

# Linux for System z

## Performance Tools for Problem Determination

Martin Schwidefsky  
 IBM Lab Böblingen, Germany  
 March 12 2014  
 Session 14559



# Trademarks & Disclaimer

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml):

IBM, the IBM logo, BladeCenter, Calibrated Vectored Cooling, ClusterProven, Cool Blue, POWER, PowerExecutive, Predictive Failure Analysis, ServerProven, System p, System Storage, System x, System z, WebSphere, DB2 and Tivoli are trademarks of IBM Corporation in the United States and/or other countries. For a list of additional IBM trademarks, please see <http://www.ibm.com/legal/copytrade.shtml>.

The following are trademarks or registered trademarks of other companies: Java and all Java based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries or both Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. UNIX is a registered trademark of The Open Group in the United States and other countries or both. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. InfiniBand is a trademark of the InfiniBand Trade Association.

Other company, product, or service names may be trademarks or service marks of others.

NOTES: Linux penguin image courtesy of Larry Ewing ([lewing@isc.tamu.edu](mailto:lewing@isc.tamu.edu)) and The GIMP

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Users of this document should verify the applicable data for their specific environment. IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Information is provided "AS IS" without warranty of any kind. All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area. All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

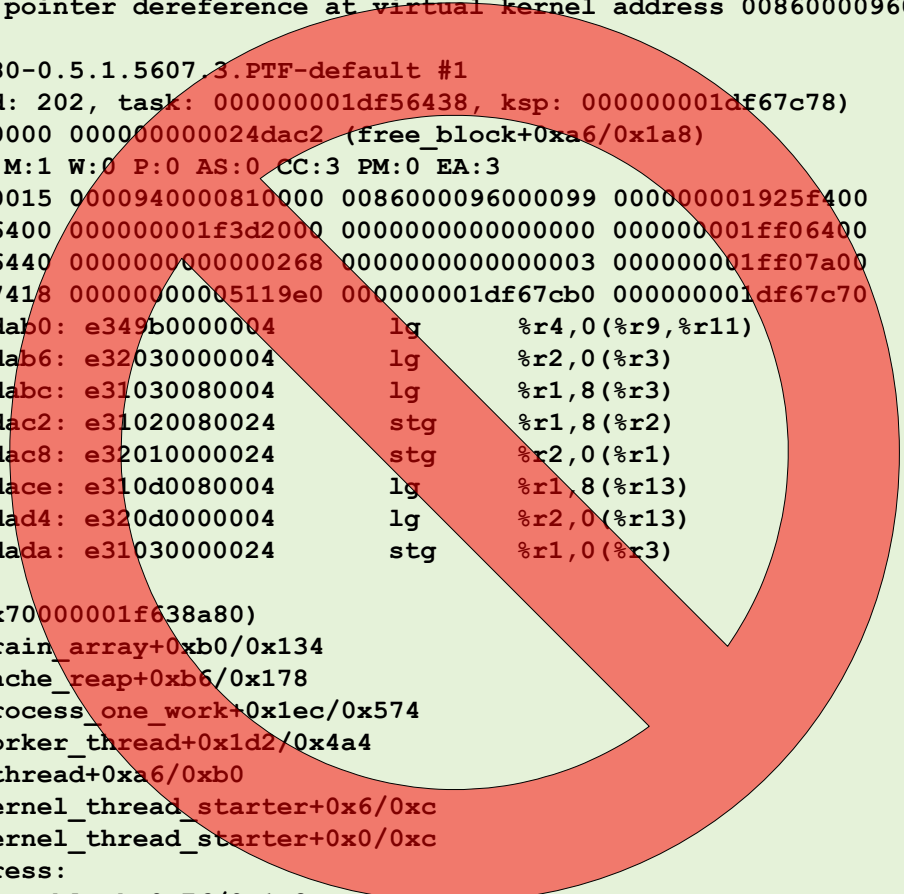
Prices are suggested US list prices and are subject to change without notice. Starting price may not include a hard drive, operating system or other features. Contact your IBM representative or Business Partner for the most current pricing in your geography. Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use. The information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any

# Agenda

- Introduction
- Start the analysis
- Formulate a theory
- Looking into the problem area
  - CPU related issues
  - Memory related issues
  - I/O related issues
  - Networking related issues
- A look at monitoring

# What this is not about

## ■ Functional problems / crashes

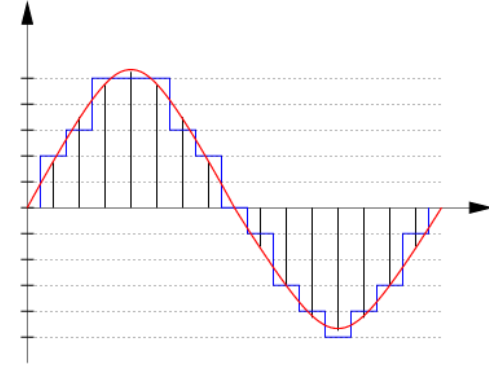


```

Unable to handle kernel pointer dereference at virtual kernel address 0086000096000000
Oops: 0038 [#1] SMP
CPU: 1 Not tainted 3.0.80-0.5.1.5607.3.PTF-default #1
Process kworker/1:4 (pid: 202, task: 000000001df56438, ksp: 000000001df67c78)
Krn1 PSW : 0404300180000000 00000000024dac2 (free_block+0xa6/0x1a8)
R:0 T:1 IO:0 EX:0 Key:0 M:1 W:0 P:0 AS:0 CC:3 PM:0 EA:3
Krn1 GPRS: 0000000000000015 0000940000810000 0086000096000099 000000001925f400
           000000001ff06400 000000001f3d2000 0000000000000000 000000001ff06400
           000000001ff06440 0000000000000268 0000000000000003 000000001ff07a00
           000000001fbb7418 00000000005119e0 000000001df67cb0 000000001df67c70
Krn1 Code: 000000000024dab0: e349b00000004      lg      %r4,0(%r9,%r11)
           000000000024dab6: e320300000004      lg      %r2,0(%r3)
           000000000024dabc: e310300800004      lg      %r1,8(%r3)
>000000000024dac2: e310200800024      stg     %r1,8(%r2)
           000000000024dac8: e320100000024      stg     %r2,0(%r1)
           000000000024dace: e310d00800004      lg      %r1,8(%r13)
           000000000024dad4: e320d00000004      lg      %r2,0(%r13)
           000000000024dada: e310300000024      stg     %r1,0(%r3)
Call Trace:
([<0700000001f638a80>] 0x70000001f638a80)
[<000000000024e004>] drain_array+0xb0/0x134
[<000000000024e32a>] cache_reap+0xb6/0x178
[<000000000016ccf4>] process_one_work+0x1ec/0x574
[<000000000016d542>] worker_thread+0xd2/0x4a4
[<0000000000176ac2>] kthread+0xa6/0xb0
[<00000000004e926e>] kernel_thread_starter+0x6/0xc
[<00000000004e9268>] kernel_thread_starter+0x0/0xc
Last Breaking-Event-Address:
[<000000000024da92>] free_block+0x76/0x1a8
  
```

## General thoughts on performance analysis

- Things that are always to consider
  - Monitoring will impact the system
  - Most data gathering averages over a period of time
    - this flattens peaks
  - Define the problem
    - which parameter(s) from the application/system indicates the problem
    - which range is considered bad, what is considered good
  - Monitor the good case and save the results
    - comparison good vs. bad can save a lot of time
- 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



# Start the analysis

“Taking a quick look”

## The “ps” tool

- Reports a snapshot of the currently running processes
- Usage: “ps axlf”, many more options available
- Example output

```
#> ps axlf
F    UID      PID  PPID  PRI   NI     VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
1      0        2      0   20    0        0      0 kthrea  S    ?          0:00 [kthreadd]
1      0        3      2   20    0        0      0 smpboo  S    ?          0:00  \_ [ksoftirqd/0]
5      0        4      2   20    0        0      0 worker  S    ?          0:00  \_ [kworker/0:0]
1      0        5      2    0 -20        0      0 worker  S<   ?          0:00  \_ [kworker/0:0H]
...
4      0         1      0   20    0   9672   5096 Sys_ep  Ss   ?          0:01 /sbin/init
4      0    2023      1   20    0   3584   1400 hrtime  Ss   ?          0:00 /usr/sbin/crond -n
4      0    2024      1   20    0   2840    892 pause   Ss   ?          0:00 /usr/sbin/atd -f
...
4      0    2280      1   20    0  10892   3344 poll_s  Ss   ?          0:00 /usr/sbin/sshd -D
4      0    2316    2280   20    0  14472   4700 poll_s  Ss   ?          0:00  \_ sshd: root@pts/0
4      0    2318    2316   20    0 106392   2216 wait    Ss   pts/0      0:00      \_ -bash
0      0    2350    2318   20    0 105576   1056 -        R+   pts/0      0:00          \_ ps axlf
1      0    2351    2318   20    0 106392    796 sleep_  D+   pts/0      0:00          \_ -bash
...
0      0    2368    2318   20    0   2192    360 -        R    pts/0     110:04 /usr/bin/app
...
```

## The “top” tool

- Shows resource usage on process level
- Usage: `“top -b -d [interval in sec] > [outfile]”`
- Shows
  - CPU utilization
  - Detailed memory usage
- Hints
  - Parameter -b enables to write the output for each interval to 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 instance
  - Use the 'W' key to write current configuration to ~/.toprc
    - becomes the default



