SCR2043 OPERATING SYSTEMS

Name	Aland Fryad	Marks
Student ID :	qiu23-0457	
Section	: Software Engineering	

This lab assessment is designed to test your understanding and skills on some basic concepts and tools related to process monitoring and management in operating system. Please follow the instructions carefully and submit your answers in this word document and rename the file as os-lab-assessment02-studentname-matricno.docx.

Essential Steps Before Starting Lab Assessment 2:

1. Download necessary source codes:

   Use the wget command to retrieve the following source code files to your Linux (or WSL or MacOS) environment:

```
wget -O mainprocess.c https://rebrand.ly/mainprocess_c wget -O
subprocess1.c https://rebrand.ly/subprocess1_c wget -O
subprocess2.c https://rebrand.ly/subprocess2_c
```

2. Compile the source files:

   Use the gcc compiler to create executable files from the source code.

```
gcc mainprocess.c -o mainprocess gcc
subprocess1.c -o subprocess1 gcc
subprocess2.c -o subprocess2
```

3. Execute the dummy processes:
   Run all the dummy processes

```
./mainprocess &
```

   Press enter two times.

4. The dummy processes are running for 2 hours. If you took longer than 2 hours on questions 1-9, please restart the main process with ./mainprocess &.

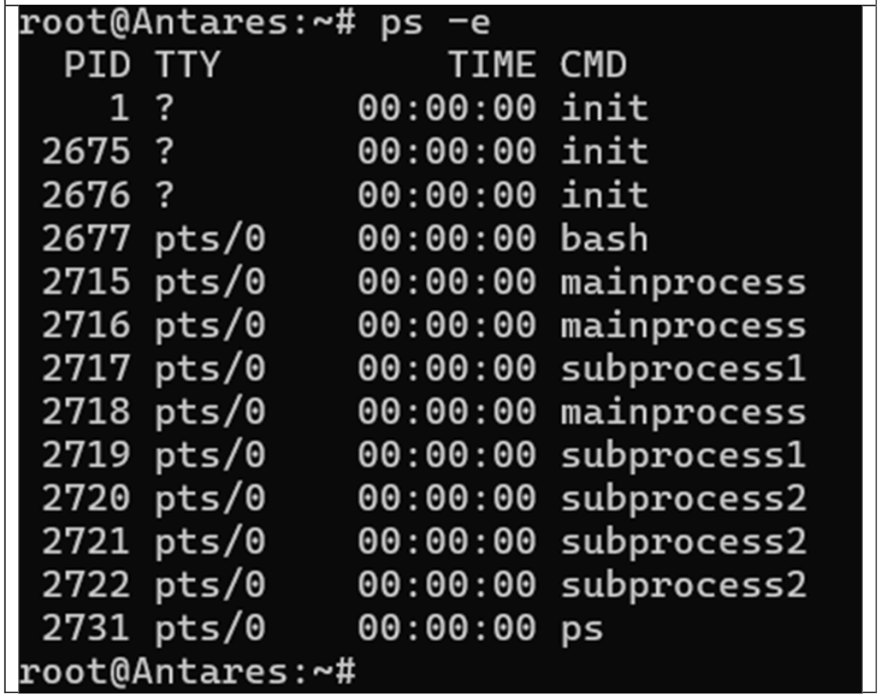
Lab Assessment 2 : Linux Process Monitoring and Management

Instructions:

1. Carefully execute each command as instructed in the questions.
2. Write down the exact command used for each task.
3. Capture a screenshot of the command's output.

Question 1

Use the ps command with the appropriate option to display a complete list of all running processes within the Linux operating system.

