Process Management

Introduction to processes
Process priorities
Background jobs

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Terminology

- **process**
  Programma of commando dat een bepaalde job uitvoert.

- **PID**
  Alle processes hebben een process id.
  Dit is een uniek nummer (tussen 0 en 65535).

- **PPID**
  Ieder process heeft een parent process, met een parent PID.
  Het child process wordt vaak gestart door het parent process.

Terminology

- **init** Het init process heeft als PID 1, wordt gestart door de kernel zelf en heeft geen parent process. Sinds upstart noemt dit proces ook systemd.

- **kill** Als een process stopt, sterft het process. Als je een process wil stoppen, moet je het “killen”.

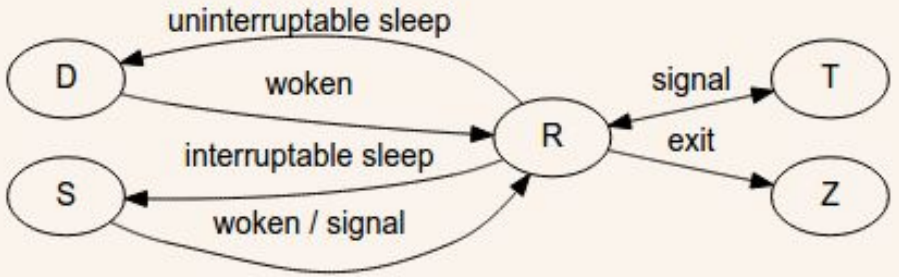
- **daemon** Een process dat start bij het opstarten van je systeem.

Terminology

- **zombie**
  Een “killed” process dat nog steeds zichtbaar is.

```
PROCESS STATE CODES
   R  running or runnable (on run queue)
   D  uninterruptible sleep (usually IO)
   S  interruptible sleep (waiting for an event to complete)
   Z  defunct/zombie, terminated but not reaped by its parent
   T  stopped, either by a job control signal or because
      it is being traced
```

A process starts its life in an R "running" state and finishes after its parent reaps it from the Z "zombie" state.



Basic Process Management

- **$$** Shell parameter: bevat het huidig process ID
- **$PPID** Shell variabele: bevat het parent ID

```
student@UbuntuDesktop:~$ echo $$ $PPID
2574 2564
```

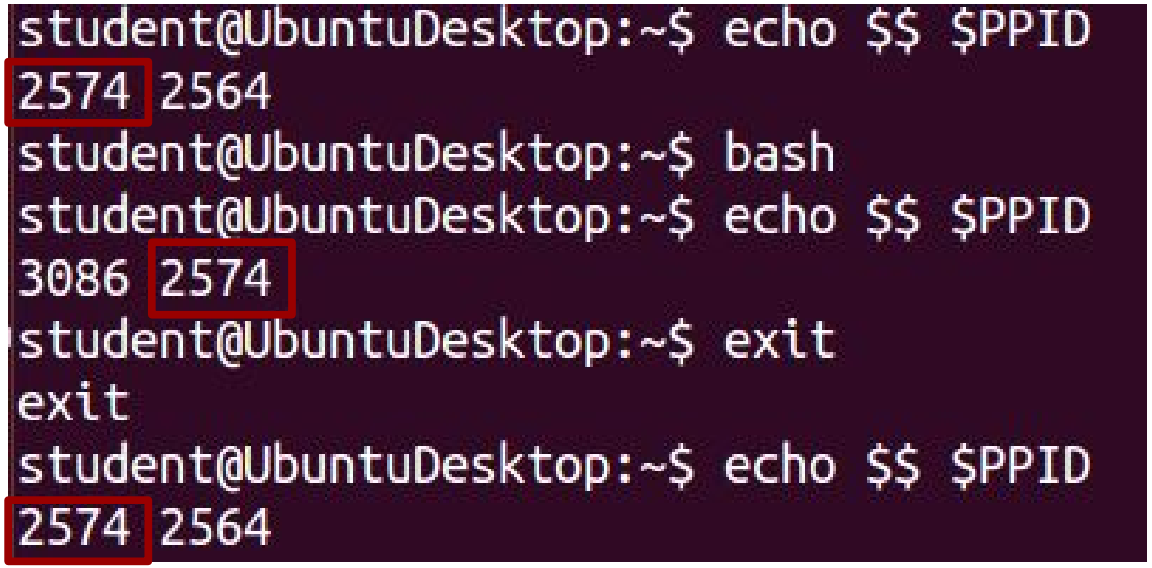
Basic Process Management

- **pidof** Vind alle process id's op naam.

```
student@UbuntuDesktop:~$ pidof bash
2574
student@UbuntuDesktop:~$ xclock &
[1] 3063
student@UbuntuDesktop:~$ xclock &
[2] 3064
student@UbuntuDesktop:~$ xclock &
[3] 3065
student@UbuntuDesktop:~$ pidof xclock
3065 3064 3063
```

Basic Process Management

- **parent and child**
  leder process (buiten init) heeft een parent process.