# The “top” tool

## ■ Output example

```
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	0	migration	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.00	0	0	0	ksoftirqd	ksoftirqd/0
4	root	10	-5	0	0	0	S	0.0	0.0	0:00.13	0	0	0	worker_th	events/0
5	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	0	0	0	worker_th	khelper

## ■ Hints

- virtual memory:  $VIRT = SWAP + RES$  unit KB
- physical memory used:  $RES = CODE + DATA$  unit KB
- shared memory:  $SHR$  unit KB
- The “htop” tool is an alternative “top” with extra features

## The “hyptop” tool

- Show hypervisor performance data on System z
  - Check CPU and overhead statistics of your and sibling images
- Usage: “hyptop”
- Shows
  - CPU load and management overhead
  - Memory usage (only under z/VM)
  - Can show image overview or single image details
- Hints
  - Good “first view” tool for a look outside of a single image
  - Requirements
    - For z/VM the guest needs class B
    - For LPAR “Global performance data control” needs to be enabled

# The “hyptop” tool

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

11:12:56 CPU-T: UN(64)							
system (str)	#cpu (#)	cpu (%)	Cpu+ (hm)	online (dmh)	memuse (GiB)	memmax (GiB)	wcur (#)
R3729003	6	399.11	2:24	0:03:05	11.94	12.00	100
R3729004	6	399.07	2:24	0:03:05	11.94	12.00	100
R3729001	6	398.99	2:26	0:03:09	11.95	12.00	100
R3729005	6	398.76	2:24	0:03:05	11.94	12.00	100
R3729009	4	398.62	2:22	0:03:05	4.20	6.00	100
R3729008	4	398.49	2:22	0:03:05	4.21	6.00	100
R3729007	4	398.39	2:21	0:03:05	4.18	6.00	100
R3729010	4	398.02	2:21	0:03:05	4.18	6.00	100
R3729002	6	397.99	2:24	0:03:05	11.94	12.00	100
R3729006	4	393.09	2:21	0:03:05	4.17	6.00	100
R3729012	2	117.37	0:43	0:03:05	0.25	2.00	100
R3729014	2	117.27	0:44	0:03:05	0.25	2.00	100
R3729011	2	117.13	0:43	0:02:37	0.25	2.00	100
R3729013	2	117.08	0:43	0:03:05	0.25	2.00	100
R3729015	2	116.63	0:43	0:03:05	0.25	2.00	100
VMSEVVU	1	0.00	0:00	0:03:10	0.01	0.03	1500
VMSEVRP	1	0.00	0:00	0:03:10	0.01	0.06	1500
VMSEVRV	1	0.00	0:00	0:03:10	0.01	0.03	1500
RACFVM	1	0.00	0:00	0:03:10	0.01	0.02	100
OPERSYMP	1	0.00	0:00	0:03:10	0.01	0.03	100
TCPIP	1	0.00	0:00	0:03:10	0.01	0.12	3000
DTCVSW2	1	0.00	0:00	0:03:10	0.01	0.03	100
OPERATOR	1	0.00	0:00	0:03:10	0.00	0.03	100
RSCS	1	0.00	0:00	0:03:09	0.00	0.03	100
RSCSDNS	1	0.00	0:00	0:03:10	0.00	0.03	100
AUTOVM	1	0.00	0:00	0:03:10	0.00	0.03	100
GCS	1	0.00	0:00	0:03:10	0.00	0.02	100
LGLOPR	1	0.00	0:00	0:03:10	0.00	0.03	100
DIRMAINT	1	0.00	0:00	0:30:10	0.01	0.03	100
DTCVSW1	1	0.00	0:00	0:30:10	0.01	0.03	100

service guest weights

## The “vmstat” tool

- Report virtual memory statistics
- Usage: “vmstat [interval in sec]”
- Shows
  - Data per time interval
  - CPU utilization
  - Disk I/O
  - Memory usage / swapping
- Output example

```
#> vmstat 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
```

- Hints
  - Shared memory usage is listed under 'cache'

## The “pidstat” tool

- Report statistics for Linux tasks
  - Identify processes with peak activity
- Usage: “pidstat [-w | -r | -d]”
- 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

# The “pidstat” tool

## ■ Example output

Time	PM	PID	cswch/s	nvcswch/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

Time	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

Time	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

# Formulate a theory

“Make a wild guess”

## Formulate a theory

- Identify the most likely problem area
  - CPU related
  - Memory related
  - I/O related
  - Networking related
  - or a combination of several factors
  
- If the first look did not show anything obvious use monitoring
  - Needs preparation to capture the event of interest, e.g. at 4pm
  - Usually a lot of data is generated that needs to be analysed
  - Monitoring will impact your system



# CPU related issues

“Who is stealing my CPU ?”

## The “strace” tool

- Trace system calls and signals
- Usage: “strace -p [process id]”
- 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
  - High overhead, high detail tool which will slow down your application
  - Option “-c” allows medium overhead by just tracking counters and durations

# The “strace” tool

## Example output

shares to rate importance

a lot, slow or failing calls?

system call name (see man pages)

```
strace -cf -p 26802
Process 26802 attached - interrupt to quit
^Process 26802 detached
```

% time	seconds	usecs/call	calls	errors	syscall
58.43	0.007430	17	450		read
24.33	0.003094	4	850	210	access
5.53	0.000703	4	190	10	open
4.16	0.000529	3	175		write
2.97	0.000377	2	180		munmap
1.95	0.000248	1	180		close
1.01	0.000128	1	180		mmap
0.69	0.000088	18	5		fdatasync
0.61	0.000078	0	180		fstat
0.13	0.000017	3	5		pause
100.00	0.012715		2415	225	total

## The “ltrace” tool

- A library call tracer
- Usage: “ltrace -p [process id]”
- 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
  - High overhead, high detail tool which will slow down your application
  - Option “-c” allows medium overhead by just tracking counters and durations
  - Option “-s” allows to combine ltrace and strace

# The “ltrace” tool

## Example output

shares to rate importance

a lot or slow calls?

library call name (see man pages)

```
ltrace -cf -p 2680
```

% time	seconds	usecs/call	calls	function
98.33	46.765660	5845707	8	pause
0.94	0.445621	10	42669	strcmp
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

## The “perf” tool

- Performance analysis tools for Linux
  - Get detailed information where & why CPU is consumed
- Usage: “perf top”, “perf stat <cmd>”, “perf record/perf diff”
- Shows
  - Sampling for CPU hotspots
    - Annotated source code along hotspots, list functions according to their usage
  - CPU event counters
  - Kernel tracepoints
- Hints
  - Without HW support only userspace can be reasonably profiled
  - “successor” of oprofile that is available with HW support (SLES11-SP2)
  - Perf HW is upstream, wait for next distribution releases
  - Won't help with I/O wait or CPU stalls

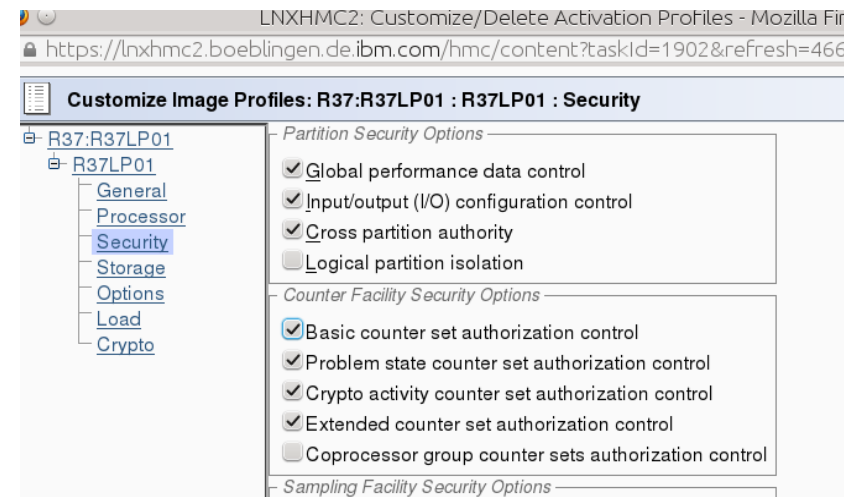
# The “perf” tool

## ■ Example output (perf diff)

```
# Baseline   Delta
# .....
#
12.14%      +8.07% [kernel.kallsyms] [k] lock_acquire
 8.96%      +5.50% [kernel.kallsyms] [k] lock_release
 4.83%      +0.38% reaim [.] add_long
 4.22%      +0.41% reaim [.] add_int
 4.10%      +2.49% [kernel.kallsyms] [k] lock_acquired
 3.17%      +0.38% libc-2.11.3.so [.] msort_with_tmp
 3.56%      -0.37% reaim [.] string_rtns_1
 3.04%      -0.38% libc-2.11.3.so [.] strncat
```

## ■ Preparation

- The cpu measurement facility needs to be enabled



# Memory related issues

“Who is eating all the memory ?”



## The “smem” tool

- Memory usage details per process/mapping
- 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: <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)

# The “smem” tool

## ■ Example output

