report HW and Software events
Perf stat - preparation

- Activate the cpu measurement facility
  - If not you'll encounter this
    Error: You may not have permission to collect stats.
    Consider tweaking /proc/sys/kernel/perf_event_paranoid
    Fatal: Not all events could be opened.
  - Check if its activated
    echo p > /proc/sysrq-trigger
dmesg
    [...] SysRq : Show Regs
    perf.ee05c5: CPU[0] CPUM_CF: ver=1.2 A=000F E=0000 C=0000
    [...]  
    - A = authorized, E=enabled (ready for use), C=controlled (currently running)
    - F = last four bits for basic, problem, crypto and extended set
Perf stat - usage

```
perf stat -B --event=cycles,instructions,r20,r21,r3,r5,sched:sched_wakeup find / -iname "*foobar*"
```

Performance counter stats for 'find / -iname *foobar*':

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>3,623,031,935</td>
</tr>
<tr>
<td>instructions</td>
<td>1,515,404,340</td>
</tr>
<tr>
<td>r20</td>
<td>1,446,545,776</td>
</tr>
<tr>
<td>r21</td>
<td>757,589,098</td>
</tr>
<tr>
<td>r3</td>
<td>705,740,759</td>
</tr>
<tr>
<td>r5</td>
<td>576,226,424</td>
</tr>
<tr>
<td>sched:sched_wakeup</td>
<td>40,675</td>
</tr>
</tbody>
</table>

6.156288957 seconds time elapsed

- **Events**
  - Cycles/Instructions globally
  - R20,R21 – Cycles/Instructions of Problem state
  - R3/R5 – Penalty cycles due for L1 instruction/data cache
  - Not only HW events, you can use any of the currently 163 tracepoints

- **Further releases will make that readable and work with few arguments**
  - Until then you can refer to this document to get the event numbers

The Load-Program-Parameter and CPU-Measurement Facilities
Java Performance in general

- “Too” many choices
  - There are many Java performance tools out there

- Be aware of common Java myths often clouding perception

- Differences
  - Profiling a JVM might hide the Java methods
  - Memory allocation of the JVM isn't the allocation of the Application
Java - Health Center

- Characteristics: Lightweight Java Virtual Machine Overview
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: IBM Support Assistant (Eclipse)
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/healthcenter
  Java Agents integrated V5SR10+, V6SR3+, usually no target install required

- Shows
  - Memory usage
  - Method Profiling
  - I/O Statistics
  - Class loading
  - Locking

- Hints
  - Low overhead, therefore even suitable for monitoring
  - Agent activation -Xhealthcenter:port=12345
  - Can trigger dumps or verbosegc for in-depth memory analysis
Example of method profiling

- **Health Center - example**

  ![Method profile - Health Center - example](image)

  - **Method profile - Health Center - example**
    - **Filter methods:**
      - **Sample:** 15
      - **Self:** 95.8%
      - **Total:** 95.8%
      - **Method:** com.ibm.tmcc.demo.ComputingResourcesConsumer.generateCpuLoad()
        - 94
        - 0.35
        - 0.78
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.addNode(org.w3c.dom.Node, int)
        - 77
        - 0.29
        - 1.27
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.nextNode()
        - 44
        - 0.16
        - 0.49
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.processNameSpaceAxesAndAttributes()
        - 42
        - 0.16
        - 1.53
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.getHandleFromNode(org.w3c.dom.Node)
        - 42
        - 0.16
        - 0.16
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.getHandleFromNode(org.w3c.dom.Node)
        - 39
        - 0.15
        - 0.18
        - java.lang.ClassLoader.defineClassImpl(java.lang.String, byte[], int, int)
        - 34
        - 0.13
        - 0.13
        - org.apache.xml.dtm.ref.DTMDetailBase.ensureSizeOfIndex(int, int)
        - 25
        - 0.093
        - 0.26
        - org.apache.xml.dtm.ref.ExpandedNameTable.getExpandedTypeID(java.lang.String)
        - 24
        - 0.089
        - 0.19
        - java.lang.ClassLoader.loadClass(java.lang.String, boolean)
        - 20
        - 0.074
        - 0.074
        - java.lang.Object.wait(int, long)
        - 17
        - 0.063
        - 0.35
        - java.lang.Thread.unsafeInterlocked(java.lang.Class)
        - 16
        - 0.06
        - 0.06
        - java.lang.String.toString()
        - 13
        - 0.048
        - 0.048
        - org.apache.xml.dtm.ref.dom2tm.DOM2DTM.getChainHashMap.get(java.lang.String)
        - 12
        - 0.045
        - 1.65
        - org.apache.xml.dtm.ref.DTMDetailBaseManagerDefault.getDTMHandleFromNode(org.w3c.dom.Node)
        - 12
        - 0.045
        - 0.045
        - sun.nio.cs.US_ASCII.encode.encode(String, java.nio.CharBuffer)
        - 11
        - 0.041
        - 0.067
        - com.ibm.cds.CDSBundleFile.getEntry(java.lang.String)
        - 11
        - 0.041
        - 0.17
        - org.apache.xml.dtm.ref.DTMDetailBase.indexNode(int, int)

  - **Invocation paths:**
    - ComputingResourcesConsumer.generateCpuLoad()
      - TMCCDemoServlet.handleHttpRequest (100%)
      - TMCCDemoServlet.doGet (100%)
      - HttpServlet.service (100%)

- **Example of method profiling**

Webcast October 2013
Java - Garbage Collection and Memory Visualizer

- Characteristics: in-depth Garbage Collection analysis
- Objective: Analyze JVM memory management
- Usage: IBM Support Assistant (Eclipse)
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/gcmv
  reads common verbosegc output, so usually no target install required

- Shows
  - Memory usage
  - Garbage Collection activities
  - Pauses
  - Memory Leaks by stale references

- Hints
  - GCMV can also compare output of two runs
  - Activate verbose logs: \texttt{-verbose:gc -Xverbosegclog:<log_file>}
- Most important values / indicators are:
  - Proportion of time spent in gc pauses (should be less than 5%)
  - For gencon: global collections $<<$ minor collections
IRQ Statistics

- Characteristics: Low overhead IRQ information
- Objective: Condensed overview of IRQ activity
- Usage: `cat /proc/interrupts` and `cat /proc/softirqs`
- Package: n/a (Kernel interface)

- Shows
  - Which interrupts happen on which CPU
  - Where softirqs and tasklets take place

