Lab 04

- Monitoring processes: ps, vmstat, pstree, top, and uptime
- Prioritising processes: nice, renice
- Configuring the cron facility: crontab

1. *What is the difference between `ps` and `ps -ef`?*

- `ps` shows the active processes for the current user
- `ps -ef` shows all active processes

2. *Explain the meaning of the column headers `UID`, `PID`, `PPID` and `CMD`*

Header	Meaning
UID	User ID, who the process belongs to
PID	Process ID, the unique number assigned to a process when it's started up
PPID	Parent Process ID
CMD	The command line being executed

3. *Write down the process identifiers of at least two processes that have the PPID of 1, representing the init process*

```
[student@UWS ~]$ ps --ppid 1
  PID TTY          TIME CMD
   82 ?        00:00:00 syslogd
   84 ?        00:00:00 klogd
   92 ?        00:00:00 rpcbind
  112 ?        00:00:00 dhcpcd
  119 ?        00:00:00 dropbear
  126 ?        00:00:00 rpc.statd
  138 ?        00:00:00 rpc.mountd
  144 ?        00:00:00 crond
  174 ?        00:00:00 login
```

4. *Is the process `ps -ef` listed itself? If yes, what is its PID on the system?*

```
[student@UWS ~]$ ps -ef | grep student
root       174     1  0 19:42 ?        00:00:00 login -- student
student    192   174  0 19:44 hvc0     00:00:01 -bash
student    246   192  0 20:09 hvc0     00:00:00 ps -ef
student    247   192  0 20:09 hvc0     00:00:00 grep student
```

- Yes
- 246

5. *How many processes in total are currently being run by the root user?*

```
[student@UWS ~]$ ps -fu root | wc -l
45
```

6. *What is the PID of the terminal and the nano process?*

```
[student@UWS ~]$ nano &
[1] 253
[student@UWS ~]$ ps
  PID TTY          TIME CMD
  192 hvc0     00:00:02 bash
  253 hvc0     00:00:00 nano
  254 hvc0     00:00:00 ps

[1]+  Stopped                 nano
```

- nano: 253
- bash: 192

7. *What is the function of the ampersand & in the above command?*

`&` sends a process to the background.

8. *What is the PID and PPID of the terminal and the two processes nano and vi? What is the relation between the calling terminal shell and the two processes?*

```
[student@UWS ~]$ ps -ef | grep -e nano -e vi
student    253   192  0 20:11 hvc0     00:00:00 nano
student    255   192  0 20:16 hvc0     00:00:01 vi
```

`nano` and `vi` have PIDs 253 and 255 respectively, and they both share PPID 192, which is the PID of `bash`

9. *Did this kill the process vi?*

```
[student@UWS ~]$ kill 255; ps -fu student
UID        PID  PPID  C STIME TTY          TIME CMD
student    192   174  0 19:44 hvc0     00:00:02 -bash
student    253   192  0 20:11 hvc0     00:00:00 nano
student    255   192  0 20:16 hvc0     00:00:01 vi
student    275   192  0 20:20 hvc0     00:00:00 ps -fu student
```

No.

10. *Use the man kill command and check for the meaning of the qualifier -9 to find out why the process has been killed this time?*

`-9` represents a `KILL` signal, which unconditionally and immediately halts the execution of a programme.

11. *What is the PPID and PID of each `top` process?*

```
[student@UWS ~]$ ps -ef | grep top
student    200   199  0 09:54 hvc0     00:00:00 top
student    202   201  0 09:54 hvc0     00:00:00 top
student    205   201  0 09:54 hvc0     00:00:00 grep top
```

- PIDs: 200, 199

- PPIDs: 202, 201

12. *What happened to the two top processes? Were they killed too? What is the new PPID of the two top processes? Can you therefore explain what happens to orphaned processes?*

```
[student@UWS ~]$ ps -ef | grep top
student    200   199  0 09:54 hvc0     00:00:00 top
student    207   199  0 09:55 hvc0     00:00:00 grep top
```

When the parent process was killed, the child was also killed. Process 200 remains in the background as its parent was untouched. If the question requires the overall parent to be killed, i.e. PID 199 `bash`, the terminal emulator would be killed along with its child processes, and the user would be required to log in.

An orphaned process is one whose parent has terminated but continues to execute. Running `ps -ax` and finding an absence of `PID -1` allows us to confirm that there are no current zombie or orphaned processes:

```
[student@UWS ~]$ ps -ax | head
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:02 init [3]
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:00 [kworker/0:0]
    4 ?        S<     0:00 [kworker/0:0H]
    5 ?        S      0:00 [kworker/u2:0]
    6 ?        S<     0:00 [mm_percpu_wq]
    7 ?        S      0:00 [ksoftirqd/0]
    8 ?        S      0:00 [kdevtmpfs]
    9 ?        S<     0:00 [netns]
-- snip --
```

If any of the `top` processes had been orphaned and adopted by `init` we'd see them with a PID of `1`.