```
# smem -tk -c "pid user command swap vss uss pss rss"
  PID User      Command      Swap      VSS      USS      PSS      RSS
  1860 root      /sbin/agetty -s sclp_line0    0      2.1M     92.0K    143.0K    656.0K
  1861 root      /sbin/agetty -s ttysclp0 11    0      2.1M     92.0K    143.0K    656.0K
   493 root      /usr/sbin/atd -f              0      2.5M    172.0K    235.0K    912.0K
  1882 root      /sbin/udev      0      2.8M    128.0K    267.0K    764.0K
  1843 root      /usr/sbin/crond -n            0      3.4M    628.0K    693.0K     1.4M
   514 root      /bin/dbus-daemon --system -    0      3.2M    700.0K    771.0K     1.5M
   524 root      /sbin/rsyslogd -n -c 5        0    219.7M    992.0K     1.1M     1.9M
 2171 root      ./hhhptest              0      5.7G     1.0M     1.2M     3.2M
  1906 root      -bash                0    103.8M     1.4M     1.5M     2.1M
 2196 root      ./hhhptest              0      6.2G     2.0M     2.2M     3.9M
 1884 root      sshd: root@pts/0        0     13.4M     1.4M     2.4M     4.2M
     1 root      /sbin/init              0      5.8M     2.9M     3.0M     3.9M
 2203 root      /usr/bin/python /usr/bin/sm    0   109.5M     6.1M     6.2M     6.9M
```

## ■ 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

# The “smem” tool

## ■ Example output

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

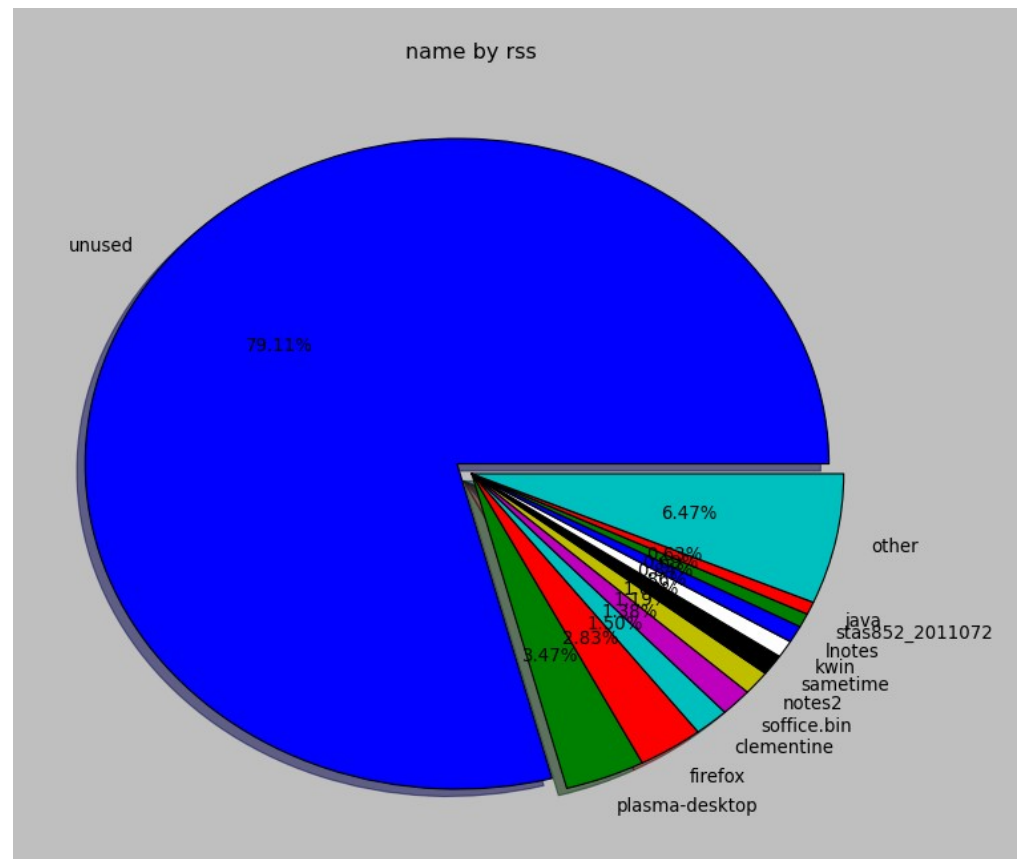
Map	Count	PIDs	Swap	VSS	USS	RSS	PSS	AVGRSS	AVGPSS
[stack:531]	1	1	0	8.0M	0	0	0	0	0
[vdso]	25	25	0	200.0K	0	132.0K	0	5.0K	0
/dev/zero	2	1	0	2.5M	4.0K	4.0K	4.0K	4.0K	4.0K
/usr/lib64/sasl2/libsasldb.so.2.0.23	2	1	0	28.0K	4.0K	4.0K	4.0K	4.0K	4.0K
/bin/dbus-daemon	3	1	0	404.0K	324.0K	324.0K	324.0K	324.0K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

## ■ 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
- Averages as there can be multiple mappers

## The “smem” tool

- Example of a memory distribution Visualization (many options)
- Warning in regard to monitoring the `/proc/<pid>/smaps` interface is expensive



## The “/proc/meminfo” interface

- How the kernel has allocated the memory

```
# cat /proc/meminfo
MemTotal:      889520 kB
MemFree:       749820 kB
Buffers:       10956 kB
Cached:        58844 kB
SwapCached:    0 kB
Active:        27140 kB
Inactive:      55760 kB
Active(anon):  13292 kB
Inactive(anon): 1184 kB
Active(file):  13848 kB
Inactive(file): 54576 kB
Unevictable:   0 kB
Mlocked:       0 kB
SwapTotal:     1048556 kB
SwapFree:      1048556 kB
Dirty:         0 kB
Writeback:     0 kB
AnonPages:     13204 kB
Mapped:        7528 kB
```

```
Shmem:         1352 kB
Slab:        39544 kB
SReclaimable:  18200 kB
SUnreclaim:    21344 kB
KernelStack:   2208 kB
PageTables: 608 kB
NFS_Unstable:  0 kB
Bounce:        0 kB
WritebackTmp:  0 kB
CommitLimit:   1493316 kB
Committed_AS:  52680 kB
VmallocTotal:  130023424 kB
VmallocUsed:    118616 kB
VmallocChunk:  129903444 kB
AnonHugePages: 0 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize:  1024 kB
```

## The “slabtop” tool

- Display kernel slab cache information in real time
- Usage: “slabtop”
- Shows
  - Active / Total object number/size
  - Objects per Slab
  - Object Name and Size
  - Objects per Slab
- Hints
  - Option -o is used for one time output e.g. to gather debug data
  - Despite slab/slob/slub in kernel its always slabtop

# The “slabtop” tool

## ■ Example output

