Lab 4 BSY PROC-2

Solution by guentgia & lichtnoa.

Task 1 – Understand Cgroups Version One

Subtask 1.1 – Default System Configuration

a) Analyze the default cgroup configuration of Ubuntu. Which subsystems are supported by the Ubuntu kernel? Explain the output.

How to list the supported subsystems:

```
$ cat /proc/cgroups

#subsys_name   hierarchy  num_cgroups  enabled
cpuset         4          1            1
cpu            3          92           1
cpuacct        3          92           1
blkio          7          92           1
memory         11         153          1
devices        6          92           1
freezer        8          2            1
net_cls        9          1            1
perf_event     5          1            1
net_prio       9          1            1
hugetlb        10         1            1
pids           12         98           1
rdma           2          1            1
```

Columns:

- subsys_name: The name of the cgroup subsystem
- hierarchy: The hierarchy ID of the subsystem
- num_cgroups: The number of cgroups currently in use for the subsystem
- enabled: Whether or not the subsystem is enabled in the kernel (1 = enabled, 0 = disabled)

Some subsystem descriptions:

- cpu: This subsystem controls CPU usage and can be used to limit the amount of CPU time that a cgroup or process can consume.
- cpuset: This subsystem assigns CPUs and memory nodes to cgroups and processes, which can be useful in a multi-processor system.
- memory: This subsystem manages memory usage and can be used to limit the amount of memory that a cgroup or process can consume.
- blkio: This subsystem controls input/output (I/O) access to block devices, such as hard disk drives, and can be used to limit the amount of I/O that a cgroup or process can perform.
- devices: This subsystem controls access to devices, such as USB drives, and can be used to restrict device access for specific cgroups or processes.
- freezer: This subsystem suspends or resumes processes in a cgroup, which can be useful for pausing or resuming system services or applications.

Alternatively one can list the files in cgroup:

```
$ cd /sys/fs/cgroup/
$ ls -lha

total 0
drwxr-xr-x 15 root root 380 Mar 30 18:48 .
drwxr-xr-x  9 root root   0 Mar 30 18:48 ..
dr-xr-xr-x  5 root root   0 Mar 30 18:48 blkio
lrwxrwxrwx  1 root root  11 Mar 30 18:48 cpu -> cpu,cpuacct
dr-xr-xr-x  5 root root   0 Mar 30 18:48 cpu,cpuacct
lrwxrwxrwx  1 root root  11 Mar 30 18:48 cpuacct -> cpu,cpuacct
dr-xr-xr-x  2 root root   0 Mar 30 18:48 cpuset
dr-xr-xr-x  5 root root   0 Mar 30 18:48 devices
dr-xr-xr-x  3 root root   0 Mar 30 18:48 freezer
dr-xr-xr-x  2 root root   0 Mar 30 18:48 hugetlb
dr-xr-xr-x  5 root root   0 Mar 30 18:48 memory
lrwxrwxrwx  1 root root  16 Mar 30 18:48 net_cls -> net_cls,net_prio
dr-xr-xr-x  2 root root   0 Mar 30 18:48 net_cls,net_prio
lrwxrwxrwx  1 root root  16 Mar 30 18:48 net_prio -> net_cls,net_prio
dr-xr-xr-x  2 root root   0 Mar 30 18:48 perf_event
dr-xr-xr-x  5 root root   0 Mar 30 18:48 pids
dr-xr-xr-x  2 root root   0 Mar 30 18:48 rdma
dr-xr-xr-x  5 root root   0 Mar 30 18:48 systemd
dr-xr-xr-x  5 root root   0 Mar 30 18:48 unified
```

b) Which hierarchies are provided by default? Which subsystems are configured at which hierarchy?

