1.5 Anwendung/Prozesse

1.5.1 Prozesse und Multitasking

Ein Prozess (englisch *task*) ist ein im Lauf befindliches Programm. Auf einem Linux-System erhält man mit den Befehlen `ps` (*process list*), `top` (*table of processes*) und `pstree` Informationen über die aktuellen Prozesse.

Multitasking bedeutet, dass mehrere Prozesse quasi gleichzeitig laufen können, obwohl nur ein Prozessor (oder wenige) für alle Prozesse zur Verfügung steht.

Damit das möglich ist, muss das Betriebssystem eine Prozessumschaltung besitzen, die jedem Prozess einen angemessenen Zeitraum zur Verfügung stellt, so dass er arbeiten kann, die anderen Prozesse jedoch auch arbeiten können. Dies ist also der Teil des Betriebssystems, der den Prozessen jeweils einen Teil der CPU-Zeit und einen Teil des RAM zuteilt; er heißt *scheduler*, seine Aufgabe ist das *scheduling* (von englisch *schedule* = Verabredung).

Bei den Betriebssystemen MS-Windows ME und 98 muss jedes Programm von sich aus regelmäßig den Scheduler aufrufen. Unterbleibt das, so werden alle anderen Prozesse blockiert. Diese Form des Multitasking heißt kooperatives Multitasking.

Bei Linux wird der Scheduler regelmäßig durch den Timer-Interrupt aufgerufen: Alle paar Millisekunden unterbricht der Timer-Baustein (ein Stück Peripherie-Hardware, heutzutage im Chipsatz integriert) den Befehlsablauf der CPU und damit den gerade laufenden Prozess (Abbildung 1). Die CPU sichert ab diesem Moment alle Register und führt anschließend ein Programmstück aus,

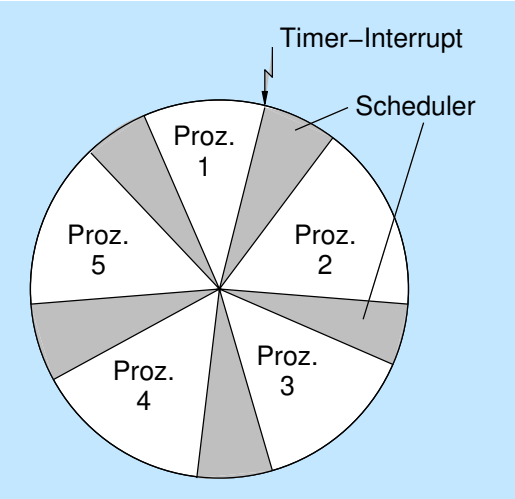

Abbildung 1: Beispiel für Multitaskting

das zum Betriebssystem, nämlich zum Scheduler gehört. Der Scheduler entscheidet nun, welcher Prozess als nächster an der Reihe ist, restauriert seine vorher gesicherten Register und führt diesen Prozess dann weiter. Wiederum nach ein paar Millisekunden wird wieder in den Scheduler gesprungen, der sucht wieder den nächsten Prozess aus und macht mit ihm weiter.

Auf diese Art erhält nacheinander jeder Prozess ein Stück Zeit (eine Zeitscheibe bzw. *time slice*). Diese Art des Multitasking heißt präemptives Multitasking, sie ist im Vergleich zuverlässiger.

Bei Linux-Systemen werden die Zeitscheiben (je nach Priorität) annähernd gleichmäßig nacheinander vergeben, etwa in der Form 1-2-3-4-1-2-3-4-1-2-3-4-.... Diese Art der Zeitvergabe heißt *round-robin*-Verfahren (benannt m.W. nach einem amerikanischen Kinderspiel ähnlich dem Plumpsack).

Für Echtzeit-Betriebssysteme (QNX, RTLinux u.a.) werden aufwändigere Scheduling-Verfahren benutzt, mit denen man eine garantierte Antwortzeit erhält.