Basic Process Management

fork and exec

Een programma wil een nieuw proces starten dat gelijktijdig uitgevoerd wordt met het proces van het programma zelf.

→ een **fork**: een process maakt een kopie van zichzelf.

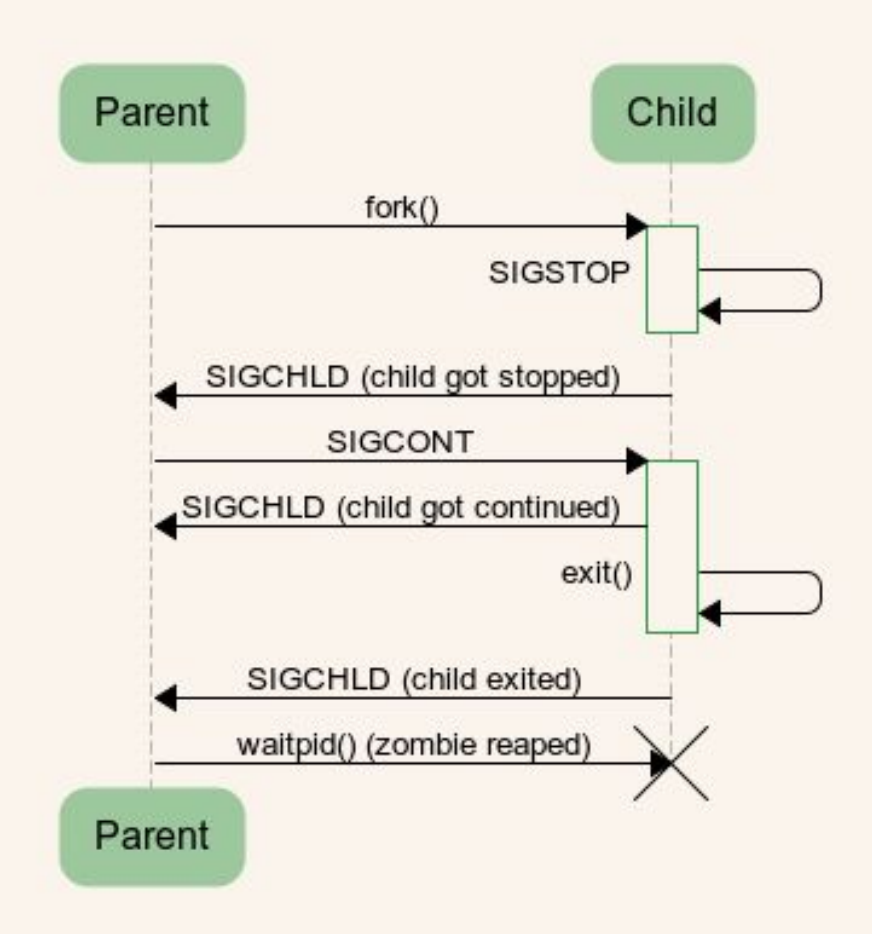
Dit is een nieuw proces dat een exacte kopie is van het proces dat opdracht gaf een fork uit te voeren, het enige verschil is de PID.

Het child process wordt meteen gestart en begint met de eerstvolgende instructie na de fork. Ook het parent process gaat verder met de uitvoering en met de eerstvolgende instructie na de fork. Er lopen nu dus twee vrijwel identieke processen die alleen een andere PID hebben.

exec vervangt het programma dat het huidige proces uitvoert. Er wordt dus geen nieuw proces gestart, maar het huidige proces start een ander programma.

