# Practice M5: System Startup and Processes Management

\* NOTE: Most of the exercises included in this practice guide are not suitable for execution in WSL or Docker environment. It is recommended that a virtual infrastructure to be used instead.

For the purpose of this practice, we will need at least one virtual machine with **AlmaLinux OS 9.x** (or **CentOS Stream 9**, or **Oracle Linux 9.x**, or **Rocky Linux 9.x**), **openSUSE Leap 15.x,** or **Debian 12.x** (or **Ubuntu Server 22.04/24.04**) installed. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All commands that we are going to use in this practice will be accompanied by the appropriate prompt. This way, it will be easier for us to understand which user, in which folder, and on which machine is executing the command.

The next steps will be executed on an **AlmaLinux** machine. If there is a big difference in the way we issue the command, or in its result between distributions, it will be stated clearly.

Please note that we start each practice with a fresh machine or set of machines.

Please note that the output from the commands shown below may differ compared to the one you get.

## Part 1

If the machine is powered off, then power it on, otherwise restart it.

Hit the **e** key, while on the boot screen asking us to make a choice what to be started. This will allow us to make temporal changes on the boot configuration.

Find the row starting with **linux** (or **linux16**)*(for* ***BIOS*** *based systems)* or **linuxefi** *(for* ***UEFI*** *based systems)* and remove the parameter **quiet** *(if present)*. This will show us all the messages generated during the load process. It is suitable for debugging. Now press **Ctrl+x** to initiate the loading process.

*Please note that* ***linux*** *is the universal option and may appear both on* ***BIOS*** *and* ***UEFI*** *based systems.*

Now, after seeing many messages on screen, our system should be ready for use. Let's log in and execute a command that will show us the same messages, but from the comfort of the command line:

[lsauser@almalinux ~]$ **dmesg**

Okay, the information is not easy to be consumed this way, we can execute this:

[lsauser@almalinux ~]$ **dmesg -H**

Now it should be better. We can search, scroll, and all other actions we can do in less. When ready, we can quit with the **q** key.

Okay, let's reboot the machine again and hit the **e** key while on the boot menu to edit the default parameters. This time we want to boot our machine into a single-user mode.

Once we enter in edit mode, go to the line stating **linux** *(or* ***linux16*** *or* ***linuxefi****)*and add at the end either **1** or **single** or **systemd.unit=rescue.target** *(we can specify another more restricted* ***emergency*** *mode with the word* ***emergency*** *or* ***systemd.unit=emergency.target****)*and then hit the **Ctrl+x** to continue with the boot process. When the loading finishes, we will be asked for the **root** password.

Enter the password and hit the **Enter** key. Then check the current mode with the **runlevel** command.

You should see confirmation of the fact that we are running in a single-user mode:

**N 1**

Once done with the examination, we can type **reboot** or hit **Ctrl+d** to reboot the system.

Let's restore the normal operation of our system.

Our next task is to change the number of seconds that the boot manager waits before initiating the loading of the default kernel.

We could also remove the **quiet** parameter from the **GRUB\_CMDLINE\_LINUX** if present in order to see the detailed information each time our system is booted.

[lsauser@almalinux ~]$ **sudo vi /etc/default/grub**

**GRUB\_TIMEOUT=30**

GRUB\_DISTRIBUTOR="$(sed 's, release .\*$,,g' /etc/system-release)"

GRUB\_DEFAULT=saved

GRUB\_DISABLE\_SUBMENU=true

GRUB\_TERMINAL\_OUTPUT="console"

GRUB\_CMDLINE\_LINUX="resume=/dev/mapper/almalinux-swap rd.lvm.lv=almalinux/root rd.lvm.lv=almalinux/swap **~~quiet~~**"

GRUB\_DISABLE\_RECOVERY="true"

GRUB\_ENABLE\_BLSCFG=true

*Note that yours may differ and the word* ***quite*** *may not be present.*

Then save the changes and quit.