1.5.2 Liste der Prozesse

Mit dem Befehl `ps` kann man sich eine Liste der aktuell auf dem System laufenden Prozesse ansehen:

Terminal
```
schueler@debian964:~$ ps alx
F  UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4    0     1     0  20   0  23356  2504 -      Ss   ?          0:02 /sbin/init
1    0     2     0  20   0      0     0 -      S    ?          0:00 /[kthreadd]
...
```

In jeder Zeile ist ein Prozess beschrieben. Zuerst ist da die Nummer des Prozesses, PID genannt. Zu jedem Prozess gehört eine User-ID (Feld ID), eine virtuelle und eine reelle RAM-Größe in kiB (Felder VSZ und RSS), einen Prozessstatus (Feld STAT[1]), eine bisherige Laufzeit (TIME) und viele weitere Informationen, die man sich mit den richtigen Optionen von `ps` anzeigen lassen kann. Tabelle 1 zeigt einen kleinen Ausschnitt dieser Optionen. Leider versucht die Linux-Version von `ps`, drei verschiedene Versionen von `ps` in sich zu vereinen. Hier sind der Übersichtlichkeit halber nur die Optionen der BSD-Version aufgeführt (erkennbar am fehlenden Minuszeichen). In

Option	heißt	Bedeutung
keine	–	nur eigene Prozesse am eigenen Terminal
a	all	auch fremde Prozesse anzeigen
x	–	auch Prozesse ohne Terminal-Verbindung
r	running	nur laufende Prozesse anzeigen
l	long	Langformat
o	output	Ausgabeformat mit angegebenen Feldern
e	env	zusätzlich Umgebungsvariablen anzeigen
f	forest	zusätzlich Prozessbaum anzeigen
w	wide	zusätzlich breite Ausgabe (ww: unbegrenzt)

Tabelle 1: Optionen für `ps` (BSD-Version)

der Option `f` in der Tabelle ist von einem Prozessbaum die Rede. Was hat es damit auf sich? Bei Linux hängen alle Prozesse in einer Baumstruktur zusammen. Mit dem Befehl `pstree` kann man sich diese Struktur etwas schöner anzeigen lassen. Der Init-Prozess mit der PID=1 ist die Wurzel (man kann auch sagen, der gemeinsame Vorfahre) aller Prozesse auf dem System. Er wird beim Systemstart quasi von Hand in die Prozesstabelle eingetragen. Alle anderen Prozesse entstehen aus dem Init-Prozess.

Dies geschieht zuerst durch Verdoppelung beim Systemaufruf `fork()`. Nach dieser Verdoppelung existieren zwei fast gleiche Prozesse. Der alte Prozess wird Elternprozess (oder Vaterprozess) genannt, den neuen nennt man Kindprozess (oder Sohnprozess). Mit zwei gleichen Prozessen kann man aber nichts anfangen. Deshalb lädt der Kindprozess sofort ein neues Programm an seine eigene Stelle. Er benutzt dazu den Systemaufruf `exec()`.

Durch diesen einen einfachen Mechanismus können innerhalb kürzester Zeit Tausende von Prozessen erzeugt werden. Welche Prozesse nun vom Init-Prozess gestartet werden sollen, kann man in speziellen Konfigurationsdateien detailliert festlegen.

Den fork-exec-Mechanismus kann man auf der Shell nicht sehen oder beobachten. Es gibt aber einen Shell-Befehl, der dem `exec()`-Aufruf nahekommt:

Terminal
```
schueler@debian964:~$ ps
  PID TTY          TIME CMD
 5706 pts/1    00:00:00 bash
```

[1] R=running, S=sleeping, T=stopped, Z=zombie, N=nice, <=not nice, s=session leader, +=foreground process group

```
 5711 pts/1    00:00:00 ps
schueler@debian964:~$ exec top
```

Beim Aufruf von `top` erhält man wie bei ps eine Prozessliste, allerdings als ständig aktualisiertes Fenster.

Terminal

```
top - 20:57:37 up 10:56,  4 users,  load average: 1,01, 1,04, 1,02
Tasks: 195 total, 1 running, 194 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,2us, 0,3sy, 0,0ni, 99,3id, 0,2wa, 0,0hi, 0,0si, 0,0st
KiB Mem:  2071956 total, 996036 used, 1055920 free,  84444 buffers
KiB Swap: 2783228 total,  92208 used, 2691020 free. 592984 cached Mem

  PID USER PR NI  VIRT   RES   SHR S %CPU %MEM  TIME+ COMMAND
 1051 root 20  0 98972 18432  6780 S  1,3  0,9  17:47.15 Xorg
 5706 sch  20  0  5132  2652  2240 R  0,3  0,1   0:00.23 top
 ...
```

Die erste Überraschung: Unter der Prozessnummer 5706 findet man nun den `top`-Befehl selbst; die Shell ist weg. Beendet man das Programm mit der Taste Q, dann wird man ausgeloggt (oder es schließt sich das Fenster, je nachdem, wie man auf dem System gearbeitet hat), weil die Shell weg ist.