Command
ps -e
Output


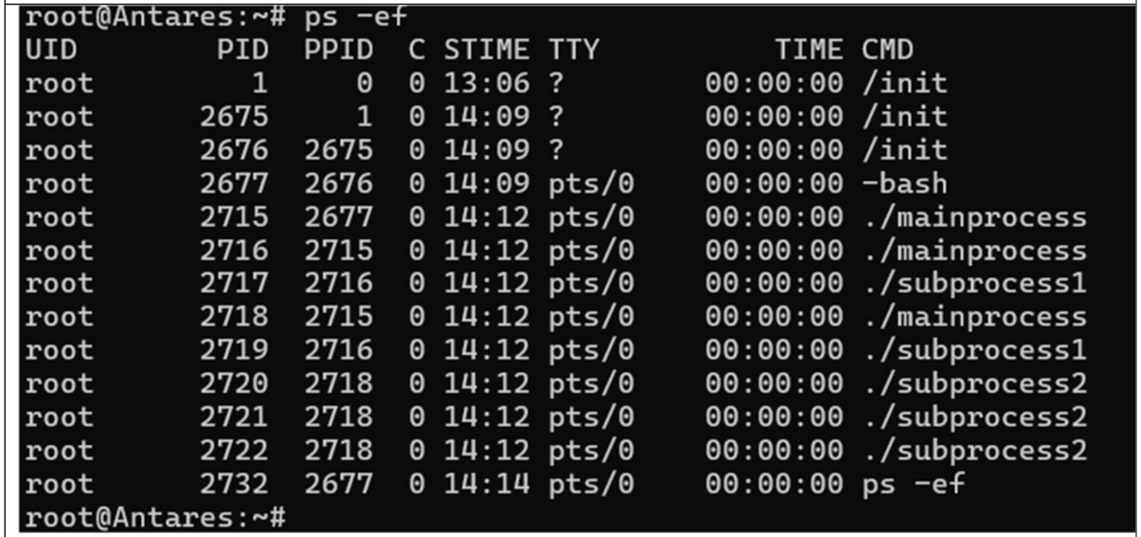
```
root@Antares:~# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
 2675 ?        00:00:00 init
 2676 ?        00:00:00 init
 2677 pts/0    00:00:00 bash
 2715 pts/0    00:00:00 mainprocess
 2716 pts/0    00:00:00 mainprocess
 2717 pts/0    00:00:00 subprocess1
 2718 pts/0    00:00:00 mainprocess
 2719 pts/0    00:00:00 subprocess1
 2720 pts/0    00:00:00 subprocess2
 2721 pts/0    00:00:00 subprocess2
 2722 pts/0    00:00:00 subprocess2
 2731 pts/0    00:00:00 ps
root@Antares:~#
```

Question 2

Employ the ps command with necessary options to unveil comprehensive details about each running process.

Command
ps -ef

Output

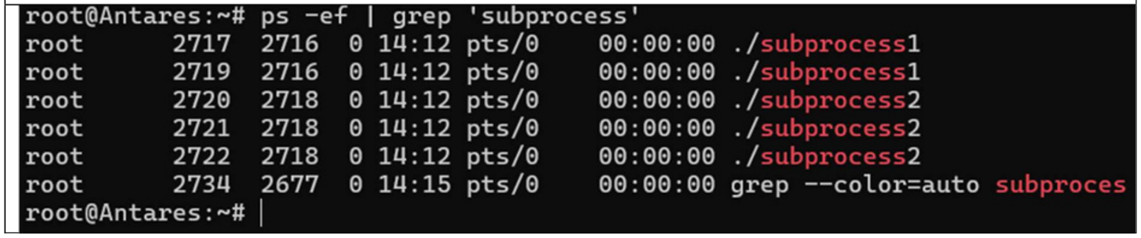


```
root@Antares:~# ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 13:06 ?        00:00:00 /init
root        2675     1  0 14:09 ?        00:00:00 /init
root        2676  2675  0 14:09 ?        00:00:00 /init
root        2677  2676  0 14:09 pts/0    00:00:00 -bash
root        2715  2677  0 14:12 pts/0    00:00:00 ./mainprocess
root        2716  2715  0 14:12 pts/0    00:00:00 ./mainprocess
root        2717  2716  0 14:12 pts/0    00:00:00 ./subprocess1
root        2718  2715  0 14:12 pts/0    00:00:00 ./mainprocess
root        2719  2716  0 14:12 pts/0    00:00:00 ./subprocess1
root        2720  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2721  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2722  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2732  2677  0 14:14 pts/0    00:00:00 ps -ef
root@Antares:~#
```

Question 3

Use the ps command with some tools to only list processes named "subprocess" and show some info about them.

