System and Network Administration - Lab 9 - Systemd

Jaffar Totanji - j.totanji@innopolis.university

Questions to answer:

1. For that, we can use:

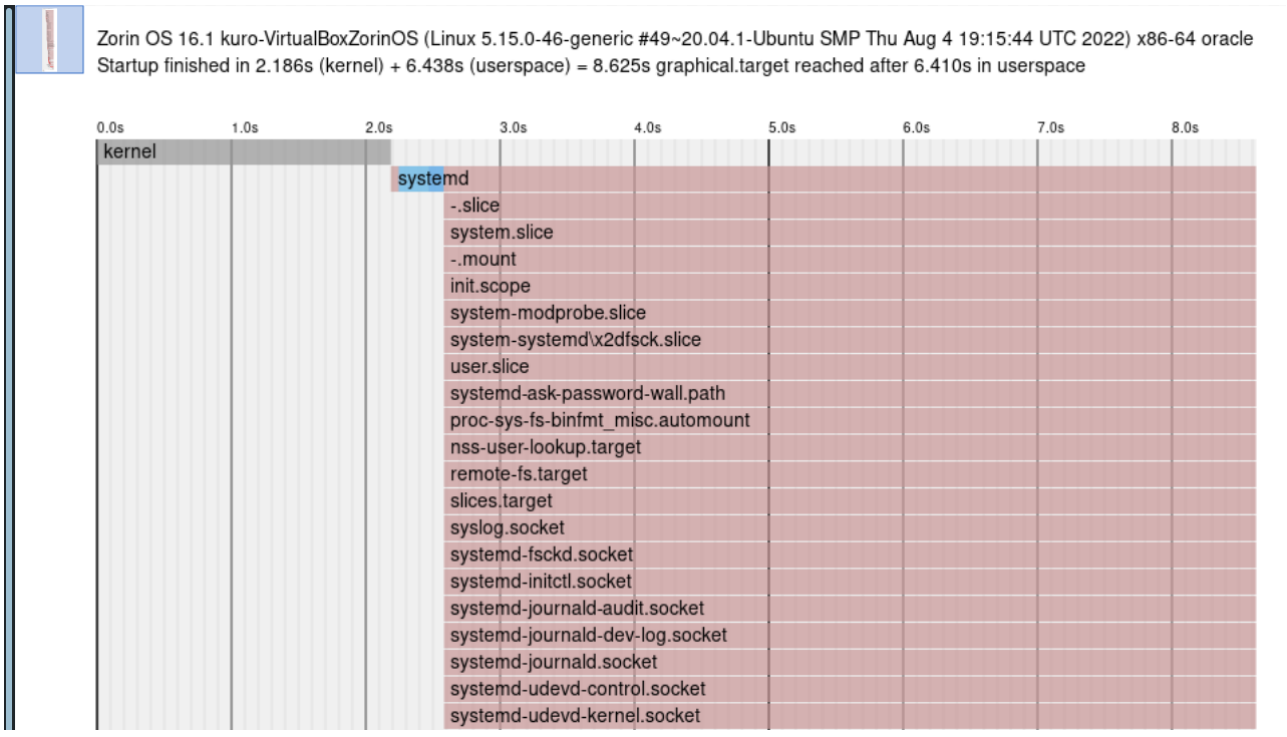
   ```
   systemd-analyze
   ```

   ```
   kuro@kuro-VirtualBoxZorinOS:~$ systemd-analyze
   Startup finished in 2.186s (kernel) + 6.438s (userspace) = 8.625s
   graphical.target reached after 6.410s in userspace
   kuro@kuro-VirtualBoxZorinOS:~$
   ```

   In order to view the services that have been started and how long it took each of them to initialize, we can use `systemd-analyze` again, but this time with its charting utility `plot`. The resulting chart is huge, so I have only included the top part of it here:

   ```
   systemd-analyze plot
   ```

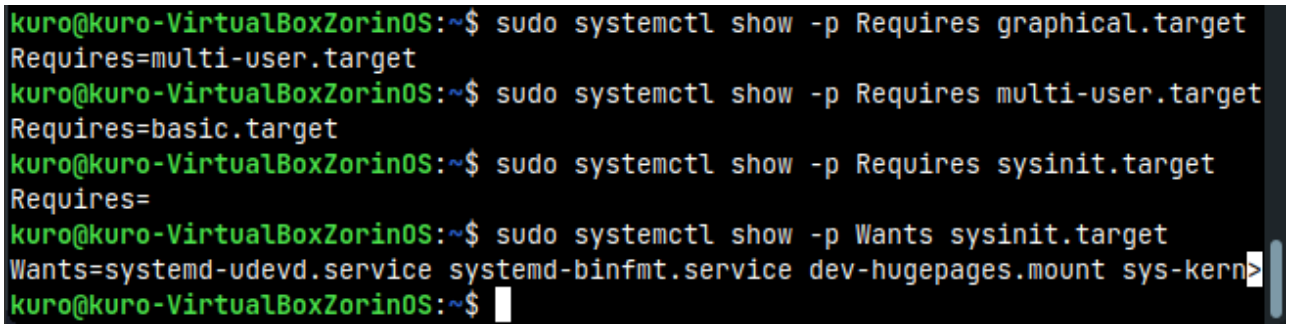
   ```
   kuro@kuro-VirtualBoxZorinOS:~$ systemd-analyze plot > ./boot.svg
   kuro@kuro-VirtualBoxZorinOS:~$
   ```

2. We can trace back the `Requires` variable of every service using `systemctl`. Our starting point looking like this:

```
systemctl show -p Requires graphical.target
```

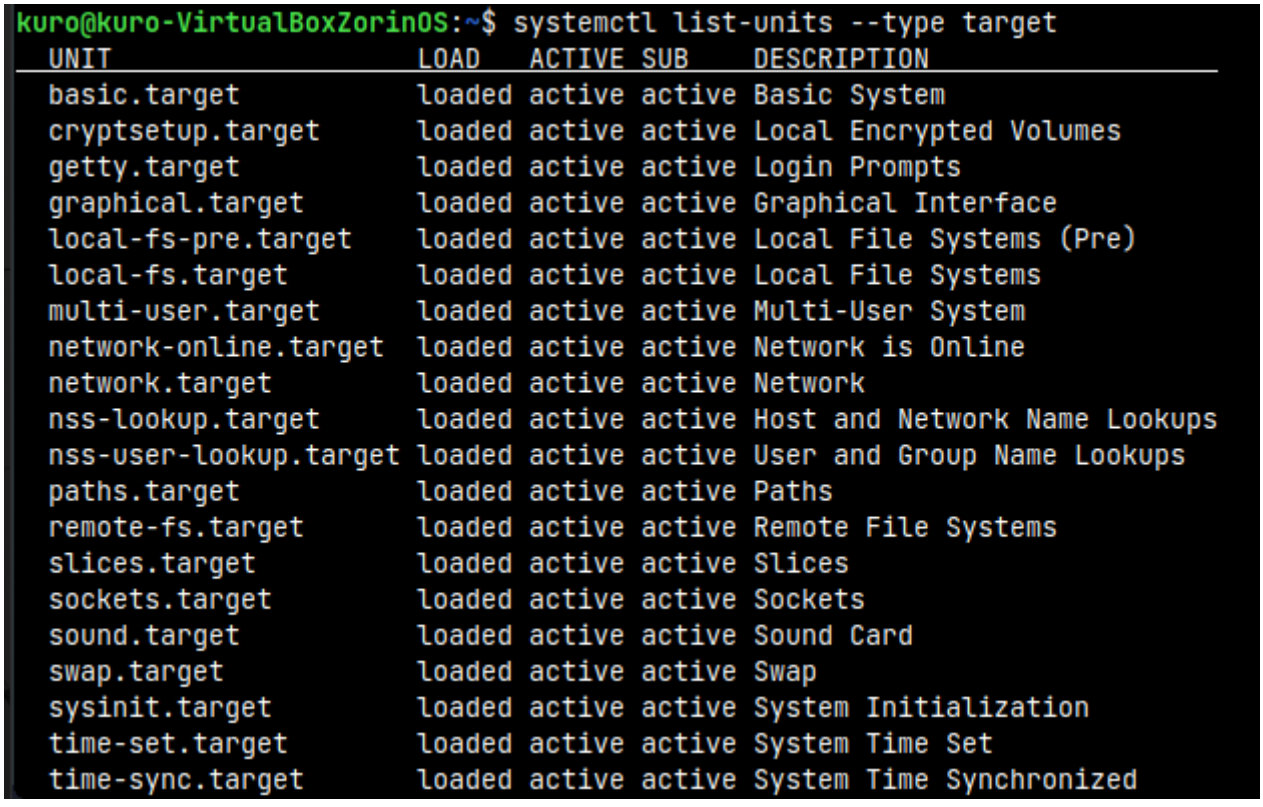
and our chain of services looks like this:



