Prozessverwaltung

Prozesse anzeigen

ps

Anzeigen aller Prozesse: [1)]

```
ps -e
```

Anzeigen aller Prozesse und Threads:

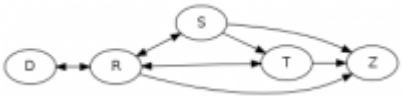
```
ps -T -e
```

Anzeigen aller Kernel-Threads:

```
ps -p 2 --ppid 2
```

Kernel-Threads die mit /nummer angezeigt werden laufen auf der CPU mit der entsprechenden Nummer. [2)]

Status



[3)]

Anzeigen aller Prozesse, incl. Benutzer: [4)]

```
ps -elf
```

Anzeige alle sortiert:

```
ps -e --sort rsz
```

Anzeige ausgewählter Felder sortiert:

```
ps -eo cmd,uname,rsz,cputime,%cpu,state --sort rsz
```

Anzeige ausgewählter Prozesse und ausgewählter Felder sortiert:

```
ps -C gzip,bzip2 -o cmd,uname,rsz,cputime,%cpu,state --sort rsz
```

Mit dem BSD Befehlssatz Alle Prozess, mit User und MEM/CPU-Statistik, als Wide (volle Kommandozeile) und als Baum (f)

```
ps ax
ps aux
ps auxw
ps auxwf
```

Anzeige aller Prozesse incl. cgroup:

```
alias psc='ps xawf -eo pid,user,cgroup,args'
psc
```

pgrep, lsof, pstree, top

Anzeige aller zip Prozesse:

```
pgrep -l zip
```

Anzeigen, welche Dateien ein Prozess geöffnet hat (`lsof` = list open files):

```
lsof -p 4711
```

oder

```
lsof -c update-notifier
```

Prozesse baumartig darstellen:

```
pstree
pstree -p
pstree -up
```

Prozessbaum der eigenen Shell (long)

```
pstree -pul -s $$
```

Prozesse nach Ressourcenverbrauch sortiert anzeigen:

```
top
```

Modernere Variante von top:

```
htop
```

Prozess Prioritäten

Priorität anzeigen

```
nice
```

oder

```
ps -p $$ -o pid,nice,cmd
```

Prozess mit um 5 abgesenkter Priorität starten:

```
nice -5 bash
```

Priorität eines laufenden Prozesses auf 3 ändern:

```
renice 3
```

Signale

Prozess normal beenden, Daten werden gepeichert:

```
kill 4711
kill -TERM 4711
```

Prozess sofort beenden, ohne speichern:

```
kill -KILL 4711
kill -9 4711
```

Prozess bitten, die Konfigurationsdatei neu einzulesen [5)]:

```
kill -HUP 4711
```

aktuelle Shell informieren, daß sich die Fenstergröße geändert hat

```
kill -WINCH $$
```

Signal für aktuelle Shell umdefinieren:

```
trap 'echo bäah!' TERM
kill -TERM $$
```

Alle zip-Prozesse beenden:

```
pkill zip
```

Beispiel: Zombie erzeugen

```
bash # vorher eine neue Shell aufmachen, damit das Terminal nicht schließt
```

```
sleep 30 &
exec sleep 60
```

Diesen dann von einem anderen Terminal aus beobachten

```
ps --forest -lf -t anderes_tty
```

Wieviel Speicher ist noch frei?

```
free -m
```

```
             total       used       free     shared    buffers     cached
Mem:           503        296        206          0         26        102
-/+ buffers/cache:        167        336
Swap:         1592          0       1592
```

Hier das ganze mal beschriftet, damit klar ist, wovon ich unten rede:

```
free -m
```

```
             total       used       free     shared    buffers     cached
Mem:           (1)        (2)        (3)        (4)        (5)        (6)
-/+ buffers/cache:        (7)        (8)
Swap:          (9)       (10)       (11)
```

1. Die physikalische Menge flüchtigen Speichers, abzüglich dessen, was für den Kernel und dessen Datenstrukturen draufgeht.
2. Der benutzte Speicher, also alles, was für Programme, deren Libraries, Datenhaltung draufgeht. Außerdem der Buffer- und der Pagecache.
3. Unbenutzter Speicher. Niemand hat dort Dinge abgelegt. Der Speicher kann direkt benutzt werden, wenn man welchen braucht. Wobei er u. U. natürlich initialisiert werden sollte
4. Der durch Shared Memory belegte Speicher.
5. Der durch den Buffer-Cache belegte Speicher
6. Der durch den Page Cache belegte Speicher
7. Der belegte Speicher von (2) abzüglich der Summe aus (5) und (6)
8. Der freie Speicher von (3) plus der Summe aus (5) und (6)
9. Die gesamte Menge an Swapspace, die zur Verfügung steht.
10. Der belegte Swapspace
11. Der noch freie Swapspace, also (9) abzüglich (10)