```
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
```

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

How is kernel memory managed by the sl[auo]b allocator used

- Named memory pools or generic kmalloc pools
- Active/total objects and their size
- growth/shrinks of caches due to workload adaption

## I/O related issues

“Why are disk always to slow?”



## The “iostat” tool

- Report input/output statistics for devices and partitions
- Usage: “`iostat -xtdk [interval in sec]`”
- 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

# The “iostat” tool

## ■ Example output

Time: 10:56:35 AM

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
dasda	0.19	1.45	1.23	0.74	64.43	9.29	74.88	0.01	2.65	0.80	0.16
dasdb	0.02	232.93	0.03	9.83	0.18	975.17	197.84	0.98	99.80	1.34	1.33

Time: 10:56:36 AM

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
dasda	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dasdb	0.00	1981.55	0.00	339.81	0.00	9495.15	55.89	0.91	2.69	1.14	38.83

Time: 10:56:37 AM

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
dasda	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dasdb	0.00	2055.00	0.00	344.00	0.00	9628.00	55.98	1.01	2.88	1.19	41.00

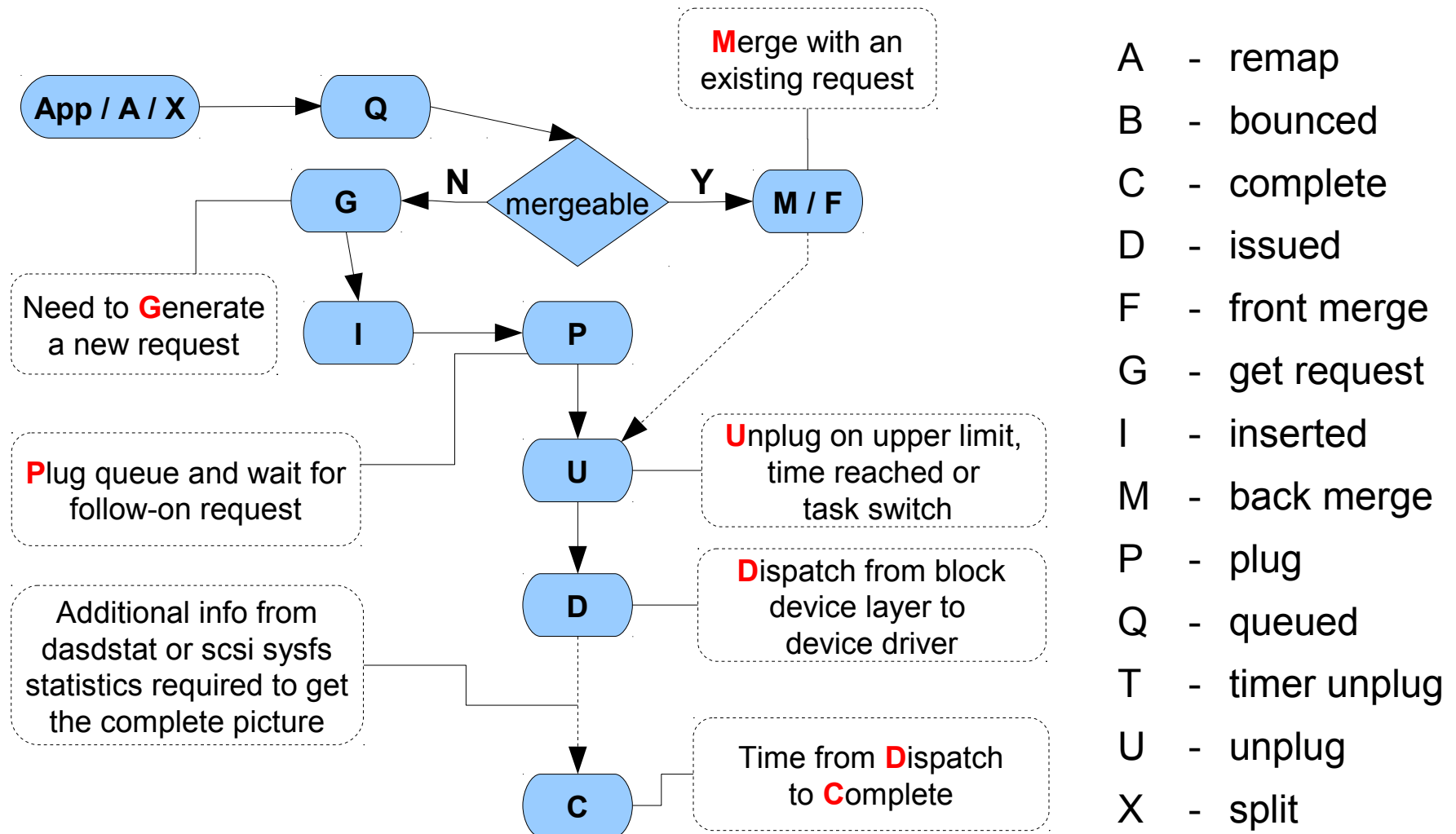
## The “blktrace” tool

- Generate traces of the I/O traffic on block devices
- Usage:
  - “blktrace -d [device(s)]”
  - “blktrace -st [commontracefilepart]”
- 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

## The “blktrace” tool

- Often its easy to identify that I/O is slow
  - but where?
  - and why?
- 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

# The “blktrace” tool – block device events (simplified)



# The “blktrace” tool

## ■ Example output

- 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

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

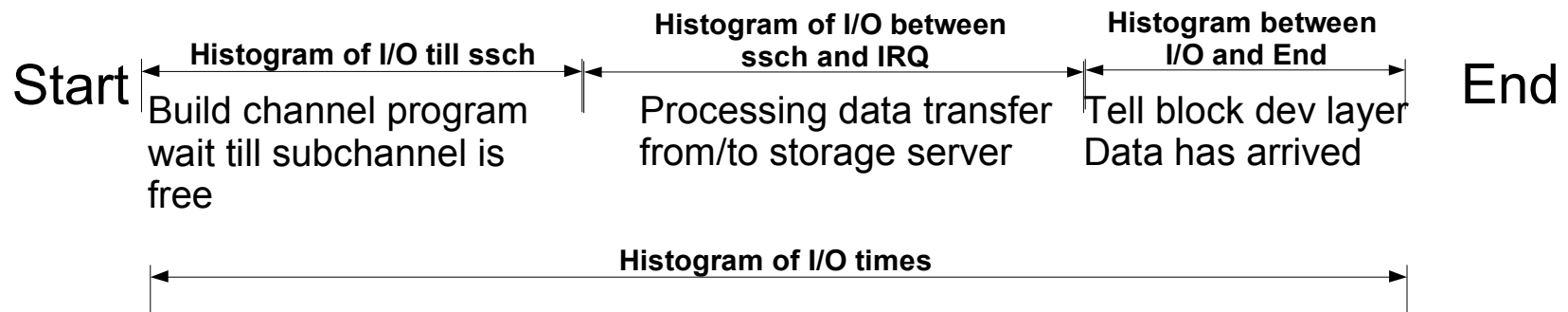


Action codes

## The DASD statistics

- 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

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



## The FCP statistics

- Collects statistics of I/O operations on FCP devices on request base
- Usage:
  - CONFIG\_STATISTICS=y must be enable for the kernel build
  - debugfs is mounted at /sys/kernel/debug/
  - for each FCP device there is a LUN directory  
`/sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>`
  - to enable, do “echo on=1 > definition”
  - to disable, do “echo on=0 > definition”
  - to reset, do “echo data=reset > definition”
  - to view, do “cat 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



# The 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



## IRQ statistics

- Condensed overview of IRQ activity
- Usage: `cat /proc/interrupts` and `cat /proc/softirqs`
- 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

	CPU0	CPU1	CPU2	CPU3	
EXT:	21179	24235	22217	22959	
I/O:	1542959	340076	356381	325691	
CLK:	15995	16718	15806	16531	[EXT] Clock Comparator
EXC:	255	325	332	227	[EXT] External Call
EMS:	4923	7129	6068	6201	[EXT] Emergency Signal
TMR:	0	0	0	0	[EXT] CPU Timer
TAL:	0	0	0	0	[EXT] Timing Alert
PFL:	0	0	0	0	[EXT] Pseudo Page Fault
DSD:	0	0	0	0	[EXT] DASD Diag
VRT:	0	0	0	0	[EXT] Virtio
SCP:	6	63	11	0	[EXT] Service Call
IUC:	0	0	0	0	[EXT] IUCV
CPM:	0	0	0	0	[EXT] CPU Measurement
CIO:	163	310	269	213	[I/O] Common I/O Layer Interrupt
QAI:	1 541 773	338 857	354 728	324 110	[I/O] QDIO Adapter Interrupt
DAS:	1023	909	1384	1368	[I/O] DASD
[...] 3215, 3270, Tape, Unit Record Devices, LCS, CLAW, CTC, AP Bus, Machine Check					

## IRQ statistics

- 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

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

# Networking related issues

“Is there anybody out there?”

## The “netstat” tool

- Print network connections, routing tables, interface statistics and more
- Usage: `“netstat -eeapn”` list connections  
`“netstat -s”` display summary statistics
- Shows
  - Information about each connection and various connection states
  - Information in regard to each protocol
  - Amount of incoming and outgoing packages
  - Various error states, for example TCP segments retransmitted!
- Hints
  - Inodes and program names are useful to reverse-map ports to applications
  - Option -s shows accumulated values since system start
  - There is always a low amount of packets in error or resets
  - Dropped segments show up on the sender side as retransmits
  - Use `sadc/sar` to identify the device

## The “netstat” tool

- Example output “netstat -s”