The file */proc/self/mountinfo* shows the mounted cgroup hierarchies and their associated subsystems:

```
$ cat /proc/self/mountinfo | grep cgroup
35 26 0:29 / /sys/fs/cgroup ro,nosuid,nodev,noexec shared:9 - tmpfs tmpfs
ro,mode=755
36 35 0:30 / /sys/fs/cgroup/unified rw,nosuid,nodev,noexec,relatime
shared:10 - cgroup2 cgroup2 rw,nsdelegate
37 35 0:31 / /sys/fs/cgroup/systemd rw,nosuid,nodev,noexec,relatime
shared:11 - cgroup cgroup rw,xattr,name=systemd
40 35 0:34 / /sys/fs/cgroup/rdma rw,nosuid,nodev,noexec,relatime shared:15 -
cgroup cgroup rw,rdma
41 35 0:35 / /sys/fs/cgroup/cpu,cpuacct rw,nosuid,nodev,noexec,relatime
shared:16 - cgroup cgroup rw,cpu,cpuacct
42 35 0:36 / /sys/fs/cgroup/cpuset rw,nosuid,nodev,noexec,relatime shared:17
- cgroup cgroup rw,cpuset
43 35 0:37 / /sys/fs/cgroup/perf_event rw,nosuid,nodev,noexec,relatime
shared:18 - cgroup cgroup rw,perf_event
44 35 0:38 / /sys/fs/cgroup/devices rw,nosuid,nodev,noexec,relatime
shared:19 - cgroup cgroup rw,devices
45 35 0:39 / /sys/fs/cgroup/blkio rw,nosuid,nodev,noexec,relatime shared:20
- cgroup cgroup rw,blkio
46 35 0:40 / /sys/fs/cgroup/freezer rw,nosuid,nodev,noexec,relatime
shared:21 - cgroup cgroup rw,freezer
47 35 0:41 / /sys/fs/cgroup/net_cls,net_prio rw,nosuid,nodev,noexec,relatime
shared:22 - cgroup cgroup rw,net_cls,net_prio
48 35 0:42 / /sys/fs/cgroup/hugetlb rw,nosuid,nodev,noexec,relatime
shared:23 - cgroup cgroup rw,hugetlb
49 35 0:43 / /sys/fs/cgroup/memory rw,nosuid,nodev,noexec,relatime shared:24
- cgroup cgroup rw,memory
50 35 0:44 / /sys/fs/cgroup/pids rw,nosuid,nodev,noexec,relatime shared:25 -
cgroup cgroup rw,pids
```

Each cgroup hierarchy can support multiple subsystems. For example, the cpu hierarchy can support both cpu and cpuacct subsystems.

c) Navigate to the default CPU hierarchy and check how many cgroups are present. What may be the purpose of these cgroups? Who created them? You may find the command systemd-cgls useful.

```
$ cd /sys/fs/cgroup/cpu
$ systemd-cgls cpu

Controller cpu; Control group /:
├─user.slice
│ ├─14620 sshd: ubuntu [priv]
│ ├─14623 /lib/systemd/systemd --user
│ ├─14624 (sd-pam)
│ ├─14741 sshd: ubuntu@pts/0
│ ├─14742 -bash
```

```
  │ ├─15242 systemd-cgls cpu
  │ └─15243 pager
  ├─init.scope
  │ └─1 /sbin/init
  └─system.slice
    ├─irqbalance.service
    │ └─736 /usr/sbin/irqbalance --foreground
    ├─systemd-networkd.service
    │ └─611 /lib/systemd/systemd-networkd
    ├─systemd-udevd.service
    │ └─403 /lib/systemd/systemd-udevd
    ├─cron.service
    │ └─722 /usr/sbin/cron -f
    ├─system-serial\x2dgetty.slice
    │ └─763 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0 vt220
    ├─polkit.service
    │ └─800 /usr/lib/policykit-1/polkitd --no-debug
    ├─networkd-dispatcher.service
    │ └─738 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-
  triggers
    ├─multipathd.service
    │ └─503 /sbin/multipathd -d -s
    ├─accounts-daemon.service
    │ └─719 /usr/lib/accountsservice/accounts-daemon
    ├─systemd-journald.service
    │ └─367 /lib/systemd/systemd-journald
    ├─atd.service
    │ └─749 /usr/sbin/atd -f
    ├─ssh.service
    │ └─854 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
    ├─snapd.service
    │ └─13758 /usr/lib/snapd/snapd
    ├─rsyslog.service
    │ └─744 /usr/sbin/rsyslogd -n -iNONE
    ├─systemd-resolved.service
    │ └─628 /lib/systemd/systemd-resolved
    ├─udisks2.service
    │ └─748 /usr/lib/udisks2/udisksd
    ├─dbus.service
    │ └─728 /usr/bin/dbus-daemon --system --address=systemd: --nofork --
  nopidfile --systemd-activation --syslog-only
    ├─systemd-timesyncd.service
    │ └─536 /lib/systemd/systemd-timesyncd
    ├─system-getty.slice
    │ └─769 /sbin/agetty -o -p -- \u --noclear tty1 linux
    └─systemd-logind.service
      └─746 /lib/systemd/systemd-logind
```