Command
ps -ef | grep ‘subprocess’
Output



```
root@Antares:~# ps -ef | grep 'subprocess'
root        2717  2716  0 14:12 pts/0    00:00:00 ./subprocess1
root        2719  2716  0 14:12 pts/0    00:00:00 ./subprocess1
root        2720  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2721  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2722  2718  0 14:12 pts/0    00:00:00 ./subprocess2
root        2734  2677  0 14:15 pts/0    00:00:00 grep --color=auto subproces
root@Antares:~# |
```

Question 4

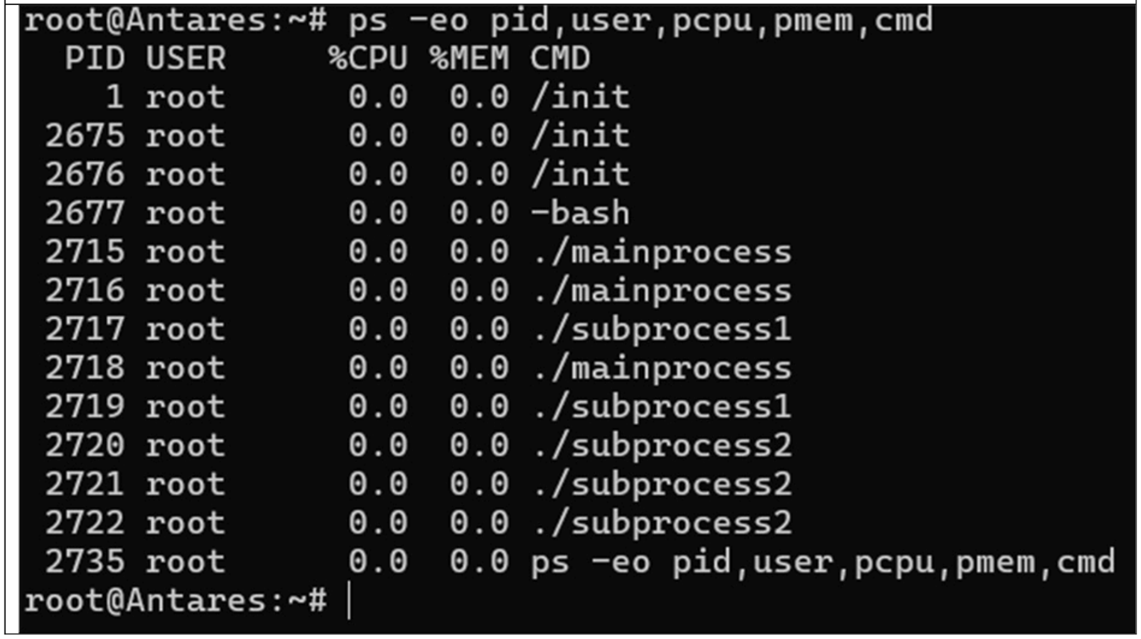
Execute the ps command, specifying options that reveal only the following columns:

- Process ID (pid)
- Owner of the process (user)
- CPU percentage (pcpu)
- Memory percentage (pmem)
- Command (cmd)