```
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.
```

## The socket statistics “ss” tool

- Another utility to investigate sockets
- Usage: “ss -aempi”
- Shows
  - Socket options
  - Socket receive and send queues
  - Inode, socket identifiers
- Example 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



## The “iptraf” tool

- Interactive colorful IP LAN monitor
- Usage: “iptraf”
- Shows
  - Live information on network devices / connections
  - 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

# The “iptraf” tool

- 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)

Packet  
sizes

**IPtraf**  
Packet size brackets for interface eth0

Packet Size Bracket	Count	Packet Size Bracket	Count
1 to 75:	2274	751 to 825:	2
76 to 150:	37	826 to 900:	0
151 to 225:	25	901 to 975:	3
226 to 300:	84	976 to 1050:	1
301 to 375:	10	1051 to 1125:	6
376 to 450:	27	1126 to 1200:	1
451 to 525:	16	1201 to 1275:	2
526 to 600:	38	1276 to 1350:	5
601 to 675:	5	1351 to 1425:	2864
676 to 750:	4	1426 to 1500+:	7

Interface MTU is 1500 bytes, not counting the data-link header  
Maximum packet size is the MTU plus the data-link header length  
Packet size computations include data-link headers, if any

	Total Packets	Total Bytes	Incoming Packets	Incoming Bytes	Outgoing Packets	Outgoing Bytes
<b>Total:</b>	44	11089	30	9101	14	1988
<b>IP:</b>	44	10473	30	8681	14	1792
<b>TCP:</b>	19	4120	9	3483	10	637
<b>UDP:</b>	25	6353	21	5198	4	1155
<b>ICMP:</b>	0	0	0	0	0	0
<b>Other IP:</b>	0	0	0	0	0	0
<b>Non-IP:</b>	0	0	0	0	0	0
<b>Total rates:</b>		1.0 kbits/sec 1.2 packets/sec		<b>Broadcast packets:</b> 21 <b>Broadcast bytes:</b> 5492		
<b>Incoming rates:</b>		0.7 kbits/sec 0.6 packets/sec				
<b>Outgoing rates:</b>		0.3 kbits/sec 0.6 packets/sec		<b>IP checksum errors:</b> 0		

IF  
details

# The “tcpdump” tool

- analyze packets of applications manually
- Usage: “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

13:30:00.338239 IP p10lp35.boeblingen.de.ibm.com.ssh > pserver1.boeblingen.de.ibm.com.38620: Flags [P.], seq 3142:3222, ack 2262, win 2790, options [nop,nop,TS val 346996 ecr 972996696], length 80
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
  - Also supported by wireshark
- Usually you have to know what you want to look for

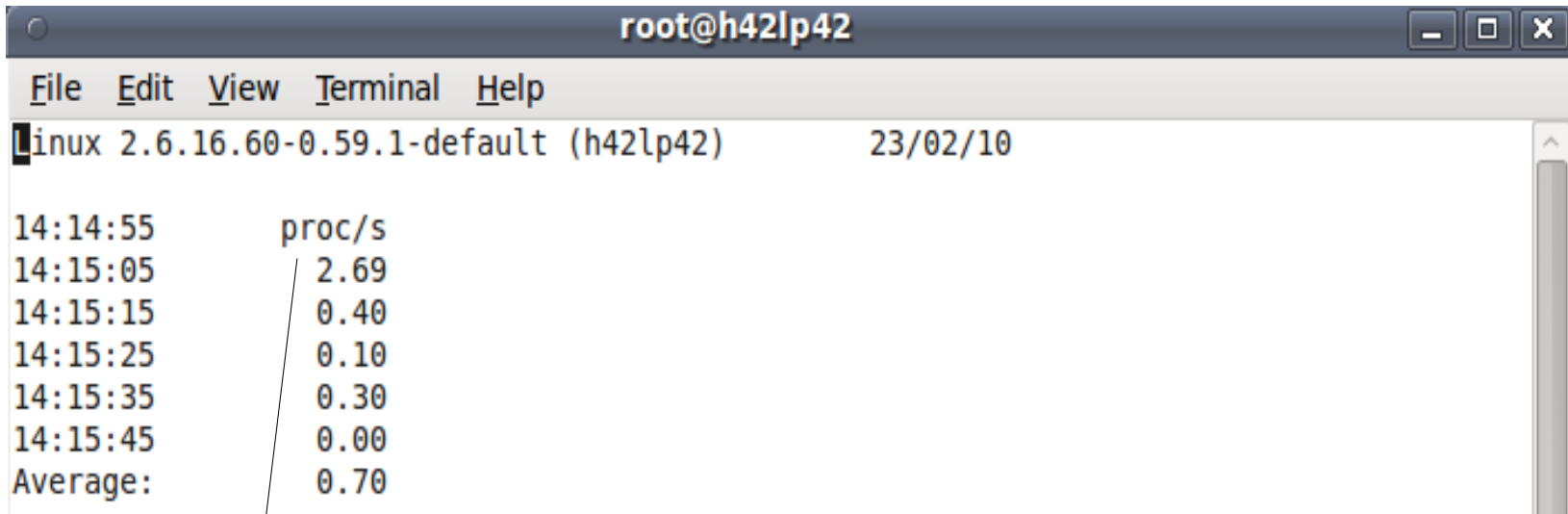
# Monitoring

“For the hard problems”

## The “SADC/SAR” tool

- System activity data collector and analysis
  - Suitable for permanent system monitoring and detailed analysis
- Usage: `“/usr/lib64/sa/sadc [-S XALL] [interval in sec] [outfile]”`  
`“sar -A -f [outfile]”`
- Shows
  - Reports statistics data over time and creates average values for each item
  - CPU and memory utilization
  - Disk I/O overview and on device level
  - Network I/O and errors on device level
  - ... and much more
- Hints
  - Shared memory is listed under 'cache'
  - [outfile] is a binary file, which contains all values. It is formatted using sar
  - sadc parameter “-S XALL” enables the gathering of further optional data
    - enables the creation of item specific reports, e.g. network only
    - enables the specification of a start and end time → time of interest

## The “SAR” output – processes created

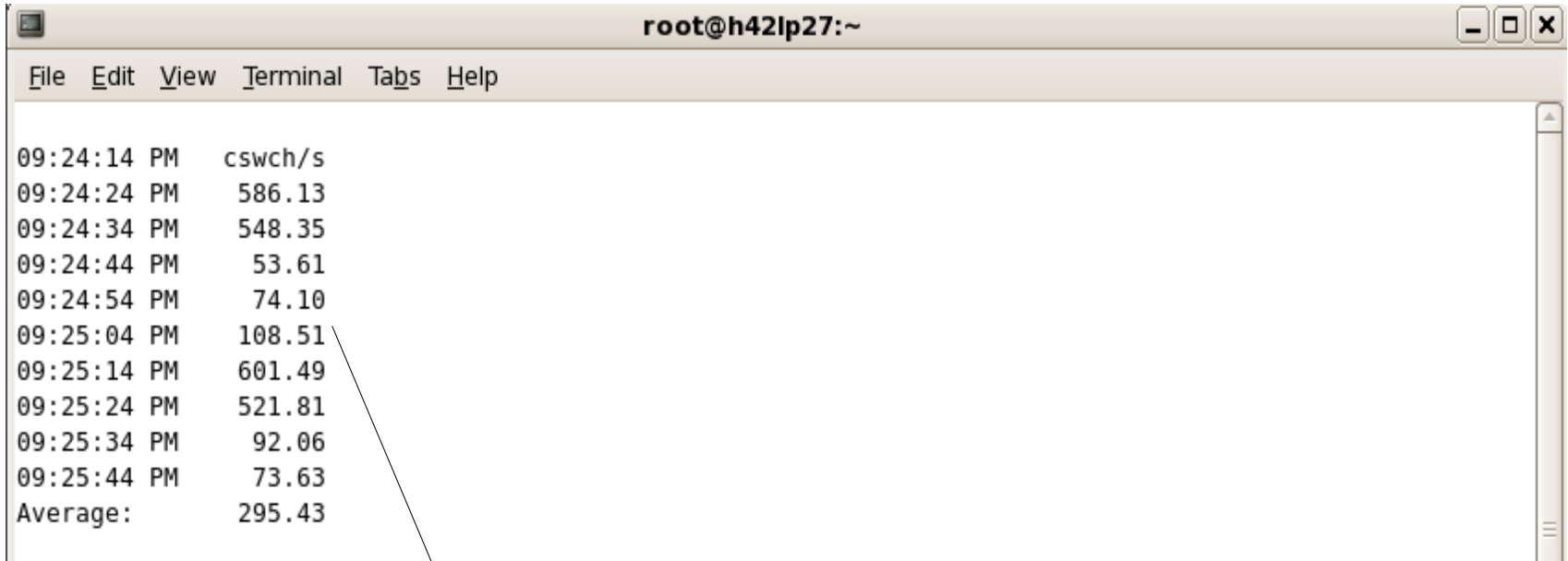


A terminal window titled 'root@h42lp42' showing the output of the 'sar' command for processes created per second. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The prompt is 'Linux 2.6.16.60-0.59.1-default (h42lp42)' and the date is '23/02/10'. The output shows a table with two columns: time and proc/s. The values are: 14:14:55 (proc/s), 14:15:05 (2.69), 14:15:15 (0.40), 14:15:25 (0.10), 14:15:35 (0.30), 14:15:45 (0.00), and Average: (0.70). A line points from the 'Average:' row to a text box below.

Time	proc/s
14:14:55	
14:15:05	2.69
14:15:15	0.40
14:15:25	0.10
14:15:35	0.30
14:15:45	0.00
Average:	0.70

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.

## The “SAR” output – context switch rate



A terminal window titled "root@h42lp27:~" displays the output of the 'sar' command for context switch rate. The output shows a table with two columns: time (HH:MM:SS PM) and cswch/s. The values range from 53.61 to 601.49. An average of 295.43 is shown at the bottom. A line points from the 'Average' row to a callout box.

