Processen

In dit hoofdstuk maken we nader kennis met processen en daarbij behorende commando's

Doelstelling

Aan het eind van dit hoofdstuk is de cursist bekend met:

- wat een proces is
- de begrippen *parent* en *child* in relatie tot processen
- processen te starten en te stoppen
- informatie m.b.t. een proces op te vragen
- het verschil tussen interne en externe commando's
- achtergrondprocessen en job control
- de bij al bovenstaande acties behorende commando's

Een proces

- Een proces is een programma (commando) in uitvoer
- Een proces bevind zich in het werkheugen (of in swap-space) van de computer
- Bij inloggen wordt een shell gestart wat op dat moment een proces is
- Een proces (bv. uw shell proces) kan andere processen starten

---

ouder / kind

Net als bij de directorystructuur kennen we bij processen ook een ouder / kind (parent / child) relatie

- een parent proces heeft een ander (kind) proces gestart
- meerdere kind processen mogelijk
- kind processen kunnen zelf ook weer processen starten (en zijn dan zowel parent als child)
- elk proces heeft een uniek ID (proces id: PID)

proces 261 (`ls`) is child van proces 260 (`shell`)
proces 260 (`shell`) is parent van proces 261 (`ls`)

fork en exec

In voorgaand voorbeeld

- de shell kopieert (forked) zichzelf in memory waarmee een nieuw kindproces ontstaat
- het nieuwe memory wordt overschreven (met exec) met de code van `ls`. Hiermee is het `ls` proces een kind proces van de shell.
- als `ls` klaar is sterft dat proces en wordt het parent proces (de shell) weer actief
- een nieuwe commando kan gegeven worden en de cyclus herhaald zich.

Indien geen fork wordt gedaan:

de shell overschrijft zichzelf (met exec) met de code van `ls`, er onstaat geen nieuw proces

als `ls` klaar is sterft dat proces (dus ook je shell) en wordt je uitgelogt (indien je shell een login-shell was)

Het ps commando

De status van processen is te zien met het `ps` commando:

```
ps [optie]...
```

- staat voor proces status
- toont informatie over processen die actief zijn. Zonder optie wordt alleen de eigen processen getoond
- Van oudsher verschil in gebruik van opties tussen de verschillende unix-en. Op veel Linux-en zijn deze ”verenigd”:
  - opties kunnen zowel met hyphen (Unix system V ps) als zonder (BSD ps) gegeven worden
  - kan conflict opleveren: `ps -aux` (!)
- `-e` toon alle processen
- `-f` geef processen weer in volledig (full) formaat:

```
oscar@sammie:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
oscar      674  2704  4 11:34 pts/3    00:00:00 /bin/bash
oscar      751   674  0 11:34 pts/3    00:00:00 ps -f
```

top, utmp, wtmp, btmp

`top` toont overzichtelijk de status van een system: actieve processen, memory gebruik, load...

Standaard worden de meest cpu intensieve processen getoond. Deze volgorde is te wijzigen met bepaalde toetsen (bv. `M` voor sorten op memory usage. Raadpleeg de manpage van uw top!

- `-b` niet interactieve (batch) mode (handig in scripts)

`/var/run/utmp`: bevat huidige status van het systeem (uptime, wie zijn er ingelogd/uitgelogd, runlevel, ...)

`/var/log/wtmp`: ”historische” `utmp` (bv. gebruikt door commando: `last`)

`/var/log/btmp`: bevat foute login pogingen. Maak zichtbaar met het commando: `lastb`

uname

geeft informatie over het systeem

`uname [OPTION]...`

`-a` geef alle informatie

`-s` geef de kernel

`-r` geef de kernel release

`-v` geef de kernel versie

`-n` geef de hostnaam

`-o` geef het OS

`-i` geef het hardware platform

`-p` geef de processor

Bijvoorbeeld:

```
oscar@cws001:~$ uname -a
    Linux cws001 3.2.0-41-generic-pae #66-Ubuntu SMP Thu Apr 25
   03:50:20 UTC 2013 i686 i686 i386 GNU/Linux
oscar@cws001:~$
```

jobs

Behalve de oorspronkelijke bourne shell kennen vrijwel alle shells ”job control”.

- Processen zijn jobs.
- Een proces is veelal een *foreground* job: gedurende de uitvoer moet u wachten. Als de job (of proces) beeindigd is krijgt u uw shell-prompt weer terug en kunt u een nieuw proces (job) starten.
- Duurt het proces lang en wilt u direct uw prompt terug dan kunt u het proces in de achtergrond (*background*) starten. De background job doet z'n werk terwijl u vanaf de prompt direct weer nieuwe commando's kan uitvoeren.

---

background jobs

- Duurt een proces lang en wilt u direct uw prompt terug dan kunt u het proces in de achtergrond (*background*) starten. De background job doet z'n werk terwijl u vanaf de prompt direct weer nieuwe commando's kan uitvoeren.
- Door een ampersand (&) achter een commando te plaatsen start u deze in de achtergrond. Het job-*nummer* en PID worden getoond en u krijgt direct uw prompt terug:

```
$ sleep 100 &
[1] 268
$ ”volgende commando”
```

- Wanner de job klaar is wordt dit getoond:

```
[1]+ Done sleep 100 &
```

- Er kunnnen meer jobs in de achtergrond gestart worden, het job-nummer loopt op.
- Er is geen user-interactie met de achtergrond job mogelijk: het proces wordt losgekoppeld van uw keyboard. Input zal dus bv. uit een file gelezen moeten worden. Output gaat nog wel naar het scherm en om te voorkomen dat dit uw scherm ”vervuild” kan de output (net als de input) het beste *geredirect* worden (hierover meer in hoofdstuk 12)
- Achtergrond processen zijn vaak zinvol als ze veel tijd nodig hebben, duur van het proces niet zozeer van belang is en als ze zelfstandig (zonder user interactie) kunnen runnen. Zeker als u maar over 1 terminal beschikt.