We can then use

```
systemctl list-units --type target
```

to view some info about these services as well as others:



- `graphical.target`: `multi-user.target` with GUI.
- `multi-user.target`: All services running, but command-line interface (CLI) only.
- `sysinit.target`: System Initialization.

Let's first start with the meaning of "Wants" and "Requires" from the `man` page:

- `Requires`: Configures requirement dependencies on other units. If this unit gets activated, the units listed here will be activated as well. If one of the other units gets deactivated or its activation fails, this unit will be deactivated.

- `Wants`:A weaker version of `Requires`. Units listed in this option will be started if the configuring unit is. However, if the listed units fail to start or cannot be added to the transaction, this has no impact on the validity of the transaction as a whole. This is the recommended way to hook start-up of one unit to the start-up of another unit.

Focusing on the last 2 lines of the description of `Wants`, `sysinit.target` basically initializes our system aka starts it up, and in order for it to do that, it needs to start up lots of other services for the system to be considered running, and if one of them fails, the transaction is still valid as a whole.

3. The first step is to create a script that starts web server on a certain port and gives us the required statistical info. I did that in the form of 2 scripts, on that prints out the statistics `script_1.sh` and another than runs the previous script on a web server `script.sh`:

script.sh

```
#!/bin/bash

printf "System Uptime:\n"
uptime -p
echo "------------------------------------------------------------"

printf "Inode Usage:\n"
df -i 2> /dev/null
echo "--------------------|---------------------------------------"

printf "Usage Statistics:\n"
memory_usage=$(free -m | awk 'NR==2{printf "Memory Usage: %s/%sMB (%.2f%%)\n", $3,$2,$3*100/$2 }')
disk_usage=$(df -h 2> /dev/null | awk '$NF=="/"{printf "Disk Usage: %d/%dGB (%s)\n", $3,$2,$5}')
cpu_usage=$(top -bn1 | grep load | awk '{printf "CPU Usage: %.2f\n", $(NF-2)}')
cur_date=`date +%b_%d_%Y_%H_%M_%S`

echo "$cur_date: $memory_usage || $disk_usage || $cpu_usage"
echo "------------------------------------------------------------"

printf "Last 15 lines of /val/log/syslog:\n"
tail -15 /var/log/syslog
echo "------------------------------------------------------------"
```

script_1.sh

```
#!/bin/bash

while true;
        do echo -e "HTTP/1.1 200 OK\n\n$(source script_1.sh)" \
        | nc -l -k -p 8080 -q 1;
done
```