- Hints
  - Recent Versions (SLES11-SP2) much more useful due to better naming
  - If interrupts are unintentionally unbalanced
  - If the amount of interrupts matches I/O
    - This can point to non-working IRQ avoidance
IRQ Statistics

- **Example**
  - Network focused on CPU zero (in this case unwanted)
  - Scheduler covered most of that avoiding idle CPU 1-3
  - But caused a lot migrations, IPI's and cache misses

<table>
<thead>
<tr>
<th></th>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT:</td>
<td>21179</td>
<td>24235</td>
<td>22217</td>
<td>22959</td>
</tr>
<tr>
<td>I/O:</td>
<td>1542959</td>
<td>340076</td>
<td>356381</td>
<td>325691</td>
</tr>
<tr>
<td>CLK:</td>
<td>15995</td>
<td>16718</td>
<td>15806</td>
<td>16531</td>
</tr>
<tr>
<td>EXC:</td>
<td>255</td>
<td>325</td>
<td>332</td>
<td>227</td>
</tr>
<tr>
<td>EMS:</td>
<td>4923</td>
<td>7129</td>
<td>6068</td>
<td>6201</td>
</tr>
<tr>
<td>TMR:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TAL:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PFL:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DSD:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VRT:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SCP:</td>
<td>6</td>
<td>63</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>IUC:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPM:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CIO:</td>
<td>163</td>
<td>310</td>
<td>269</td>
<td>213</td>
</tr>
<tr>
<td>QAI:</td>
<td>1541</td>
<td>338</td>
<td>354</td>
<td>324</td>
</tr>
<tr>
<td>DAS:</td>
<td>1023</td>
<td>909</td>
<td>1384</td>
<td>1368</td>
</tr>
</tbody>
</table>

[EXT] Clock Comparator
[EXT] External Call
[EXT] Emergency Signal
[EXT] CPU Timer
[EXT] Timing Alert
[EXT] Pseudo Page Fault
[EXT] DASD Diag
[EXT] Virtio
[EXT] Service Call
[EXT] IUCV
[EXT] CPU Measurement
[I/O] Common I/O Layer Interrupt
[I/O] QDIO Adapter Interrupt
[I/O] DASD

[...] 3215, 3270, Tape, Unit Record Devices, LCS, CLAW, CTC, AP Bus, Machine Check
IRQ Statistics II

- Also softirqs can be tracked which can be useful to
  - check if tasklets execute as intended
  - See if network, scheduling and I/O behave as expected

<table>
<thead>
<tr>
<th></th>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>498</td>
<td>1522</td>
<td>1268</td>
<td>1339</td>
</tr>
<tr>
<td>TIMER</td>
<td>5640</td>
<td>914</td>
<td>664</td>
<td>643</td>
</tr>
<tr>
<td>NET_TX</td>
<td>15</td>
<td>16</td>
<td>52</td>
<td>32</td>
</tr>
<tr>
<td>NET_RX</td>
<td>18</td>
<td>34</td>
<td>87</td>
<td>45</td>
</tr>
<tr>
<td>BLOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BLOCK_IOPOLL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TASKLET</td>
<td>13</td>
<td>10</td>
<td>44</td>
<td>20</td>
</tr>
<tr>
<td>SCHED</td>
<td>8055</td>
<td>702</td>
<td>403</td>
<td>445</td>
</tr>
<tr>
<td>HRTIMER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RCU</td>
<td>5028</td>
<td>2906</td>
<td>2794</td>
<td>2564</td>
</tr>
</tbody>
</table>
Java - Jinsight

- Characteristics: zoomable call stack
- Objective: Analyze method call frequency and duration
- Usage: `jinsight_trace -tracemethods <yourProgram> <yourProgramArgs>`
- Package: RHEL: n/a SLES: n/a WWW: IBM alphaworks

- Shows
  - Call Stack and time

- Hints
  - Significant slowdown, not applicable to production systems
  - No more maintained, but so far still working
Jinsight Execution View

- Threads in columns, select one to zoom in
Many horizontal stages mean deep call stacks
Long vertical areas mean long method execution
Rectangles full of horizontal lines can be an issue
Perf profiling

- Perf example how-to
  - We had a case where new code caused cpus to scale badly
  - `perf record "workload"`
    - Creates a file called `perf.data` that can be analyzes
  - We used `perf diff` on both data files to get a comparison

- “Myriad” of further options/modules
  - Live view with `perf top`
  - Perf sched for an integrated analysis of scheduler tracepoints
  - Perf annotate to see samples alongside code
  - Perf stat for a counter based analysis
  - [...]
Perf profiling

- Perf example (perf diff)
  - found a locking issue causing increased cpu consumption

<table>
<thead>
<tr>
<th># Baseline</th>
<th>Delta</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.14%</td>
<td>+8.07%</td>
<td>[kernel.kallsyms] lock_acquire</td>
</tr>
<tr>
<td>8.96%</td>
<td>+5.50%</td>
<td>[kernel.kallsyms] lock_release</td>
</tr>
<tr>
<td>4.83%</td>
<td>+0.38%</td>
<td>reaim add_long</td>
</tr>
<tr>
<td>4.22%</td>
<td>+0.41%</td>
<td>reaim add_int</td>
</tr>
<tr>
<td>4.10%</td>
<td>+2.49%</td>
<td>[kernel.kallsyms] lock_acquired</td>
</tr>
<tr>
<td>3.17%</td>
<td>+0.38%</td>
<td>libc-2.11.3.so msort_with_tmp</td>
</tr>
<tr>
<td>3.56%</td>
<td>-0.37%</td>
<td>reaim string_rtns_1</td>
</tr>
<tr>
<td>3.04%</td>
<td>-0.38%</td>
<td>libc-2.11.3.so strncat</td>
</tr>
</tbody>
</table>
HTOP

- Characteristics: Process overview with extra features
- Objective: Get a understanding about your running processes
- Usage: `htop`
- Package: RHEL: n/a SLES: n/a WWW: http://htop.sourceforge.net/
- Shows
  - Running processes
  - CPU and memory utilization
  - Accumulated times
  - I/O rates
  - System utilization visualization

- Hints
  - Htop can display more uncommon fields (in menu)
  - Able to send signals out of its UI for administration purposes
  - Processes can be sorted/filtered for a more condensed view
## htop