Time	cswch/s
09:24:14 PM	cswch/s
09:24:24 PM	586.13
09:24:34 PM	548.35
09:24:44 PM	53.61
09:24:54 PM	74.10
09:25:04 PM	108.51
09:25:14 PM	601.49
09:25:24 PM	521.81
09:25:34 PM	92.06
09:25:44 PM	73.63
Average:	295.43

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

## The “SAR” output – CPU utilization

Per CPU values:

watch out for

system time (kernel)

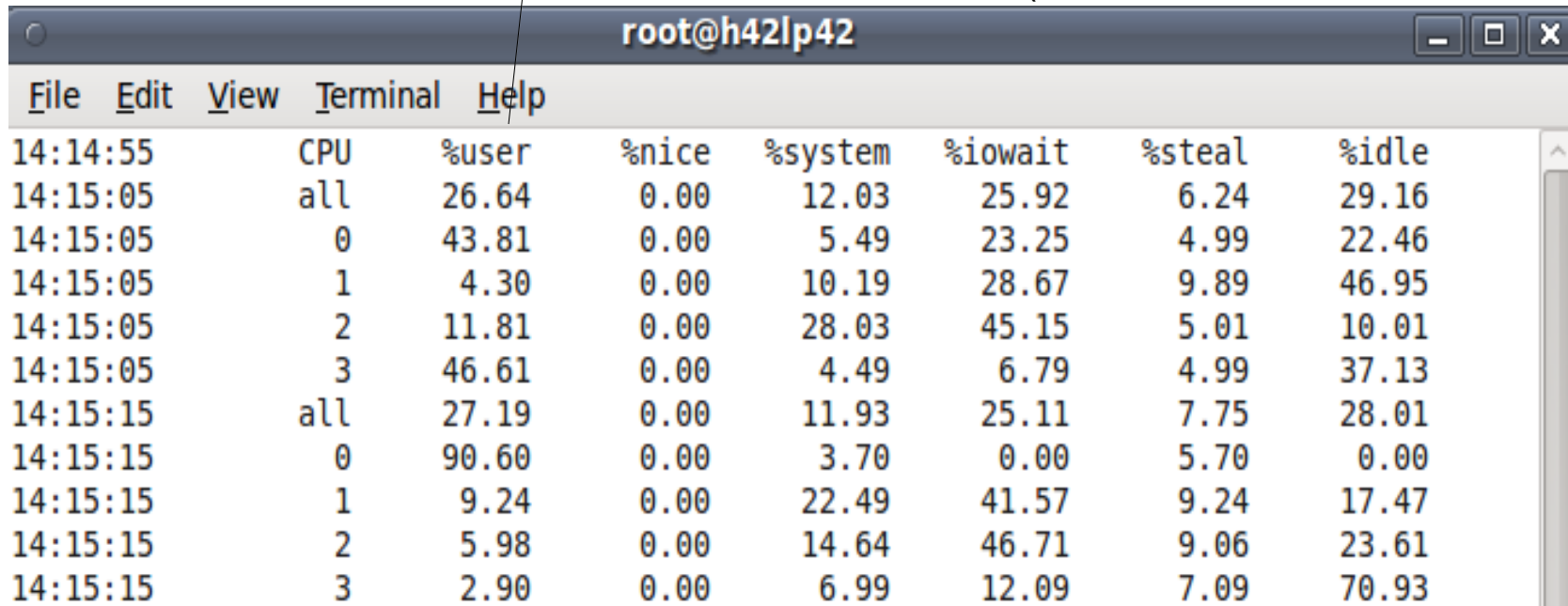
user (applications)

irq/soft (kernel, interrupt handling)

idle (nothing to do)

iowait time (runnable but waiting for I/O)

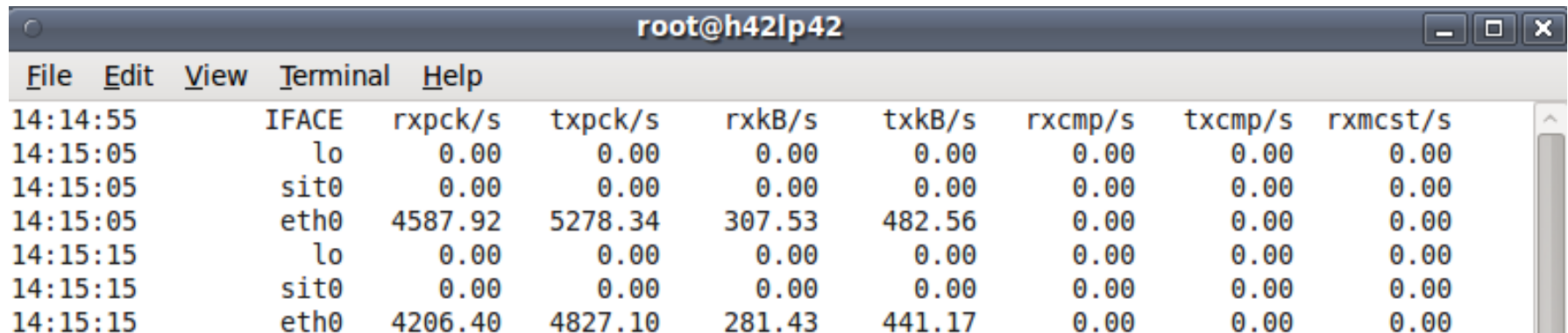
steal time (runnable but utilized somewhere else)



root@h42lp42							
	File	Edit	View	Terminal	Help		
14:14:55	CPU	%user	%nice	%system	%iowait	%steal	%idle
14:15:05	all	26.64	0.00	12.03	25.92	6.24	29.16
14:15:05	0	43.81	0.00	5.49	23.25	4.99	22.46
14:15:05	1	4.30	0.00	10.19	28.67	9.89	46.95
14:15:05	2	11.81	0.00	28.03	45.15	5.01	10.01
14:15:05	3	46.61	0.00	4.49	6.79	4.99	37.13
14:15:15	all	27.19	0.00	11.93	25.11	7.75	28.01
14:15:15	0	90.60	0.00	3.70	0.00	5.70	0.00
14:15:15	1	9.24	0.00	22.49	41.57	9.24	17.47
14:15:15	2	5.98	0.00	14.64	46.71	9.06	23.61
14:15:15	3	2.90	0.00	6.99	12.09	7.09	70.93



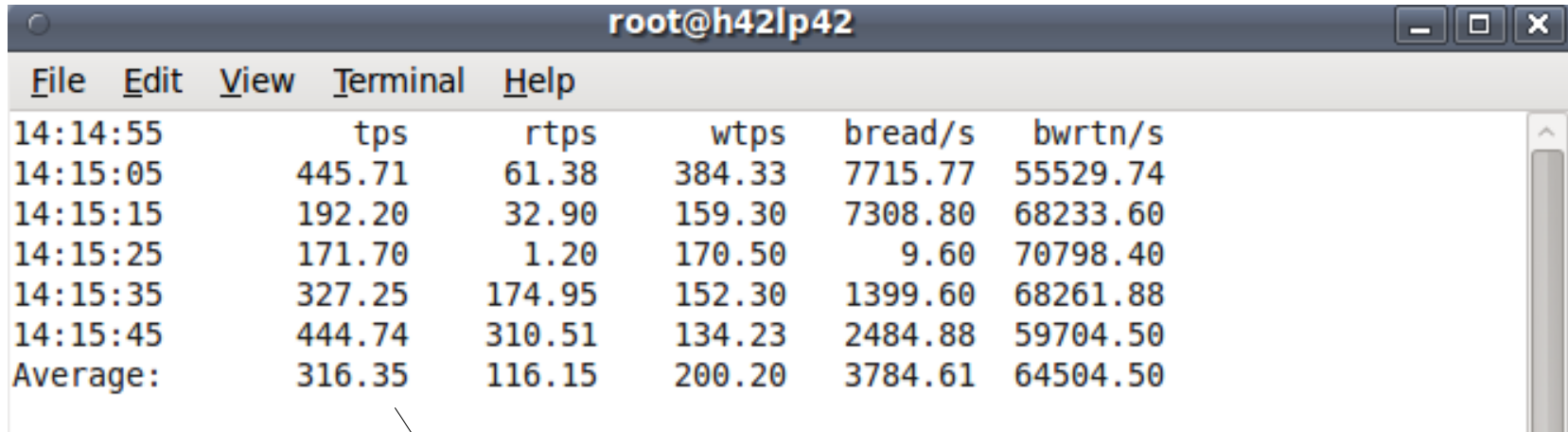
## The “SAR” output – network traffic



TIME	I/FACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcast/s
14:14:55								
14:15:05	lo	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:15:05	sit0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:15:05	eth0	4587.92	5278.34	307.53	482.56	0.00	0.00	0.00
14:15:15	lo	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:15:15	sit0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:15:15	eth0	4206.40	4827.10	281.43	441.17	0.00	0.00	0.00

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.

## The “SAR” output – disk I/O overall

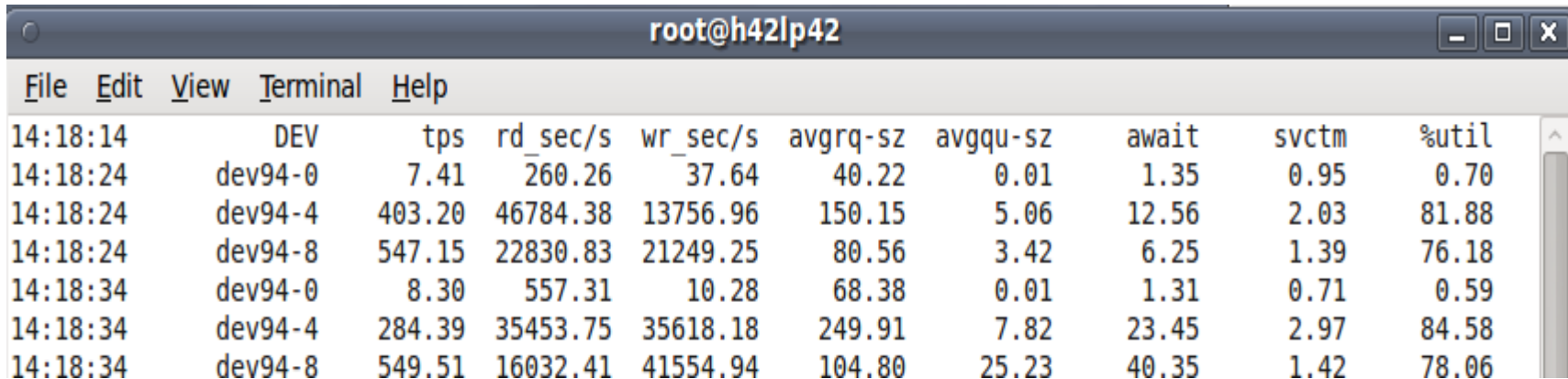


	tps	rtps	wtps	bread/s	bwrtn/s
14:14:55					
14:15:05	445.71	61.38	384.33	7715.77	55529.74
14:15:15	192.20	32.90	159.30	7308.80	68233.60
14:15:25	171.70	1.20	170.50	9.60	70798.40
14:15:35	327.25	174.95	152.30	1399.60	68261.88
14:15:45	444.74	310.51	134.23	2484.88	59704.50
Average:	316.35	116.15	200.20	3784.61	64504.50

Overview of

- operations per second
- transferred amount

## The “SAR” output – disk I/O per device



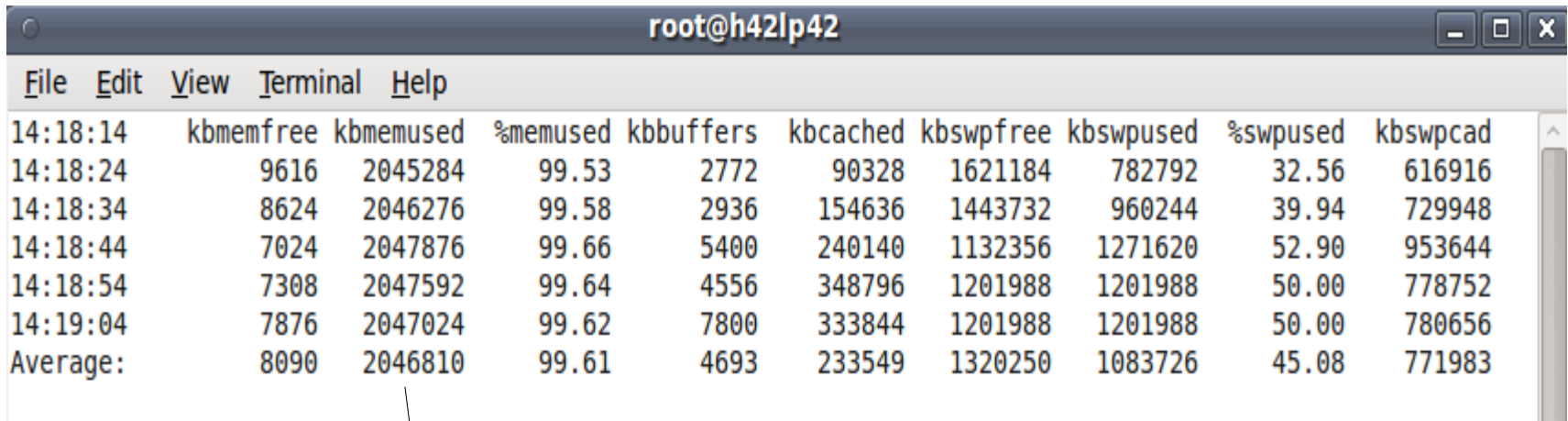
	DEV	tps	rd_sec/s	wr_sec/s	avgrq-sz	avgqu-sz	await	svctm	%util
14:18:14	DEV	tps	rd_sec/s	wr_sec/s	avgrq-sz	avgqu-sz	await	svctm	%util
14:18:24	dev94-0	7.41	260.26	37.64	40.22	0.01	1.35	0.95	0.70
14:18:24	dev94-4	403.20	46784.38	13756.96	150.15	5.06	12.56	2.03	81.88
14:18:24	dev94-8	547.15	22830.83	21249.25	80.56	3.42	6.25	1.39	76.18
14:18:34	dev94-0	8.30	557.31	10.28	68.38	0.01	1.31	0.71	0.59
14:18:34	dev94-4	284.39	35453.75	35618.18	249.91	7.82	23.45	2.97	84.58
14:18:34	dev94-8	549.51	16032.41	41554.94	104.80	25.23	40.35	1.42	78.06

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.

## The “SAR” output – memory statistics

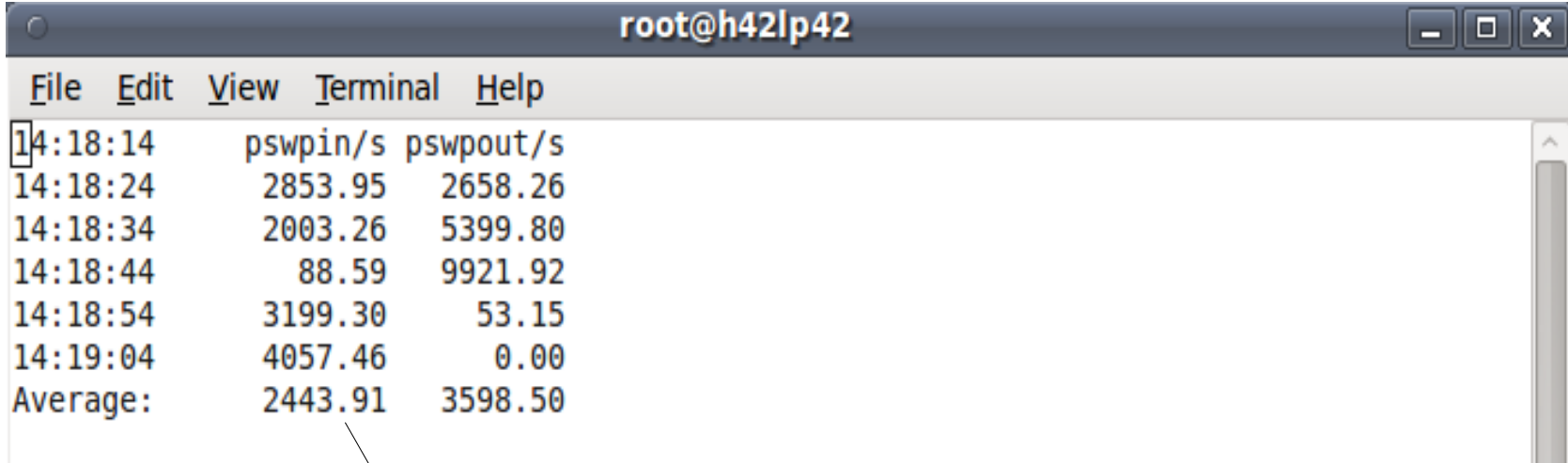


	kbmemfree	kbmemused	%memused	kbbuffers	kbcached	kbswpfree	kbswpused	%swpused	kbswpcad
14:18:14									
14:18:24	9616	2045284	99.53	2772	90328	1621184	782792	32.56	616916
14:18:34	8624	2046276	99.58	2936	154636	1443732	960244	39.94	729948
14:18:44	7024	2047876	99.66	5400	240140	1132356	1271620	52.90	953644
14:18:54	7308	2047592	99.64	4556	348796	1201988	1201988	50.00	778752
14:19:04	7876	2047024	99.62	7800	333844	1201988	1201988	50.00	780656
Average:	8090	2046810	99.61	4693	233549	1320250	1083726	45.08	771983

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

Same for swap – to use swap is actually good, but to access it (swpin/-out) all the time is bad.

## The “SAR” output – memory pressure / swap



	pswpin/s	pswpout/s
14:18:14		
14:18:24	2853.95	2658.26
14:18:34	2003.26	5399.80
14:18:44	88.59	9921.92
14:18:54	3199.30	53.15
14:19:04	4057.46	0.00
Average:	2443.91	3598.50

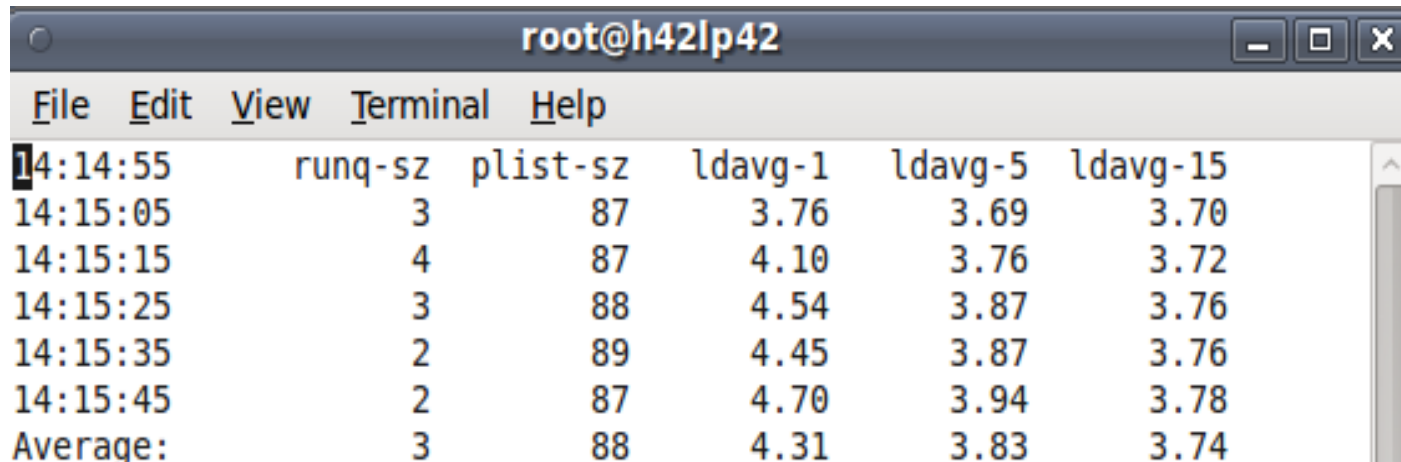
The percentage seen before can be high,  
But the swap rate shown here should be low.  