Das bedeutet also, dass im Falle ganz oben insgesamt 167M von Programmen und ähnlichem belegt ist und ich theoretisch 336M zusätzliche belegen könnte. Dann wäre aber der PageCache auf 0 und der Buffer Cache ebenfalls. Da das sehr auf die Performance ginge, würde der Kernel vorher bereits einige Programme, die selten laufen (z. B. ein unbeschäftigter exim oder irgendwelche anderen Daemons, die selten laufen) in den Swap verschieben. Zum schnellen Überblick, wie viel Speicher denn noch frei ist (also eigentlich, wie viel Speicher denn noch nutzbar ist), ist die Zahl unter (8) am besten als Indikator zu gebrauchen.

Beispiel: Cache füllen

In einem Fenster Speicher beobachten:

```
watch free -m
```

Im zweiten Fenster unsinnig Dateien lesen:

```
grep -r lakdsjf /usr/ 2>/dev/null >/dev/null
```

Beispiel: Cache leeren

In einem Fenster Speicher beobachten:

```
watch free -m
```

Im zweiten Fenster ein Programm viel Speicher verbrauchen lassen:

```
dd bs=1M count=1500 < /dev/zero | sort > /dev/null
```

Oder einfach:

```
echo 3 > /proc/sys/vm/drop_caches
```

Beispiel: Speicher füllen

Prozess starten, der 2000MB verbraucht:

```
dd if=/dev/zero bs=1M count=2000 | perl -e '$slurp=<>; sleep'
```

oder

```
dd if=/dev/zero bs=1M count=2000 | read slurp
```

[1)]
oft sieht man auch die BSD-Syntax:

```
ps ax
```

[2)]
Beispiele für Kernel-Threads (die Liste ist weder aktuell noch vollständig):

- migration: Used to move processes between CPUs - for load balancing
- ksoftirqd: Kernel healper thread to handle softirqs that can't be handled immediately
- watchdog: Notices if the system appears to be hung
- events: Handles kernel workqueues that aren't handled by other threads

- khelper: used as a schedulable context for stuff that call_user_mode_helper wants to run; like /sbin/hotplug or modprobe
- kblockd: used for running low-level disk operations
- kacpid: deals with ACPI - such as for power saving functions
- khubd: used for configuring USB
- kseriod: used for handling serial io
- khungtaskd: used for detecting hung tasks
- pdflush: sends memory to disk - to free RAM
- kswapd: swaps processes to disk
- aio: handles asynchronous IO
- kpsmoused: handles mouse IO
- scsi_eh: SCSI event handler
- qla2xxx_0_dpc: qla device driver thread
- scsi_wq: scsi work queue handler
- fc_wq: fiber chanel work queue handler
- fc_dl: fiber chanel handler
- kstriped: helper thread for striped volumes
- ksnapd: helper thread for handling dm-snaps (disk snapshots)
- kjournald: helper thread for handling filesystem journals
- kauditd: kernel thread for handling audit events
- kmpathd: multipath daemon helper thread
- ib_*: infiniband threads
- kworker: handles delayed work from throughout the kernel where the work is not being handled by one of the other queues. The number of kworker threads vary over time based on the amount of work in the work queues.
- kondemand: Used by ondemand CPU frequency govenor

[3]

aus `man waitpid`: *A child that terminates, but has not been waited for becomes a „zombie“. The kernel maintains a minimal set of information about the zombie process (PID, termination status, resource usage information) in order to allow the parent to later perform a wait to obtain information about the child. As long as a zombie is not removed from the system via a wait, it will consume a slot in the kernel process table, and if this table fills, it will not be possible to create further processes. If a parent process terminates, then its „zombie“ children (if any) are adopted by init(1), (or by the nearest „subreaper“ process ...; init(1) automatically performs a wait to remove the zombies.*

[4]

```
ps aux
```

[5]

das funktioniert nur bei den meisten Serverdiensten