The purpose of these cgroups is to manage CPU resource allocation and utilization for different processes and groups of processes. By creating separate cgroups for different processes or groups of processes, system administrators can enforce resource limits, prioritize CPU usage, and prevent certain processes from consuming too much CPU time and impacting system performance.

The cgroups under the CPU hierarchy can be created by different system components, such as systemd or other process management tools.

d) How many processes are in cgroup user.slice/system.slice/init.slice?

Check status of slice:

```
$ systemctl status user.slice
● user.slice - User and Session Slice
     Loaded: loaded (/lib/systemd/system/user.slice; static; vendor preset:
enabled)
     Active: active since Thu 2023-04-13 15:49:05 UTC; 2min 14s ago
       Docs: man:systemd.special(7)
      Tasks: 7
     Memory: 219.8M
     CGroup: /user.slice
             └─user-1000.slice
               ├─session-1.scope
               │ ├─1383 sshd: ubuntu [priv]
               │ ├─1516 sshd: ubuntu@pts/0
               │ ├─1517 -bash
               │ ├─1536 systemctl status user.slice
               │ └─1537 pager
               └─user@1000.service
                 └─init.scope
                   ├─1402 /lib/systemd/systemd --user
                   └─1407 (sd-pam)
```

- user.slice: 7
- system.slice: 57
- init.slice: inactive (dead)