Basic Process Management

- **fork and exec**



Basic Process Management

- **fork and exec**

fork →

exec →

```
student@UbuntuDesktop:~$ echo $$
3163
student@UbuntuDesktop:~$ sh
$ echo $$ $PPID
3223 3163
$ exec bash
student@UbuntuDesktop:~$ echo $$ $PPID
3223 3163
student@UbuntuDesktop:~$ exit
exit
student@UbuntuDesktop:~$ echo $$
3163
```

Basic Process Management

- **Werken met ps**

man ps

```
PS(1)                          User Commands                          PS(1)

NAME
       ps - report a snapshot of the current processes.
```

```
student@UbuntuDesktop:~$ ps ax | head -4
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:03 /sbin/init splash
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:03 [ksoftirqd/0]
```

```
student@UbuntuDesktop:~$ ps --pid 1 -o pid,cmd,comm
  PID CMD                         COMMAND
    1 /sbin/init splash           systemd
```

Basic Process Management

- **Werken met ps**

```
student@UbuntuDesktop:~$ echo $$ $PPID
3163 3154
student@UbuntuDesktop:~$ bash
student@UbuntuDesktop:~$ echo $$ $PPID
3288 3163
student@UbuntuDesktop:~$ bash
student@UbuntuDesktop:~$ echo $$ $PPID
3300 3288
student@UbuntuDesktop:~$ ps f
  PID TTY      STAT   TIME COMMAND
 3163 pts/11   Ss     0:00 bash
 3288 pts/11   S      0:00  \_ bash
 3300 pts/11   S      0:00      \_ bash
 3315 pts/11   R+     0:00          \_ ps f
student@UbuntuDesktop:~$ exit
exit
student@UbuntuDesktop:~$ ps f
  PID TTY      STAT   TIME COMMAND
 3163 pts/11   Ss     0:00 bash
 3288 pts/11   S      0:00  \_ bash
 3317 pts/11   R+     0:00      \_ ps f
student@UbuntuDesktop:~$ exit
exit
student@UbuntuDesktop:~$ ps f
  PID TTY      STAT   TIME COMMAND
 3163 pts/11   Ss     0:00 bash
 3318 pts/11   R+     0:00  \  ps f
```

Basic Process Management

pgrep

processen op naam zoeken

```
student@UbuntuDesktop:~$ sleep 1000 &
[1] 3320
student@UbuntuDesktop:~$ pgrep sleep
3320
student@UbuntuDesktop:~$ ps -C sleep
  PID TTY          TIME CMD
 3320 pts/11   00:00:00 sleep
```

het commando van een proces ook tonen

```
student@UbuntuDesktop:~$ sleep 2000 &
[2] 3324
student@UbuntuDesktop:~$ pgrep -l sleep
3320 sleep
3324 sleep
```

Basic Process Management

top

Ordent processen naargelang gebruik van CPU of andere properties.

```
top - 13:43:20 up  2:52,  2 users,  load average: 0,05, 0,04, 0,05
Tasks: 306 total,   2 running, 304 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,0 us,  0,3 sy,  0,0 ni, 97,4 id,  0,0 wa,  0,3 hi,  0,0 si,  0,0 st
KiB Mem:   1010460 total,   887800 used,   122660 free,    60036 buffers
KiB Swap:  1046524 total,     9832 used,  1036692 free.   215340 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2229 student   20   0 1272016 180228  34892 S  1,3 17,8  11:05.41 compiz
 1103 root      20   0  282536  67644  13248 S  0,3  6,7   1:30.54 Xorg
 3334 student   20   0   29520   1828   1204 R  0,3  0,2   0:00.14 top
    1 root      20   0   33780   2776   1404 S  0,0  0,3   0:01.62 init
    2 root      20   0       0      0      0 S  0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S  0,0  0,0   0:02.01 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0,0  0,0   0:00.33 rcu_sched
```

h → help q → quit

Signalling Processes

`kill`

Process stoppen.

```
student@UbuntuDesktop:~$ sleep 1000 &
[1] 3337
student@UbuntuDesktop:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
student     3163  3154  0 13:06 pts/11   00:00:00 bash
student     3337  3163  0 13:51 pts/11   00:00:00 sleep 1000
student     3338  3163  0 13:51 pts/11   00:00:00 ps -f
student@UbuntuDesktop:~$ kill 3337
[1]+  Terminated              sleep 1000
student@UbuntuDesktop:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
student     3163  3154  0 13:06 pts/11   00:00:00 bash
student     3339  3163  0 13:51 pts/11   00:00:00 ps -f
```

kill 3337 = kill -15 3337 = kill -s SIGTERM 3337

Signalling Processes

list signals

Draaiende processen kunnen signals ontvangen van andere processen of van users.

```
kill -l
```

`kill -1`

`SIGHUP`

Process laten weten dat het de configuratiefile moet herlezen.

(Afhankelijk van het proces kan dit wel of niet, sommige processen moeten stoppen en starten. Zie documentatie van het programma/Daemon!)