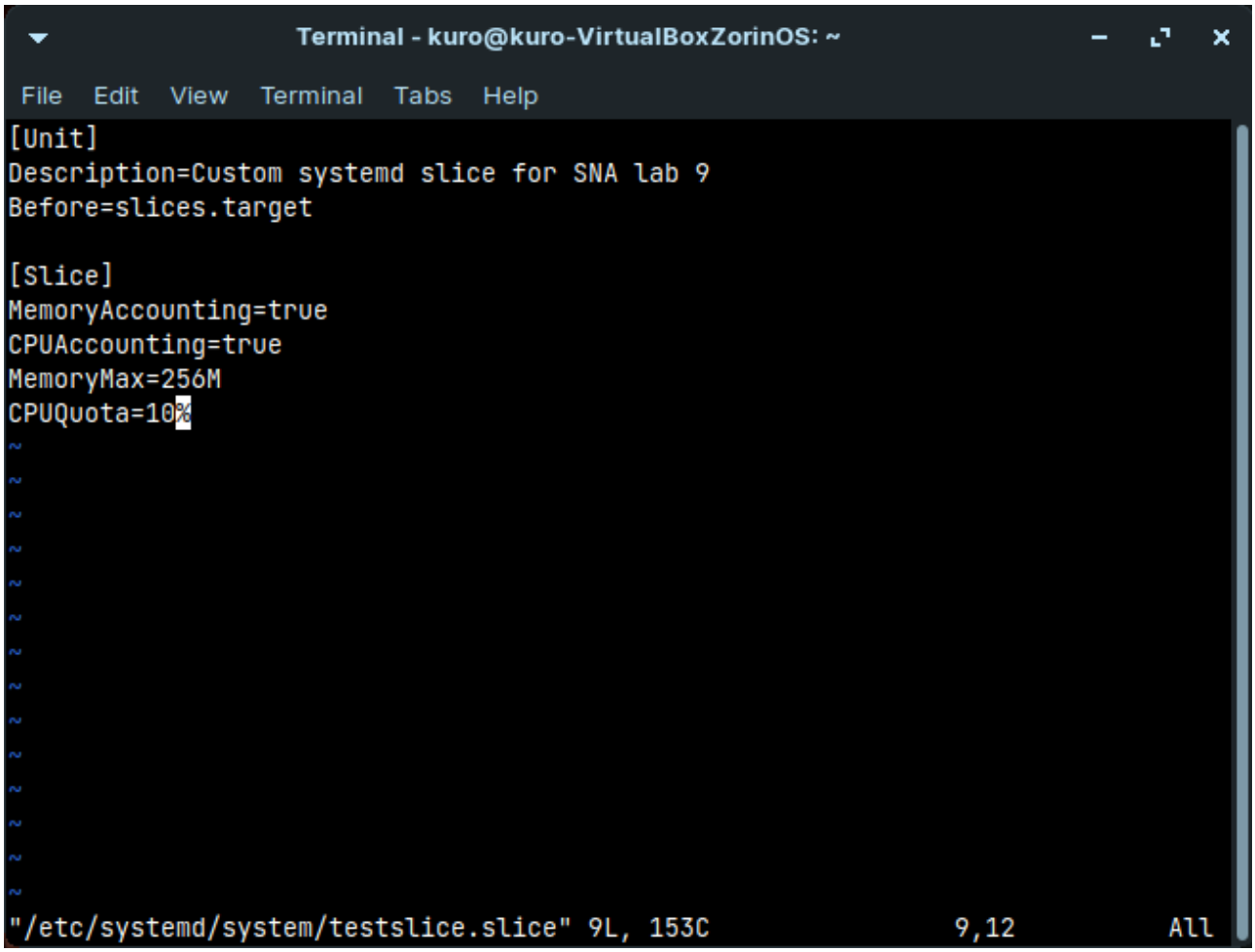
Then, we need to create a systemd service to accomodate the script at run it at start up and a slice to control resource usage. That is done exactly as described in the lab:

```
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl daemon-reload
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl restart shellscript.service
kuro@kuro-VirtualBoxZorinOS:~$ sudo vim /lib/systemd/system/shellscript.service
kuro@kuro-VirtualBoxZorinOS:~$ sudo vim /etc/systemd/system/testslice.slice
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl daemon-reload
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl restart shellscript.service
kuro@kuro-VirtualBoxZorinOS:~$
```