13. *Describe the output that you see. How much approximate CPU time (in %) does the cruncher program use while it is still running?*

![[Pasted image 20211005120713.png]]

`top` appears in the terminal window with a cascading series of processes stemming from `init`, leading to `cruncher` and `top`, before a series of processes related to `[kthreadd]` appears. In CPU time %, `cruncher` appears to use between 3-5%.

14. *In what order does the top command display the processes?*

`top` displays processes in %CPU descending in a tree format, i.e. a `cruncher`, when using significant CPU time, will appear as a child of `init`. If `cruncher` is the most intensive process running at the time, `init` will appear at the top of the list despite its current %CPU reading of ~0.

15. *How many processes are running on your machine in total? How many of them are sleeping?*

```
[root@UWS ~]# top
top - 11:14:24 up  1:30,  1 user,  load average: 0.00, 0.03, 0.01
Tasks:  49 total,   1 running,  47 sleeping,   1 stopped,   0 zombie
-- snip --
```

- 49 total
- 47 sleeping

16. *The 'Tasks' row has space for zombie processes. Do you have any idea what a zombie in a Unix-like system might refer to?*

When a process is killed it informs its parent of its pending termination. If this is successful its PID will be removed from `ps`. If the parent fails to acknowledge the termination, for whatever reason, the process becomes a zombie process. A zombie process has been terminated, (and therefore) cannot be killed, and its resources have been freed. Either init adopts the zombie process, or the process is cleared on boot.

17. *What is the priority number of the program cruncher? Does it change at all?*

20, and the priority value does not appear to change throughout the programme's lifetime.

18. *What is the highest priority that any of the processes gets assigned?*

- BSD C Shell & standalone: 20
- System V C Shell: 39

19. *What priority level is given to the majority of the processes?*

20

20. *Is any swap space used at this time?*

`0.0GiB`

21. *Explain briefly what kind of information vmstat displays here? What do the two qualifiers: 10 and 4 stand for?*

Statistics pertaining to the system's virtual memory. `vmstat n` will cause the programme to refresh every `n` seconds, where 10 is every ten seconds, and 4 is every four seconds.

22. *What numbers change in the vmstat display? What does the number in the `r` column represent?*

With a number of `cruncher` programmes executing in the background:

- columns `r, in, cs, us, sy,` and `id` all change at least once.
- `r` represents the number of runnable processes which are either running, or waiting for run time.

23. *What happens, while running the application programs, to the last three entries shown below the cpu heading that are labelled: `us sy id`? To which value do they always add up? Can you interpret their meaning?*

The values of `us, sy`, and `id` each add up to 100. These numbers vary over time, with `id` appearching to change once at upon initial execution of the `vmstat` command, and when the background instances of `cruncher` appear to have each completed processing.

```
[root@UWS ~]# vmstat 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 4  0      0 495352      0   4952    0    0     0     0  904   78  9 15 77  0  0
 5  0      0 495352      0   4952    0    0     0     0 3608  368 36 64  0  0  0
 4  0      0 495384      0   4952    0    0     0     0 3626  369 37 63  0  0  0
 4  0      0 495260      0   4952    0    0     0     0 3636  366 39 61  0  0  0
 4  0      0 495352      0   4952    0    0     0     0 3638  370 38 62  0  0  0
 0  0      0 496160      0   4944    0    0     0     0 2753  276 28 47 25  0  0
```

24. *What happens to the number associated with the `us` column once all cruncher sessions have terminated?*

It rests at zero.

25. *What information does uptime provide?*

```
[root@UWS ~]# uptime
 11:55:33 up 12 min,  1 user,  load average: 0.44, 0.83, 0.46
```

The length of time that the system has been running, the number of logged in users, the time and date the system came online, and information about the system's average load.

26. *What is the name and the significance of the process that is displayed in the leftmost position*

`init`. Its significance is that it is the direct, indirect parent of all processes running on the system.

27. *Write down all linking processes between init and pstree. Could you obtain the same information using the ps command?*

`init --> login --> bash --> pstree`

A clumsy way of performing this would be to find the process ID of some programme and work backward:

```
[student@UWS ~]$ sleep 600 &
[1] 238
[student@UWS ~]$ ps -o ppid= -p 238
  192
[student@UWS ~]$ ps -o ppid= -p 192
  174
[student@UWS ~]$ ps -o ppid= -p 174
    1
```

28. *What are the nice numbers of some processes: e.g. init , kthread, cron and udevd?*

```
[student@UWS ~]$ top -b -n 1 | grep "NI\|cron\|init\|kthread\|udevd"
  PID USER      PR  NI    VIRT    RES  %CPU  %MEM     TIME+ S COMMAND
    1 root      20   0    2.1m   0.7m   0.0   0.1   0:02.84 S init [3]
  144 root      20   0    2.1m   0.8m   0.0   0.2   0:00.11 S  - /usr/sbin/crond
-f
  333 student   20   0    2.9m   1.0m   0.0   0.2   0:00.03 S          - grep
NI\|cron\|init\|kthread\|udevd
    2 root      20   0    0.0m   0.0m   0.0   0.0   0:00.01 S [kthreadd]
```

They're all 0.

29. *What is the nice number (NI) given by the system to the cruncher script?*

20.

