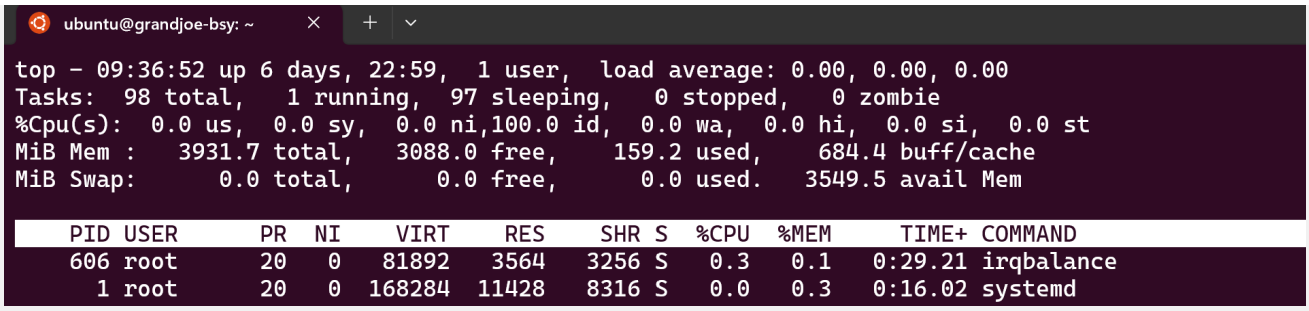
Lab 2 PROC-1

Task 1 – Programs and Processes

Subtask 1.1 - Process Basics

Using “top”, explain the elements of the “Summary Display”



Up time, user sessions, CPU usage, Memory usage, etc.

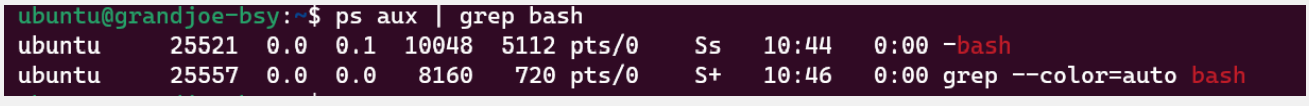
Which process has PID 1 and PID 2? Who is the owner of these processes?

PID1: Systemd (root):

PID2: kthreadd (root):

What is the PID of your terminal? Who is the owner?

```
ps aux | grep bash
```



Show the Environmental Variables of your terminal. Add a variable named Name and assign your surname to it.

show: `env`

new env variable: `export NAME=Grand`

Study the “tail” program and find out how to monitor the evolution of a file in real-time. Use this tool to monitor system information available in the following

file: /var/log/syslog. Start a second terminal and observe the output of your monitoring session. What do you notice?

```
tail -f /var/log/syslog
```

```
ubuntu@grandjoe-bsy:~$ tail -f /var/log/syslog
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Downloading…: 90%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Downloading…: 92%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Downloading…: 94%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Downloading…: 96%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Downloading…: 98%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Idle…: 100%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Idle…: 100%
Mar 17 09:49:07 grandjoe-bsy fwupdmgr[10322]: Successfully downloaded new metadata: 0 local devices supported
Mar 17 09:49:07 grandjoe-bsy systemd[1]: fwupd-refresh.service: Succeeded.
Mar 17 09:49:07 grandjoe-bsy systemd[1]: Finished Refresh fwupd metadata and update motd.
Mar 17 09:57:10 grandjoe-bsy systemd[1]: Started Session 195 of user ubuntu.
```

On the second terminal, what is the PID and the state of your monitoring process (tail)?

```
ps aux | grep tail
```

```
ubuntu@grandjoe-bsy:~$ ps aux | grep tail
ubuntu      10504  0.0  0.0   7264   580 pts/0    S+   09:56   0:00 tail -f /var/log/syslog
```

(PID: 10504)

Show the process hierarchy of your observer process (tail)

```
pstree -p
```

Terminal (kill) the terminal that runs the tail command and explain what happens.