Command

ps -eo pid,user,pcpu,pmem,cmd

Output



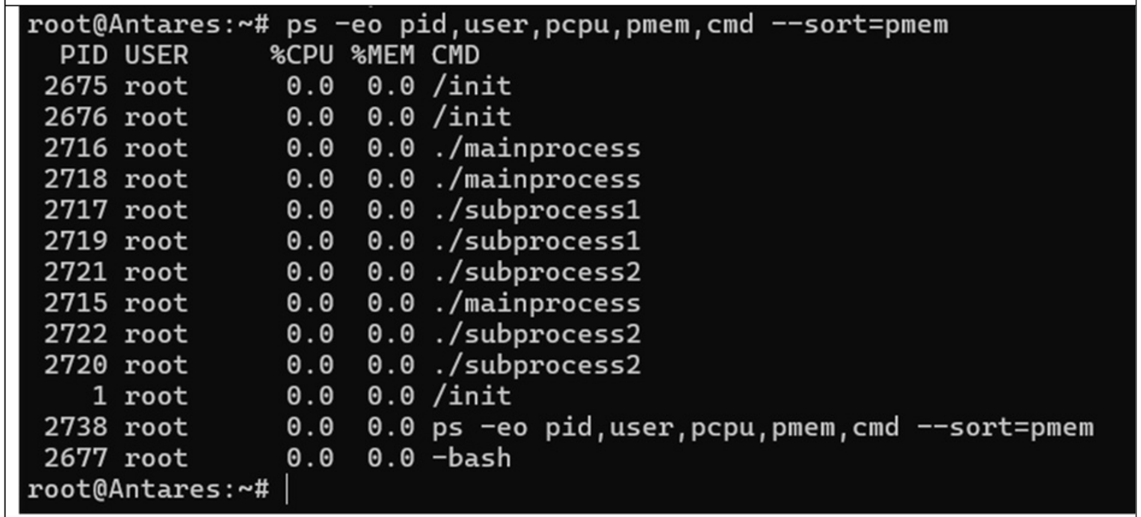
```
root@Antares:~# ps -eo pid,user,pcpu,pmem,cmd
  PID USER     %CPU %MEM CMD
    1 root      0.0  0.0 /init
 2675 root      0.0  0.0 /init
 2676 root      0.0  0.0 /init
 2677 root      0.0  0.0 -bash
 2715 root      0.0  0.0 ./mainprocess
 2716 root      0.0  0.0 ./mainprocess
 2717 root      0.0  0.0 ./subprocess1
 2718 root      0.0  0.0 ./mainprocess
 2719 root      0.0  0.0 ./subprocess1
 2720 root      0.0  0.0 ./subprocess2
 2721 root      0.0  0.0 ./subprocess2
 2722 root      0.0  0.0 ./subprocess2
 2735 root      0.0  0.0 ps -eo pid,user,pcpu,pmem,cmd
root@Antares:~# |
```

Question 5
Building on the ps command used in Question 4, can you add an option to sort the listed processes by their memory usage (pmem)?

Command

ps -eo pid,user,pcpu,pmem,cmd –-sort=pmem

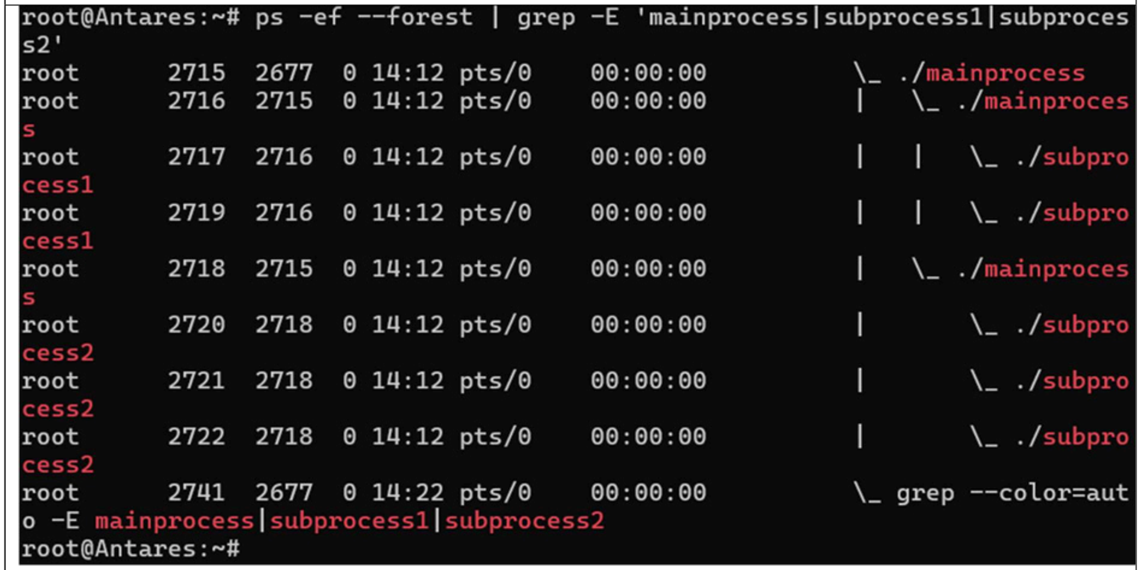
Output



```
root@Antares:~# ps -eo pid,user,pcpu,pmem,cmd --sort=pmem
  PID USER     %CPU %MEM CMD
 2675 root      0.0  0.0 /init
 2676 root      0.0  0.0 /init
 2716 root      0.0  0.0 ./mainprocess
 2718 root      0.0  0.0 ./mainprocess
 2717 root      0.0  0.0 ./subprocess1
 2719 root      0.0  0.0 ./subprocess1
 2721 root      0.0  0.0 ./subprocess2
 2715 root      0.0  0.0 ./mainprocess
 2722 root      0.0  0.0 ./subprocess2
 2720 root      0.0  0.0 ./subprocess2
    1 root      0.0  0.0 /init
 2738 root      0.0  0.0 ps -eo pid,user,pcpu,pmem,cmd --sort=pmem
 2677 root      0.0  0.0 -bash
root@Antares:~# |
```

