Operating System fundamentals

Process management

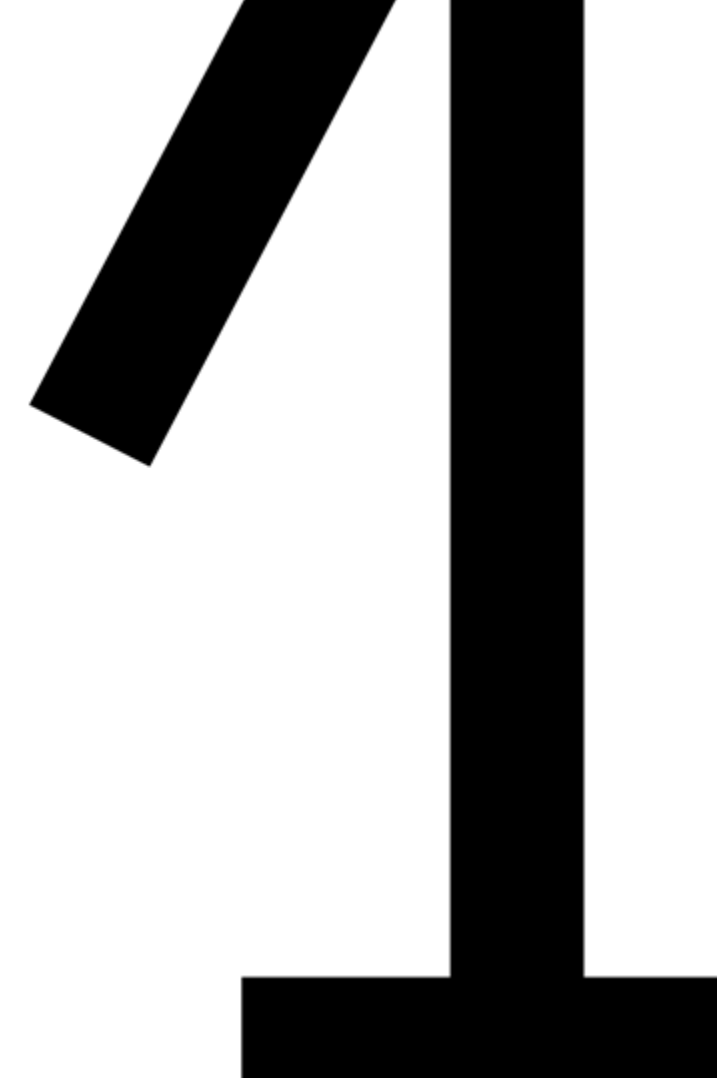
KdG Karel de Grote Hogeschool

Contents

Course text

- Chapter 12 Monitor and Manage Linux Processes
  - RedHat chapter 8
    - Process States and Lifecycle
    - Control Jobs
    - Kill Processes
    - Monitor Process Activity
  - KdG chapter 12: Schedule Linux Processes (not in RedHat)