Signalling Processes

`kill -15`

`SIGTERM`

standard kill

Wordt uitgevoerd als er geen signal wordt meegegeven.

`kill -9`

`SIGKILL`

Wordt niet naar het proces gestuurd, maar naar de kernel.

De kernel zal het proces stoppen.

“sure kill”

Signalling Processes

`kill -19`

`SIGSTOP`

Met `SIGSTOP` wordt een proces gepauzeerd (suspended).

Zo'n proces gebruikt geen cpu cycles, maar blijft in het geheugen.

`kill -18`

`SIGCONT`

Een gepauzeerd proces kan terug gereanimeerd worden met `SIGCONT`.

Signalling Processes

`pkill`

`pkill slee`
zou ook gaan, want pkill werkt met pattern matching

```
student@UbuntuDesktop:~$ pgrep sleep
3320
3324
student@UbuntuDesktop:~$ pkill sleep
[1]-  Terminated              sleep 1000
[2]+  Terminated              sleep 2000
```

`killall`

stuurt SIGTERM naar alle processen met de opgegeven naam (geen pattern matching)

```
student@UbuntuDesktop:~$ sleep 1000 &
[1] 3376
student@UbuntuDesktop:~$ sleep 1500 &
[2] 3377
student@UbuntuDesktop:~$ jobs
[1]-  Running                 sleep 1000 &
[2]+  Running                 sleep 1500 &
student@UbuntuDesktop:~$ killall sleep
[1]-  Terminated              sleep 1000
[2]+  Terminated              sleep 1500
```

Signalling Processes

Signalling vanuit top

Start firefox en enkele ander applicaties
Zoek naar de processen van firefox

```
student@UbuntuDesktop:~$ ps -x | grep firefox
 3528 ?        Sl     0:03 /usr/lib/firefox/firefox
 3654 pts/0    S+     0:00 grep --color=auto firefox
```

of pgrep -l firefox

Start top en druk op “K” voor kill

```
PID to signal/kill [default pid = 2229] 3528
```

```
Send pid 3528 signal [15/sigterm] 15
```

Zoek opnieuw naar de processen van firefox

```
student@UbuntuDesktop:~$ ps -x | grep firefox
 3656 pts/0    S+     0:00 grep --color=auto firefox
```

of pgrep -l firefox

Priority and Nice Values

Ieder process heeft een priority en een nice waarde.

Hogere nice waarde = hogere priority waarde

Hogere priority waarde = minder CPU tijd

Je kan dit beïnvloeden met `nice` en `renice`

De verhouding tussen nice, priority en de overeenkomstige CPU-tijd is niet éénduidig en hangt af van dynamische factoren zoals het aantal processen en moeilijke wiskundige formules. Er is dus geen vaste relatie.

Priority and Nice Values

We bekijken met “top” vier processen die elk evenveel CPU innemen en samen 100% van de CPU innemen

top zonder argumenten uitgevoerd

```
top - 16:24:38 up  5:33,  3 users,  load average: 1,83, 0,69, 0,34
Tasks: 318 total,   3 running, 313 sleeping,   2 stopped,   0 zombie
%Cpu(s): 38,2 us, 61,8 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   1010460 total,   887128 used,   123332 free,     6976 buffers
KiB Swap:  1046524 total,    60692 used,   985832 free.   170888 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 3841 student   20   0   11668    616    520 S 23,1  0,1   0:38.57 proj33
 3850 student   20   0   11668    616    520 R 23,1  0,1   0:38.57 proj33
 3853 student   20   0   11668    616    520 R 23,1  0,1   0:17.18 proj42
 3855 student   20   0   11668    616    520 S 22,8  0,1   0:17.17 proj42
```

CPU idle time is 0.0

Onze processen gebruiken de volledige CPU-kracht

Priority and Nice Values

top -p

Monitoring van specifieke processen

```
top -p 3841,3850,3853,3855
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3841	**student**	**20**	**0**	**11668**	**616**	**520**	**R**	**22,6**	**0,1**	**12:27.83**	**proj33**
3850	student	20	0	11668	616	520	S	22,3	0,1	12:27.82	proj33
3853	**student**	**20**	**0**	**11668**	**616**	**520**	**R**	**22,3**	**0,1**	**12:06.43**	**proj42**
3855	**student**	**20**	**0**	**11668**	**616**	**520**	**R**	**22,3**	**0,1**	**12:06.44**	**proj42**

4 processen – zelfde prioriteit – moeten vechten voor processortijd

Priority and Nice Values

renice

Met renice kan je de nice waarde wijzigen van draaiende processen.

Hier +8 voor de proj33-processen

