

# LPIC-1 101-400 – Lesson 5

## 103.5 Create, monitor and kill processes



# Processes

- A process is the copy of a program executed in the memory (RAM) at a specific moment
- Processes are created when we call a program by its name
- Every process has an integer unique id (Process ID - PID)
- Processes created by other parent processes are called child processes



# init

- It is under **/sbin/init**
- It is “the mother of all processes” because all other processes are either directly or indirectly children of **init**.
- It has a PID equal to 1
- If you kill **init** all other processes die too and the system freezes.

***Note:** in some systems (e.g. RedHat) **init** has been replaced by **systemd***



# Display running processes with `ps`

- The **ps** command can show us valuable information about running processes
- The options of **ps** can be confusing because some are compatible with the **UNIX 98** standard and others with the **BSD Unix** standard
  - **UNIX 98**: these options have a single dash
  - **BSD**: these options have no dash
  - **GNU**: these options have a double dash



# Display running processes with `ps`

- `$ ps #` show the current shell and it's processes
- `$ ps a #` display all processes that belong to terminals (tty)
- `$ ps au #` display additional fields like USER
- `$ ps aux #` display all processes including services (daemons) and system programs



# Display running processes with `ps`

- `$ ps auxw # w` will adapt the length of the output to the output of the terminal without truncating it at 80 columns
- `$ ps auxwf #` will display the hierarchical relationship between parents and children in a tree
- `$ ps eaux #` display the environment variables for each process
- `$ ps -e #` display all processes
- `$ ps -ef #` same as above but display additional columns like PPID.



# Display running processes with `ps`

- `$ ps -f -u theo` # display processes for user theo  
# in UNIX98 format

UID	PID	PPID	C	STIME	TTY	TIME	CMD
theo	1674	1	0	16:50	?	00:00:00	/usr/bin/gnome-keyring-daemon --daemonize --login
theo	1693	1669	0	16:50	?	00:00:00	gnome-session -session=classic-gnome
theo	1728	1693	0	16:50	?	00:00:00	/usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with-session
theo	1731	1	0	16:50	?	00:00:00	gnome-session -session=classic-gnome
theo	1732	1	0	16:50	?	00:00:00	/usr/bin/dbus-launch --exit-with-session gnome-session --session=classic-gnome
theo	1737	1	0	16:50	?	00:00:00	daemon --fork --print-pid 5 --print-address 7 --session
theo	1742	1	0	16:50	?	00:00:03	/usr/lib/libgconf2-4/gconfd-2
theo							/usr/lib/gnome-settings-daemon/gnome-settings-daemon



# Display running processes with `ps`

- \$ ps uU theo # display processes for user theo

```

# in BSD format
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START
TIME COMMAND
theo         1674  0.0  0.0 153844  3396 ?        Sl    16:50
0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
theo         1693  0.0  0.2 239332  8444 ?        Ss1   16:50
0:00 gnome-session --session=classic-gnome
theo         1728  0.0  0.0  12092   288 ?        Ss    16:50
0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with-
session gnome-session --session=classic-gnome
theo         1731  0.0  0.0  26400   612 ?        S     16:50
0:00 /usr/bin/dbus-launch --exit-with-session gnome-session
--session=classic-gnome
theo         1732  0.0  0.0  25820  2160 ?        Ss    16:50
0:00 //bin/dbus-daemon --fork --print-pid 5 --print-address
7 --session
theo         1737  0.0  0.1  57160  5672 ?        S     16:50
0:00 /usr/lib/libgconf2-4/gconfd-2
theo         1742  0.0  0.4 459272 17836 ?        Ss1   16:50
0:03 /usr/lib/gnome-settings-daemon/gnome-settings-daemon
```





# Display running processes with `ps`

- `$ ps -C getty` # display processes  
# of the `getty` command

PID	TTY	TIME	CMD
1046	tty4	00:00:00	getty
1052	tty5	00:00:00	getty
1067	tty2	00:00:00	getty
1069	tty3	00:00:00	getty
1074	tty6	00:00:00	getty
1465	tty1	00:00:00	getty