job control

- Een voorgrond job kunt u **stoppen** met `<Ctrl>-z`. U krijgt dan uw prompt terug.
- Om het gestopte voorgrond proces in de achtergrond verder te runnen geeft u het commando `bg`
- om een achtergrond job naar de voorgrond halen geeft u het commando: `fg`

Bij meerdere jobs kunt u deze met `fg` en `bg` op een aantal manieren aanspreken:

- %job-nummer: op basis van job nummer
- %string op basis van de aangegeven string. De naam van de job begint met de aangegeven string
- %?string op basis van de aangegeven string. Se string komt voor in de job-naam
- %+ verwijzing naar de laatst gestarte job in de achtergrond
- %- verwijzing naar de op één na laatst gestarte job in de achtergrond

  Bijvoorbeeld: `bg %3`
-

Een overzicht van background jobs krijgt u met het commando jobs:

```
$ jobs
[3]     +       Running         find . -print > /tmp/gevonden 2>&1 &
[2]     -       Running sort -o uitvoer rommelig &
[1]             Stopped sleep 100 &
```

nohup

Dit staat voor No Hangup

- Als parent verdwijnt stuurt deze een hangup (HUP) signal naar de kind processen om deze te beeindigen
- Met nohup voor een commando blijft deze runnen in de achtergrond, ook als de parent verdwijnt. Je kunt dus uitloggen.
- commando moet zelfstandig kunnen draaien (geen input van terminal nodig). De output wordt standaard naar het file `nohup.out` geschreven.

---

zombies

- Als commando klaar is staat er nog wel een entry in de proces-tabel
- De parent kan met de wait system call de exit status verkrijgen en dan wordt ook de entry uit de proces tabel opgeruimd. Als dit mis gaat blijft het proces als zombie-proces bestaan
- Met `kill` niet op te ruimen
- Z status in STAT kolom van de `ps` output
- Ook wel een ”defunct” proces genoemd

---

kill

Met `kill` kunnen signals naar een proces gestuurd worden:

```
kill -signal pid
```

`-2|INT` stuurt een interrupt signal (wat ook `<Ctrl>-c` doet)

`-9|KILL` stuurt een kill signal. Breekt een proces af zonder dat dit door het proces afgevangen kan worden. Ook wel hard kill genoemd.

`-15|TERM` stuurt een term signal. Dit is de default en kan daardoor weggelaten worden. Breekt proces af wanneer he proces dit toestaat. Ook wel soft kill genoemd.

Verstandig: eerst soft kill proberen (geef het programma een kans netjes af te sluiten), dan pas hard kill

---

killall, pkill, free

`killall` kill processen niet met PID maar met naam

`pgrep` / `pkill`: toon / kill processen niet met het PID maar met de naam.

`free` toon in gebruik en vrije memory

---

interne en externe commando's

- De shell heeft een aantal commando's ”build in”. Dit zijn dan interne commando's, zoals:
  - `cd`, `echo`, `kill`, `if`, `then`, `else`, `for`, `break`, `continue`, `...`
  - Gebruik veelal in shell-scripts (maar bv. ook `cd` en `echo`)
  - zijn geen complexe commando's: als onderdeel van de shell dienen ze (programmeertechnisch) relatief makkelijk in te passen te zijn.
- Externe commando's maken geen onderdeel uit van de shell en moeten op disk gezocht worden, enkele voorbeelden
  - `echo`, `kill`, `chmod`, `cp`, `mkdir`, `ls`, `find`, `cat`, `less`, ...
- Merk op: soms zowel een intern commando als een extern commando (bv. `echo` en `kill`)!
- De shell kijkt **eerst intern** of een commando bestaat. Zo niet dan wordt het zoekpad (de vartiable PATH) gebruikt om externe commando's te vinden. PATH bevat zoekpaden gescheiden door een dubbele punt (:)

```
$ echo $PATH
/usr/local/bin:/usr/sbin:/usr/bin:/bin
```

- De huidige werk-directory (`.`) zit standaard **niet** in uw zoekpad (en verdient ook niet de voorkeur..)

nice en renice

- Commando's hebben een bepaalde prioriteit. Deze is te zien in de output column `PRI` met het commando `ps -l`.
- Een user kan vriendelijker zijn door het commando **minder** prioriteit te geven door de *nice waarde* te **verhogen.** De nice-waarde is te zien in de output column `NI` met het commando `ps -l`
- `nice -n nice-value` *`commando`* Bv.:
  - `nice -n 19 sudo find / -name '*.old' -ls`
  - **`nice -n 10 <=> nice -10 en nice -n -5 <=> nice --5`**
- nice values varieren van -20 t/m 19. De default waarde die een proces krijgt is de waarde van de parent (meestal 0). De default waarde die `nice` geeft zonder optie is 10.
- gebruikers mogen alleen een positieve nice-value geven (mogen alleen aardiger zijn)
- de root user mag ook negatieve nice-values meegeven (**minder** nice, **meer** prioriteit)

Achteraf kan de nice value nog veranderd worden met renice:

- `renice -n nice-value -p pid`

Alle processen van een user renicen:

- `renice -n nice-value -u username`

Oefeningen

Tijd voor oefening!