Now, we must transfer the changes to the actual **GRUB** configuration, for this purpose for **BIOS** based systems, we must execute:

[lsauser@almalinux ~]$ **sudo grub2-mkconfig -o /boot/grub2/grub.cfg**

*For* ***Debian*** *and* ***Ubuntu*** *use either* ***grub-mkconfig*** *or* ***update-grub***

Or for **UEFI** based systems this *(note that the path may vary on different distributions)*:

[lsauser@almalinux ~]$ **sudo grub2-mkconfig -o /boot/efi/EFI/almalinux/grub.cfg**

Of course, should we want to see what is going to be written first before actually doing it, we could execute this:

[lsauser@almalinux ~]$ **sudo grub2-mkconfig**

Before we continue and test our new **GRUB** settings, let's experiment a bit and re-install the **GRUB** boot loader *(not that we need to do it but just for the sake of the exercise).*

This can be done, especially in our case, very easily. Just execute the following:

[lsauser@almalinux ~]$ **sudo grub2-install /dev/sda**

*For* ***Debian*** *and* ***Ubuntu*** *use* ***grub-install***

Bear in mind that we must specify the device on which the boot loader to be installed. In our case, this is the first hard drive *(more on the subject in the next module)*.

Now, let's reboot the system and see if it will work.

We are ready to restart the system with:

[lsauser@almalinux ~]$ **sudo reboot**

Now, we can see that the manager will wait for **30** seconds, and if no choice is made, it will load the default kernel.

Let's press the **c** key in order to start the **GRUB** shell.

If want to get some help, we can execute:

grub> **help**

...

ls List devices or files.

...

Let's check what devices **GRUB** sees:

grub> **ls**

Now on **BIOS** system we can execute this command to see the contents of a device:

grub> **ls (hd0,msdos1)/**

We could add options like **-a**, **-l**, **-h**, and others to modify the output.

The same way we can use the **cat** command to see file contents. For example, check the

grub> **cat (hd0,msdos1)/grub2/grub.cfg**

Once we are done exploring, we can hit the **Esc** key to return to the boot menu and continue with the loading process.

We could do one more experiment, while in the boot menu, hit the **c** key to start the **GRUB** shell again. This time, we will create a one-time boot configuration.

Knowing that we have one drive and its partition structure, we can start typing the configuration.

First, we will specify the root partition with:

grub> **set root=(hd0,msdos1)**

Then, we will enter a minimal kernel configuration with *(it may be* ***linux16*** *or* ***linuxefi****)\**:

grub> **linux /vmlinuz-4.18.0-372.9.1.el8.x86\_64 root=/dev/mapper/almalinux-root**

We can use the **Tab** key to write or finish the above statement.

*\* Please note that the exact path and file name varies between different installations and distributions.*

Of course, we have to specify the ram disk configuration as well *(it may be* ***initrd16*** *or* ***initrdefi****)\**:

grub> **initrd /initramfs-4.18.0-372.9.1.el8.x86\_64.img**

*\* Please note that the exact path and file name varies between different installations and distributions.*

Finally, enter:

grub> **boot**

And hit the **Enter** key. After a while, we will have our system booted as usual.

## Part 2

Log in as the **lsauser**.

Before continuing with the tasks, let's install a package, which will ease our life:

[lsauser@almalinux ~]$ **sudo dnf install bash-completion**

...

*The name of the package is the same for all other distributions*

Now we must log out and then log back in. Now we will be able to use tab completion while writing commands for **systemctl** for example.

Let's ask what the default target on our system is with:

[lsauser@almalinux ~]$ **systemctl get-default**

multi-user.target

[lsauser@almalinux ~]$

Now, if we want to check which targets are active:

[lsauser@almalinux ~]$ **systemctl list-units --type=target**

UNIT LOAD ACTIVE SUB DESCRIPTION

basic.target loaded active active Basic System

cryptsetup.target loaded active active Local Encrypted Volumes

getty.target loaded active active Login Prompts

...

Should we want to see the status of the system, we can do it with:

[lsauser@almalinux ~]$ **systemctl status**

● almalinux

State: running

Jobs: 0 queued

Failed: 0 units

Since: ср 2019-03-06 11:22:59 EET; 13min ago

CGroup: /

├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 22

├─user.slice

│ └─user-1000.slice

...

Alternative ways for getting information about particular service or the system manager itself is with:

[lsauser@almalinux ~]$ **systemctl show sshd**

Type=notify

Restart=on-failure

NotifyAccess=main

RestartUSec=42s

TimeoutStartUSec=1min 30s

...

[lsauser@almalinux ~]$ **systemctl show**

Version=219

Features=+PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 -SECCOMP +BLKID +ELFUTILS

Virtualization=kvm

Architecture=x86-64

FirmwareTimestampMonotonic=0

...