### Configurable utilization visualization

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>CPU%</th>
<th>MEM%</th>
<th>UTIME+</th>
<th>STIME+</th>
<th>IORR</th>
<th>IOWR</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>51931</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>142M</td>
<td>140M</td>
<td>S</td>
<td>1.0</td>
<td>1.2</td>
<td>0:00.47</td>
<td>0:00.21</td>
<td>627</td>
<td>0</td>
<td>0:00.68</td>
<td>postgres:</td>
</tr>
<tr>
<td>51962</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>157M</td>
<td>154M</td>
<td>R</td>
<td>3.0</td>
<td>1.3</td>
<td>0:00.56</td>
<td>0:00.24</td>
<td>483</td>
<td>0</td>
<td>0:00.80</td>
<td>postgres:</td>
</tr>
<tr>
<td>51981</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>170M</td>
<td>168M</td>
<td>R</td>
<td>3.0</td>
<td>1.4</td>
<td>0:00.61</td>
<td>0:00.26</td>
<td>424</td>
<td>0</td>
<td>0:00.87</td>
<td>postgres:</td>
</tr>
<tr>
<td>51921</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>164M</td>
<td>162M</td>
<td>R</td>
<td>1.0</td>
<td>1.4</td>
<td>0:00.57</td>
<td>0:00.25</td>
<td>398</td>
<td>0</td>
<td>0:00.83</td>
<td>postgres:</td>
</tr>
<tr>
<td>51953</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>169M</td>
<td>166M</td>
<td>R</td>
<td>1.0</td>
<td>1.4</td>
<td>0:00.62</td>
<td>0:00.27</td>
<td>280</td>
<td>0</td>
<td>0:00.89</td>
<td>postgres:</td>
</tr>
<tr>
<td>51934</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>174M</td>
<td>172M</td>
<td>R</td>
<td>2.0</td>
<td>1.4</td>
<td>0:00.64</td>
<td>0:00.27</td>
<td>269</td>
<td>0</td>
<td>0:00.91</td>
<td>postgres:</td>
</tr>
<tr>
<td>51923</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>156M</td>
<td>153M</td>
<td>R</td>
<td>3.0</td>
<td>1.3</td>
<td>0:00.55</td>
<td>0:00.26</td>
<td>269</td>
<td>0</td>
<td>0:00.81</td>
<td>postgres:</td>
</tr>
<tr>
<td>51933</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>154M</td>
<td>151M</td>
<td>S</td>
<td>1.0</td>
<td>1.3</td>
<td>0:00.55</td>
<td>0:00.26</td>
<td>251</td>
<td>0</td>
<td>0:00.81</td>
<td>postgres:</td>
</tr>
<tr>
<td>51942</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>178M</td>
<td>175M</td>
<td>R</td>
<td>1.0</td>
<td>1.5</td>
<td>0:00.68</td>
<td>0:00.31</td>
<td>205</td>
<td>0</td>
<td>0:00.99</td>
<td>postgres:</td>
</tr>
<tr>
<td>51946</td>
<td>postgres</td>
<td>20</td>
<td>0</td>
<td>3264M</td>
<td>139M</td>
<td>136M</td>
<td>R</td>
<td>1.0</td>
<td>1.2</td>
<td>0:00.47</td>
<td>0:00.22</td>
<td>200</td>
<td>0</td>
<td>0:00.69</td>
<td>postgres:</td>
</tr>
</tbody>
</table>

**Tasks:** 101, 80 thr; 60 running
**Load average:** 42.03 16.67 6.24
**Uptime:** 00:17:11
Tracepoints (Events)

- Characteristics: Complex interface, but a vast source of information
- Objective: In kernel latency and activity insights
- Usage: Access debugfs mount point /tracing
- Package: n/a (Kernel interface)

- Shows
  - Timestamp and activity name
  - Tracepoints can provide event specific context data
  - Infrastructure adds extra common context data like cpu, preempts depth, ...

- Hints
  - Very powerful and customizable, there are hundreds of tracepoints
    - Some tracepoints have tools to be accessed “perf sched”, “blktrace” both base on them
    - Others need custom postprocessing
  - There are much more things you can handle with tracepoints check out
    Kernel Documentation/trace/tracepoint-analysis.txt (via perf stat)
    Kernel Documentation/trace/events.txt (custom access)
Tracepoints – example I/III

- Here we use custom access since there was tool
  - We searched for 1.2ms extra latency
    - Target is it lost in HW, Userspace, Kernel or all of them
  - Workload was a simple 1 connection 1 byte←→1 byte load
  - Call “perf list” for a list of currently supported tracepoints

- We used the following tracepoints

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Tracepoint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>netif_receive_skb</td>
<td>low level receive</td>
</tr>
<tr>
<td>P</td>
<td>napi_poll</td>
<td>napi work related to receive</td>
</tr>
<tr>
<td>Q</td>
<td>net_dev_queue</td>
<td>enqueue in the stack</td>
</tr>
<tr>
<td>S</td>
<td>net_dev_xmit</td>
<td>low level send</td>
</tr>
</tbody>
</table>
Tracepoints – example II/III

-(Simplified) Script

- # full versions tunes buffer sizes, checks files, ...

```bash
echo latency-format > /sys/kernel/debug/tracing/trace_options               # enable tracing type
echo net:* >> /sys/kernel/debug/tracing/set_event                         # select specific events
echo napi:* >> /sys/kernel/debug/tracing/set_event                        # "
echo "name == ${dev}" > /sys/kernel/debug/tracing/events/net/filter       # set filters
echo "dev_name == ${dev}" > /sys/kernel/debug/tracing/events/napi/filter   # "
cat /sys/kernel/debug/tracing/trace >> ${output}                          # synchronous
echo !*:* > /sys/kernel/debug/tracing/set_event                           # disable tracing
```

- Output

```
#                _------=>
#               / _-----=> irqs-off
#              | / _----=> need-resched
#              || / _---=> hardirq/softirq
#              ||| / _--=> preempt-depth
#              |||| /     delay
#  cmd     pid |||||  
<...>-24116 0..s. 486183281us+: net_dev_xmit: dev=eth5 skbaddr=000000075b7e3e8 len=67 rc=0
<idle>-0     0..s. 486183303us+: netif_receive_skb: dev=eth5 skbaddr=00000007ecc6e00 len=53
<idle>-0     0.Ns. 486183306us+: napi_poll: napi poll on napi struct 000000007d2479a8 fordevice eth
<...>-24116 0..s. 486183311us+: net_dev_queue: dev=eth5 skbaddr=000000075b7e3e8 len=67
<...>-24116 0..s. 486183317us+: net_dev_xmit: dev=eth5 skbaddr=000000075b7e3e8 len=67 rc=0
```
Tracepoints – example III/III