# Output field of `ps`

Field Name	Description
USER/UID	Name of the user under whom the process is running
PID	Process ID
%CPU/C	CPU utilization. Percentage for BSD, integer for UNIX98
%MEM	Memory Utilization
PPID	Parent Process ID
VSZ	Virtual Memory Size
RSS	Resident Set Size
TTY	tty used by the process
STAT	Process Status
START/STIME	Time when process started
TIME	Additive time of CPU utilization
COMMAND/ CMD	Program name (optionally with options and arguments)

# Hierarchical presentation of processes with `pstree`

- `$ pstree # hierarchical presentation of the parent-child relationship of processes`

## Options:

- `-a # show options and arguments of processes`
- `-n # use numeric sorting instead of alphabetic`
- `-p # show process id (PID)`



# Dynamic display of running processes with `top`

- **top** is an interactive command that displays the most active processes
- It sorts based on CPU utilization by default
- Besides CLI options it accepts interactive commands
- It also displays processes stats system utilization, CPU, Memory, uptime, etc
- We can quit **top** by pressing **q**



# Dynamic display of running processes with `top`

```
top - 18:51:33 up 2:02, 3 users, load average: 0.00, 0.01, 0.05
Tasks: 164 total, 1 running, 162 sleeping, 0 stopped, 1 zombie
Cpu(s): 2.0%us, 1.5%sy, 0.0%ni, 95.9%id, 0.6%wa, 0.0%hi, 0.0%si,
0.0%st
Mem: 3987872k total, 1597792k used, 2390080k free, 69636k buffers
Swap: 3506172k total, 0k used, 3506172k free, 541848k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1386	root	20	0	369m	91m	64m	S	2	2.4	3:14.83	Xorg
1998	theo	20	0	113m	32m	16m	S	2	0.8	2:46.11	npviewer.bin
2311	theo	20	0	324m	17m	11m	S	1	0.4	0:07.12	gnome-terminal
1985	theo	20	0	234m	19m	14m	S	1	0.5	1:14.67	plugin-containe



# Dynamic display of running processes with `top`

- In the `top` header we can find the following information:
  - Uptime
  - Load average
  - Number of users
  - Total number of processes and statistics based on process status
  - CPU utilization statistics
  - Memory and Swap space utilization statistics



# Dynamic display of running processes with `top`

## Options:

- **-b** # useful for sending stdout data to other commands or files. Useful with **-n**
- **-d 1** # update screen every 1s instead of the default 5s
- **-i** # do not display inactive processes
- **-n 5** # close after 5 successive displays



# Dynamic display of running processes with `top`

## Interactive commands:

- **Spacebar**: reload display
- **h**: help
- **k**: kill process ( with signal 15)
- **n**: change number of displayed processes
- **q**: exit **top**
- **r**: change command priority (nice/renice)
- **s**: change renewal period
- **M**: sort based on memory utilization
- **P**: sort based on cpu utilization





# Run commands to foreground and background

- `$ find / -exec grep -i linux {} \;`  
`Ctrl-Z` # suspend the command above
- `$ bg` # re-activate the command in the background. If there are more than one suspended commands we can choose which one to send to background
- `$ find / -name Linux &` # the ampersand operator (&) will send the command straight to background



# Run commands to foreground and background

- `$ jobs # display background or suspended commands`
- `$ fg 2 # restore the 2nd background or suspended command to the foreground. If we do not set a number it will choose the first one`
- The background processes are still child processes of the shell it created them, and can be terminated if the shell is terminated
- `$ nohup find / # detach find from the parent shell and redirect stdout and stderr to the nohup.out file.`

# Signal processes with `kill`

- The command **kill**, in spite of its name, is not just used for killing processes but to send different signals to them. These signals can trigger different responses from the processes. It takes **PID(s)** as argument
- There are 64 different signals, in total, and we will study only the most significant of them. Signals can be expressed with names or numbers
- `$ kill -l # show the names and numbers of all signals`



# Signal processes with `kill`

Signal Name	Number	Description
SIGHUP/ HUP	1	Hang up. Forces a service to re-read its configuration and closes interactive applications
SIGINT/ INT	2	Interrupt. Stops the execution of a command just like when we press Ctrl-C
SIGKILL/ KILL	9	Kill. Kills without mercy! The process is instantly terminated with the risk of losing data. <b>Use only in emergency</b>
SIGTERM/ TERM	15	Terminate. The programs closes gracefully, just like when closed normally. It will first finish any pending tasks, save the data and terminate. This is the most preferable method of closing programs. It is the default signal of <b>kill</b>
SIGTSTP/ TSTP	20	TTY Stop. The execution of a program is suspended and the process expects the next signal (CONT) to resume. Sent when Ctrl-Z is pressed
SIGCONT/ CONT	18	Continue. Send to suspended processes to resume at the same state before being suspended

# Signal processes with `kill`

- `$ kill 3459` # terminate PID 3459 use `ps` first to find the process you are looking for
- `$ kill -15 3459` # = `kill -TERM 3459;`  
`kill -SIGTERM 3459`. All these commands have the same effect since signal 15 is the default
- `$ kill -9 2523` # = `kill -[SIG]KILL`.  
**Kill process without warning!** Any unsaved data will be lost and temporary files will remain in the filesystem

*Note: the only processes that can resist SIGKILL, are the **Zombie** processes. These are marked with **Z** in the **STAT** field of `ps` and just like fictional Zombies they can not be killed because they are already dead!*

# Signal processes with `kill`

- `$ kill -s HUP 1498 # = kill -1.` if the process is a daemon (system service) like Apache it will be forced to re-read its configuration, without restarting. Interactive commands like `top` will be terminated gracefully
- `$ kill -s 15 $(cat /var/run/cups/cupsd.pid) #` gracefully terminate daemon `cupsd`



# Signal processes based on process name with `killall`

- `killall` works in a similar fashion , just like, `kill` except that we give the process name as an argument instead of the PID
- `killall` will send the signal to all processes with the same name
- The default signal is **15 ( SIGTERM)** and options are the same as `kill`
- ```
$ killall apache2 # terminate with 15 all Apache daemon processes
```

“My favorite OS? ... LINUX without a doubt. Get this! It even has a “killall” command!!”

~ George RR Martin ~



# Find a Process ID with `pgrep`

- `pgrep` will return all the PID based on the process name
- `$ pgrep apache # return the PID`  
# of all processes  
# matching "apache"