> The parent processes "bash" and "sshd" get killed:
>
> The "tail" process is still running as an orphan and gets assigned the init process as new parent. It will clean up the now unnecessary "tail" process and send a sigterm.
>
> If we run "ps -aux | grep tail" we dont see the "tail -f" command:

```
ubuntu@bsy-vm:~$ ps -aux | grep tail
ubuntu      9844  0.0  0.0   8160   672 pts/1    S+   07:40   0:00 grep --color=auto tail
```

Subtask 1.2 – Job Control

Study the “Job Control” section of the bash man page. Start three observer processes and put all of them into the background. Display the list of background jobs and explain the output. Move job number 2 into foreground and press Ctrl + z. List again the job list. What is the difference now?

`vim /etc/temp/1 &` use **&** for starting the process in the background

```
ubuntu@grandjoe-bsy:~$ jobs
[1]   Stopped                 vim /etc/temp/1
[2]-  Stopped                 vim /etc/temp/2
[3]+  Stopped                 vim /etc/temp/3
ubuntu@grandjoe-bsy:~$
```

`fg %2` put process 2 in the foreground `Ctrl + Z` to pause it again

```
ubuntu@grandjoe-bsy:~$ fg %2
vim /etc/temp/2

[2]+  Stopped                 vim /etc/temp/2
ubuntu@grandjoe-bsy:~$ jobs
[1]   Stopped                 vim /etc/temp/1
[2]+  Stopped                 vim /etc/temp/2
[3]-  Stopped                 vim /etc/temp/3
ubuntu@grandjoe-bsy:~$
```

Last post job marked with **+**

Immidiate post job marked with **-**

Subtask 1.3 – Process Creation

HelloWorld.c

Revisit the layout of a binary and print the sizes of sections “Text”, “Data” and “BSS”. What is the difference between the “Berkley” and the “Gnu” format?

`size HelloWorld --format=berkeley` oder `size HelloWorld --format=sysv`

```
ubuntu@grandjoe-bsy:~$ size HelloWorld --format=berkeley
   text    data     bss     dec     hex filename
   1572     600       8    2180     884 HelloWorld
ubuntu@grandjoe-bsy:~$ size HelloWorld --format=sysv
HelloWorld  :
section               size    addr
.interp                 28     792
.note.gnu.property      32     824
.note.gnu.build-id      36     856
.note.ABI-tag           32     892
.gnu.hash               36     928
.dynsym                168     968
.dynstr                130    1136
.gnu.version            14    1266
.gnu.version_r          32    1280
.rela.dyn              192    1312
.rela.plt               24    1504
.init                   27    4096
.plt                    32    4128
.plt.got                16    4160
.plt.sec                16    4176
.text                  389    4192
.fini                   13    4584
.rodata                 23    8192
.eh_frame_hdr           68    8216
.eh_frame              264    8288
.init_array              8   15800
.fini_array              8   15808
.dynamic               496   15816
.got                    72   16312
.data                   16   16384
.bss                     8   16400
.comment                43       0
Total                 2223
```

GNU lists every section

Use objdump and display all section headers. Revisit the sections above and study the elements of each section. Disassemble the “Text” section.