1.5.3 Signale (Kommunikation)

Mit Hilfe von Signalen kann man mit Prozessen kommunizieren. Ein Signal kann von verschiedenen Quellen kommen:

a) vom Benutzer, der z. B. Strg - C drückt

b) von der Hardware, die z. B. eine Division durch null erkannt hat

c) vom Prozess selbst, der sich z. B. beenden möchte

d) von einem anderen Prozess, der eine Nachricht schicken möchte

Hier ist vor allem die letzte Variante interessant: Der Benutzer mit seinem Shell-Prozess möchte einem anderen laufenden Prozess etwas mitteilen. Dies geschieht mit dem Befehl `kill`:

Terminal

```
schueler@debian964:~$ sleep 20 &
[1] 5909
schueler@debian964:~$ kill 5909
[1]-  Beendet                 sleep 30
```

In der ersten Zeile wird ein Programm im Hintergrund gestartet. Dieses Programm tut nichts außer 20 Sekunden zu warten und sich dann selbst zu beenden. Die folgende Zeile zeigt die PID. In der zweiten Zeile wird an den Prozess ein Signal geschickt. Der Prozess beendet sich daraufhin. Der Benutzer root darf jedem Prozess ein Signal schicken; andere Benutzer dürfen nur eigene Prozesse ansprechen.

Es gibt 64 Signale, die mit Nummer und Namen bezeichnet werden können. Standardmäßig schickt `kill` das Signal Nummer 15 mit dem Namen SIGTERM; damit kann ein Programm aufgefordert werden, sich zu beenden. Ein Programm kann jedoch selbst festlegen, was es bei Eintreffen eines Signals macht. Es gibt ein Standardverhalten, das kann aber überschrieben werden. Ausnahmen sind die Signale Nummer 9 (SIGKILL) und Nummer 19 (SIGSTOP), die nicht abgefangen werden können. Häufig benutzte Signale sind in Tabelle 2 aufgeführt. Mit dem `kill`-Befehl kann man jedes der 64 Signale an einen Prozess schicken, und zwar auf mehrere verschiedene Arten:

Nr.	Name	Std.-Aktion	Std.-Meldung	Bedeutung
1	HUP	Beenden	Hangup	Verbindung (z. B. SSH) beendet
2	INT	Beenden	Unterbrechung	Strg - C von der Tastatur
3	QUIT	Abbruch	Verlassen	Abbruchsignal von der Tastatur
4	ILL	Abbruch	Ungültiger Maschinenbef.	Falscher CPU-Befehl
6	ABRT	Abbruch	Abgebrochen	Programmabbr. vom Prog. selbst
8	FPE	Abbruch	Gleitkomma-Ausnahme	Gleitkomma-Fehler
9	KILL	Beenden	Getötet	Sofortiger Abbruch
10	USR1	Beenden	Benutzerdef. Signal 1	—
11	SEGV	Abbruch	Speicherzugriffsfehler	Schutzverletzung
12	USR2	Beenden	Benutzerdef. Signal 2	—
13	PIPE	Beenden	Datenüberg. unterbr.	FIFO-Fehler
14	ALRM	Beenden	Der Wecker klingelt	Zeitsignal erhalten
15	TERM	Beenden	Beendet	Prozess beenden
17	CHLD	Ignorieren	—	Kindprozess beendet oder gestoppt
18	CONT	Weiterlaufen	—	Gestoppter Prozess läuft weiter
19	STOP	Anhalten	Angehalten	Laufender Prozess wird gestoppt
20	TSTP	Anhalten	Angehalten	Laufender Proz. d. Strg - Z gest.