30. *Note the actual outputs for real, user and sys times for the cruncher script after it has finished executing.*

```
[root@UWS ~]#
real    0m15.022s
user    0m4.394s
sys     0m9.989s
```

31. *Note the real, user and sys times again and comment on any difference in the real time value*

```
[root@UWS ~]# time nice -n 19 ./cruncher > /dev/null
real    0m14.351s
user    0m4.453s
sys     0m9.710s
```

This second command was able to perform 0.671s faster than the previous command, possibly due to the increased processor time enjoyed by the process as a result of the reduction in nice value.

32. *Why do you think, user and sys time are not so different when compared to the values obtained after the first run of cruncher?*

The processing time is the factor affected by the reduction in the process' nice value, the time it takes for the system to read the file and the time it takes to process system calls will be the same.

33. *Note the actual outputs for real, user and sys time again. Comment on any improvement in performance.*

```
[root@UWS ~]# time nice -n -20 ./cruncher > /dev/null
real    0m14.649s
user    0m4.582s
sys     0m9.965s
[root@UWS ~]#
```

There was no real effect on performance, presumably because there are so few active processes running on the system.

35. *Note the actual outputs of the time command for real, user and sys. How do the time differences between nice 19 and nice -20 compare to the time differences on a quiet system?*

```
[root@UWS ~]# time nice -n 19 ./cruncher > /dev/null &
[2] 9044
[root@UWS ~]# time nice -n -20 ./cruncher > /dev/null &
[3] 9110
[root@UWS ~]#
real    0m13.012s
user    0m4.497s
sys     0m8.321s

real    0m27.496s
user    0m4.467s
sys     0m9.770s
```

The `nice -20` command completed roughly twice as fast as the `nice 19` command.

36. *Note the output of the renice command.*

```
[root@UWS ~]# time nice -n 19 ./cruncher > /dev/null &
[2] 9850
[root@UWS ~]# renice -20 9850
9850 (process ID) old priority 0, new priority -20
[root@UWS ~]#
real    0m14.696s
user    0m4.178s
sys     0m9.481s
```

37. *Estimate the performance gain of the above action.*

After reconfiguring the nice value, the process runs about twice as fast as it would with its initial nice value.

38. *Which file contains a list of users who are allowed to submit requests to the crontab facility?*

If it exists, `cron.allow`.

39. *Which file contains a list of users who are NOT allowed to submit request to the crontab facility? If a user's name appears in both lists, which list takes precedence?*

`cron.deny`, and `cron.allow` respectively.

40. *Explain the result of the above command in detail.*

```
[root@UWS etc]# du -k /home
4       /home/student/Desktop
4       /home/student/.config/procps
8       /home/student/.config
20      /home/student
24      /home
```

This command lists the sizes of a directory and its subdirectory in a given unit - in this case, `KB`.

41. *What does the piping in the `sort –nr` and the `head –5` do to the `du -k /home` output? Explain in detail.*

- `sort –nr` sorts some input in reverse numerical order
- `head -5` receives a test stream and outputs the top five lines of that stream

42. *Note the output of the above command.*

```
[root@UWS etc]# crontab -l
5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 * * * * du -k /home | sort -nr | head
-5 >> /root/test.dat
```

43. *What happens to the size of /root/test.dat over time? Do you see any danger in having a steadily growing output file in a directory?*

The cronjob uses the append operator `>>`, meaning every five minutes another five lines will be added to the file, an disk space, while cheap, is finite!

44. *What would be the entry in /tmp/entries to perform the command once every day at 23:15?*

```
15 23 * * * du -k /home | sort -nr | head -5 >> /root/test.dat
```

45. *What would be required to perform it at 4:30pm on the 21 st of January every year?*

```
30 16 21 1 * du -k /home | sort -nr | head -5 >> /root/test.dat
```

46. *Find an alternative cron syntax that could be used to specify that a job be performed every 5 minutes. Note the alternative notation as the answer in your logbook.*

```
*/5 * * * * * du -k /home | sort -nr | head -5 >> /root/test.dat
```

47. *In which different time-periods are the system cron-jobs ordered?*

```
[root@UWS etc]# ls -lah | grep cron
drwxr-xr-x    2 root     root          60 Nov 11  2020 cron.d
drwxr-xr-x    2 root     root          37 Nov 11  2020 cron.daily
drwxr-xr-x    2 root     root          37 Nov 11  2020 cron.hourly
drwxr-xr-x    2 root     root          37 Nov 11  2020 cron.monthly
drwxr-xr-x    2 root     root          37 Nov 11  2020 cron.weekly
```

- daily
- hourly
- monthly
- weekly

48. *Note an example of a cron-job service task that is found in the cron.daily directory.*

```
[root@UWS etc]# ls -lah cron.daily/
total 8
drwxr-xr-x    2 root     root          37 Nov 11  2020 .
drwxr-xr-x   17 root     root        1.3K Dec 10  2020 ..
```

There aren't any!

49. *What is the exact meaning of the -r character in the crontab -r command line?*

The -r option causes the current crontab to be removed.