We can switch our system to a different target state with:

[lsauser@almalinux ~]$ **sudo systemctl isolate runlevel1.target**

...

After the switch is done, we must enter the root password in order to enter the ***maintenance mode*** or press **Ctrl+d** to continue (return to ***normal mode***). Let's instead enter the root password and check the mode.

We can explore a bit, and then execute:

[root@almalinux ~]# **shutdown -r now**

Let's explore the system journal a bit. For this purpose, we must execute:

[lsauser@almalinux ~]$ **journalctl**

-- Logs begin at ср 2019-03-06 11:53:35 EET, end at ср 2019-03-06 11:55:12 EET. --

мар 06 11:53:35 almalinux systemd-journal[88]: Runtime journal is using 6.1M (max allowed 49.5M, trying to leave 74.3M free of 489.3M availab

мар 06 11:53:35 almalinux kernel: Initializing cgroup subsys cpuset

мар 06 11:53:35 almalinux kernel: Initializing cgroup subsys cpu

...

As we can see it shows events from the earliest to most recent ones. Usually, we would like to have recent events on top, so we can do it with:

[lsauser@almalinux ~]$ **journalctl --reverse**

-- Logs begin at ср 2019-03-06 11:53:35 EET, end at ср 2019-03-06 11:55:12 EET. --

мар 06 11:55:12 almalinux sshd[3327]: pam\_unix(sshd:session): session opened for user lsauser by (uid=0)

мар 06 11:55:12 almalinux systemd-logind[2585]: New session 1 of user lsauser.

мар 06 11:55:12 almalinux systemd[1]: Started Session 1 of user lsauser.

мар 06 11:55:12 almalinux systemd[1]: Created slice User Slice of lsauser.

...

We can add arguments on the command line, for example to filter all events for a single service:

[lsauser@almalinux ~]$ **journalctl --reverse \_SYSTEMD\_UNIT=sshd.service**

-- Logs begin at ср 2019-03-06 11:53:35 EET, end at ср 2019-03-06 11:55:12 EET. --

мар 06 11:55:11 almalinux sshd[3327]: Accepted password for lsauser from 10.0.2.2 port 52648 ssh2

мар 06 11:53:50 almalinux sshd[3077]: Server listening on :: port 22.

мар 06 11:53:50 almalinux sshd[3077]: Server listening on 0.0.0.0 port 22.

[lsauser@almalinux ~]$

It could happen, so that we need information about how much time it took for the system to boot up:

[lsauser@almalinux ~]$ **systemd-analyze time**

Startup finished in 696ms (kernel) + 4.570s (initrd) + 11.344s (userspace) = 16.612s

[lsauser@almalinux ~]$

We can omit the **time** argument, and the output will be the same.

The same tool can show us different perspective on the system boot process:

[lsauser@almalinux ~]$ **systemd-analyze blame**

3.086s dev-mapper-almalinux\x2droot.device

2.242s lvm2-monitor.service

2.136s NetworkManager-wait-online.service

1.579s tuned.service

...

Alternatively, we can see the so-called critical chain with:

[lsauser@almalinux ~]$ **systemd-analyze critical-chain**

The time after the unit is active or started is printed after the "@" character.

The time the unit takes to start is printed after the "+" character.

multi-user.target @11.323s

└─tuned.service @9.743s +1.579s

└─network.target @9.735s

└─network.service @8.985s +743ms

└─NetworkManager-wait-online.service @6.833s +2.136s

└─NetworkManager.service @6.654s +171ms

└─network-pre.target @6.643s

└─firewalld.service @5.310s +1.327s

└─polkit.service @5.015s +246ms

└─basic.target @4.982s

└─sockets.target @4.981s

└─dbus.socket @4.979s

└─sysinit.target @4.974s

└─systemd-update-utmp.service @4.953s +20ms

└─auditd.service @4.623s +317ms

└─systemd-tmpfiles-setup.service @4.530s +81ms

└─rhel-import-state.service @4.247s +280ms

└─local-fs.target @4.245s

└─boot.mount @3.128s +1.036s

└─local-fs-pre.target @3.124s

└─lvm2-monitor.service @791ms +2.242s

└─lvm2-lvmetad.service @936ms

└─lvm2-lvmetad.socket @742ms

└─-.slice

[lsauser@almalinux ~]$

We can even make the tool draw a graphic for us, but we will need the **dot** tool, which is not installed by default with the **minimal profile**. First, we must check what package provides the tool:

[lsauser@almalinux ~]$ **dnf provides dot**

...