- Example postprocessed

<table>
<thead>
<tr>
<th></th>
<th>SUM</th>
<th>COUNT</th>
<th>AVERAGE</th>
<th>MIN</th>
<th>MAX</th>
<th>STD-DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2Q</td>
<td>8478724</td>
<td>1572635</td>
<td>5.39</td>
<td>4</td>
<td>2140</td>
<td>7.41</td>
</tr>
<tr>
<td>Q2S</td>
<td>12188675</td>
<td>1572638</td>
<td>7.65</td>
<td>3</td>
<td>71</td>
<td>4.89</td>
</tr>
<tr>
<td>S2R</td>
<td>38562294</td>
<td>1572636</td>
<td>24.42</td>
<td>1</td>
<td>2158</td>
<td>9.08</td>
</tr>
<tr>
<td>R2P</td>
<td>4197486</td>
<td>1572633</td>
<td>2.57</td>
<td>1</td>
<td>43</td>
<td>2.39</td>
</tr>
<tr>
<td>SUM</td>
<td>63427179</td>
<td>1572635</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Confirmed that ~all of the 1.2 ms were lost inside Linux (not in the fabric)
- Confirmed that it was not at/between specific function tracepoints
  - Eventually it was an interrupt locality issue causing bad caching
vmstat

- Characteristics: Easy to use, high-level information
- Objective: First and fast impression of the current state
- Usage: vmstat [interval in sec]
- Package: RHEL: sysstat.s390x SLES: sysstat
- Output sample:

```
vstat 1
procs -------memory-------- ---swap--- ------io---- -system-- -----cpu-----
r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
2 2 0 4415152 64068 554100 0 0 4 63144 350 55 29 64 0 3 4
3 0 0 4417632 64832 551272 0 0 0 988 125 60 32 67 0 0 1
3 1 0 4415524 68100 550068 0 0 0 5484 212 66 31 64 0 4 1
3 0 0 4411804 72188 549592 0 0 0 8984 230 42 32 67 0 0 1
3 0 0 4405232 72896 555592 0 0 0 16 105 52 32 68 0 0 0
```

- Shows
  - Data per time interval
  - CPU utilization
  - Disk I/O
  - Memory usage/Swapping

- Hints
  - Shared memory usage is listed under 'cache'
sadc/sar

- Characteristics: Very comprehensive, statistics data on device level
- Objective: Suitable for permanent system monitoring and detailed analysis
- Usage (recommended):
  - monitor /usr/lib64/sa/sadc [-S XALL] [interval in sec] [outfile]
  - View sar -A -f [outfile]
- Package: RHEL: sysstat.s390x SLES: sysstat
- Shows
  - CPU utilization
  - Disk I/O overview and on device level
  - Network I/O and errors on device level
  - Memory usage/Swapping
  - … and much more
  - Reports statistics data over time and creates average values for each item
- Hints
  - sadc parameter "-S XALL" enables the gathering of further optional data
  - Shared memory is listed under 'cache'
  - [outfile] is a binary file, which contains all values. It is formatted using sar
    - enables the creation of item specific reports, e.g. network only
    - enables the specification of a start and end time → time of interest
Processes created per second usually small except during startup. If constantly at a high rate your application likely has an issue. Be aware – the numbers scale with your system size and setup.
SAR - Context Switch Rate

Context switches per second usually < 1000 per cpu except during startup or while running a benchmark if > 10000 your application might have an issue.
SAR - CPU utilization

Per CPU values:
- watch out for system time (kernel)
- user (applications)
- irq/soft (kernel, interrupt handling)
- idle (nothing to do)
- iowait time (Runnable but waiting for I/O)
- steal time (Runnable but utilized somewhere else)

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>%user</th>
<th>%nice</th>
<th>%system</th>
<th>%iowait</th>
<th>%steal</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:14:55</td>
<td>CPU</td>
<td>26.64</td>
<td>0.00</td>
<td>12.03</td>
<td>25.92</td>
<td>6.24</td>
<td>29.16</td>
</tr>
<tr>
<td>14:15:05</td>
<td>all</td>
<td>43.81</td>
<td>0.00</td>
<td>5.49</td>
<td>23.25</td>
<td>4.99</td>
<td>22.46</td>
</tr>
<tr>
<td>14:15:05</td>
<td>0</td>
<td>4.30</td>
<td>0.00</td>
<td>10.19</td>
<td>28.67</td>
<td>9.89</td>
<td>46.95</td>
</tr>
<tr>
<td>14:15:05</td>
<td>1</td>
<td>11.81</td>
<td>0.00</td>
<td>28.03</td>
<td>45.15</td>
<td>5.01</td>
<td>10.01</td>
</tr>
<tr>
<td>14:15:05</td>
<td>2</td>
<td>46.61</td>
<td>0.00</td>
<td>4.49</td>
<td>6.79</td>
<td>4.99</td>
<td>37.13</td>
</tr>
<tr>
<td>14:15:15</td>
<td>all</td>
<td>27.19</td>
<td>0.00</td>
<td>11.93</td>
<td>25.11</td>
<td>7.75</td>
<td>28.01</td>
</tr>
<tr>
<td>14:15:15</td>
<td>0</td>
<td>90.60</td>
<td>0.00</td>
<td>3.70</td>
<td>0.00</td>
<td>5.70</td>
<td>0.00</td>
</tr>
<tr>
<td>14:15:15</td>
<td>1</td>
<td>9.24</td>
<td>0.00</td>
<td>22.49</td>
<td>41.57</td>
<td>9.24</td>
<td>17.47</td>
</tr>
<tr>
<td>14:15:15</td>
<td>2</td>
<td>5.98</td>
<td>0.00</td>
<td>14.64</td>
<td>46.71</td>
<td>9.06</td>
<td>23.61</td>
</tr>
<tr>
<td>14:15:15</td>
<td>3</td>
<td>2.90</td>
<td>0.00</td>
<td>6.99</td>
<td>12.09</td>
<td>7.09</td>
<td>70.93</td>
</tr>
</tbody>
</table>
SAR - Network traffic

