7 Prozesse

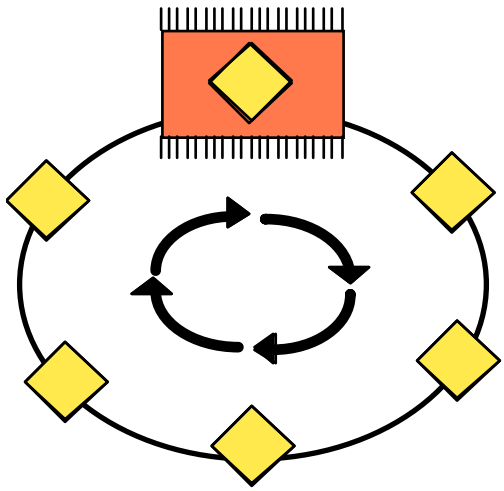
Das Betriebssystem UNIX ist fähig, mehrere Aufgaben scheinbar gleichzeitig zu erledigen. Dies wird mit Multitasking bezeichnet.

Auf einem UNIX-Rechner können hundert oder mehr Personen zur selben Zeit arbeiten und jede Person hat den Eindruck, (fast) alleine am Rechner zu sein. Dabei ist die Maschine vielleicht auch noch ein Webserver und beantwortet mehrere Webanfragen in jeder Sekunde.

Um die verschiedenen Aufgaben "gleichzeitig" und voneinander getrennt auszuführen, bedient sich ein Betriebssystem sogenannter Prozesse. Ein UNIX-Prozess ist eine Umgebung, in der ein Programm abläuft.

Wenn ein Benutzer etwa die Files in seinem Directory auflisten will, startet er das `ls`-Programm. Dieses Programm besteht aus vielen einzelnen Anweisungen, die die CPU eine nach der anderen ausführt. So wird das Directory im Filesystem gesucht und gelesen, die Eigenschaften der Files darin werden ermittelt und die Informationen werden formatiert und zum Bildschirm geschickt. Der `ls`-Prozess kann während der Arbeit etliche Male unterbrochen werden: er muss die CPU an andere Prozesse abtreten und sich (falls er noch nicht fertig ist) in eine Warteschlange (Queue) einreihen.

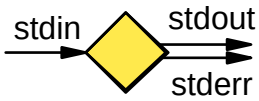
Das Prozessmanagement ist eine der Hauptaufgaben des Kernels. Er teilt die CPU den wartenden Prozessen zu und stellt sicher, dass keiner sie länger als eine bestimmte Zeit beansprucht. Dieses System wird *preemptive multitasking* (verdrängend) genannt. Ein Wechsel der CPU von einem Prozess zu einem anderen ist ein *context switch*.



```
kali> vmstat 5
Virtual Memory Statistics: (pagesize = 8192)
  procs    memory              intr        cpu
  r  w  u  act  free wire      in  sy  cs  us  sy  id
  2176 25   41K  12K 8766     62  4K 164  33   3  64
  2176 25   41K  12K 8511     58  2K 182   4   5  90
  3176 25   40K  13K 8570     73 601 192   4   3  93
  2178 25   40K  13K 8572     64 995 184   1   4  95
```

Damit der Prozess ein Programm ausführen kann und er umgekehrt auch vom UNIX-System gesteuert werden kann, hat er verschiedene Eigenschaften:

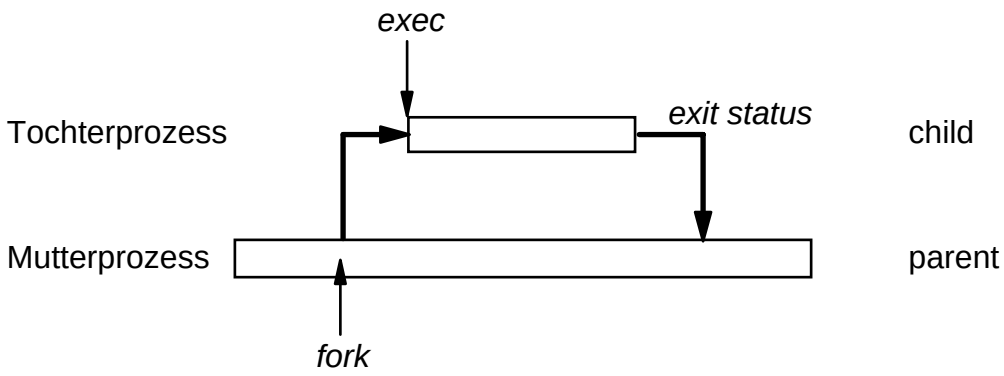
Name		**Beschreibung**
Prozessnummer	`pid`	Eine durchlaufende Nummer, unter welcher der Prozess dem Kernel bekannt ist.
Programm	`code`	Instruktionen, die von der CPU ausgeführt werden
Variable	`data`	Daten, mit denen gearbeitet wird
Stapel	`stack`	Datenstrukturen zur internen Steuerung
Umgebung	`environment`	Terminal-Typ, Home-Directory, etc.
Besitzer	`owner`, `group`	Wer hat den Prozess gestartet und welche Permissions werden angewendet (*real* und *effective uid* / *gid*)
Filedeskriptoren	`stdin`, `stdout` und `stderr`	Eingabe, Ausgabe und Fehler. Über diese Kanäle (Files) kommuniziert ein Prozess mit der Umwelt.