Then we can install the package:

[lsauser@almalinux ~]$ **sudo dnf install graphviz**

...

*The name of the package is the same for all other distributions*

And now we are ready to execute the graph generation command:

[lsauser@almalinux ~]$ **systemd-analyze dot sshd.service | dot -Tsvg > sshd.svg**

Color legend: black = Requires

dark blue = Requisite

dark grey = Wants

red = Conflicts

green = After

[lsauser@almalinux ~]$

We can create another image, for example one, showing service initialization:

[lsauser@almalinux ~]$ **systemd-analyze plot > systemd.svg**

Of course, if we don't have a desktop environment, it would be difficult to see what's inside the files, but we can always copy them via **ssh** on our host.

Let's explore one target file:

[lsauser@almalinux ~]$ **cat /usr/lib/systemd/system/multi-user.target**

# This file is part of systemd.

#

# systemd is free software; you can redistribute it and/or modify it

# under the terms of the GNU Lesser General Public License as published by

# the Free Software Foundation; either version 2.1 of the License, or

# (at your option) any later version.

[Unit]

Description=Multi-User System

Documentation=man:systemd.special(7)

Requires=basic.target

Conflicts=rescue.service rescue.target

After=basic.target rescue.service rescue.target

AllowIsolate=yes

[lsauser@almalinux ~]$

And one service file:

[lsauser@almalinux ~]$ **cat /usr/lib/systemd/system/sshd.service**

[Unit]

Description=OpenSSH server daemon

Documentation=man:sshd(8) man:sshd\_config(5)

After=network.target sshd-keygen.service

Wants=sshd-keygen.service

[Service]

Type=notify

EnvironmentFile=/etc/sysconfig/sshd

ExecStart=/usr/sbin/sshd -D $OPTIONS

ExecReload=/bin/kill -HUP $MAINPID

KillMode=process

Restart=on-failure

RestartSec=42s

[Install]

WantedBy=multi-user.target

[lsauser@almalinux ~]$

Alternatively, we can use this command to examine a unit file:

[lsauser@almalinux ~]$ **systemctl cat sshd.service**

# /usr/lib/systemd/system/sshd.service

[Unit]

Description=OpenSSH server daemon

Documentation=man:sshd(8) man:sshd\_config(5)

After=network.target sshd-keygen.service

Wants=sshd-keygen.service

...

Now that we are familiar with **systemd** and we have seen so many ways to reboot or power-off our system, let's take a closer look at the connection between all those utilities and **systemd**.

First, let's check where the executable artefact for the **reboot** command resides:

[lsauser@almalinux ~]$ **whereis reboot**

reboot: /usr/sbin/reboot /usr/share/man/man8/reboot.8.gz

Then, assuming that the others are stored in the same folder, let's check what they really are:

[lsauser@almalinux ~]$ **ls -l /usr/sbin/{reboot,shutdown,halt,poweroff}**

... /usr/sbin/halt -> ../bin/systemctl

... /usr/sbin/poweroff -> ../bin/systemctl

... /usr/sbin/reboot -> ../bin/systemctl

... /usr/sbin/shutdown -> ../bin/systemctl

So, it appears that they are symbolic links to **systemctl** and when executed are causing the system to be switched to a different target.

## Part 3

Let's execute the following command:

[lsauser@almalinux ~]$ **ping abv.bg**

PING abv.bg (194.153.145.104) 56(84) bytes of data.

64 bytes from abv.bg (194.153.145.104): icmp\_seq=1 ttl=63 time=1.24 ms

...

After 10 seconds or so, press **Ctrl+z** keys to stop the task. Now let's ask for a long listing of all processes in our session with a tree representation of relations between them:

[lsauser@almalinux ~]$ **ps lf**

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

0 1000 11648 11647 20 0 115476 2096 do\_wai Ss pts/0 0:00 -bash

4 1000 11701 11648 20 0 132640 1712 do\_sig T pts/0 0:00 \\_ ping abv.bg

0 1000 11709 11648 20 0 153232 1524 - R+ pts/0 0:00 \\_ ps lf

[lsauser@almalinux ~]$

We can experiment with different sets of arguments to **ps** in order to explore various result sets. For example, put a dash (**-**) in front of the **lf** option. Then use just the **l** (lowercase letter L) symbol. Finally, ask for all possible options and their meaning by executing the command with **--help** option and note that there are multiple ways to specify options and to achieve one and the same thing.