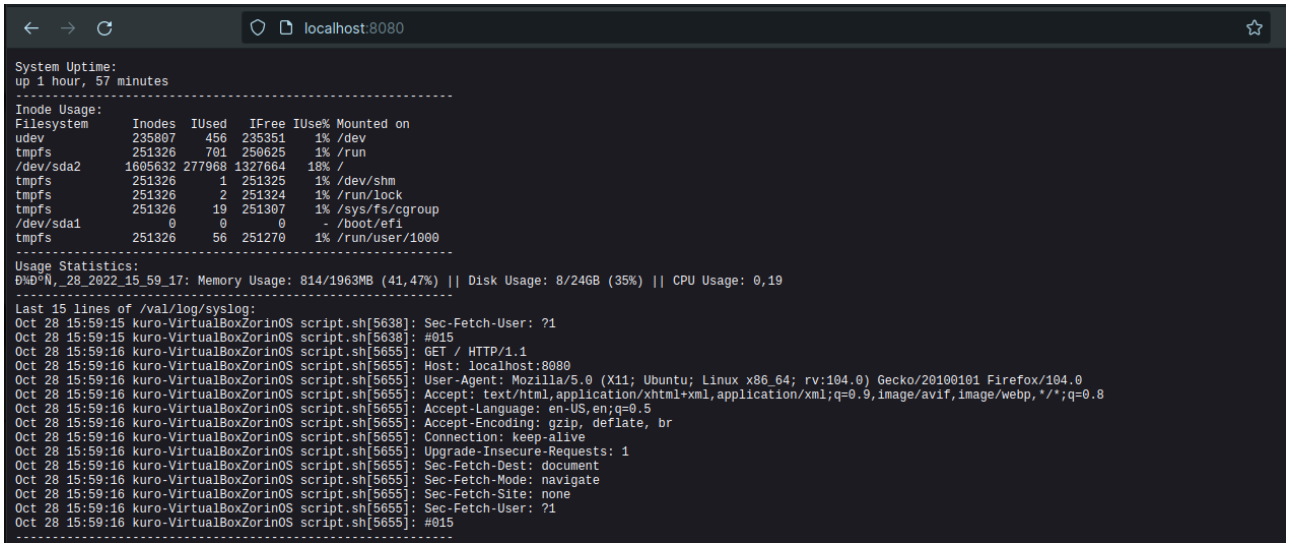
Terminal - kuro@kuro-VirtualBoxZorinOS: ~