```
20955  
29064  
29420  
29433  
29862
```

- Options:
- `-f` search the whole command line not just the name
- `-x` exact match





# Signal a process by pattern with `kill`

- `kill` is a sister of `pgrep` and they share almost the same options. It signals the processes based on the match
  - `$ kill apache # terminate all processes matching "apache" in their name`
  - `$ kill -x apache2 # terminate all processes matching exactly "apache2"`
  - `$ kill -f start # terminate all processes matching "start" in their name or options/arguments`
- 

# Show memory and swap utilization with `free`

- `$ free # show usage of memory and swap space in bytes`

## Options:

- `-b # show in bytes (default)`
- `-k # show in kilobytes`
- `-m # show in megabytes`
- `-g # show in gigabytes`
- `-s 2 # renew every 2 seconds`



# Show system's running time with `uptime`

- `$ uptime` # show current time, total system running time since the system started (`uptime`), number of users and load average
- Options:
- `-V` # the only option of `uptime` shows the current version of the command



# Show system's running time with `uptime`

- `$ uptime`

14:34:03 up 10:43, 4 users, load average: 1.73, 0.50, 7.98

In a system with just one CPU these results are interpreted as:

- **1.73**: 73% overloaded system the last minute (0.73 processes had to wait in queue)
- **0.50**: 50% underloaded system the last 5 minutes (no process had to wait in queue)
- **7.98**: 698% overloaded the last 15 minutes (6.98 processes had to wait in queue)



# Multiplex shells with `screen`

- **screen** allows the parallel execution of multiple shell on the same terminal
- **screen** sessions are persistent after being detached from the terminal
- A **screen** session can be resumed later to check on the progress of a process etc.



# Multiplex shells with `screen`

- `$ screen # launch screen`  
`Ctrl-a d # detach screen from terminal`
- `$ screen -r # re-attach detached screen session`
- `$ screen -r # if there are more than one detached screen  
# session you will be prompted to choose one`

There are several suitable screens on:

```
13466.pts-0.srv (06/27/2018 05:44:55 PM) (Detached)
13396.pts-0.srv (06/27/2018 05:38:38 PM) (Detached)
13346.pts-0.srv (06/27/2018 05:37:31 PM) (Detached)
```

Type "screen [-d] -r [pid.]tty.host" to resume one of them.

- `$ screen -r 13466 # re-attach screen session with PID 13466`
- `$ screen -d 25676 # detach session already attached to another terminal`



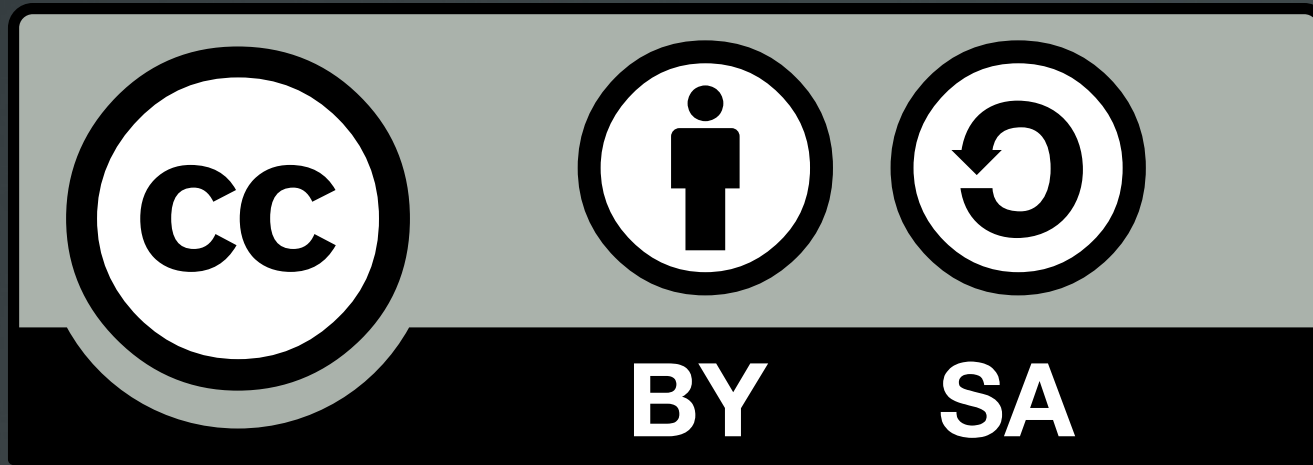
# Multiplex shells with `screen`

## Screen Key Bindings:

- **Ctrl-a a** # rebinds **Ctrl-a** to home
- **Ctrl-a c** # Create a new parallel shell
- **Ctrl-a Ctrl-a** # switches to the previous shell
- **Ctrl-a '** # Select shell to switch
- **Ctrl-a "** # show active shells and select one
- **Ctrl-a 5** # Switch to shell 5
- **Ctrl-a d** # detach screen from current terminal
- **Ctrl-a n** # go to next shell
- **Ctrl-a p** # go to previous shell



# License



The work titled "LPIC-1 101-400 – Lesson 5" by Theodotos Andreou is distributed with the Creative Commons Attribution ShareAlike 4.0 International License.