Now, we can start another tool which will help us to explore what's going on:

[lsauser@almalinux ~]$ **top**

Help information for different keys can be received by pressing the **h** key. We can exit by the **q** key, but this time we will close the tool by pressing the **Ctrl+z** key combination.

So far, we should have two stopped tasks. Let's see if this is correct:

[lsauser@almalinux ~]$ **jobs**

[1]- Stopped ping abv.bg

[2]+ Stopped top

[lsauser@almalinux ~]$ **jobs -l**

[1]- 11701 Подтиснат ping abv.bg

[2]+ 11713 Подтиснат (сигнал) top

[lsauser@almalinux ~]$

In order to continue the job #1 (the **ping** command), we must execute:

[lsauser@almalinux ~]$ **fg 1**

ping abv.bg

64 bytes from abv.bg (194.153.145.104): icmp\_seq=4 ttl=63 time=2.49 ms

64 bytes from abv.bg (194.153.145.104): icmp\_seq=5 ttl=63 time=1.15 ms

64 bytes from abv.bg (194.153.145.104): icmp\_seq=6 ttl=63 time=1.06 ms

^Z

[1]+ Stopped ping abv.bg

[lsauser@almalinux ~]$

If we press **Ctrl+z** again, we will stop it once more.

Now we can switch its execution to background with:

[lsauser@almalinux ~]$ **bg**

[1]+ ping abv.bg &

[lsauser@almalinux ~]$ 64 bytes from abv.bg (194.153.145.104): icmp\_seq=7 ttl=63 time=1.42 ms

64 bytes from abv.bg (194.153.145.104): icmp\_seq=8 ttl=63 time=1.58 ms

...

We can use just **bg** without any parameters as this job had the focus (indicated by **the plus sign**). Now because the command is running in background, we can't use **Ctrl+z** any more to stop it. We can type **fg 1** and hit **Enter** key, to move it to foreground, and then either **Ctrl+z** to stop it, or **Ctrl+c** to interrupt it.

Now let's bring the stopped top job back:

[lsauser@almalinux ~]$ **fg**

No arguments are needed, as this is the only stopped job.

Let's quit the **top** utility and try another one which offers greater level of interactivity

We may need to install an extra repository - to so called EPEL repository:

[lsauser@almalinux ~]$ **sudo dnf install epel-release**

...

*No need to add additional repository for the other distributions*

It is not installed by default, so we will install it first:

[lsauser@almalinux ~]$ **sudo dnf install -y htop**

*The name of the package is the same for all other distributions*

Once we have it, we can start it and explore it:

[lsauser@almalinux ~]$ **htop**

We can start any program in background mode by modifying its command line:

[lsauser@almalinux ~]$ **ping softuni.bg > /tmp/pingout.txt &**

[1] 11850

[lsauser@almalinux ~]$ **jobs**

[1]+ Running ping softuni.bg > /tmp/pingout.txt &

[lsauser@almalinux ~]$ **ps l**

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

0 1000 11648 11647 20 0 115664 2188 do\_wai Ss pts/0 0:00 -bash

4 1000 11850 11648 20 0 132640 1712 poll\_s S pts/0 0:00 ping softuni.bg

0 1000 11852 11648 20 0 153236 1528 - R+ pts/0 0:00 ps l

[lsauser@almalinux ~]$

As we know, we can use the tail command to follow information added to a file. We can use this to check if our **ping** command is working as expected:

[lsauser@almalinux ~]$ **tail -f /tmp/pingout.txt**

64 bytes from softuni.bg (217.174.159.195): icmp\_seq=125 ttl=63 time=1.29 ms

64 bytes from softuni.bg (217.174.159.195): icmp\_seq=126 ttl=63 time=1.99 ms

...