Tabelle 2: Wichtige Signale

Terminal
```
schueler@debian964:~$ kill -9 12345   # oder -KILL 12345 oder -SIGKILL
schueler@debian964:~$ kill -s 9 12345 # oder -s KILL 12345
schueler@debian964:~$ kill -n 9 12345 # oder -n KILL 12345
```

Will man auf einer Bash-Shell ein Signal abfangen, kann man das mit dem `trap`-Befehl machen:

Terminal
```
schueler@debian964:~$ trap "echo Hallo" 10
schueler@debian964:~$ kill -s 10 $$  # $$ ist d.PID d.aktuellen Shell
Hallo
```

1.5.4 Startmöglichkeiten von Prozessen

Systemintern werden Prozesse durch `fork()` und `exec()` gestartet. Für Benutzer gibt es andere Möglichkeiten, neue Prozesse zu erzeugen:

- `shell`: Prozessstart durch Eingabe des Namens der ausführbaren Datei
- `atd`: Prozessstart zu beliebiger Zeit durch den AT-Dienst
- `crond`: Regelmäßiger Start durch den CRON-Dienst
- `inittab`: Dauerhafter Lauf eines Programmes unter Kontrolle des Init-Prozesses

Bei der Shell wiederum gibt es viele Möglichkeiten, den Prozess mit Sonderwünschen aufzurufen. Hier sind nur einige der Möglichkeiten aufgeführt:

a) `time programmname`: Die Zeit, die der Prozess läuft, wird gemessen

b) `nohup programmname`: Der Prozess erhält kein HUP-Signal, wenn der Benutzer sich ausloggt und kann damit weiterlaufen (ähnlich: `disown` und `screen`)

c) `timeout programmname`: Der Prozess wird spätestens nach einer bestimmten Zeit mit einem Signal abgebrochen

d) `watch programmname`: Der Prozess wird alle zwei Sekunden neu gestartet mit leerem Bildschirm; man erhält regelmäßig eine aktuelle Ausgabe

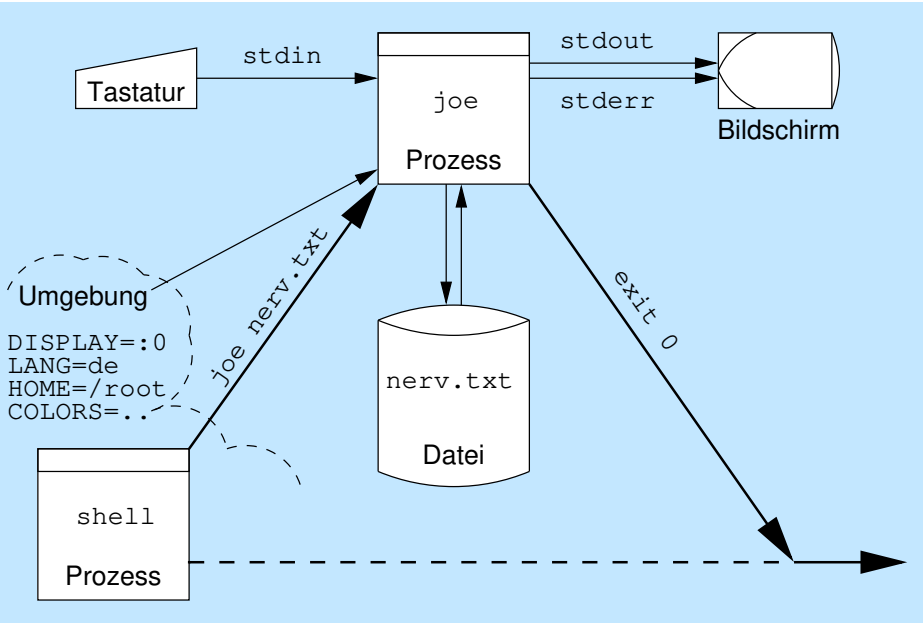

Abbildung 2: Schnittstellen eines Prozesses

1.5.5 Schnittstellen eines Prozesses

Wie in Abbildung 2 gezeigt, kann ein Prozess mit seiner Umwelt über verschiedene Möglichkeiten kommunizieren[2]:

- Der Aufruf selbst mit den Befehlszeilen-Argumenten, der Umgebung und dem Exitstatus des Programms bildet die erste Ebene (Aufruf-Ebene).
- Eine zweite Ebene ist die Benutzer-Interaktion mit Tastatur-Eingaben und Bildschirm-Ausgaben (Dialog-Ebene).
- Eine dritte Ebene ist der Zugriff des Programms auf Dateien, Geräte und Netzwerk-Resourcen (Ebene des Hintergrundverkehrs).