Per interface statistic of packets/bytes
You can easily derive average packet sizes from that.
Sometimes people expect - and planned for – different sizes.
Has another panel for errors, drops and such events.
### Overview of SAR – Disk I/O I – overall

<table>
<thead>
<tr>
<th>Time</th>
<th>tps</th>
<th>rtps</th>
<th>wtps</th>
<th>bread/s</th>
<th>bwrt/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:14:55</td>
<td>316.35</td>
<td>116.15</td>
<td>200.20</td>
<td>3784.61</td>
<td>64504.50</td>
</tr>
<tr>
<td>14:15:05</td>
<td>445.71</td>
<td>61.38</td>
<td>384.33</td>
<td>7715.77</td>
<td>55529.74</td>
</tr>
<tr>
<td>14:15:15</td>
<td>192.20</td>
<td>32.90</td>
<td>159.30</td>
<td>7308.80</td>
<td>68233.60</td>
</tr>
<tr>
<td>14:15:25</td>
<td>171.70</td>
<td>1.20</td>
<td>170.50</td>
<td>9.60</td>
<td>70798.40</td>
</tr>
<tr>
<td>14:15:35</td>
<td>327.25</td>
<td>174.95</td>
<td>152.30</td>
<td>1399.60</td>
<td>68261.88</td>
</tr>
<tr>
<td>14:15:45</td>
<td>444.74</td>
<td>310.51</td>
<td>134.23</td>
<td>2484.88</td>
<td>59704.50</td>
</tr>
</tbody>
</table>

Overview of:
- operations per second
- transferred amount
Is your I/O balanced across devices? Imbalances can indicate issues with a LV setup.

tps and avgrq-sz combined can be important. Do they match your sizing assumptions?

Await shows the time the application has to wait.
SAR - Memory statistics - the false friend

Be aware that high %memused and low kbmemfree is no indication of a memory shortage (common mistake).

Same for swap – to use swap is actually good, but to access it (swapin/-out) all the time is bad.
SAR - Memory pressure - Swap

The percentage seen before can be high, but the swap rate shown here should be low. Ideally it is near zero after a rampup time. High rates can indicate memory shortages.
SAR - Memory pressure – faults and reclaim

Don't trust pgpgin/-out absolute values
Faults populate memory
Major faults need I/O
Scank/s is background reclaim by kswap/flush (modern)
Scand/s is reclaim with a “waiting” allocation
Steal is the amount reclaimed by those scans
SAR - System Load

Runqueue size are the currently runnable programs. It's not bad to have many, but if they exceed the amount of CPUs you could do more work in parallel.

Plist-sz is the overall number of programs, if that is always growing you have likely a process starvation or connection issue.

Load average is a runqueue length average for 1/5/15 minutes.
iostat

- Characteristics: Easy to use, information on disk device level
- Objective: Detailed input/output disk statistics
- Usage: `iostat -xtdk [interval in sec]`
- Package: RHEL: sysstat.s390x SLES: sysstat

- Shows
  - Throughput
  - Request merging
  - Device queue information
  - Service times

- Hints
  - Most critical parameter often is `await`
    - average time (in milliseconds) for I/O requests issued to the device to be served.
    - includes the time spent by the requests in queue and the time spent servicing them.
  - Also suitable for network file systems
iostat

### Output sample:

<table>
<thead>
<tr>
<th>Time: 10:56:35 AM</th>
<th>Device:</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dasda</td>
<td>0.19</td>
<td>1.45</td>
<td>1.23</td>
<td>0.74</td>
<td>64.43</td>
<td>9.29</td>
<td>74.88</td>
<td>0.01</td>
<td>2.65</td>
<td>0.80</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>dasdb</td>
<td>0.02</td>
<td>232.93</td>
<td>0.03</td>
<td>9.83</td>
<td>0.18</td>
<td>975.17</td>
<td>197.84</td>
<td>0.98</td>
<td>99.80</td>
<td>1.34</td>
<td>1.33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time: 10:56:36 AM</th>
<th>Device:</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dasda</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>dasdb</td>
<td>0.00</td>
<td>1981.55</td>
<td>0.00</td>
<td>339.81</td>
<td>0.00</td>
<td>9495.15</td>
<td>55.89</td>
<td>0.91</td>
<td>2.69</td>
<td>1.14</td>
<td>38.83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time: 10:56:37 AM</th>
<th>Device:</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dasda</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>dasdb</td>
<td>0.00</td>
<td>2055.00</td>
<td>0.00</td>
<td>344.00</td>
<td>0.00</td>
<td>9628.00</td>
<td>55.98</td>
<td>1.01</td>
<td>2.88</td>
<td>1.19</td>
<td>41.00</td>
</tr>
</tbody>
</table>
DASD statistics

- Characteristics: Easy to use, very detailed
- Objective: Collects statistics of I/O operations on DASD devices
- Usage:
  - enable: echo on > /proc/dasd/statistics
  - show:
    - Overall cat /proc/dasd/statistics
    - for individual DASDs tunedasd -P /dev/dasda
- Package: n/a for kernel interface, s390-tools for dasdstat
- Shows:
  - various processing times:

Start

Histogram of I/O till ssch
Build channel program
wait till subchannel is free

Histogram of I/O between
ssch and IRQ
Processing data transfer
from/to storage server

Histogram between
I/O and End
Tell block dev layer
Data has arrived

Histogram of I/O times

End

New Tool “dasdstat” available to handle that all-in-one
DASD statistics - report

- Sample:

  \[ 8 \times 512b = 4KB \leq \text{request size} < 1 \times 512b = 8KB \quad \text{1ms} \leq \text{response time} < 2 \text{ ms} \]

<table>
<thead>
<tr>
<th>Request Size (512B)</th>
<th>&lt;4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1k</th>
<th>2k</th>
<th>4k</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>64k</th>
<th>128k</th>
</tr>
</thead>
<tbody>
<tr>
<td>29432</td>
<td>202</td>
<td>252</td>
<td>184</td>
<td>178</td>
<td>220</td>
<td>270</td>
<td>225</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Histogram of sizes (512B secs)

- Hints

  - Also shows data per sector which usually only confused
FCP statistics

- Characteristics: Detailed latency information (SLES9 and SLES10)
- Objective: Collects statistics of I/O operations on FCP devices on request base, separate for read/write
- Package: n/a (Kernel interface)