Ideally it is near zero after a rampup time.  
High rates can indicate memory shortages.

# The “SAR” output – memory pressure faults and reclaim

h37lp19 : root										
File	Edit	View	Scrollback	Bookmarks	Settings	Help				
10:12:15 AM	pgpgin/s	pgpgout/s	fault/s	majflt/s	pgfree/s	pgscank/s	pgscand/s	pgsteal/s	%vmeff	
10:12:17 AM	109.45	336.32	634.83	1.99	4710.95	0.00	0.00	0.00	0.00	
10:12:19 AM	174.00	18.00	109.00	1.00	76.50	0.00	0.00	0.00	0.00	
10:12:21 AM	0.00	18.00	36.00	0.00	71.00	0.00	0.00	0.00	0.00	
10:12:23 AM	826.00	327910.00	1697.00	8.50	64659.00	66066.50	5424.50	64285.50	89.92	
10:12:25 AM	577.11	715393.03	43.28	1.49	178377.61	110505.47	96352.24	178305.97	86.20	
10:12:27 AM	588.12	679320.79	43.07	1.49	169312.87	101317.82	94495.54	169250.00	86.43	
10:12:29 AM	1040.00	688822.00	62.00	2.50	171417.50	99329.50	100065.50	171355.50	85.94	
10:12:31 AM	698.04	663082.35	45.59	2.45	165792.65	93984.80	95946.57	165715.69	87.25	
10:12:33 AM	1212.12	624048.48	84.34	4.55	155524.75	90932.32	87934.85	155378.28	86.87	
10:12:35 AM	595.07	215950.74	68.47	2.46	54027.09	27919.70	32992.61	53903.45	88.49	
10:12:37 AM	558.00	159790.00	43.50	1.50	38183.00	18968.50	21232.00	38122.50	94.83	
10:12:39 AM	1569.85	21949.75	102.51	4.02	5976.38	3144.72	2990.95	5868.84	95.65	
10:12:41 AM	1081.55	527207.77	213.59	1.46	134243.20	65822.33	90253.40	134170.87	85.97	
10:12:43 AM	1718.59	702936.68	62.31	2.51	176173.37	86268.34	118320.10	176107.54	86.08	
10:12:45 AM	1237.44	683623.65	42.86	1.48	171228.57	83624.14	114011.33	171166.01	86.61	
10:12:47 AM	1269.39	699144.90	44.39	1.53	173979.08	89181.63	112045.41	173909.69	86.42	
10:12:49 AM	1691.54	677327.36	62.19	2.49	171114.93	89499.00	104974.13	171048.26	87.95	
10:12:51 AM	1979.70	285857.87	143.15	4.57	69777.16	37740.61	44834.01	69590.86	84.28	
10:12:53 AM	458.00	20.00	57.00	1.00	156.50	98.00	0.00	96.00	97.96	
10:12:55 AM	433751.72	5944.83	1818.23	24.14	109168.97	3466.01	210573.89	108376.85	50.63	
10:12:57 AM	924042.00	336.00	248.50	26.50	231500.50	2384.50	461443.00	231027.50	49.81	
10:12:59 AM	906810.00	214.00	225.00	25.00	226950.50	2117.00	447010.00	226605.50	50.45	
10:13:01 AM	917504.00	180.00	206.00	38.00	230020.50	2225.00	460268.00	229486.50	49.62	
10:13:03 AM	865062.00	348.00	464.00	58.00	216892.50	7677.50	419680.50	215976.00	50.54	
10:13:05 AM	12.06	20.10	42.21	0.00	267.84	160.80	16.08	176.88	100.00	
10:13:07 AM	770.15	123.38	201.99	9.95	266.17	131.34	0.00	130.85	99.62	
10:13:09 AM	484.42	20.10	64.32	2.51	263.82	192.96	0.00	192.96	100.00	
10:13:11 AM	16.00	20.00	38.50	0.50	96.00	0.00	0.00	0.00	0.00	
10:13:13 AM	0.00	20.00	36.00	0.00	95.50	0.00	0.00	0.00	0.00	
10:13:15 AM	2700.00	318.00	698.00	9.00	22343.00	301.00	0.00	287.00	95.35	
10:13:17 AM	2.00	20.00	36.00	0.00	60.50	0.00	0.00	0.00	0.00	
10:13:19 AM	0.00	20.00	36.00	0.00	61.50	0.00	0.00	0.00	0.00	
10:13:21 AM	5558.21	662674.63	2378.11	9.95	147166.17	77842.79	94354.23	146721.39	85.21	
10:13:23 AM	1039.22	1052684.31	60.78	2.45	260784.31	116591.18	193280.39	260718.14	84.14	
10:13:25 AM	611.76	1075607.84	42.14	1.06	266601.47	121706.57	107347.55	265036.76	84.83	

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

## The “SAR” output – system load



	runq-sz	plist-sz	ldavg-1	ldavg-5	ldavg-15
14:14:55					
14:15:05	3	87	3.76	3.69	3.70
14:15:15	4	87	4.10	3.76	3.72
14:15:25	3	88	4.54	3.87	3.76
14:15:35	2	89	4.45	3.87	3.76
14:15:45	2	87	4.70	3.94	3.78
Average:	3	88	4.31	3.83	3.74

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.

## The “dstat” tool

- Versatile tool for generating system resource statistics
- Usage: `dstat -tv -aio -disk-util -n -net-packets -i -ipc  
-D total,[diskname] -top-io [...] [interval]`
- Short: `dstat -vtin`
- 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



# The “dstat” tool

Report nice bugs to [bug-coreutils.gnu.org](mailto:bug-coreutils.gnu.org)  
 GNU coreutils home page: <<http://www.gnu.org/software/coreutils/>>  
 General help using GNU software: <<http://www.gnu.org/gethelp/>>  
 For complete documentation, run: `info coreutils 'nice invocation'`  
 [root@r3729001 ~]# `nice -n 1 dstat -tv --aio --disk-util -n --net-packets -i --ipc -D total,sda --top-io --noupdate 5`