We can exit the follow mode with **Ctrl+c**

There are many ways to stop the **ping** command, but let's use this:

[lsauser@almalinux ~]$ **pkill ping**

[1]+ Terminated ping softuni.bg > /tmp/pingout.txt

[lsauser@almalinux ~]$

Instead of killing all processes with name **ping**, we can ask for their **PIDs** with:

[lsauser@almalinux ~]$ **pgrep sleep**

11866

...

Let's simulate similar situation, but we will use different method for process interruption:

[lsauser@almalinux ~]$ **ping softuni.bg > /tmp/pingout.txt &**

[1] 11858

[lsauser@almalinux ~]$ **kill -9 11858**

[lsauser@almalinux ~]$

[1]+ Безусловно прекъснат ping softuni.bg > /tmp/pingout.txt

[lsauser@almalinux ~]$

There are two additional and interesting utilities – **pstree** and **killall**, but they are not installed by default. Let's check which package provides them and then install it:

[lsauser@almalinux ~]$ **dnf provides pstree**

...

psmisc-22.20-15.el7.x86\_64 : Utilities for managing processes on your system

Хранилище : base

Съвпадащ от:

Име на файл : /usr/bin/pstree

[lsauser@almalinux ~]$ **sudo dnf install -y psmisc**

...

*The name of the package is the same for all other distributions*

Let's create another session and start two nested bash sessions in it:

[lsauser@almalinux ~]$ **bash**

[lsauser@almalinux ~]$ **sleep 10000&**

[1] **12009**

[lsauser@almalinux ~]$ **bash**

[lsauser@almalinux ~]$

Now, let's return to the first session and execute:

[lsauser@almalinux ~]$ **ps al**

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

0 1000 11648 11647 20 0 115664 2244 do\_wai Ss pts/0 0:00 -bash

4 0 11708 1 20 0 110092 860 n\_tty\_ Ss+ tty1 0:00 /sbin/agetty --noclear tty1 linux

0 1000 11979 **11978** 20 0 115440 2008 do\_wai Ss pts/1 0:00 -bash

0 1000 11996 11979 20 0 115440 2048 do\_wai S pts/1 0:00 bash

0 1000 **12009 11996 20 0 107952 616 hrtime S pts/1 0:00 sleep 10000**

0 1000 12010 11996 20 0 115440 2016 n\_tty\_ S+ pts/1 0:00 bash

0 1000 12029 11648 20 0 153236 1528 - R+ pts/0 0:00 ps al

[lsauser@almalinux ~]$ **pstree 11978**

sshd───bash───bash─┬─bash

└─sleep

Now, if we exit all bash sessions and close the second **ssh** session, then return to the first and execute:

[lsauser@almalinux ~]$ **ps alx | grep sleep**

0 1000 **12009** 1 20 0 107952 616 hrtime S ? 0:00 sleep 10000

0 1000 12075 11648 20 0 112712 1000 - R+ pts/0 0:00 grep --color=auto sleep

[lsauser@almalinux ~]$

We will notice that the parent process **ID** of our sleep command has changed to **1**, which is the **systemd**'s main process or the process of all processes.

In fact, we can easily check what is running under **PID 1** with:

[lsauser@almalinux ~]$ **ps -fp 1**

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 11:53 ? 00:00:06 /usr/lib/systemd/systemd --switched-root --system --deserialize 22

[lsauser@almalinux ~]$

Let's explore a bit the priority change tools. First, let's start a sleep command and check its default niceness level:

[lsauser@almalinux ~]$ **sleep 20000&**

[1] **12080**

[lsauser@almalinux ~]$ **ps l**

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

0 1000 11648 11647 20 0 115664 2244 do\_wai Ss pts/0 0:00 -bash

0 1000 **12080 11648 20 0** 107952 620 hrtime S pts/0 0:00 sleep 20000

0 1000 12088 11648 20 0 153236 1528 - R+ pts/0 0:00 ps l

[lsauser@almalinux ~]$

As we can see the default niceness value is 0. Let's try to make it nicer:

[lsauser@almalinux ~]$ **renice 5 12080**

12080 (process ID) old priority 0, new priority 5

[lsauser@almalinux ~]$ **ps l | grep sleep**

0 1000 12080 11648 25 5 107952 620 hrtime SN pts/0 0:00 sleep 20000

0 1000 12091 11648 20 0 112712 996 pipe\_w S+ pts/0 0:00 grep --color=auto sleep

[lsauser@almalinux ~]$

Now, let's increase again the level, and then try to lower it:

[lsauser@almalinux ~]$ **renice 10 12080**

12080 (process ID) old priority 5, new priority 10

[lsauser@almalinux ~]$ **renice 0 12080**

renice: failed to set priority for 12080 (process ID): Отказан достъп

[lsauser@almalinux ~]$ **sudo renice 0 12080**

12080 (process ID) old priority 10, new priority 0

[lsauser@almalinux ~]$ **ps l | grep sleep**

0 1000 12080 11648 20 0 107952 620 hrtime S pts/0 0:00 sleep 20000

0 1000 12100 11648 20 0 112712 1000 pipe\_w S+ pts/0 0:00 grep --color=auto sleep

[lsauser@almalinux ~]$

We can see that even though we are the owner of the process, we can neither restore its niceness level nor decrease it. In order to do this, we must use **sudo**.

If want to start a process with custom niceness level, we can use the nice command.

Monitoring of the system includes also resource availability or usage tracking.

For example, we can check the utilization of the memory with:

[lsauser@almalinux ~]$ **free**

total used free shared buff/cache available

Mem: 1014972 100984 229964 6824 684024 728992

Swap: 839676 0 839676

[lsauser@almalinux ~]$ **free -h**

total used free shared buff/cache available

Mem: 991M 98M 224M 6,7M 667M 712M

Swap: 819M 0B 819M

[lsauser@almalinux ~]$

In a similar fashion there is command that will return the free space on hard disk drives:

[lsauser@almalinux ~]$ **df**

Файлова с-ма 1K-блокове Заети Свободни Изп% Монтирана на

/dev/mapper/almalinux-root 6486016 1450992 5035024 23% /

devtmpfs 495588 0 495588 0% /dev

tmpfs 507484 0 507484 0% /dev/shm

tmpfs 507484 6824 500660 2% /run

tmpfs 507484 0 507484 0% /sys/fs/cgroup

/dev/sda1 1038336 134848 903488 13% /boot

tmpfs 101500 0 101500 0% /run/user/1000

[lsauser@almalinux ~]$ **df -h**

Файлова с-ма Размер Заети Свобод Изп% Монтирана на

/dev/mapper/almalinux-root 6,2G 1,4G 4,9G 23% /

devtmpfs 484M 0 484M 0% /dev

tmpfs 496M 0 496M 0% /dev/shm

tmpfs 496M 6,7M 489M 2% /run

tmpfs 496M 0 496M 0% /sys/fs/cgroup

/dev/sda1 1014M 132M 883M 13% /boot

tmpfs 100M 0 100M 0% /run/user/1000

[lsauser@almalinux ~]$

In a similar way, we can monitor the estimated file space usage. For example, we can check the **/etc** folder in human readable format:

[lsauser@almalinux ~]$ **sudo du -h -s /etc**

31M /etc

We can modify the command to show how big each element in **/etc** is and sort them by size:

[lsauser@almalinux ~]$ **sudo du -h -s /etc/\* | sort -h**

0 /etc/alternatives

0 /etc/binfmt.d

...

1008K /etc/pki

7,6M /etc/udev

19M /etc/selinux

[lsauser@almalinux ~]$

There are also other utilities. For example, there are many utilities that end on **\*top** or on **\*mon**.

Let's check for three of them:

[lsauser@almalinux ~]$ **dnf provides atop bmon nmon**

...

atop-2.3.0-8.el7.x86\_64 : An advanced interactive monitor to view the load on system and process level

Хранилище : epel

bmon-3.6-1.el7.x86\_64 : Bandwidth monitor and rate estimator

Хранилище : epel

nmon-16g-3.el7.x86\_64 : Nigel's performance Monitor for Linux

Хранилище : epel

[lsauser@almalinux ~]$ **sudo dnf install -y nmon atop bmon**

...

*No need to add additional repositories to any of the other distributions. The names of the packages are the same*

We can start each one of them and check what it returns.

There are even more utilities for monitoring, few (**iostat**, **pidstat** & **sar**) of them are in one package called **sysstat**. You may need to install the package.

*No need to add additional repositories to any of the other distributions. The names of the packages are the same*

You can play with them. Please note that some utilities show the information in real time, while for others there is a need to pass some time in order to have enough data gathered.