Question 6
Construct a command using ps, suitable options, and any additional tools to visualize the hierarchical structure (tree-like) of the following processes:

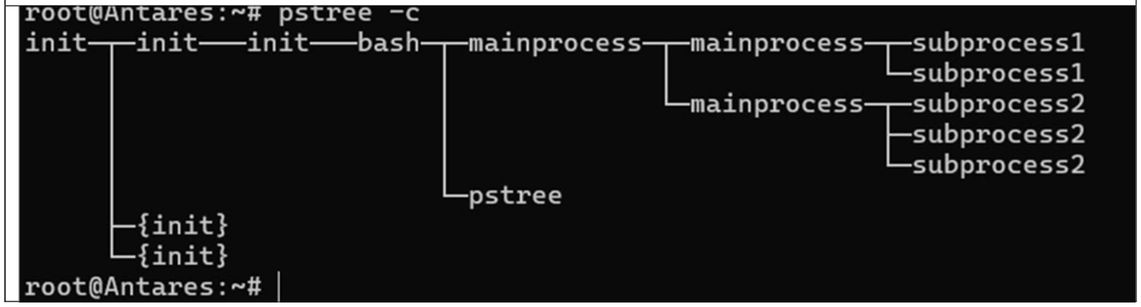
- ☐ "mainprocess"
- ☐ "subprocess1"
- ☐ "subprocess2"

Command
ps -ef --forest | grep -E 'mainprocess|subprocess1|subprocess2'
Output


```
root@Antares:~# ps -ef --forest | grep -E 'mainprocess|subprocess1|subproces
s2'
root      2715  2677  0 14:12 pts/0    00:00:00          \_ ./mainprocess
root      2716  2715  0 14:12 pts/0    00:00:00          |   \_ ./mainproces
s
root      2717  2716  0 14:12 pts/0    00:00:00          |   |   \_ ./subpro
cess1
root      2719  2716  0 14:12 pts/0    00:00:00          |   |   \_ ./subpro
cess1
root      2718  2715  0 14:12 pts/0    00:00:00          |   \_ ./mainproces
s
root      2720  2718  0 14:12 pts/0    00:00:00          |       \_ ./subpro
cess2
root      2721  2718  0 14:12 pts/0    00:00:00          |       \_ ./subpro
cess2
root      2722  2718  0 14:12 pts/0    00:00:00          |       \_ ./subpro
cess2
root      2741  2677  0 14:22 pts/0    00:00:00          \_ grep --color=aut
o -E mainprocess|subprocess1|subprocess2
root@Antares:~#
```