1.5.6 Vordergrund, Hintergrund und Job Control

Ein Prozess kann auf der Shell im Vordergrund oder im Hintergrund gestartet werden. Der Normalfall ist der Start im Vordergrund. Wenn man die Befehlszeile jedoch mit einem `'&'`-Zeichen beendet, startet der Prozess im Hintergrund.

Terminal

```
schueler@debian964:~$ xeyes &
[1] 6180
schueler@debian964:~$
```

Der Unterschied zum normalen Vordergrund-Prozess liegt nur darin, dass hier wieder sofort die Befehlszeile der Shell zur Verfügung steht. Man kann also sofort wieder auf der Tastatur den nächsten Befehl eingeben. Mit der Position des Fensters auf der GUI haben die Begriffe Vordergrund und Hintergrund hier nichts zu tun (Abbildung 3).

Angezeigt werden die Prozess-ID 6180 und ein Job-Handle für diesen Hintergrundprozess, eine fortlaufende Nummer, beginnend mit eins. Mit dem `kill`-Befehl kann der Prozess wieder beendet

[2]Reinhard Fößmeier: Die Schnittstellen von UNIX-Programmen. Tips zur Programm-Organisation unter UNIX. Reihe Informationstechnik und Datenverarbeitung. Springer, Heidelberg 1991.

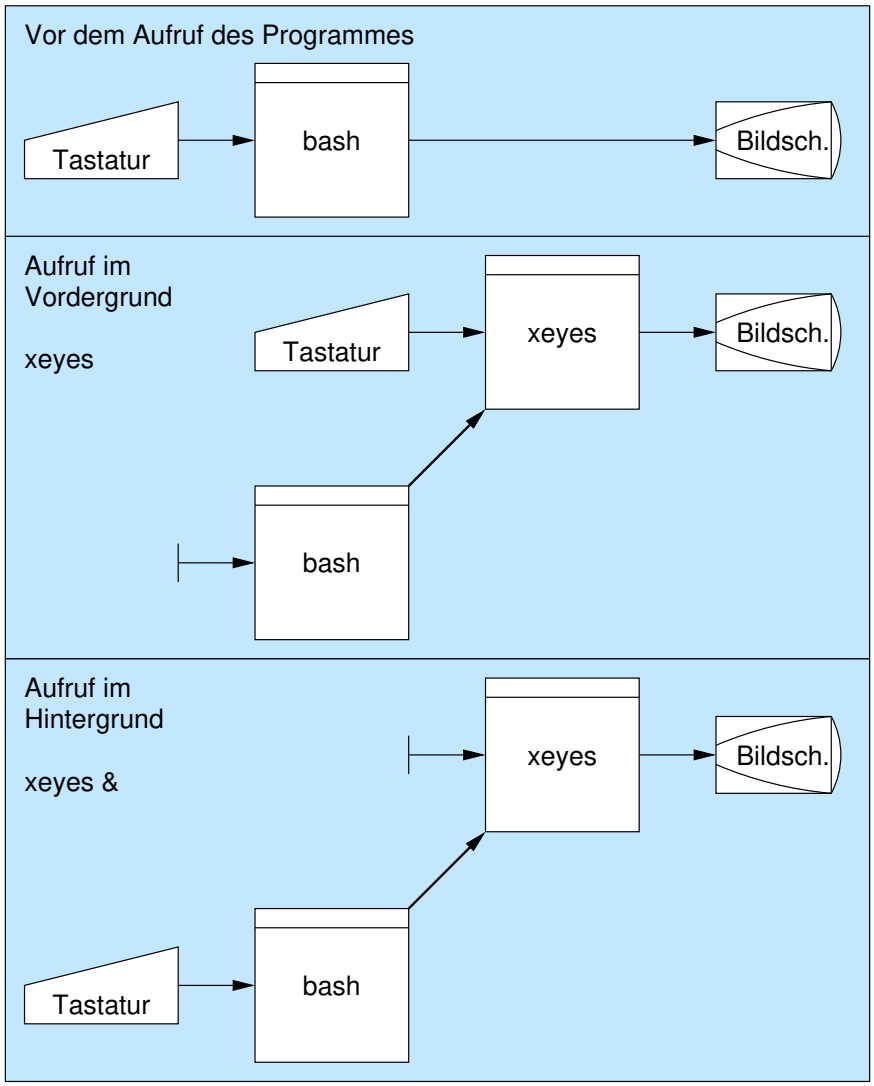

Abbildung 3: Vorder- und Hintergrundprozess und Dialogebene

werden. Dabei kann man entweder die Prozess-ID angeben oder ein Prozentzeichen gefolgt vom Job-Handle:

Terminal
```
schueler@debian964:~$ kill %1
[1]+  Beendet                 xeyes
schueler@debian964:~$
```

Ein Problem kann es sein, wenn man einen Prozess aus Versehen im Vordergrund statt im Hintergrund gestartet hat. Man kann ihn zwar mit Strg - C abbrechen, aber nicht bei jedem Prozess empfiehlt sich das. Dann muss man zweistufig vorgehen: Mit der Tastenkombination Strg - Z schickt man das Programm in den Zustand „angehalten“. Anschließend lässt man ihn mit dem `bg`-Befehl im Hintergrund weiterlaufen. Abbildung 4 zeigt alle vorhandenen Möglichkeiten. Sie werden oft unter dem Namen *job control* zusammengefasst. Ebenso kann man einen Prozess, den

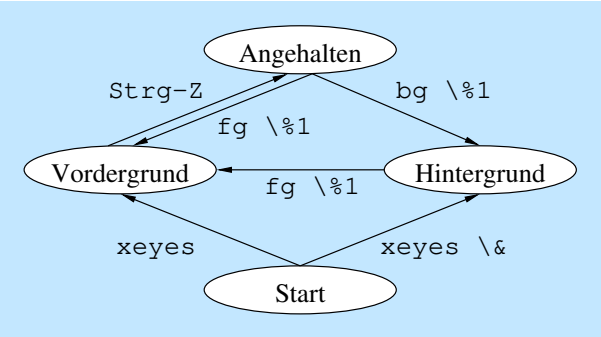

Abbildung 4: Job Control

man aus Versehen im Hintergrund gestartet hat und der eine Eingabe erwartet, mit Hilfe des `fg`-Befehls im Vordergrund weiterlaufen lassen. Mit dem Befehl `jobs` kann man sich alle aktuell laufenden Hintergrundprozesse auflisten lassen.

1.5.7 Prioritäten setzen und verändern

1.5.7.1 PR-Wert Unter Linux hat jeder Prozess eine bestimmte Priorität. Diese Priorität wird mit Hilfe des Zahlenwertes PR ausgedrückt. Bei Benutzerprozessen liegt der Wert oft bei PR=20. Bei Systemprozessen kann der Wert anders sein.

Man kann die Priorität ermitteln mit `top`. Dort sind in in der Spalte PR die entsprechenden Werte angegeben. Oder man nimmt ps zur Anzeige:

Terminal
```
schueler@debian964:~$ ps -axl|less
F UID   PID PPID PRI  NI    VSZ   RSS WCHAN STAT TTY   TIME COMMAND
5   0    10    2 -100  -      0     0 -     S    ?     0:00 [watchdog/0]
1   0    21    2  25   5      0     0 -     SN   ?     0:00 [ksmd]
4   0 1051 1045  20   0 112840 31408 -     Ssl+ tty7  3:57 /usr/bin/X
```

PR=0 bedeutet höchste Priorität, ein solcher Prozess wird vorrangig abgearbeitet und bekommt am häufigsten CPU-Zeit. PR=39 ist dagegen die niedrigste Priorität.

Außerdem kann es noch besonders vorrangige Prozesse geben, die nach Echtzeitverfahren abgearbeitet werden Sie bekommen einen negativen Prioritätswert zwischen -100 und -1[3] [4] [5].

$$-100 \leq PR \leq 39$$

[3] Bei `top` steht bei einem solchen Prozess in der Spalte PR statt der negativen Zahl das Wort RT.

[4] Um keine negativen PR-Werte anzeigen zu müssen, addieren manche Programme in der Ausgabe einfach den Wert 100 hinzu (*mapping*).

[5] Beim `ps`-Befehl mit der Option `-eo comm,pid,pri,ni` wird in der Spalte PR der Wert $-PR+39$ ausgegeben. Die Zahlenwerte liegen dadurch alle über 0 (zwischen 0 und 139).