For example, we can ask for **CPU** utilization using the **sar** tool, with:

[lsauser@almalinux ~]$ **sar -u**

Linux 3.10.0-957.el7.x86\_64 (almalinux) 6.03.2019 \_x86\_64\_ (1 CPU)

16,20,01 CPU %user %nice %system %iowait %steal %idle

16,30,01 all 0,03 0,00 0,14 0,01 0,00 99,83

Average: all 0,03 0,00 0,14 0,01 0,00 99,83

[lsauser@almalinux ~]$

Use man to explore how each tool is used. Experiment with the others as well.

Let’s try the **watch** command. We can use it to monitor the progress of a process. It is typically used with other commands that do not offer the option to track the execution progress over time.

For example, we may want to see what is happening with our running processes over time. In fact, we will see a snapshot of the situation every N seconds.

First, let’s prepare the playground.

We will simulate a long running process by executing the following command:

[lsauser@almalinux ~]$ **for i in $(seq 1 100); do echo "Progress: $i" > /tmp/a.txt; sleep 3; done &**

The above will run in the background for about 300 seconds and every 3 seconds will increase the number stored in the file.

Now, can execute the following to monitor the progress of the above:

[lsauser@almalinux ~]$ **watch -n 5 cat /tmp/a.txt**

With the above, we will check every 5 seconds the contents of the file and thus be informed how the running process is progressing.

Once we are done with the observation, we can press **Ctrl + C** to stop the **watch** command.

Now, let’s try the **screen** and **tmux** commands.

*Please note that they may not be installed on your system. Use the package manager to install them. The packages have the same name as the corresponding command. For* ***Red Hat****-based distributions, you will need the* ***EPEL*** *repository (for the screen package).*

We typically use either of these commands to start long running processes like installation of software, creating a backup, etc. We can use them in our everyday tasks as well. The idea is to ensure that our tasks will continue their execution even if our session gets closed. In a situation like this, both **screen** and **tmux** will allow us to continue with or reconnect to our existing session.

We will see the same simplified scenario with both commands. Let’s start with the **screen** command:

[lsauser@almalinux ~]$ **screen**

*On distributions like* ***Debian****,* ***openSUSE****, and* ***Ubuntu****, you will be presented with a welcome screen. Just hit* ***Space*** *or* ***Return*** *to continue.*

Now, we are in. Let’s use the **ping** command to simulate long running process that we want to continue even if our session gets closed:

[lsauser@almalinux ~]$ **ping localhost**

Let’s close the session (without interrupting the command) by pressing **Ctrl + a** and then **d**. This will detach the session.

Close the SSH session to the VM:

[lsauser@almalinux ~]$ **exit**

Now, open a new SSH session.

Once we are in, in order to see what other sessions exist, we can execute:

[lsauser@almalinux ~]$ **screen -ls**

Then, in order to connect to the existing session, we must execute:

[lsauser@almalinux ~]$ **screen -r <session-number>**

Now, press **Ctrl + c** to interrupt **ping** execution and close the session with:

[lsauser@almalinux ~]$ **exit**

If we check again for other **screen** sessions with:

[lsauser@almalinux ~]$ **screen -ls**

We will see that there aren’t any.

Should you want to, you can repeat the same with **tmux**. In case you do, then follow the next steps.

Let’s continue with the **tmux** command:

[lsauser@almalinux ~]$ **tmux**

Now, we are in. Let’s use the **ping** command to simulate long running process that we want to continue even if our session gets closed:

[lsauser@almalinux ~]$ **ping localhost**

Let’s close the session (without interrupting the command) by pressing **Ctrl + b** and then **d**. This will detach the session.

Close the SSH session to the VM:

[lsauser@almalinux ~]$ **exit**

Now, open a new SSH session.

Once we are in, in order to see what other sessions exist, we can execute:

[lsauser@almalinux ~]$ **tmux ls**

Then, in order to connect to the existing session, we must execute:

[lsauser@almalinux ~]$ **tmux attach-session -t <session-number>**

Now, press **Ctrl + c** to interrupt **ping** execution and close the session with:

[lsauser@almalinux ~]$ **exit**

If we check again for other **tmux** sessions with:

[lsauser@almalinux ~]$ **tmux ls**

We will see that there aren’t any.

Of course, there is much more to explore for both (**screen** and **tmux**) commands and every other command that is part of this module, but this is the bare minimum that we should know. Feel free to go further/deeper.