Question 7

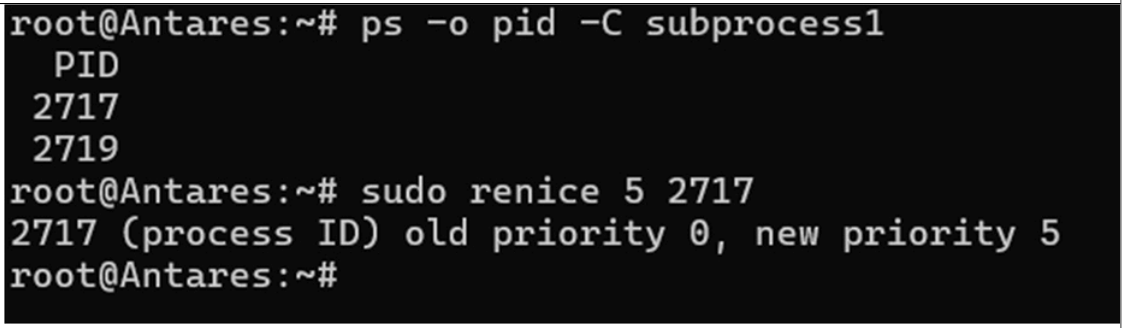
Use pstree command with option that show the number of threads to each process.

Command
pstree -c
Output


```
root@Antares:~# pstree -c
init─┬─init───init───bash─┬─mainprocess─┬─mainprocess─┬─subprocess1
     │                    │             │             └─subprocess1
     │                    │             └─mainprocess─┬─subprocess2
     │                    │                           ├─subprocess2
     │                    │                           └─subprocess2
     │                    └─pstree
     ├─{init}
     └─{init}
root@Antares:~#
```


Question 8

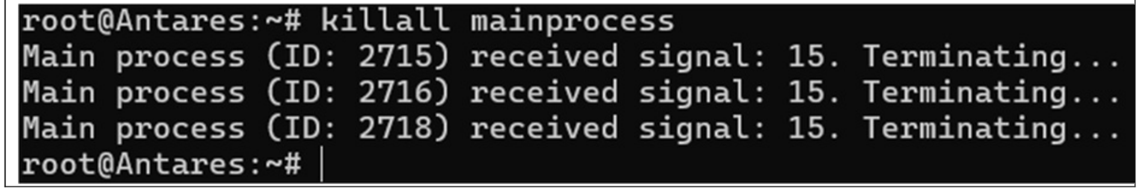
Use renice command to change priority level of one of process “subprocess1”.

Command
sudo renice 5 2717
Output


```
root@Antares:~# ps -o pid -C subprocess1
  PID
 2717
 2719
root@Antares:~# sudo renice 5 2717
2717 (process ID) old priority 0, new priority 5
root@Antares:~#
```

Question 9

Terminate all running processes with the name “mainprocess”.

Command
killall mainprocess
Output


```
root@Antares:~# killall mainprocess
Main process (ID: 2715) received signal: 15. Terminating...
Main process (ID: 2716) received signal: 15. Terminating...
Main process (ID: 2718) received signal: 15. Terminating...
root@Antares:~# |
```

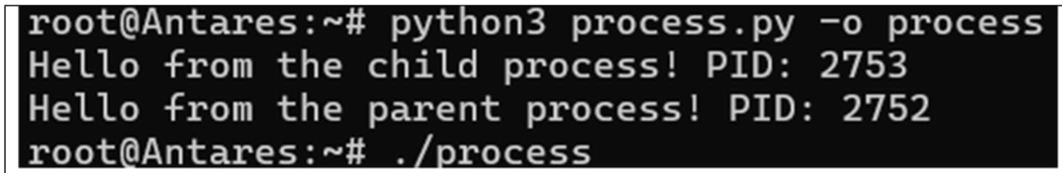
Question 10

Write a short C or Python code (choose only one language) demonstrating multiprocessing with fork() and wait(). Compile and/or run the code. Show the output.

Source Code:

```
Nano process.py

import multiprocessing
import os
def child_process():

    print(f"Hello from the child process! PID: {os.getpid()}")

if __name__ == "__main__":
    # Create a new process
    process = multiprocessing.Process(target=child_process)
    process.start() # Start the child process
    process.join() # Wait for the child process to finish

    print(f"Hello from the parent process! PID: {os.getpid()}")


python3 process.py -o process
```

Output:



```
root@Antares:~# python3 process.py -o process
Hello from the child process! PID: 2753
Hello from the parent process! PID: 2752
root@Antares:~# ./process
```