Process States and Lifecycle



KdG
Karel de Grote Hogeschool

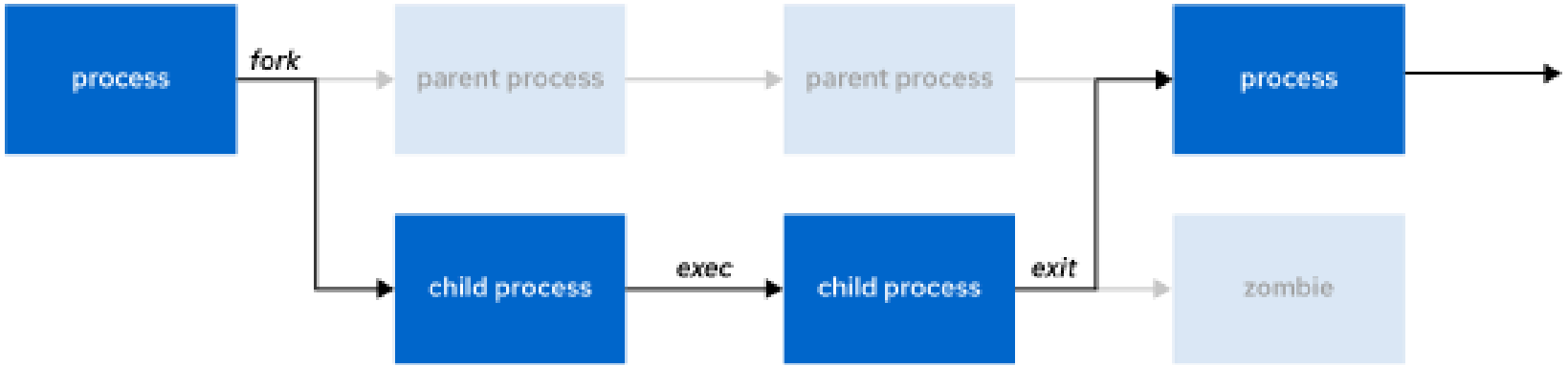
Processes

- What is a process?
- What properties does a process have?
  - id
  - state
  - memory (code/data/stack/heap)
  - security: owner, privileges
  - system resources
  - priority
  - …
- processes can start other processes
  - fork() -> duplicates a process
  - exec() -> replaces a process

Scheduling

- There are hundreds of active processes
- There are only a limited number of “cores”
- So how can the computer execute all these processes?
  - “round robin preemptive scheduling”