e) Run the following commands and explain the output: ps xawf -eo pid,user,cgroup,args

```
$ ps xawf -eo pid,user,cgroup,args
    PID USER     CGROUP                   COMMAND
      2 root     -                        [kthreadd]
      3 root     -                         \_ [rcu_gp]
      4 root     -                         \_ [rcu_par_gp]
      6 root     -                         \_ [kworker/0:0H-kblockd]
      9 root     -                         \_ [mm_percpu_wq]
     10 root     -                         \_ [ksoftirqd/0]
     11 root     -                         \_ [rcu_sched]
     12 root     -                         \_ [migration/0]
     13 root     -                         \_ [idle_inject/0]
     14 root     -                         \_ [cpuhp/0]
     15 root     -                         \_ [cpuhp/1]
     16 root     -                         \_ [idle_inject/1]
```

```
 17 root     -                    \_ [migration/1]
 18 root     -                    \_ [ksoftirqd/1]
 20 root     -                    \_ [kworker/1:0H-kblockd]
 21 root     -                    \_ [cpuhp/2]
 22 root     -                    \_ [idle_inject/2]
 23 root     -                    \_ [migration/2]
 24 root     -                    \_ [ksoftirqd/2]
 26 root     -                    \_ [kworker/2:0H-kblockd]
 27 root     -                    \_ [cpuhp/3]
 28 root     -                    \_ [idle_inject/3]
 29 root     -                    \_ [migration/3]
 30 root     -                    \_ [ksoftirqd/3]
 32 root     -                    \_ [kworker/3:0H-kblockd]
 33 root     -                    \_ [kdevtmpfs]
 34 root     -                    \_ [netns]
 35 root     -                    \_ [rcu_tasks_kthre]
 36 root     -                    \_ [kauditd]
 37 root     -                    \_ [khungtaskd]
 38 root     -                    \_ [oom_reaper]
 39 root     -                    \_ [writeback]
 40 root     -                    \_ [kcompactd0]
 41 root     -                    \_ [ksmd]
 42 root     -                    \_ [khugepaged]
 90 root     -                    \_ [kintegrityd]
 91 root     -                    \_ [kblockd]
 92 root     -                    \_ [blkcg_punt_bio]
 93 root     -                    \_ [tpm_dev_wq]
 94 root     -                    \_ [ata_sff]
 95 root     -                    \_ [md]
 96 root     -                    \_ [edac-poller]
 97 root     -                    \_ [devfreq_wq]
 98 root     -                    \_ [watchdogd]
101 root     -                    \_ [kswapd0]
102 root     -                    \_ [ecryptfs-kthrea]
104 root     -                    \_ [kthrotld]
105 root     -                    \_ [acpi_thermal_pm]
107 root     -                    \_ [scsi_eh_0]
108 root     -                    \_ [scsi_tmf_0]
109 root     -                    \_ [scsi_eh_1]
110 root     -                    \_ [scsi_tmf_1]
112 root     -                    \_ [vfio-irqfd-clea]
113 root     -                    \_ [ipv6_addrconf]
123 root     -                    \_ [kstrp]
126 root     -                    \_ [kworker/u9:0]
139 root     -                    \_ [charger_manager]
185 root     -                    \_ [cryptd]
251 root     -                    \_ [raid5wq]
291 root     -                    \_ [jbd2/vda1-8]
292 root     -                    \_ [ext4-rsv-conver]
327 root     -                    \_ [hwrng]
351 root     -                    \_ [kworker/2:1H-kblockd]
387 root     -                    \_ [kworker/1:1H-kblockd]
398 root     -                    \_ [kworker/3:1H-kblockd]
499 root     -                    \_ [kaluad]
500 root     -                    \_ [kmpath_rdacd]
501 root     -                    \_ [kmpathd]
502 root     -                    \_ [kmpath_handlerd]
```

```
    512 root     -                              \_ [loop0]
    515 root     -                              \_ [loop1]
    518 root     -                              \_ [kworker/0:1H-kblockd]
   1700 root     -                              \_ [kworker/1:1-events]
   1828 root     -                              \_ [kworker/1:0-
cgroup_destroy]
   1955 root     -                              \_ [loop3]
   2360 root     -                              \_ [loop4]
   2404 root     -                              \_ [kworker/0:3-events]
   2529 root     -                              \_ [loop5]
   2983 root     -                              \_ [kworker/u8:1-
events_unbound]
   2984 root     -                              \_ [kworker/u8:2-
events_power_efficient]
   3064 root     -                              \_ [kworker/2:0-events]
  14808 root     -                              \_ [kworker/0:0-
cgroup_destroy]
  14855 root     -                              \_ [kworker/3:1-
cgroup_destroy]
  15106 root     -                              \_ [kworker/2:2-
cgroup_destroy]
  15111 root     -                              \_ [kworker/3:2-events]
      1 root     10:memory:/init.scope,8:cpu /sbin/init
    366 root     10:memory:/system.slice/sys /lib/systemd/systemd-journald
    401 root     10:memory:/system.slice/sys /lib/systemd/systemd-udevd
    503 root     10:memory:/system.slice/mul /sbin/multipathd -d -s
    536 systemd+ 10:memory:/system.slice/sys /lib/systemd/systemd-timesyncd
    613 systemd+ 10:memory:/system.slice/sys /lib/systemd/systemd-networkd
    629 systemd+ 10:memory:/system.slice/sys /lib/systemd/systemd-resolved
    720 root     10:memory:/system.slice/acc
/usr/lib/accountsservice/accounts-daemon
    723 root     10:memory:/system.slice/cro /usr/sbin/cron -f
    724 message+ 10:memory:/system.slice/dbu /usr/bin/dbus-daemon --system -
-address=systemd: --nofork --nopidfile --systemd-activat
    731 root     10:memory:/system.slice/irq /usr/sbin/irqbalance --
foreground
    733 root     10:memory:/system.slice/net /usr/bin/python3
/usr/bin/networkd-dispatcher --run-startup-triggers
    735 syslog   10:memory:/system.slice/rsy /usr/sbin/rsyslogd -n -iNONE
    737 root     10:memory:/system.slice/sys /lib/systemd/systemd-logind
    739 root     10:memory:/system.slice/udi /usr/lib/udisks2/udisksd
    740 daemon   10:memory:/system.slice/atd /usr/sbin/atd -f
    755 root     10:memory:/system.slice/sys /sbin/agetty -o -p -- \u --
keep-baud 115200,38400,9600 ttyS0 vt220
    767 root     10:memory:/system.slice/sys /sbin/agetty -o -p -- \u --
noclear tty1 linux
    792 root     10:memory:/system.slice/pol /usr/lib/policykit-1/polkitd --
no-debug
    867 root     10:memory:/system.slice/ssh sshd: /usr/sbin/sshd -D
[listener] 0 of 10-100 startups
  14833 root     10:memory:/user.slice/user-  \_ sshd: ubuntu [priv]
  14956 ubuntu   10:memory:/user.slice/user-      \_ sshd: ubuntu@pts/0
  14957 ubuntu   10:memory:/user.slice/user-          \_ -bash
  15118 ubuntu   10:memory:/user.slice/user-              \_ ps xawf -eo
pid,user,cgroup,args
   2002 root     10:memory:/system.slice/sna /usr/lib/snapd/snapd
```

```
  14849 ubuntu   10:memory:/user.slice/user- /lib/systemd/systemd --user
  14857 ubuntu   10:memory:/user.slice/user-  \_ (sd-pam)
```