Usage:
- enable
  - CONFIG_STATISTICS=y must be set in the kernel config file
  - debugfs is mounted at /sys/kernel/debug/
  - For a certain LUN in directory /sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>
    issue `echo on=1 > definition` (turn off with on=0, reset with data=reset)
- view
  - cat /sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>/data

Hint
- FCP and DASD statistics are not directly comparable, because in the FCP case many I/O requests can be sent to the same LUN before the first response is given. There is a queue at FCP driver entry and in the storage server
FCP statistics

- Shows:
  - Request sizes in bytes (hexadecimal)
  - Channel latency Time spent in the FCP channel in nanoseconds
  - Fabric latency processing data transfer from/to storage server incl. SAN in nanoseconds
  - (Overall) latencies whole time spent in the FCP layer in milliseconds

- Calculate the pass through time for the FCP layer as
  pass through time = overall latency - (channel latency + fabric latency)
  → Time spent between the Linux device driver and FCP channel adapter inclusive in Hypervisor
FCP statistics example

cat /sys/kernel/debug/statistics/zfcp-0.0.1700-0x5005076303010482-0x4014400500000000/data

... request_sizes_scsi_read 0x1000 1163 request_sizes_scsi_read 0x80000 805 request_sizes_scsi_read 0x54000 47 request_sizes_scsi_read 0x2d000 44 request_sizes_scsi_read 0x2a000 26 request_sizes_scsi_read 0x57000 25 request_sizes_scsi_read 0x1e000 25

... latencies_scsi_read <=1 1076 latencies_scsi_read <=2 205 latencies_scsi_read <=4 575 latencies_scsi_read <=8 368 latencies_scsi_read <=16 0

... channel_latency_read <=160000 0 channel_latency_read <=320000 983 channel_latency_read <=640000 99 channel_latency_read <=1280000 115 channel_latency_read <=2560000 753 channel_latency_read <=5120000 106 channel_latency_read <=10240000 141 channel_latency_read <=20480000 27 channel_latency_read <=40960000 0

... fabric_latency_read <=10000000 1238 fabric_latency_read <=20000000 328 fabric_latency_read <=40000000 522 fabric_latency_read <=80000000 136 fabric_latency_read <=160000000 0

request size 4KB, 1163 occurrences

response time <= 1ms

Channel response time <= 32μs = all below driver

Fabric response time <= 1ms = once leaving the card
netstat

- Characteristics: Easy to use, connection information
- Objective: Lists connections
- Usage: `netstat -eeapn`
- Package: RHEL: net-tools SLES: net-tools

- Shows
  - Information about each connection
  - Various connection states

- Hints
  - Inodes and program names are useful to reverse-map ports to applications
netstat -s

- Characteristics: Easy to use, very detailed information
- Objective: Display summary statistics for each protocol
- Usage: `netstat -s`

- Shows
  - Information to each protocol
  - Amount of incoming and outgoing packages
  - Various error states, for example TCP segments retransmitted!

- Hints
  - Shows accumulated values since system start, therefore mostly the differences between two snapshots are needed
  - There is always a low amount of packets in error or resets
  - Retransmits occurring only when the system is sending data
    When the system is not able to receive, then the sender shows retransmits
  - Use sadc/sar to identify the device
netstat -s

- Output sample:

  Tcp:
  15813 active connections openings
  35547 passive connection openings
  305 failed connection attempts
  0 connection resets received
  6117 connections established
  81606342 segments received
  127803327 segments send out
  288729 segments retransmitted
  0 bad segments received.
  6 resets sent
Socket statistics

- Characteristics: Information on socket level
- Objective: Check socket options and weird connection states
- Usage: `ss -aempi`
- Package: RHEL: iproute-2 SLES: iproute2
- Shows
  - Socket options
  - Socket receive and send queues
  - Inode, socket identifiers

- Sample output
  ```
  ss -aempi
  State    Recv-Q Send-Q   Local Address:Port      Peer Address:Port
  LISTEN   0    128      :::ssh                  :::*
  users:(("sshd",959,4)) ino:7851 sk:ef858000  mem:(r0,w0,f0,t0)
  ```

- Hints
  - Inode numbers can assist reading strace logs
  - Check long outstanding queue elements
Top

- Characteristics: Easy to use
- Objective: Shows resource usage on process level
- Usage: `top -b -d [interval in sec] > [outfile]`
- Package: RHEL: procps SLES: procps

- Shows
  - CPU utilization
  - Detailed memory usage

- Hints
  - Parameter -b enables to write the output for each interval into a file
  - Use -p [pid1, pid2,...] to reduce the output to the processes of interest
  - Configure displayed columns using 'f' key on the running top program
  - Use the 'W' key to write current configuration to ~/.toprc
    → becomes the default
top (cont.)

- See ~/.toprc file in backup

Output sample:

```
top - 11:12:52 up 1:11,  3 users,  load average: 1.21, 1.61, 2.03
Tasks:  53 total,   5 running,  48 sleeping,  0 stopped,  0 zombie
Cpu(s):  3.0%us,  5.9%sy,  0.0%ni,  79.2%id,  9.9%wa,  0.0%hi,  1.0%si,  1.0%st
Mem:  5138052k total,  801100k used,  4336952k free,  447868k buffers
Swap:  88k total,  0k used,  88k free,  271436k cached

PID USER      PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  P  SWAP   DATA  WCHAN     COMMAND
3224 root      18   0  1820  604  444  R  2.0  0.0   0:00.56 0  1216  252 -  dbench
3226 root      18   0  1820  604  444  R  2.0  0.0   0:00.56 0  1216  252 -  dbench
2737 root      16   0  9512 3228 2540  R  1.0  0.1   0:00.46 0  6284  868 -  sshd
3225 root      18   0  1820  604  444  R  1.0  0.0   0:00.56 0  1216  252 -  dbench
3230 root      16   0  2652 1264  980  R  1.0  0.0   0:00.01 0  1388  344 -  top
1 root      16   0   848  304  256  S  0.0  0.0   0:00.54 0  544  232 select  init
2 root      RT   0  0  0  0  S  0.0  0.0   0:00.00 0  0  0migration  migration/0
3 root      34  19  0  0  0  S  0.0  0.0   0:00.00 0  0  0ksoftirqd ksoftirqd/0
4 root      10  -5  0  0  0  S  0.0  0.0   0:00.13 0  0  0worker_th events/0
5 root      20  -5  0  0  0  S  0.0  0.0   0:00.00 0  0  0worker_th khelper
```