```

----system----  ---procs---  -----memory-usage-----  ---paging--  -dsk/total--  -dsk/sda--  ---system--  ----total-cpu-usage----  async  sda-
time    run  blk new|  used  buff  cach  free|  in   out|  read  writ:  read  writ|  int   csw  |usr sys  idl  wai  hiq  siq|  #aio|util|
17-07 17:41:18| 0.0  0  38|1303M 13.5M 10.4G 57.4M| 0    0|4137k 14M: 124k 337k| 0    4968  | 4  3  92  0  0  1| 0    |1.59|
17-07 17:41:24| 13   0  0|1307M 13.5M 10.4G 57.2M| 0    0|1708k 30k:  45k   0| 0    16k| 33  9  55  0  0  3| 0    |0.20|
17-07 17:41:28| 9.4 0.2  0|1311M 13.5M 10.4G 59.0M| 0    0|1626k 19k:  60k   0| 0    15k| 63 15  17  0  0  5| 0    |0.20|
17-07 17:41:34| 13   0 0.2|1313M 13.5M 10.4G 59.5M| 0    0|1325k 11k:  32k   0| 0    11k| 71 17  6  0  0  6| 0    |0.40|
17-07 17:41:39| 3.6  0  0|1317M 13.6M 10.4G 60.8M| 0    0|1258k 23k:  26k   0| 0    16k| 76 18  0  0  0  6| 0    |0.40|
17-07 17:41:44| 13   0  0|1318M 13.6M 10.4G 53.1M| 0    0|1601k 16k:  37k   0| 0    14k| 75 19  0  0  0  6| 0    |0.40|
17-07 17:41:49| 11   0 0.2|1322M 13.6M 10.4G 82.2M| 0    0| 811k 16k:  31k   0| 0    13k| 67 28  0  0  0  5| 0    |0.20|
17-07 17:41:53| 12  0.4  0|1324M 13.6M 10.4G 68.7M| 0    0| 909k 3277B: 40k   0| 0    9820  |26  6  65  0  0  2| 0    |0.20|
  
```

similar to vmstat

similar to iostat  
(also per device)

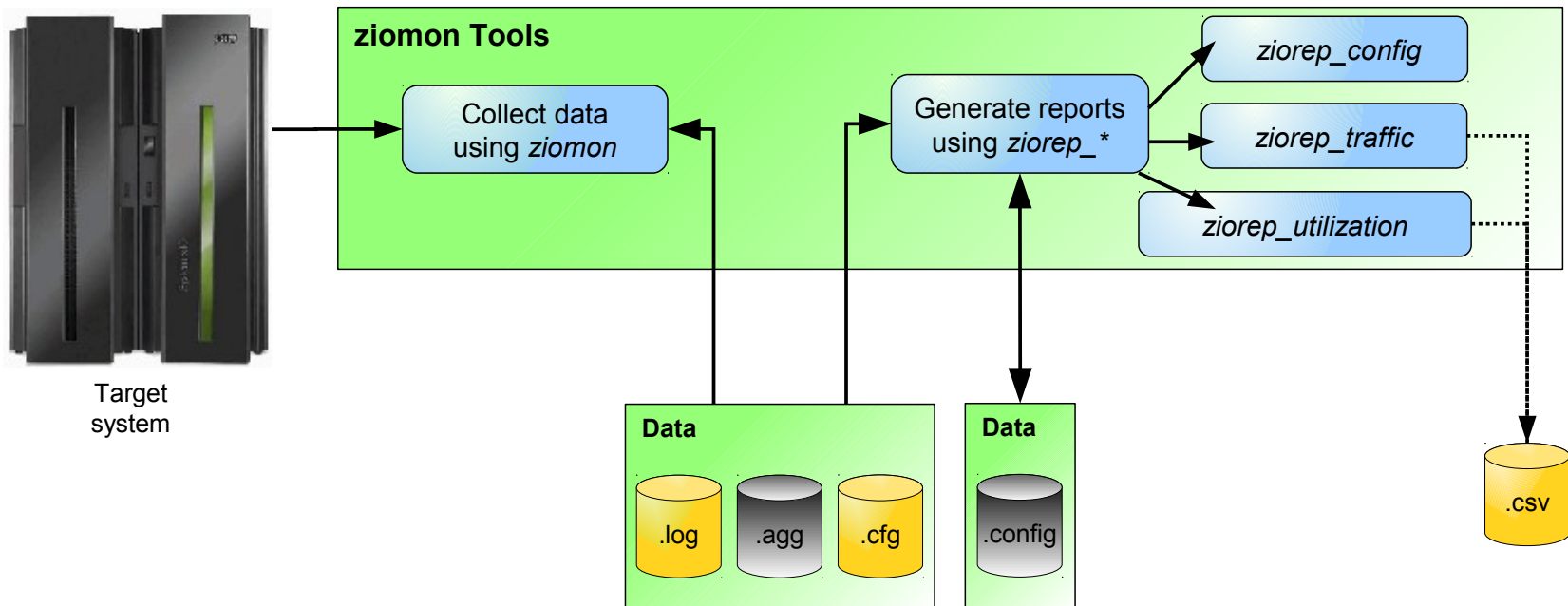
new in live tool

```

--net/total-  -pkt/total-  inter  --sysv-ipc-  ----most-expensive----
writ: read  writ|  int   csw  |usr sys idl  wai  hiq  siq|  #aio|util|  recv  send|#recv #send|  1  |msg sem shm|  i/o process
14M: 124k 337k| 0    4968  | 4  3  92  0  0  1| 0    |1.59|  0    0|  0    0| 300  | 0  35  1|sshd      15M 25M
30k:  45k   0| 0    16k| 33  9  55  0  0  3| 0    |0.20| 21B 426B|0.40 0.40| 81  | 0  35  1|postgres: p 78k  0
19k:  60k   0| 0    15k| 63 15  17  0  0  5| 0    |0.20| 10B 148B|0.20 0.20| 74  | 0  35  1|postgres: p 75k  0
11k:  32k   0| 0    11k| 71 17  6  0  0  6| 0    |0.40| 142B 148B|0.60 0.20| 62  | 0  35  1|postgres: p 141k 0
23k:  26k   0| 0    16k| 76 18  0  0  0  6| 0    |0.40| 133B 151B|0.60 0.20| 75  | 0  35  1|postgres: p 32k  0
16k:  37k   0| 0    14k| 75 19  0  0  0  6| 0    |0.40| 10B 151B|0.20 0.20| 73  | 0  35  1|postgres: p 152k 0
16k:  31k   0| 0    13k| 67 28  0  0  0  5| 0    |0.20| 162B 161B|0.80 0.40| 53  | 0  35  1|postgres: p 22k  0
3277B: 40k   0| 0    9820  |26  6  65  0  0  2| 0    |0.20| 10B 151B|0.20 0.20| 41  | 0  35  1|postgres: p 110k 0
  
```

# The ziomon

- Analyze your FCP based I/O
- Usage: “ziomon” → “ziorep\*”



- 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
  - PDF: <http://www.vm.ibm.com/education/lvc/LVC0425.pdf>
  - Replay: <http://ibmstg.adobeconnect.com/p7zvdjz0yye/>

## Questions

- Further information is available at
  - Linux on System z – Tuning hints and tips  
<http://www.ibm.com/developerworks/linux/linux390/perf/index.html>
  - Live Virtual Classes for z/VM and Linux  
<http://www.vm.ibm.com/education/lvc/>



***Martin Schwidefsky***

*Linux on System z  
Development*

*Schönaicher Strasse 220  
71032 Böblingen, Germany*

*Phone +49 (0)7031-16-2247  
[schwidefsky@de.ibm.com](mailto:schwidefsky@de.ibm.com)*