The command lists all running processes along with their PID, username of the creator, cgroup name and command line arguments for starting the process.

The options xawf specify the following:

- x: list all processes when used together with the *a* option
- a: include processes from all users
- w: wide output format
- f: full format display

f) Check the configuration for a process called “cron” using “ps” and “grep”. Explain both, the configuration for the “cron” and “ps” process, in terms of systemd and cgroup configuration.

Check cron:

```
$ ps xawf -eo pid,user,cgroup | grep cron
723 root
10:memory:/system.slice/cron.service,8:cpu,cpuacct:/system.slice/cron.servic
e,6:blkio:/system.slice/cron.service,5:pids:/system.slice/cron.service,2:dev
ices:/system.slice/cron.service,1:name=systemd:/system.slice/cron.service,0:
:/system.slice/cron.service
```

-> system.slice

Check ps:

```
$ ps xawf -eo pid,user,cgroup,args | grep ps
    867 root     10:memory:/system.slice/ssh sshd: /usr/sbin/sshd -D
[listener] 0 of 10-100 startups
   1572 ubuntu   10:memory:/user.slice/user-              \_ ps xawf -eo
pid,user,cgroup,args
   1573 ubuntu   10:memory:/user.slice/user-              \_ grep --
color=auto ps
```

-> user.slice

g) Verify your understanding by using the following commands: systemd-cgtop and systemd-cgls

systemd-cgtop:
Displays the current resource usage of cgroups in a hierarchical view:

- Real-time overview of the CPU, memory, and IO usage of cgroups and their associated processes.
- By default top-level cgroup hierarchy.

```
$ systemd-cgtop
Control Group                          Tasks   %CPU   Memory  Input/s Output/s
user.slice                                 6    2.3   119.2M        -        -
/                                        148    1.0     1.2G        -        -
system.slice                              55    0.1   959.7M        -        -
system.slice/irqbalance.service            2    0.0   908.0K        -        -
system.slice/multipathd.service            7    0.0    13.6M        -        -
init.scope                                 1      -     8.1M        -        -
system.slice/accounts-daemon.service       3      -     8.6M        -        -
system.slice/atd.service                   1      -   640.0K        -        -
system.slice/boot-efi.mount                -      -    36.0K        -        -
system.slice/cloud-final.service           -      -     1.7M        -        -
system.slice/cloud-init.service            -      -    14.4M        -        -
system.slice/cron.service                  1      -     5.5M        -        -
...
```

Can you identify the “cron” service?

To display the memory usage of cron.service one can run the following command:

```
$ systemd-cgtop | grep cron
system.slice/cron.service                  1      -     5.3M        -        -
```

systemd-cgls
Displays (recursively) the cgroup hierarchy in a tree-like format, showing the parent-child relationships between cgroups.