File Edit View Terminal Tabs Help

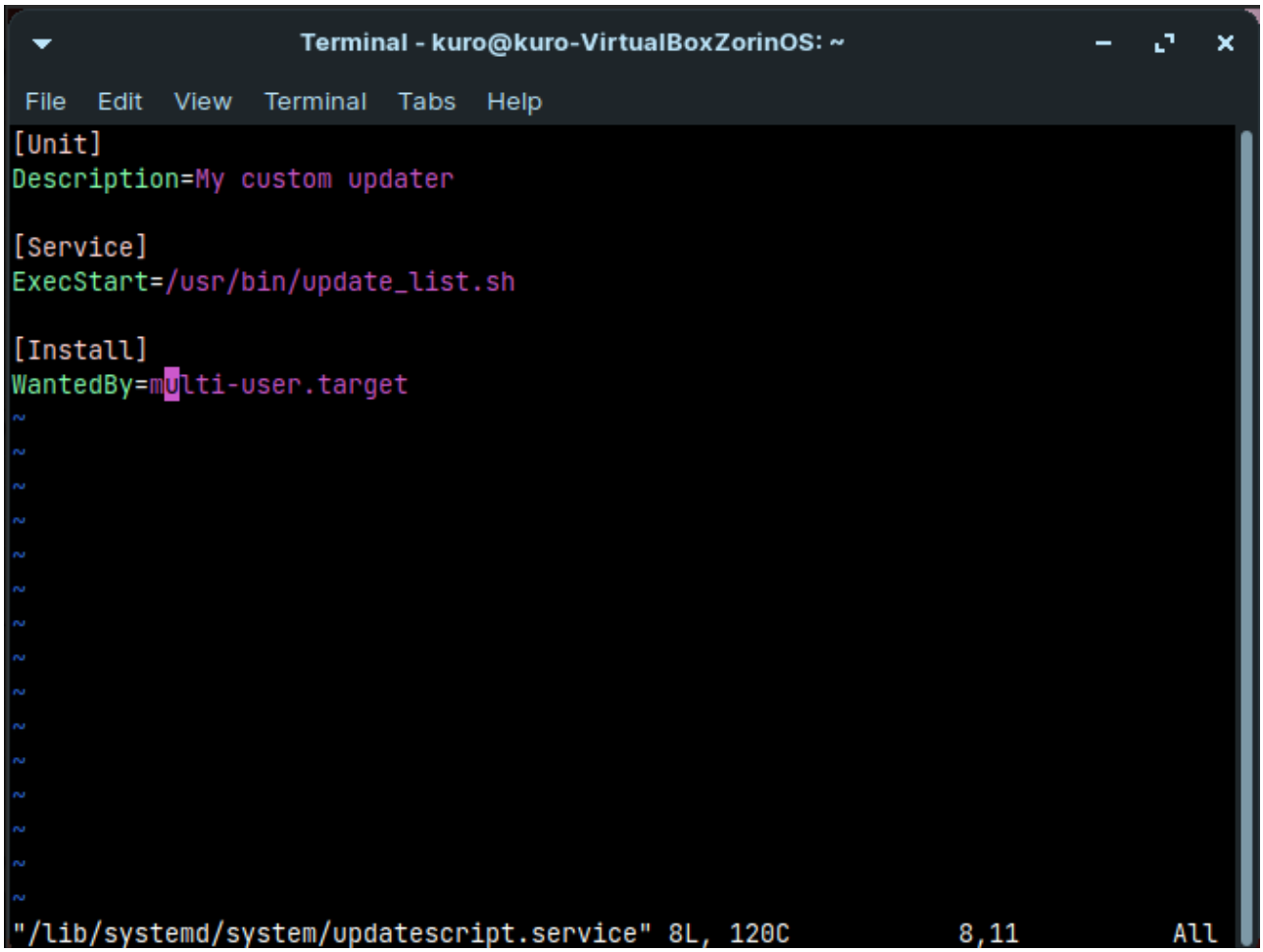
```
[Unit]
Description=Lab 9 SNA

[Service]
ExecStart=/usr/bin/script.sh
Slice=testslice.slice

[Install]
WantedBy=multi-user.target
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/lib/systemd/system/shellscript.service" 9L, 129C          6,21          All
```