1.5.7.2 NI-Wert Nun gibt es die Möglichkeit, für einen Prozess die Priorität zu ändern. Das passiert mit Hilfe des NICE-Wertes (abgekürzt NI). Ein Benutzerprozess mit NI=5 bekommt die Priorität PR=20+5=25.

$$PR = 20 + NI$$

Je höher der NICE-Wert ist, desto höher ist dadurch auch der PR-Wert. Der Prozess bekommt also weniger CPU-Zeit und ist damit netter zu den anderen Prozessen.

1.5.7.3 `nice` Beim Start eines Prozesses kann man den NI-Wert dieses Prozesses setzen. Dazu dient das Programm `nice`:

Terminal

```
schueler@debian964:~$ nice xeyes &
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4681 2785  30 10 6612 3804 poll_s SN   pts/1 0:00 xeyes
```

Standardmäßig wird der NI-Wert auf NI=10 gesetzt. Damit ist PR=30. Mit der Option `-n` kann man andere Werte einstellen:

Terminal

```
schueler@debian964:~$ nice -n 4 xeyes &
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4686 2785  24  4 6612 3848 poll_s SN   pts/1 0:00 xeyes
```

Bei positiven NI-Werten ist das Programm netter. Was ist, wenn man für sein Programm mehr CPU-Zeit haben möchte? Das müsste mit einem negativen Wert für NI gehen:

Terminal

```
schueler@debian964:~$ nice -n -4 xeyes &
nice: die Priorität kann nicht gesetzt werden: Keine Berechtigung
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4690 2785  20  0 6612 3732 poll_s S    pts/1 0:00 xeyes
```

Man braucht also bei `nice` für einen *negativen* NI-Wert Administrator-Rechte:

Terminal

```
schueler@debian964:~$ su
Passwort:
root@debian964:~# nice -n -4 xeyes &
root@debian964:~# exit
schueler@debian964:~$ ps -axl|grep xeyes
F  UID  PID PPID PRI  NI  VSZ  RSS WCHAN STAT TTY   TIME COMMAND
4    0 4707    1  16  -4 6612 3820 -     S<   pts/1 0:00 xeyes
```

1.5.7.4 `renice` Wie kann man nun nachträglich bei einem bereits laufenden Programm den NI-Wert ändern? Dazu gibt es das Programm `renice`. Hier gibt man bei der Option `-n` den neuen NI-Wert an:

Terminal

```
schueler@debian964:~$ xeyes &
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4841 2785  20  0 6612 3840 poll_s S    pts/1 0:00 xeyes
schueler@debian964:~$ renice -n 3 4841
4841 (process ID) old priority 0, new priority 3
schueler@debian964:~$ ps -axl|less
```

```
F   UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0  1000 4841 2785  23  3 6612 3840 poll_s SN   pts/1 0:01 xeyes
schueler@debian964:~$ renice -n 7 4841
4841 (process ID) old priority 3, new priority 7
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4841 2785  27  7 6612 3840 poll_s SN   pts/1 0:01 xeyes
```

Bei `renice` braucht man immer dann Administrator-Rechte, wenn man den NI-Wert *herabsetzen* will:

Terminal

```
schueler@debian964:~$ renice -n 6 4841
renice: failed to set priority for 4841 (PID): Keine Berechtigung
schueler@debian964:~$ su
Passwort:
root@debian964:~# nice -n 6 xeyes &
4841 (process ID) old priority 7, new priority 6
root@debian964:~# exit
schueler@debian964:~$ ps -axl|less
F  UID  PID  PPID PRI NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4841  2785  26  6 6612 3840 poll_s SN   pts/1 0:08 xeyes
```

Genauso kann man als Administrator negative NI-Werte für einen Prozess festlegen:

Terminal

```
schueler@debian964:~$ su
Passwort:
root@debian964:~# nice -n -11 xeyes &
4841 (process ID) old priority 6, new priority -11
root@debian964:~# exit
schueler@debian964:~$ ps -axl|less
F  UID  PID PPID PRI  NI  VSZ  RSS WCHAN  STAT TTY   TIME COMMAND
0 1000 4841 2785   9 -11 6612 3840 poll_s S<   pts/1 0:10 xeyes
```

1.5.7.5 Vergleich zwischen `nice` und `renice`

Tabelle 3 zeigt eine vergleichende Übersicht über `nice` und `renice`. Wie man in den letzten beiden Zeilen sieht, ist die alte Aufrufart der