- It can be used to inspect the hierarchy and see the subsystems associated with each cgroup.
- By default, systemd-cgls displays the entire cgroup hierarchy, use --unit|--user-unit [UNIT...] for showing a specific
- By default, empty control groups are not shown, use --all for showing also empty ones.

```
$ systemd-cgls
Control group /:
```

```
-.slice
├─user.slice
│ └─user-1000.slice
│   ├─user@1000.service
│   │ └─init.scope
│   │   ├─21878 /lib/systemd/systemd --user
│   │   └─21883 (sd-pam)
│   └─session-158.scope
│     ├─21865 sshd: ubuntu [priv]
│     ├─21994 sshd: ubuntu@pts/0
│     ├─21997 -bash
│     ├─22083 systemd-cgls
│     └─22084 pager
├─init.scope
│ └─1 /sbin/init
└─system.slice
  ├─irqbalance.service
  │ └─736 /usr/sbin/irqbalance --foreground
  ├─systemd-networkd.service
  │ └─611 /lib/systemd/systemd-networkd
  ├─systemd-udevd.service
  │ └─403 /lib/systemd/systemd-udevd
  ├─cron.service
  │ └─722 /usr/sbin/cron -f
  ├─system-serial\x2dgetty.slice
  │ └─serial-getty@ttyS0.service
  │   └─763 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0
vt220
  ├─polkit.service
  │ └─800 /usr/lib/policykit-1/polkitd --no-debug
  ├─networkd-dispatcher.service
  │ └─738 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-
triggers
  ├─multipathd.service
  │ └─503 /sbin/multipathd -d -s
  ├─accounts-daemon.service
  │ └─719 /usr/lib/accountsservice/accounts-daemon
  ├─systemd-journald.service
  │ └─367 /lib/systemd/systemd-journald
  ├─atd.service
  │ └─749 /usr/sbin/atd -f
  ├─ssh.service
  │ └─854 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
  ├─snapd.service
  │ └─13758 /usr/lib/snapd/snapd
  ├─rsyslog.service
  │ └─744 /usr/sbin/rsyslogd -n -iNONE
  ├─systemd-resolved.service
  │ └─628 /lib/systemd/systemd-resolved
  ├─udisks2.service
  │ └─748 /usr/lib/udisks2/udisksd
  ├─dbus.service
  │ └─728 /usr/bin/dbus-daemon --system --address=systemd: --nofork --
nopidfile --systemd-activation --syslog-only
  ├─systemd-timesyncd.service
  │ └─536 /lib/systemd/systemd-timesyncd
  ├─system-getty.slice
```

```
  | └─getty@tty1.service
  |   └─769 /sbin/agetty -o -p -- \u --noclear tty1 linux
  └─systemd-logind.service
    └─746 /lib/systemd/systemd-logind
```

Can you identify the “cron” service?

```
$ systemd-cgls --unit cron.service
Unit cron.service (/system.slice/cron.service):
└─722 /usr/sbin/cron -f
```

Subtask 1.2 – Default System Configuration by Systemd

a) Obviously, systemd creates a number of cgroups. Identify the respective unit files and explain their configuration.

Command to list all the slice units loaded in the system:

```
$ systemctl list-units --type slice
  UNIT                          LOAD   ACTIVE SUB    DESCRIPTION
  -.slice                       loaded active active Root Slice
  system-getty.slice            loaded active active system-getty.slice
  system-modprobe.slice         loaded active active system-modprobe.slice
  system-serial\x2dgetty.slice loaded active active system-
serial\x2dgetty.slice
  system-systemd\x2dfsck.slice loaded active active system-
systemd\x2dfsck.slice
  system.slice                  loaded active active System Slice
  user-1000.slice               loaded active active User Slice of UID 1000
  user.slice                    loaded active active User and Session Slice
  ...
```

- -.slice: This is the root slice of the systemd hierarchy, which includes all other slices and units.
- system-getty.slice: This slice includes all the getty processes that are used to provide console access to the system. It is managed by the getty@.service unit, which starts a getty process for each virtual console.