- Hints
  - virtual memory: VIRT = SWAP + RES unit KB
  - physical memory used: RES = CODE + DATA unit KB
  - shared memory SHR unit KB
Linux ps command

- Characteristics: very comprehensive, statistics data on process level
- Objective: reports a snapshot of the current processes
- Usage: “ps axlf”
- Package: RHEL: procps SLES: procps

```
PID  TID  NLWP  POL USER   TTY      NI PRI PSR P STAT WCHAN        START     TIME %CPU %MEM   VSZ   SZ   RSS - COMMAND
871  871  1 TS  root     ?       -5 29 0 * S< kauditd_thread 10:01 00:00:00 0.0 0.0 0 0 0 - [kauditd]
2835 2835 1 TS  root     pts/2    0 23 0 * Ss+ read_chan 10:38 00:00:00 0.0 0.0 5140 824 2644 - bash
3437 3437 1 TS  root     pts/1    0 23 0 * S+ wait4 11:39 00:00:00 0.0 0.0 1816 248 644 - dbench 3
3438 3438 1 TS  root     pts/1    0 20 0 0 R+  - 11:39 00:00:24 33.1 0.0 1820 252 604 - dbench 3
3439 3439 1 TS  root     pts/1    0 20 0 0 R+  - 11:39 00:00:23 32.8 0.0 1820 252 604 - dbench 3
3440 3440 1 TS  root     pts/1    0 20 0 0 R+  - 11:39 00:00:23 31.8 0.0 1820 252 604 - dbench 3
...```

- Hints
  - Do not specify blanks inside the -o format string
  - Many more options available
**Lszcrypt / icastats**

- **Characteristics:** overview of s390 crypto HW and libica usage
- **Objective:** am I really using my crypto hardware
- **Usage:** "lczcrypt -VV[V]" “cat /proc/icastats"
- **Package:** RHEL: s390utils-base SLES: s390-tools

```
lszcrypt -VV
  card02: CEX3C  online  hwtype=9  depth=8  request_count=443
  card03: CEX3A  offline hwtype=8  depth=8  request_count=0
```

```
<table>
<thead>
<tr>
<th>function</th>
<th># hardware</th>
<th># software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-224</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-256</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-384</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RANDOM</td>
<td>187109</td>
<td>0</td>
</tr>
<tr>
<td>MOD EXPO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RSA CRT</td>
<td>93554</td>
<td>0</td>
</tr>
<tr>
<td>DES ENC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DES DEC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3DES ENC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3DES DEC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AES ENC</td>
<td>2574106</td>
<td>0</td>
</tr>
<tr>
<td>AES DEC</td>
<td>2075854</td>
<td>0</td>
</tr>
<tr>
<td>CMAC GEN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CMAC VER</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- Never assume your HW correctly is used until you confirmed it
  - If not going via libica (e.g. Java pkcs#11 direct to HW you won't see it in icastat)
**lsluns**

- **Characteristics:** overview of multipathing
- **Objective:** check your multipath setup hierarchy
- **Usage:** “lsluns -a”
- **Package:** RHEL: s390utils-base SLES: s390-tools

```bash
lsluns -a
adapter = 0.0.1700
  port = 0x500507630900c7c1
  lun = 0x4020402100000000 /dev/sg0  Disk IBM:2107900
  lun = 0x4020402200000000 /dev/sg1  Disk IBM:2107900
  lun = 0x4020402300000000 /dev/sg2  Disk IBM:2107900
  lun = 0x4021402100000000 /dev/sg3  Disk IBM:2107900
  lun = 0x4021402200000000 /dev/sg4  Disk IBM:2107900
  lun = 0x4021402300000000 /dev/sg5  Disk IBM:2107900

adapter = 0.0.1780
  port = 0x500507630903c7c1
  lun = 0x4020402100000000 /dev/sg17 Disk IBM:2107900
  lun = 0x4020402200000000 /dev/sg23 Disk IBM:2107900
  lun = 0x4020402300000000 /dev/sg32 Disk IBM:2107900
  lun = 0x4021402100000000 /dev/sg39 Disk IBM:2107900
  lun = 0x4021402200000000 /dev/sg43 Disk IBM:2107900
  lun = 0x4021402300000000 /dev/sg46 Disk IBM:2107900

[...]
```

- Lsluns provides a hierarchical view which often easily identifies missing paths, adapters or similar imbalances
- Adapter to WWPN associations can have concurring targets
  - Low overhead, max fallback capability, best performance, ...
Multipath -ll

- Characteristics: overview of multipathing
- Objective: check your multipath setup configuration
- Usage: “multipath -ll”
- Package: RHEL: device-mapper-multipath SLES: multipath-tools

```
multipath -ll
swap-3of6 (36005076309fffc7c1000000000002022) dm-2 IBM ,2107900
size=256G features='0' hwhandler='0' wp=rw
 `-- policy='service-time 0' prio=0 status=active
   `- 0:0:20:1075986464 sdb  8:16  active ready running
   `- 1:0:22:1075986464 sdx  65:112 active ready running
   `- 2:0:21:1075986464 sdh  8:112 active ready running
   `- 3:0:20:1075986464 sdn  8:208 active ready running
   `- 4:0:26:1075986464 sdz  65:144 active ready running
   `- 5:0:19:1075986464 sdy  65:128 active ready running
   `- 7:0:25:1075986464 sdac 65:192 active ready running
   `- 6:0:24:1075986464 sdad  65:208 active ready running
[...]```

- This also reports multipath.conf inconsistencies
- Check all reported parameters are what you thought them to be
  - For example (in)famous rr_min_io renaming
Systemtap

- Characteristics: tool to “tap” into the kernel for analysis
- Objective: analyze in kernel values or behavior that otherwise would be inaccessible or require a modification/recompile cycle
- Usage (mini example): “stap -v -e 'probe vfs.read {printf("read performed\n"); exit()}’”
- Package: RHEL: systemtap + systemtap-runtime SLES: systemtap
  Also requires kernel debuginfo and source/devel packages

- Procedural and C-like language based on two main constructs
  – Probes – “catching events”
    • On functions, syscalls or single statements via file:linenumber
  – Functions – “what to do”
    • Supports local and global variables
    • Program flow statements if, loops, …

- Tapsets provide pre written probe libraries

- Fore more check out “Using SystemTap on Linux on System z” from Mike O’Reilly
  https://share.confex.com/share/118/webprogram/Handout/Session10452/atlan.pdf
lsqeth

- Characteristics: overview of network devices
- Objective: check your network devices basic setup
- Usage: “lsqeth -p”
- Package: RHEL: s390-utils-base SLES: s390-tools

<table>
<thead>
<tr>
<th>lsqeth -p</th>
<th>devices</th>
<th>CHPID</th>
<th>interface</th>
<th>cardtype</th>
<th>port</th>
<th>chksum</th>
<th>prio-q'ing</th>
<th>rtr4</th>
<th>rtr6</th>
<th>lay'2</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.e000/0.0.e001/0.0.e002</td>
<td>x84</td>
<td>eth1</td>
<td>OSD_10GIG</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>0.0.e100/0.0.e101/0.0.e102</td>
<td>x85</td>
<td>eth2</td>
<td>OSD_10GIG</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>0.0.f200/0.0.f201/0.0.f202</td>
<td>x6B</td>
<td>eth0</td>
<td>OSD_1000</td>
<td>0</td>
<td>hw</td>
<td>always_q_2</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

- Check for layer, offload, and buffer counts
  - More buffers are usually better especially for massive amounts of concurrent connections
**Ethtool I**

- **Characteristics**: overview of network device capabilities / offload settings
- **Objective**: check your network device (offload) settings
- **Usage**: “ethtool <dev>”, “ethtool -k <dev>”
- **Package**: RHEL: ethtool SLES: ethtool