Processes

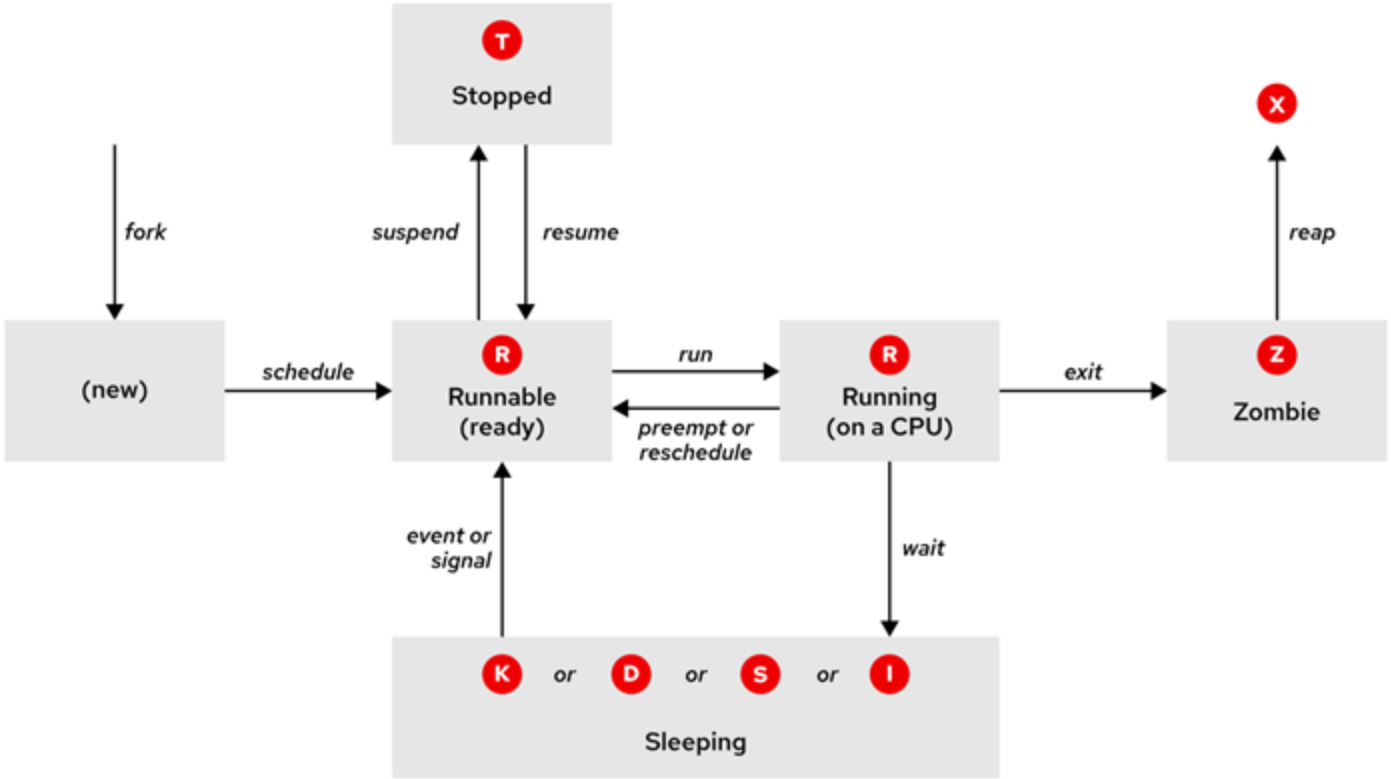

process
fork
parent process
parent process
process
child process
exec
child process
exit
zombie

The “pstree” command

- The **pstree** command shows the hierarchy of processes
- Starts with “systemd” (PID=1)

```
systemd-+-NetworkManager---2*[{NetworkManager}]
        |-VBoxDRMClient---4*[{VBoxDRMClient}]
        |-VBoxService---8*[{VBoxService}]
        |-auditd---{auditd}
        |-chronyd
        |-crond
        |-dbus-broker-lau---dbus-broker
        |-firewalld---{firewalld}
        |-login---bash---pstree
        |-nm-dispatcher---3*[{nm-dispatcher}]
        |-rsyslogd---2*[{rsyslogd}]
        |-sshd
        |-systemd---(sd-pam)
        |-systemd-hostnam
        |-systemd-journal
        |-systemd-logind
        `-systemd-udevd
```