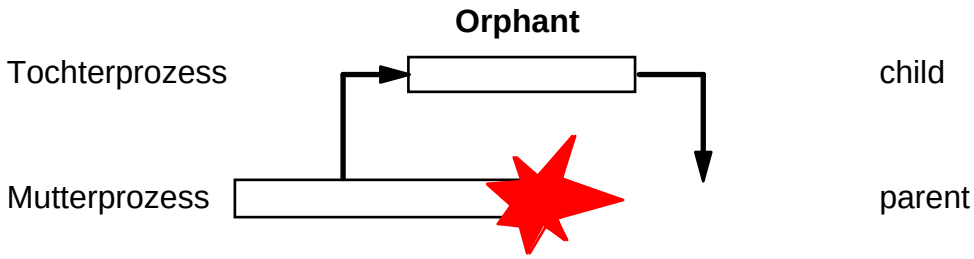
In modernen UNIX-Systemen werden die Prozesse noch in feinere Einheiten aufgespalten, die sog. *threads*. Mit threads wird das Programm in Bereiche unterteilt, die sich die prozess-privaten Daten teilen aber unabhängig voneinander ablaufen können.

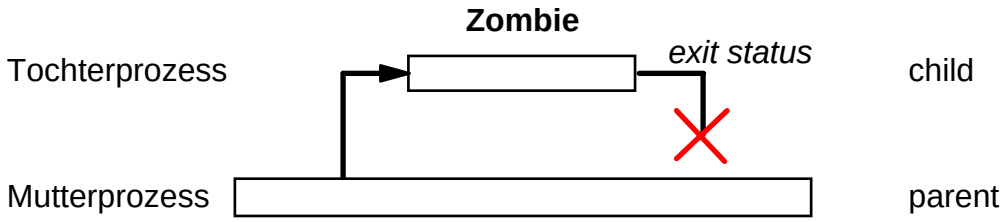
7.1 Prozessbeziehungen

Ein Prozess wird von einem anderen Prozess gestartet und hört wieder auf zu existieren, wenn seine Aufgabe beendet ist oder wenn er von aussen gestoppt wird. Die Beziehungen der Prozesse untereinander werden oft mit Familienverwandschaften verglichen:



Während der Tochterprozess aktiv ist, kann der Mutterprozess warten oder andere Arbeiten verrichten. Er muss aber von der Beendigung der Tochter Kenntnis nehmen.





7.1.1 Dämonen

Ein Dämon (*daemon*) ist in UNIX ein Prozess, der nicht einer Login-Session eines Benutzers zugeordnet ist und meist während der ganzen Lebensdauer des Systems aktiv ist. Dämonen werden beim Einschalten des Systems gestartet. Einige bekannte Dämonen sind:

Name	Funktion
`inetd`	Internet daemon: Empfängt einkommende Netzverbindungen und startet entsprechende Dienste (telnet, ftp, etc.)
`sendmail`	Empfängt und versendet Mail
`cron`	lässt bestimmte Aufgaben periodisch ablaufen
`httpd`	Webserver

Ein Dämon ist im allgemeinen ein **Server**-Prozess: er nimmt rechnerweite bzw. netzwerkweite (Internet) Aufgaben wahr.

7.2 Prozesse auflisten

Eine Liste der Prozesse liefert das `ps`-*Kommando (process status*):

```
kali> ps
   PID TTY      S           TIME CMD
 12657 ttyp3    IW +     0:00.17 -tcsh (tcsh)
 16150 ttyp7    S        0:00.17 -tcsh (tcsh)
 22882 ttypf    IW +     0:00.16 -tcsh (tcsh)
kali>
```