```
ethtool eth1
Settings for eth1:
  Supported ports: [ FIBRE ]
  Supported link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Half 1000baseT/Full
                         10000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Advertised link modes:  10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Half 1000baseT/Full
                          10000baseT/Full
  Advertised pause frame use: No
  Advertised auto-negotiation: Yes
  Speed: 10000Mb/s
  Duplex: Full
  Port: FIBRE
  PHYAD: 0
  Transceiver: internal
  Auto-negotiation: on
  Link detected: yes
```

- **Check e.g. announced speeds**
Ethtool II

- Offload Settings via “ethtool -k <dev>”
- Changes via upper case “-K”

```
ethtool -k eth1
Features for eth1:
rx-checksumming: off [fixed]
tx-checksumming: off
  tx-checksum-ipv4: off [fixed]
tx-checksum-ip-generic: off [fixed]
tx-checksum-ipv6: off [fixed]
tx-checksum-fcoe-crc: off [fixed]
tx-checksum-sctp: off [fixed]
scatter-gather: off
  tx-scatter-gather: off [fixed]
tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
  tx-tcp-segmentation: off [fixed]
tx-tcp-ecn-segmentation: off [fixed]
tx-tcp6-segmentation: off [fixed]
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: off [fixed]
tx-vlan-offload: off [fixed]
[...]
ntuple-filters: off [fixed]
receive-hashing: off [fixed]
highdma: off [fixed]
rx-vlan-filter: on [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-udp_tnl-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
```

- In some cases external influences like layer2 prevent most offloads (the example here)
TCPDump

- Characteristics: dumps network traffic to console/file
- Objective: analyze packets of applications manually
- Usage: “tcpdump ...”
- Package: RHEL: tcpdump SLES: tcpdump

```
tcpdump host pserver1
 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
 listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

```
13:30:00.326581 IP pserver1.boeblingen.de.ibm.com.38620 > p10lp35.boeblingen.de.ibm.com.ssh: Flags [.], ack 3142, win 102, options [nop,nop,TS val 972996696 ecr 346994], length 0
```

```
```

```
13:30:00.375491 IP pserver1.boeblingen.de.ibm.com.38620 > p10lp35.boeblingen.de.ibm.com.ssh: Flags [.], ack 3222, win 102, options [nop,nop,TS val 972996709 ecr 346996], length 0
```

```
[...]
^C
31 packets captured
31 packets received by filter
0 packets dropped by kernel
```

- Not all devices support dumping packets in older distribution releases
  - Also often no promiscuous mode
- Check flags or even content if your expectations are met
- -w flag exports captured unparsed data to a file for later analysis in libpcap format
  - Also supported by wireshark
- Usually you have to know what you want to look for
Wireshark

- Characteristics: Analyzes captured network traffic
- Objective: In depth analysis of handshakes, missing replies, protocols, ...
- Usage: Dump in libpcap or pcap-ng format (tcpdump, dumpcap)
  then analyze on remote system via “wireshark”
- Package: RHEL: wireshark SLES: wireshark

- No “direct” invocation on System z usually
  - e.g. on RH6 there is not even a wireshark binary
- Scrolling huge files on Remote X isn't fun anyway
  - Capturing tools are available

- Custom columns and profiles are important to visualize what you want to look for
- For more details you might start at
  - The share sessions of Mathias Burkhard
    https://share.confex.com/share/121/webprogram/Session13282.html
  - Official documentation http://www.wireshark.org/docs/wsug_html/
Wireshark example

- 1. Dump via "tcpdump -w" or wireshark’s "dumpcap"
- 2. Analyze on remote system via "wireshark"

```
tcpdump host pserver1 -w traceme
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C
40 packets captured
40 packets received by filter
0 packets dropped by kernel
```

scp to my system

```
wireshark traceme
```

![Wireshark Packet Capture](image-url)

Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: IBM_... [source MAC address]
> Internet Protocol Version 4, Src: 9.152.140.75 (9.152.140.75), Dst: 9.152.140.6 (9.152.140.6)
> Transmission Control Protocol, Src Port: ssh (22), Dst Port: 39528 (39528), Seq: 22, Ack: 22, Len: 0
ziomon

- Characteristics: in depth zfcp based I/O analysis
- Objective: Analyze your FCP based I/O
- Usage: “ziomon” → “ziorep*”
- Package: RHEL: s390utils(-ziomon) SLES: s390-tools

Be aware that ziomon can be memory greedy if you have very memory constrained systems
The has many extra functions please check out the live virtual class of Stephan Raspl
- Replay: http://ibmstg.adobeconnect.com/p7zvdjz0yye/
Questions

- Further information is available at
  - Linux on System z – Tuning hints and tips
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/

Christian Ehrhardt
Linux on System z
Performance Evaluation

Research & Development
Schönaicher Strasse 220
71032 Böblingen, Germany

ehrhardt@de.ibm.com