Process states



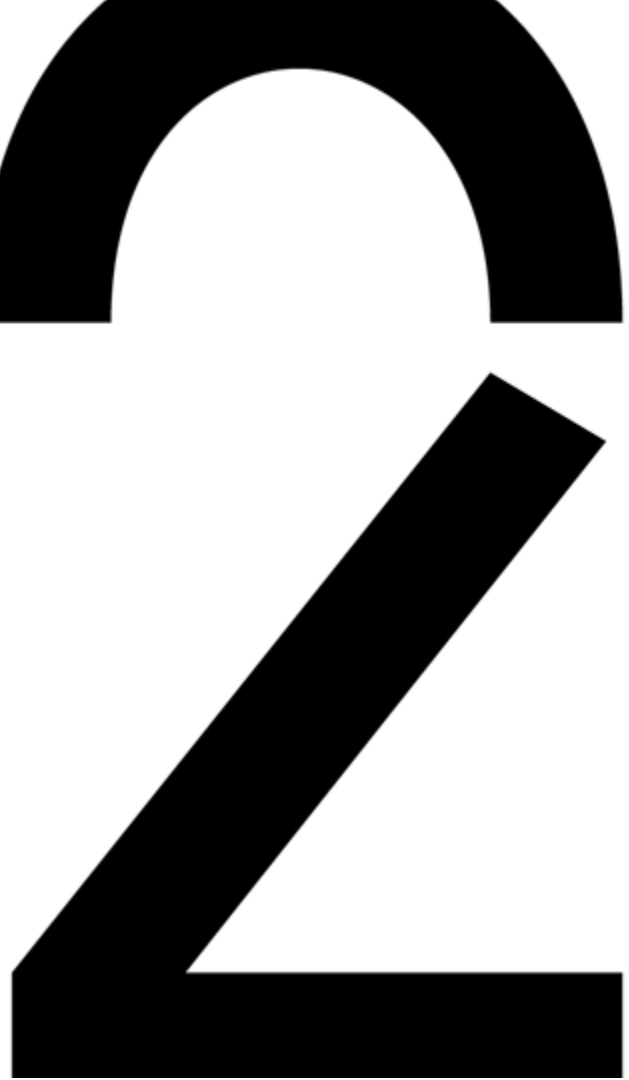
The “ps” command

- “**ps aux**” or **"ps –ef"** shows for all processes:
  - user identification (USER)
  - process identification (PID)
  - CPU usage (%CPU)
  - memory usage (%MEM)
  - proces standard input/output (TTY)
  - proces state (STAT)
  - start time (START)
  - amount of CPU time used (TIME)
  - executing command (COMMAND)

Different variants

- **ps aux**
  - used most frequently
- **ps lax** also shows:
  - parent process id (PPID)
  - priority (PRI)
  - nice level (NI)
- **ps fax** also shows:
  - hierarchy

Control Jobs



Jobs

- a “job” is all processes that are started in order to execute a command
- example:
  - cat /etc/passwd | cut -d “:” -f 1 | sort -u >users.txt
  - how many processes are started here?
- you can start multiple jobs in 1 terminal
  - only 1 job has access to the keyboard (foreground)
  - the other jobs have access to the terminal for output (background)

Jobs in the background and foreground

- Use **&** to start a job in the background:

```
[user@host ~]$ sleep 10000 &
[1] 5947
```

- Get a list of all jobs:

```
[user@host ~]$ jobs
[1]+ Running    sleep 10000 &
```

- Bring a job to the foreground:

```
[user@host ~]$ fg %1
sleep 10000
```

Jobs in the background and foreground

- Ending a job in the foreground with ctrl-c

```
[user@host ~]$ sleep 10000
^C
```

- Stopping a job in the foreground with ctrl-z

```
[user@host ~]$ sleep 10000
^Z
[1]+  Stopped     sleep 10000
```

- Activating a stopped job in the background:

```
[user@host ~]$ bg %1
[1]+ sleep 10000 &
```

Job and session information

```
[user@host ~]$ ps j
  PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
   850    1472    1472    1472 tty1         1581 Ss    1000   0:00 -bash
  1472    1579    1579    1472 tty1         1581 S     1000   0:00 sleep 10000
  1472    1581    1581    1472 tty1         1581 R+    1000   0:00 ps j
```

- process parent ID (PPID)
- process group leader, first process in pipeline (PGID)
- session leader (SID)

Process signals

Process signals

- You can send a signal to a process
  - using the keyboard
    - ctrl-c: INT
    - ctrl-z: TSTP
    - ctrl-\: QUIT
  - using a command (**kill, killall, pkill**)
    - e.g.: kill -9 1453 # unconditional stop of process 1453
    - e.g.: kill -SIGTERM 8476 # clean stop of process 8476

Signals

Signal	Name	Definition
1	HUP	`Hangup` : Reports termination of the controlling process of a terminal. Also requests process re-initialization (configuration reload) without termination.
2	INT	`Keyboard interrupt` : Causes program termination. It can be blocked or handled. Sent by pressing the INTR (Interrupt) key sequence (Ctrl+c).
3	QUIT	`Keyboard quit` : Similar to SIGINT; adds a process dump at termination. Sent by pressing the QUIT key sequence (kbd:[Ctrl+\]).
9	KILL	`Kill, unblockable` : Causes abrupt program termination. It cannot be blocked, ignored, or handled; consistently fatal.
15 *default*	TERM	`Terminate` : Causes program termination. Unlike SIGKILL, it can be blocked, ignored, or handled. The "clean" way to ask a program to terminate; it allows the program to complete essential operations and self-cleanup before termination.
18	CONT	`Continue` : Sent to a process to resume if stopped. It cannot be blocked. Even if handled, it always resumes the process.
19	STOP	`Stop, unblockable` : Suspends the process. It cannot be blocked or handled.
20	TSTP	`Keyboard stop` : Unlike SIGSTOP, it can be blocked, ignored, or handled. Sent by pressing the suspend key sequence (Ctrl+z).