objdump -h HelloWorld

```
ubuntu@grandjoe-bsy:~$ objdump -h HelloWorld

HelloWorld:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000000318  0000000000000318  00000318  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.gnu.property 00000020  0000000000000338  0000000000000338  00000338  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000000358  0000000000000358  00000358  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .note.ABI-tag 00000020  000000000000037c  000000000000037c  0000037c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     00000024  00000000000003a0  00000000000003a0  000003a0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .dynsym       000000a8  00000000000003c8  00000000000003c8  000003c8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .dynstr       00000082  0000000000000470  0000000000000470  00000470  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .gnu.version  0000000e  00000000000004f2  00000000000004f2  000004f2  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .gnu.version_r 00000020  0000000000000500  0000000000000500  00000500  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .rela.dyn     000000c0  0000000000000520  0000000000000520  00000520  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .rela.plt     00000018  00000000000005e0  00000000000005e0  000005e0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .init         0000001b  0000000000001000  0000000000001000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .plt          00000020  0000000000001020  0000000000001020  00001020  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .plt.got      00000010  0000000000001040  0000000000001040  00001040  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .plt.sec      00000010  0000000000001050  0000000000001050  00001050  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .text         00000185  0000000000001060  0000000000001060  00001060  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 16 .fini         0000000d  00000000000011e8  00000000000011e8  000011e8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 17 .rodata       00000017  0000000000002000  0000000000002000  00002000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 18 .eh_frame_hdr 00000044  0000000000002018  0000000000002018  00002018  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 19 .eh_frame     00000108  0000000000002060  0000000000002060  00002060  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 20 .init_array   00000008  0000000000003db8  0000000000003db8  00002db8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .fini_array   00000008  0000000000003dc0  0000000000003dc0  00002dc0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .dynamic      000001f0  0000000000003dc8  0000000000003dc8  00002dc8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 23 .got          00000048  0000000000003fb8  0000000000003fb8  00002fb8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000004000  0000000000004000  00003000  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000008  0000000000004010  0000000000004010  00003010  2**0
                  ALLOC
 26 .comment      0000002b  0000000000000000  0000000000000000  00003010  2**0
                  CONTENTS, READONLY
```

Write a program that when running as a process, creates another process by “forking” (man fork) itself. Use the library function “sleep()” (man sleep) to put both processes to sleep for 30 seconds before they terminate. Check the memory usage of both processes and explain. You may find “getpid()” useful, see man getpid.

fork.c

```
ubuntu@grandjoe-bsy:~$ ./fork
Hello from parent with id: 23224
Hello from child with id: 23225
```

`ps aux | grep fork` to see the memory usage:

```
ubuntu@grandjoe-bsy:~/Praktikum_2$ ps aux | grep fork
message+     659  0.0  0.0   7648  4808 ?        Ss   Mar24   0:00 /usr/bin/dbus-daemon --s
: --nofork --nopidfile --systemd-activation --syslog-only
ubuntu      9209  0.0  0.0   2488   576 pts/0    T    07:31   0:00 ./fork
ubuntu      9210  0.0  0.0   2488    72 pts/0    T    07:31   0:00 ./fork
ubuntu      9216  0.0  0.0   8160   724 pts/0    S+   07:32   0:00 grep --color=auto fork
```

Note that the second process (child process) uses only 72kb instead of 576kb as the **Copy-On-Write** mode is used (default).

Subtask 1.4 – Zombie

Zombie.c

```
ubuntu@grandjoe-bsy:~$ ps aux | grep Zombie
ubuntu     23936  0.0  0.0   2488   580 pts/1    S+   17:11   0:00 ./Zombie
ubuntu     23937  0.0  0.0      0     0 pts/1    Z+   17:11   0:00 [Zombie] <defunct>
```

Zombie-process marked with

Subtask 1.5 – Zombie or no Zombie

Orphan.c

```
ubuntu@grandjoe-bsy:~/Praktikum_2$ ./Orphan
Hello from parent with id: 24037
Parent exiting now!
Hello from child with id: 24038
```

```
ubuntu@grandjoe-bsy:~/Praktikum_2$ ps aux | grep Orphan
ubuntu     24038  0.0  0.0   2488    72 pts/0    S    17:17   0:00 ./Orphan
ubuntu     24042  0.0  0.0   8160   720 pts/1    S+   17:17   0:00 grep --color=auto Orphan
```

Note that only the child process is shown. When you look at the hierarchy you see the new parent of the oprhan:

```
ubuntu@grandjoe-bsy:~/Praktikum_2$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─Orphan
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─2*[agetty]
```