- system-modprobe.slice: This slice includes all the modprobe processes that are used to load kernel modules. It is managed by the systemd-modules-load.service unit, which loads kernel modules at boot time.
- system-serial\x2dgetty.slice: This slice includes all the serial getty processes that are used to provide serial console access to the system. It is managed by the serial-getty@.service unit, which starts a getty process for each serial console.
- system-systemd\x2dfsck.slice: This slice includes the systemd-fsck-root.service unit, which runs a file system check on the root file system during system boot.
- system.slice: This slice includes all the system services and processes that are not associated with a specific user. It is managed by various system services, including systemd-journald.service, systemd-udevd.service, and dbus.service, among others.
- user-1000.slice: This slice includes all the processes and services associated with user ID 1000. It is managed by the user@1000.service unit, which starts user-specific services and processes.
- user.slice: This slice includes all the processes and services associated with any user on the system. It is managed by the user@.service unit, which starts user-specific services and processes for any user that logs in.

Bonus:
The init.scope slice is created by systemd at boot time and provides a separate resource management environment for the system initialization process. This allows systemd to manage the initialization process in a controlled and efficient manner, and to ensure that critical system services and processes are started in the correct order.

It is also used to manage system shutdown and reboot operations. When the system is shut down or rebooted, systemd sends a signal to the init process to trigger the shutdown sequence. This allows systemd to coordinate the shutdown of all running processes and services in a controlled and graceful manner.

b) Modify the user.slice configuration such that all processes created by users do not receive more than 20% of CPU share. Verify your configuration by creating more load than your system can handle, e.g. by creating a number of "wc /dev/zero &" background processes (& send the process directly into the background). You may find man man systemd.resource-control useful. What happened to the "cron" process?

To modify the CPU share limit for processes in the user.slice cgroup without directly modifying the default configuration file user.slice, one can create a custom systemd service file in the /etc/systemd/system directory:

1. Create a new file in the /etc/systemd/system directory with a name ending in .slice.conf, for example custom-user.slice.conf.

```
$ sudo nano /etc/systemd/system/user.slice
```

3. Add the following content:

```
[Unit]
Description=Custom User Slice

[Slice]
CPUQuota=20%
```

4. Save the file and exit the text editor.
5. Reload the systemd configuration to apply the changes:

```
$ sudo systemctl daemon-reload
```

6. Test the configuration by creating more load than your system can handle.

```
$ wc /dev/zero &
```

Checking the CPU shares:

```
$ top
...
PID  USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1789 ubuntu    20   0    7240    656    592 R   6.5   0.0   0:01.49 wc
1790 ubuntu    20   0    7240    652    588 R   6.5   0.0   0:01.21 wc
1786 ubuntu    20   0    7240    660    592 R   2.1   0.0   0:01.41 wc
1788 ubuntu    20   0    7240    652    588 R   2.1   0.0   0:00.61 wc
1787 ubuntu    20   0    7240    656    588 R   1.8   0.0   0:00.66 wc
1791 ubuntu    20   0    7240    656    592 R   1.5   0.0   0:00.44 wc
   1 root      20   0  104960  12632   8352 S   0.0   0.2   0:03.18 systemd
   2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
```

```
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
...
```

As for the "cron" process, it should not be affected by the changes to the user.slice configuration, since it runs in a separate cgroup. The changes we made only affect processes running in the user.slice cgroup. However, if the "cron" process is running as a user-level process (rather than as a system service), then it would be subject to the same CPU quota limit as other user-level processes.

c) Free one wc process from the limits imposed by the user.slice related cgroup.

First solution:

1. Move the process to a different cgroup using the systemd-run command. For example, you can run:

```
$ sudo systemd-run -p CPUAccounting=false -p CPUQuota=100% wc /dev/zero
```

This command creates a new transient unit for a wc /dev/zero process and sets the CPUQuota parameter to 100%, effectively removing the CPU limit imposed by the user.slice cgroup.

3. Verify that the process has been moved to the new cgroup by checking its cgroup using the cat /proc//cgroup command. The output should show the new cgroup that you specified in the systemd-run command.

```
PID   USER    PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
22403 root    20   0    7240    660    592 R 100.0   0.0   1:23.40 wc
22327 ubuntu  20   0    7240    656    592 R   5.8   0.0   0:25.02 wc
22334 ubuntu  20   0    7240    596    528 R   1.0   0.0   0:14.34 wc
22255 ubuntu  20   0    7240    656    592 R   0.6   0.0   0:18.33 wc
```

```
$ cat /proc/22403/cgroup
12:pids:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
11:memory:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
10:hugetlb:/
9:net_cls,net_prio:/
8:freezer:/
7:blkio:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
6:devices:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
5:perf_event:/
4:cpuset:/
```

```
3:cpu,cpuacct:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
2:rdma:/
1:name=systemd:/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
0::/system.slice/run-r911bb82d8a624dc886f7b79d8c1f7853.service
```