```
student@UbuntuDesktop:~/procs$ renice +8 3841
3841 (process ID) old priority 0, new priority 8
student@UbuntuDesktop:~/procs$ renice +8 3850
3850 (process ID) old priority 0, new priority 8
```

Normale users kunnen een nice waarde toevoegen van 0 tot 20 aan hun eigen processen.
Enkel root kan negatieve nice waarden toekennen.
(Voorzichtig zijn met negatieve nice waarden !!)

Priority and Nice Values

impact of nice values

`top -p 3841,3850,3853,3855`

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3853	**student**	**20**	**0**	**11668**	**616**	**520**	**R**	**42,3**	**0,1**	**16:17.74**	**proj42**
3855	student	20	0	11668	616	520	S	42,3	0,1	16:17.75	proj42
3841	**student**	**28**	**8**	**11668**	**616**	**520**	**R**	**7,0**	**0,1**	**13:44.30**	**proj33**
3850	student	28	8	11668	616	520	S	7,0	0,1	13:43.94	proj33

`nice`

nice werkt hetzelfde als renice, maar wordt gebruikt bij het starten van een proces

Background Processes

jobs

jobs toont de jobs (processen) die in de background draaien in je huidige shell.

```
student@UbuntuDesktop:~$ jobs
student@UbuntuDesktop:~$
```

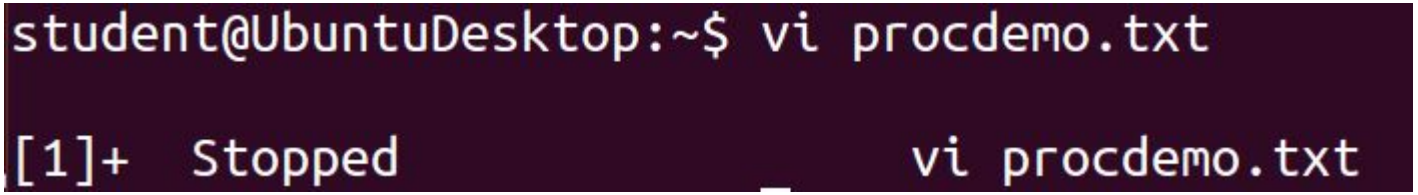
Standaard draaien er geen jobs in de background.

Background Processes

control-Z

Sommige processen kan je pauzeren (stoppen) naar de background met ctrl-Z

→ `SIGSTOP`



Background Processes

& ampersand

Processen die gestart worden met een & achteraan, worden naar de background gebracht waar ze blijven uitvoeren

Ook deze jobs kunnen getoond worden met het commando jobs.

```
student@UbuntuDesktop:~$ find / > allfiles.txt 2> /dev/null &
[2] 3993
student@UbuntuDesktop:~$ jobs
[1]+  Stopped                 vi procdemo.txt
[2]-  Running                 find / > allfiles.txt 2> /dev/null &
```

Background Processes

jobs -p

Om de PID van de background processen te tonen.

```
student@UbuntuDesktop:~$ sleep 500 &
[1] 3994
student@UbuntuDesktop:~$ sleep 400 &
[2] 3995
student@UbuntuDesktop:~$ jobs -p
3994
3995
student@UbuntuDesktop:~$ ps $(jobs -p)
  PID TTY      STAT   TIME COMMAND
 3994 pts/11   S      0:00 sleep 500
 3995 pts/11   S      0:00 sleep 400
```

Background Processes

fg

Om een background proces naar de foreground te brengen en opnieuw door te laten uitvoeren.

De parameter is het nummer van de background job.

```
student@UbuntuDesktop:~$ jobs
[1]   Running                 sleep 1000 &
[2]-  Running                 sleep 1000 &
[3]+  Running                 sleep 2000 &
student@UbuntuDesktop:~$ fg 3
sleep 2000
```

Background Processes

bg

Om een background proces dat gepauzeerd is terug te starten en op de background te houden.

```
student@UbuntuDesktop:~$ sleep 5000 &
[1] 4029
student@UbuntuDesktop:~$ sleep 3000
^Z
[2]+  Stopped                 sleep 3000
student@UbuntuDesktop:~$ jobs
[1]-  Running                 sleep 5000 &
[2]+  Stopped                 sleep 3000
student@UbuntuDesktop:~$ bg 2
[2]+ sleep 3000 &
student@UbuntuDesktop:~$ jobs
[1]-  Running                 sleep 5000 &
[2]+  Running                 sleep 3000 &
```