Task 2 – Multi-Threading

- if less than 2 args, stop
- create stack with following features:
  - `NULL`: anywhere you want
  - `STACK_SIZE`: the size
  - `PROT_READ |PROT_WRITE`: read / write access permitted
  - `MAP_PRIVATE | MAP_ANONYMOUS | MAP_STACK`: only seen by this process, mapping to heap, allocate mapping at a suitable address for process / thread
  - `-1`: file descriptor (-1 = none)
  - `0`: offset
- clone() creates child that will start his execution in childFunc entry point and top of stack is passed to child
- parent waits for the child to finish

How can you (developer) define what should be shared between parent and child?

Within the clone function the developer can define what can be shared -> here stack.

Task 3 - Kernel Threads

`top` PID 2 is used by kthreadd

Identify Kernel threads:

```
ps -efH
```

All processes which are within []

See on which cpu a thread is running

In the file /proc//stat the 39 variable shows the cpu id the thread / program is running on:

```
ubuntu@grandjoe-bsy:/proc/3$ cat /proc/3/stat | awk '{print $39}'
0
ubuntu@grandjoe-bsy:/proc/3$ cat /proc/cpuinfo | grep processor
processor       : 0
processor       : 1
ubuntu@grandjoe-bsy:/proc/3$
```

Select any thread called “Kworker” and explain the state by querying it with the ps command.

```
ubuntu@bsy-vm:~$ sudo cat /proc/6/status
Name:   kworker/0:0H-kblockd
Umask:  0000
State:  I (idle)
Tgid:   6
Ngid:   0
Pid:    6
PPid:   2
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups:
NStgid: 6
NSpid:  6
NSpgid: 0
NSsid:  0
Threads:        1
SigQ:   0/15608
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: ffffffffffffffff
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
NoNewPrivs:     0
Seccomp:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   1
Cpus_allowed_list:      0
Mems_allowed:   00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        4
nonvoluntary_ctxt_switches:     0
ubuntu@bsy-vm:~$ sudo cat /proc/6/task/
cat: /proc/6/task/: Is a directory
ubuntu@bsy-vm:~$ sudo ls /proc/6/task/
6
```

```
ubuntu@bsy-vm:~$ ps -aux | grep kworker
root           6  0.0  0.0      0     0 ?        I<   Mar11   0:00 [kworker/0:0H-kblockd]
root          20  0.0  0.0      0     0 ?        I<   Mar11   0:00 [kworker/1:0H-kblockd]
root         112  0.0  0.0      0     0 ?        I<   Mar11   0:00 [kworker/u5:0]
root         171  0.0  0.0      0     0 ?        I<   Mar11   0:00 [kworker/1:1H-kblockd]
root         276  0.0  0.0      0     0 ?        I<   Mar11   0:00 [kworker/0:1H-kblockd]
root       16412  0.0  0.0      0     0 ?        I    17:42   0:00 [kworker/0:0-events]
root       17069  0.0  0.0      0     0 ?        I    20:18   0:00 [kworker/1:1-events]
root       17315  0.0  0.0      0     0 ?        I    21:26   0:00 [kworker/u4:0-events_unbound]
root       17338  0.0  0.0      0     0 ?        I    21:57   0:00 [kworker/0:2-events]
root       17356  0.0  0.0      0     0 ?        I    21:57   0:00 [kworker/1:2]
root       17635  0.0  0.0      0     0 ?        I    22:10   0:00 [kworker/u4:1-events_power_efficient]
ubuntu     17775  0.0  0.0   8160  2452 pts/2    S+   22:22   0:00 grep --color=auto kworker
```

We can see that the coworker process here is "IDLE". This can also be seen at "I" in "ps". On the other hand we see its name, where it may be written to, in which address, on which CPU it may run, on which CPUs it may be scheduled

> (cpus_allowed), how many threads there are (1) also seen in proc/6/task (number of entries = number of tasks), who the parent is (ppid)...