The kill, killall, and pkill command

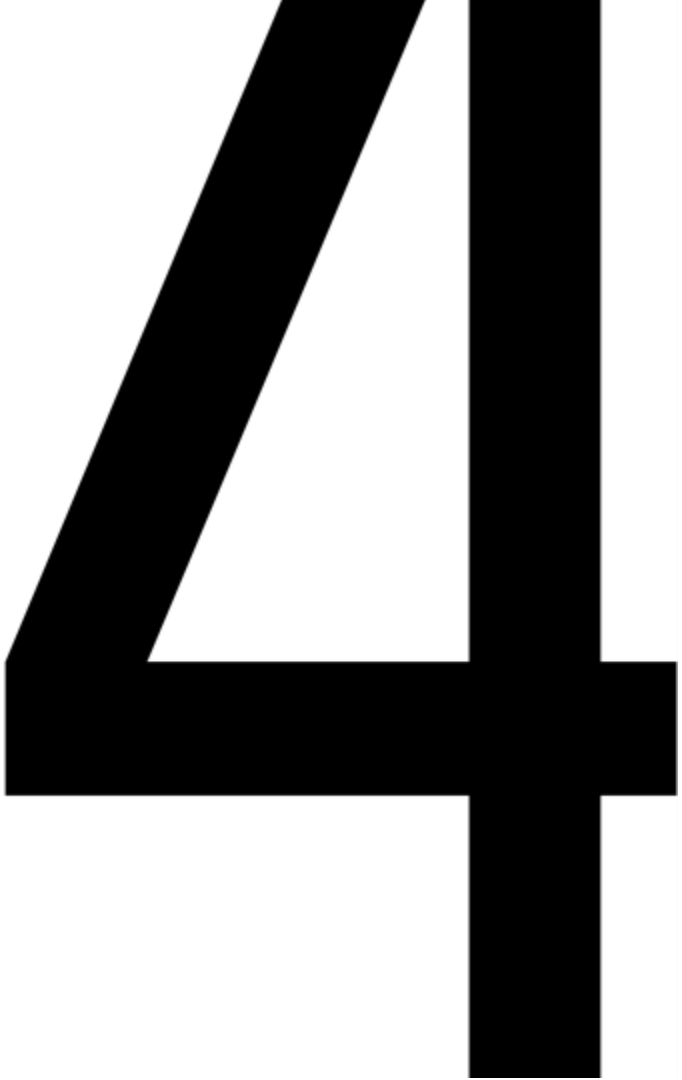
- **kill** [-signal] process-id
  - no signal -> default 15 (TERM)
  - e.g.: **kill -SIGKILL** 22676
  - you can use “KILL” or “SIGKILL”
- **killall** [-signal] process-name
  - you can refer to a process by its name
  - bv: **killall -9** bash
- **pkill** [-signal] pattern
  - you can refer to processes with a regular expression
  - bv: **pkill -SIGTERM** ‘.*sh’
  - also other options:
    - bv: **pkill -U** jim
    - bv: **pkill -t** tty3
    - …

pgrep

- If you don’t want to send a signal but just list processes
- Same options as pkill
- e.g.:

```
[ user@server ~ ] $ pgrep -l -u student
6964 bash
6998 sleep
6999 sleep
7000 sleep
```

Monitor Process Activity



Load average

- Load average = average number of processes in “Runnable” state during a certain period
- The command “**uptime**” shows this:

```
[ user@server ~ ] $ uptime
15:29:03 up 14 min,  2 users,  load average: 2.92, 4.48, 5.20
```

- load average for the last 1, 5 en 15 minutes
- the values decrease: what does this mean?
- if load average == 1
  - 1 processor with 1 core is busy 100% of the time
  - 1 processor with 4 cores busy 25% of the time
  - it is important to know the number of cores