Second posible solution:

1. Choose the PID of the wc process that you want to free from the limits imposed by the cgroup -> 1789.
2. Run the following command to move the chosen wc process out of the user.slice cgroup and into the default cgroup:

```
$ sudo sh -c 'echo 1789 > /sys/fs/cgroup/cpu/user.slice/cgroup.procs'
```

Task 2 - Create Custom Controls

Create an environment to control CPU access (CPU affinity) from scratch, i.e. not using any preconfigured cgroups.

```
$ sudo mkdir /mnt/myenv
```

Create a tmpfs under /mnt and use this directory for your cgroup hierarchy.

```
$ sudo mount -t tmpfs myenv /mnt/myenv -o size=10M
```

How many CPUs can be controlled on your system per default?

```
$ nproc
4
```

Number of cores the CPU has.

Create M processes, with M being the number of CPUs (N) on your system plus one (M=N+1). Those processes shall consume 100% CPU load.

4+1 = 5 processes

```
$ for i in {1..5}; do dd if=/dev/urandom of=/dev/null & done
[1] 23649
[2] 23650
[3] 23651
[4] 23652
[5] 23653
```

```
$ top

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
COMMAND
  23651 ubuntu    20   0    7272    788    720 R  89.7   0.0   2:23.90 dd
  23652 ubuntu    20   0    7272    720    656 R  83.7   0.0   2:22.27 dd
  23653 ubuntu    20   0    7272    772    708 R  80.0   0.0   2:23.53 dd
  23649 ubuntu    20   0    7272    784    720 R  73.7   0.0   2:21.90 dd
```

Now configure your cgroups to only allow these processes to use half of the CPUs available on your system. Mind the following prerequisite (command to be executed once the controller is mounted and the cgroup created): echo 0 > cpuset.mems

```
$ echo 0 | sudo tee /mnt/myenv/cpuset.mems
$ echo "0-1" | sudo tee /mnt/myenv/cpuset.cpus
$ sudo echo 23649 >> /mnt/myenv/cgroup.procs
$ sudo echo 23650 >> /mnt/myenv/cgroup.procs
$ sudo echo 23651 >> /mnt/myenv/cgroup.procs
$ sudo echo 23652 >> /mnt/myenv/cgroup.procs
$ sudo echo 23653 >> /mnt/myenv/cgroup.procs
$ cat cgroup.procs
23649
23650
23651
23652
23653
$ sudo echo 0 > /mnt/myenv/cpuset.mems
```

Verify and explain the effect by looking at the processes via "top".

```
$ top
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
COMMAND
  23651 ubuntu    20   0    7272    788    720 R  50.2   0.0  70:41.55 dd
  23649 ubuntu    20   0    7272    784    720 R  49.8   0.0  70:37.21 dd
  23650 ubuntu    20   0    7272    788    720 R  33.2   0.0  70:37.68 dd
```

```
  23652 ubuntu    20   0    7272    720    656 R  33.2   0.0  70:39.50 dd
  23653 ubuntu    20   0    7272    772    708 R  33.2   0.0  70:40.75 dd
```

Also double-check the CPU assignments (affinity) with the command "taskset".

```
$ taskset -pc 23649
pid 23649's current affinity list: 0,1
$ taskset -pc 23650
pid 23650's current affinity list: 0,1
$ taskset -pc 23651
pid 23651's current affinity list: 0,1
$ taskset -pc 23652
pid 23652's current affinity list: 0,1
$ taskset -pc 23653
pid 23653's current affinity list: 0,1
```

Disable those CPUs via "chcpu", that you allowed in your cpuset and explain what happens with those processes pinned onto them.

```
$ sudo chcpu -d 0,1
```

verify available cpus

```
$ lscpu
```

The processes will get migrated to cpu that is still available. if no CPUs are available in the cpuset the processes will not be able to execute until cpu becomes available. The set default core however cannot be disabled so processes can still run.