In dieser Form werden alle Prozesse angezeigt, die dem Benutzer gehören. Alle Prozesse auf der Maschine liefern die Optionen `-ef`:

```
kali> ps -ef
UID          PID   PPID    C STIME    TTY             TIME CMD
root           0      0  0.0   Jun 02 ??          18:37.94 [kernel idle]
root           1      0  0.0   Jun 02 ??           0:29.17 /sbin/init -a
root         135      1  0.0   Jun 02 ??           0:01.98 /usr/sbin/syslogd
root         137      1  0.0   Jun 02 ??           0:00.01 /usr/sbin/binlogd
root         327      1  0.0   Jun 02 ??           0:00.07 /usr/sbin/mountd -i
root         329      1  0.0   Jun 02 ??           0:00.00 /usr/sbin/nfsd -t1
root         613      1  0.0   Jun 02 ??          17:47.06 /usr/sbin/advfsd
root         622      1  0.0   Jun 02 ??           0:15.50 /usr/sbin/inetd
root         669      1  0.0   Jun 02 ??           0:05.18 /usr/sbin/cron
root         740      1  0.0   Jun 02 ??           6:52.01 /usr/dt/bin/dtlogin
root        6506    622  0.0 11:37:54 ??           0:01.07 telnetd
root        7387    622  0.0 12:20:54 ??           0:00.18 telnetd
daemon      9957  14545  0.0 14:23:42 ??           1:26.46 lpd (Worker - Print
pmdf       28176      1  0.0   Jun 12 ??           0:03.31 /pmdf/bin/job_contr
pmdf       29455      1  0.0   Jun 12 ??           0:05.41 /pmdf/bin/dispatche
root         762      1  0.0   Jun 02 console      0:00.03 /usr/sbin/getty
petr        5606   6506  0.0 11:37:54 ttyp1        0:00.05 -csh (csh)
root        6079   5606  0.0 11:38:03 ttyp1        0:00.03 -ksh (ksh)
kurs00      2123   7387  0.0 12:20:54 ttyp3        0:00.34 -csh (csh)
root       26134   2123  0.0 18:23:52 ttyp3        0:00.03 ps -ef
kali>
```

Kolonne	Bedeutung
UID	Besitzer des Prozesses
PID	eindeutige Prozessnummer (*process id*)
PPID	*pid* des Mutterprozesses (*parent pid*)
STIME	Startzeit des Prozesses
TTY	Terminal (?? wenn nicht bestimmt)
TIME	verbrauchte CPU-Zeit (`[[days-]hour:]min:sec.hh`)
CMD	Kommando

7.3 Signale

Signale (*software interrupts*) werden benutzt, um Prozesse zu beeinflussen. Ein Signal kann einen Prozess beenden, ihn stoppen und wieder starten oder ihn vom Ableben seines Tochterprozesses benachrichtigen. Der Prozess kann ein Signal abfangen, ignorieren oder am Signal sterben.

Signale werden mit dem `kill`-Kommando gesendet (ein unglücklicher Name, da der Prozess nicht unbedingt beendet wird)

```
kill -signal pid
```

Einige Signale sind

Signal	Beschreibung
`HUP`	Hangup, oft auch Konfig nochmals lesen
`INT`	Interrupt
`QUIT`	Quit mit memory dump
`KILL`	Abbruch, kann nicht abgefangen werden
`ALRM`	Timer
`TERM`	Abbruch, Prozess beenden
`STOP`	temporärer Unterbruch
`CONT`	Fortsetzung nach Unterbruch
`CLD`	Tochterprozess beendet

Die Namen der Signale werden oft mit SIG vorangestellt, z.B. SIGHUP oder SIGTERM. Das kill-Kommando verlangt jedoch die Namen ohne SIG:

```
kali> kill -STOP
```

7.4 Prozessprioritäten

Die Priorität eines Prozesses bestimmt wie schnell er verarbeitet wird. Lange Berechnungen, die kein Eingreifen des Benutzers erfordern, müssen mit niedriger Priorität laufen. Sie lassen damit Vortritt z.B. interaktiven Login-Sessions.

Eine niedrige Priorität bedeutet nicht, dass der Rechner langsamer arbeitet, sondern dass der Prozess weniger oft die CPU zugeteilt bekommt.

In UNIX wird die Priorität durch den `nice`-Wert ausgedrückt. `nice` gibt an, wie "nett" ein Prozess zu den anderen ist und ihnen den Vortritt lässt. `nice` kann Werte zwischen `-19` und `+19` annehmen, wobei hohe Werte von `nice` kleinen Prioritäten entsprechen.

nice	Bedeutung	Priorität
+19	sehr nett	kleinste Priorität
0	normal	default-Priorität
-19	gar nicht nett	höchste Priorität

```
kali> ps -e -o user,pid,nice,pri,comm
USER        PID  NI PRI COMMAND
root          0  -6  38 kernel idle
root          1   0  44 init
root        425 -12  32 xntpd
root        485  -1  43 pmgrd
root        525  -1  43 advfsd
root        534   0  44 inetd
kurs00    27231   0  44 csh
zimpet00  27112   0  44 csh
kurs00    27401   0  44 pine
kali>
```

Der `nice`-Wert eines Kommandos wird festgelegt mit

```
kali> nice -n 12 prim
```

Prozesse mit hohen `nice`-Werten werden gewöhnlich als Batch oder im Hintergrund ausgeführt.

Nur der `superuser` darf den `nice`-Wert erniedrigen, also eine höhere Prozesspriorität einstellen. Mit `renice` kann auch nachträglich die Prozesspriorität geändert werden:

```
kali> renice -n 4 2765
```