CPU information

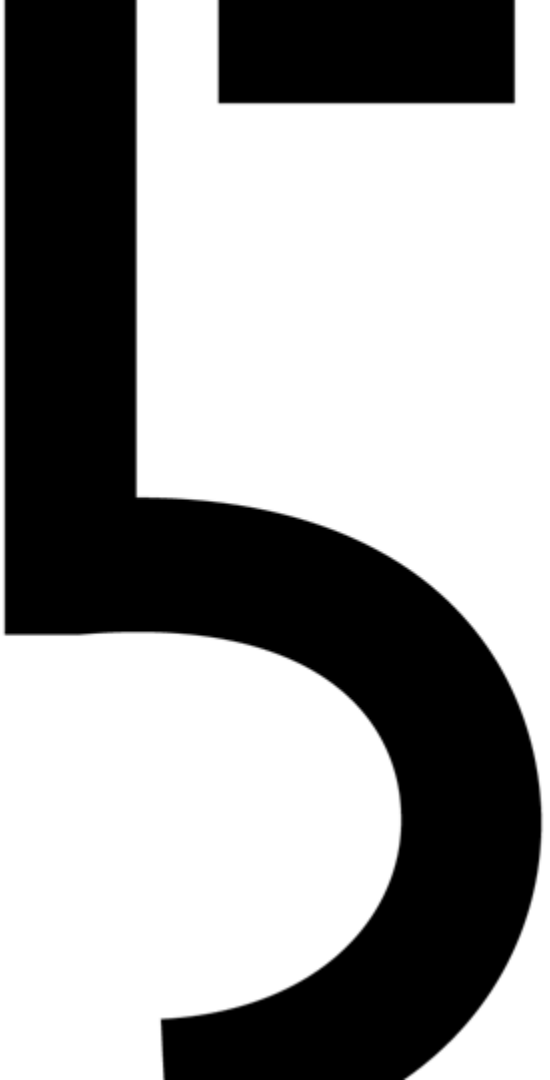
- The "**lscpu**" command gives information about the CPU:

```
[ user@server ~ ] $ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
```

Interactive overview of processes

- The "**top**" command shows information about processes, sorted by CPU usage
  - PID
  - USER
  - Virtual memory (all memory, including resident, libraries,shared)
  - Resident memory (physical memory)
  - Process state
  - CPU time - total cpu time
  - the process command

Schedule Linux Processes



KdG
Karel de Grote Hogeschool

Scheduling processes

- Sometimes you want to start a command at a certain point in time
  - use the "**at**" command
  - bv: **at** 5pm < /home/user/bin/backup.sh

Crontab

- You can view and edit your crontab with the **crontab** command
- crontab -l # shows the current crontab
- crontab -e # starts editor with crontab

```
minutes      hour     day_of_month      month    day_of_week
    command
0    16       1        12       *        tar -cf
~/backup_home.tar ~
```

- more info: https://tecadmin.net/crontab-in-linux-with-20-examples-of-cron-schedule/

Example crontab

```
#min  hour day month day command
 5    4    *    *    *  script.sh
```
→ every day at 04.05am
```
 0   16    *    *    *  tar cf home.tar ~
```
→ every day at 16:00
```
*/5 9-16  *    *   1-5 mail -s “hello” root
```
→ send a mail to root with subject “hello” every 5 minutes, between 9:00 and 16.55, monday till friday

Exercise

- **`crontab -e`**
- add the following line:

  ```
  */1 9-18 * * 1-5 /usr/bin/logger "Hello INF10x!"
  ```
- execute `journalctl -n`
- wait a minute
- execute `journalctl -n`
- delete the line again

Scheduling processes

- Sometimes you want to execute a command on a regular basis
- There is a “cron daemon” running on your system
  - every user has a “crontab”
  - contains commands and when to execute them
  - the daemon will execute the tasks when needed

Exercises

Exercises

- KdG
  - Chapter 12: exercise 1 - 7
- RedHat
  - ch08s02
  - ch08s04
  - ch08s06
  - ch08s08
  - ch08s09

KdG Karel de Grote Hogeschool