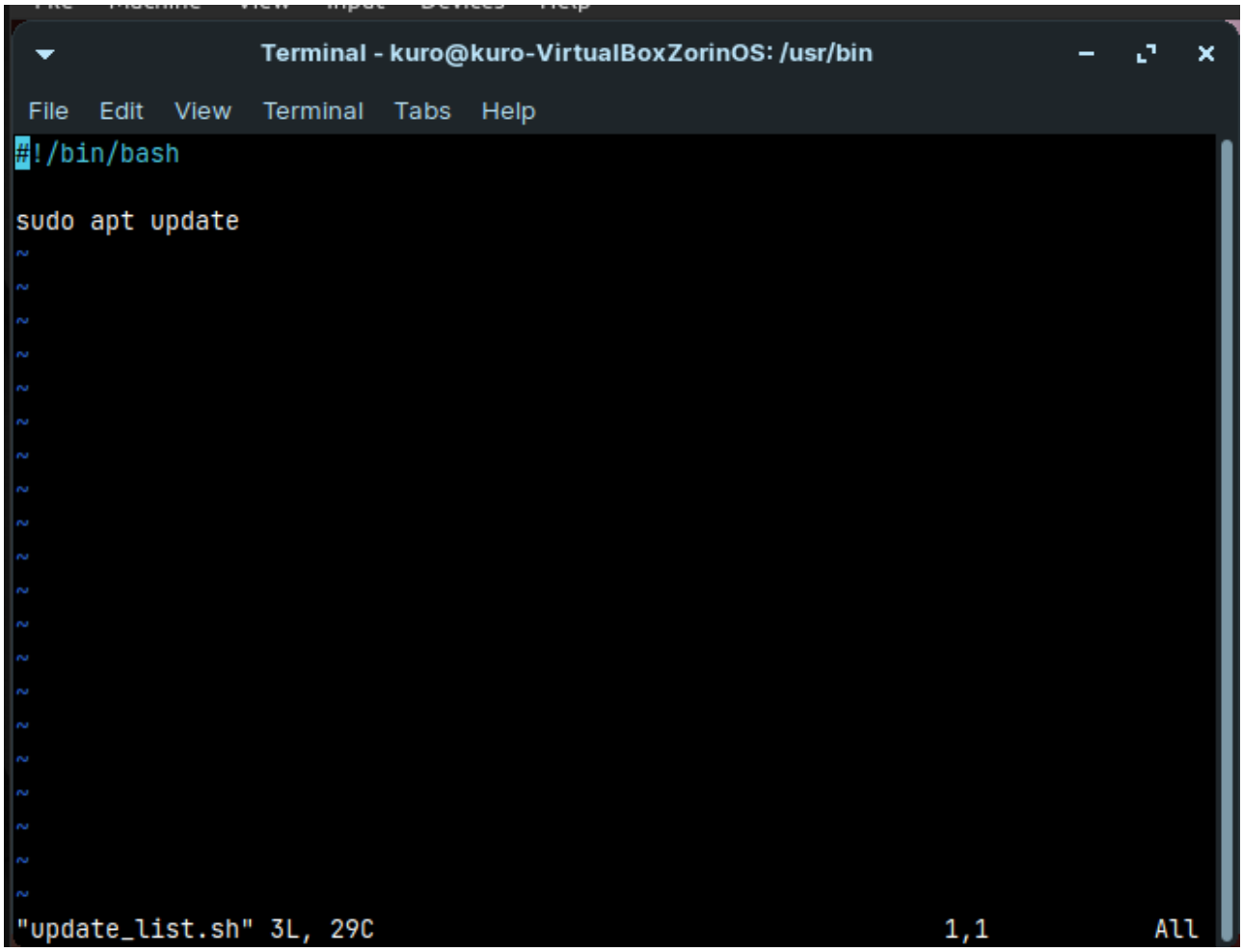
Finally, the service in action:



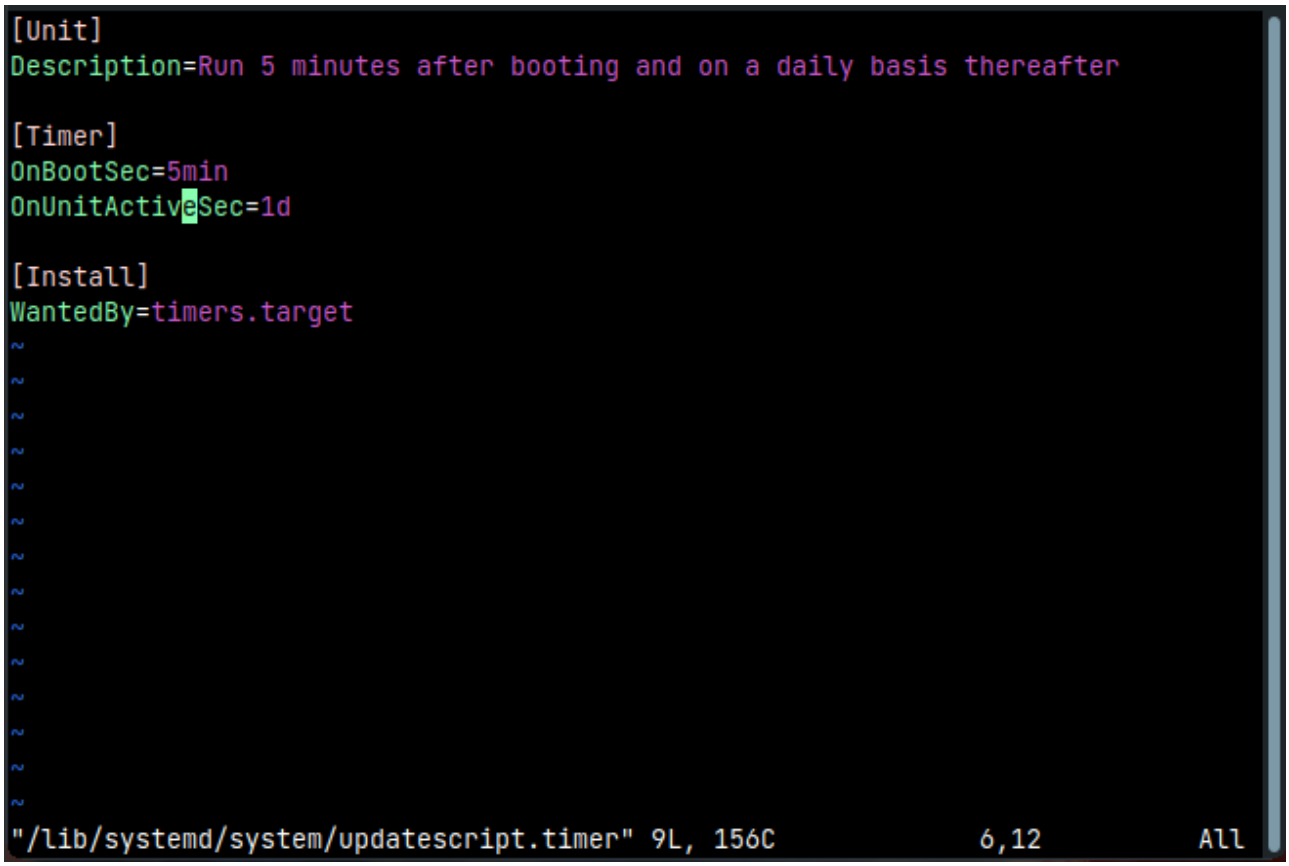
4. For that, we first need to create a service that updates the package source list:



`update_list` script simply performs the update:



To then run the service periodically, we need to create a corresponding timer for the service that will trigger after boot up and on a daily thereafter:



Finally, we need to do `systemctl daemon-reload` to make `systemd` aware of our new files, and then we need to enable and start the timer which will trigger our service as scheduled:

```
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl daemon-reload
[sudo] password for kuro:
Sorry, try again.
[sudo] password for kuro:
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl enable updatescript.timer
kuro@kuro-VirtualBoxZorinOS:~$ sudo systemctl start updatescript.timer
kuro@kuro-VirtualBoxZorinOS:~$ systemctl status updatescript.timer updatescript.service
● updatescript.timer - Run 5 minutes after booting and on a daily basis thereaf>
     Loaded: loaded (/lib/systemd/system/updatescript.timer; enabled; vendor pr>
     Active: active (waiting) since Fri 2022-10-28 18:03:25 MSK; 5min ago
    Trigger: Sat 2022-10-29 18:03:26 MSK; 23h left
   Triggers: ● updatescript.service

окт 28 18:03:25 kuro-VirtualBoxZorinOS systemd[1]: Started Run 5 minutes after >

● updatescript.service - My custom updater
     Loaded: loaded (/lib/systemd/system/updatescript.service; enabled; vendor >
     Active: inactive (dead) since Fri 2022-10-28 18:03:58 MSK; 4min 47s ago
TriggeredBy: ● updatescript.timer
   Main PID: 3831 (code=exited, status=0/SUCCESS)

окт 28 18:03:57 kuro-VirtualBoxZorinOS update_list.sh[3833]: Reading package li>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: Building dependenc>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: Reading state info>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: 139 packages can b>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: W: Failed to fetch>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: W: Failed to fetch>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: W: Failed to fetch>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: W: Failed to fetch>
окт 28 18:03:58 kuro-VirtualBoxZorinOS update_list.sh[3833]: W: Some index file>
kuro@kuro-VirtualBoxZorinOS:~$
```

End of Exercises