Programm	`nice`	`renice`
Einsatz	Prozess-Start	Laufender Prozess
NI-Wert auf 9 setzen	`nice -n 9 xeyes`	`renice -n 9 4841`
NI-Wert auf -9 setzen	`nice -n -9 xeyes`	`renice -n -9 4841`
Administrator-Rechte für ...	negativen NI-Wert	Verringern des NI-Werts
NI-Wert auf 9 setzen (alt)	`nice -9 xeyes`	`renice +9 4841`
NI-Wert auf -9 setzen (alt)	`nice --9 xeyes`	`renice -9 4841`

Tabelle 3: Vergleich `nice` und `renice`

beiden Programme (ohne die Option `-n`) unterschiedlich: Bei `nice` ist das Minuszeichen das Optionszeichen, während es bei `renice` als Vorzeichen für NI dient.

1.5.8 IO-Prioritäten setzen und verändern

Oft sind es nicht die CPU- oder RAM-Auslastung eines Prozesses, die das System verlangsamen, sondern die Ein-/Ausgabe-Aktivitäten (IO). Deshalb ist es sinnvoll, bei diesen Aktivitäten ebenfalls eine Priorität einzuführen. Das ist bei Linux so gelöst: Ein Prozess läuft in einer von drei IO-Prioritäts-Klassen:

1 realtime (Echtzeit; IO sofort)

2 best-effort (normale Klasse für Benutzerprozesse)

3 idle (IO findet nur dann statt, wenn das System frei ist)

Aus Kompatibilitätsgründen gibt es noch eine Klasse 0 (none), die aber gleichbedeutend mit Klasse 2 ist.

Innerhalb der Klasse 2 (best effort) gibt es nochmal eine IO-Prioritätsstufe mit einem Wert von 0 bis 7 (wobei 7 am nettesten ist), siehe Abbildung 5.

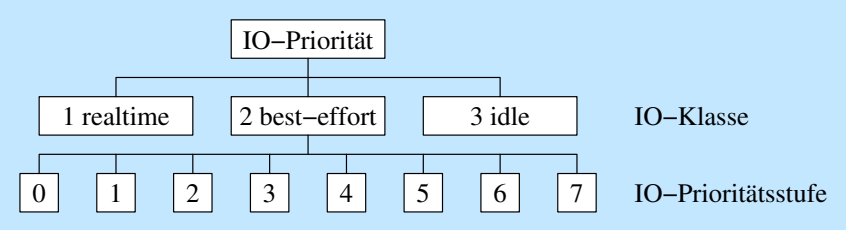

Abbildung 5: IO-Prioräten: Klassen und Stufen

Standardmäßig hängt die IO-Prioritätsstufe von der normalen Priorität ab:

$$IOPR = \frac{PR}{5} = \frac{NI + 20}{5} \tag{1}$$

Für das Anzeigen, Setzen und Verändern von Prioritäten gibt es den Befehl `ionice`:

a) Man kann die IO-Klasse und die IO-Prioritätsstufe eines laufenden Prozesses ausgeben (die Option -p dient zur Angabe der Prozess-ID):

Terminal
```
schueler@debian964:~$ ps a
...
12345 pts/4    S+     0:10 top
schueler@debian964:~$ ionice -p 12345
none: prio 4
```

b) Man kann die IO-Klasse und/oder die IO-Prioritätsstufe eines laufenden Prozesses verändern (wie bei `renice`).

Die IO-Klasse verändert man mit der Option `-c`:

Terminal
```
schueler@debian964:~$ ionice -c3 -p 12345
schueler@debian964:~$ ionice -p 12345
idle
```

Die Änderung zur Klasse 1 (Echtzeit) kann allerdings nur der Benutzer `root` vornehmen.

Mit der Option `-n` ändert man die IO-Prioritätsstufe eines Prozesses. Falls der Prozess noch nicht in der Klasse 2 ist, bewirkt diese Option, dass er in die Klasse 2 kommt:

Terminal
```
schueler@debian964:~$ ionice -n 2 -p 12345
schueler@debian964:~$ ionice -p 12345
best-effort: prio 2
```

c) Man kann einen Prozess mit einer bestimmten IO-Klasse oder mit einer bestimmten IO-Prioritätsstufe starten (wie bei `nice`):

Terminal
```
schueler@debian964:~$ ionice -n 7 firefox
```