SCR2043 OPERATING SYSTEMS

Name : Chua Jia Lin
Student ID : A23CS0069
Section : 2

Marks

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 &`.

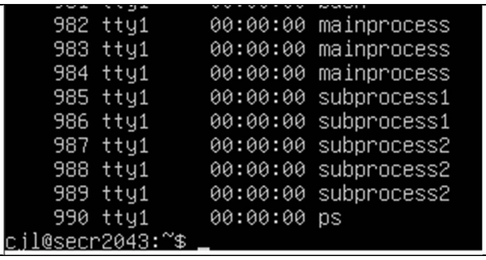
Lab Assessment 2 : Linux Process Monitoring and Management

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

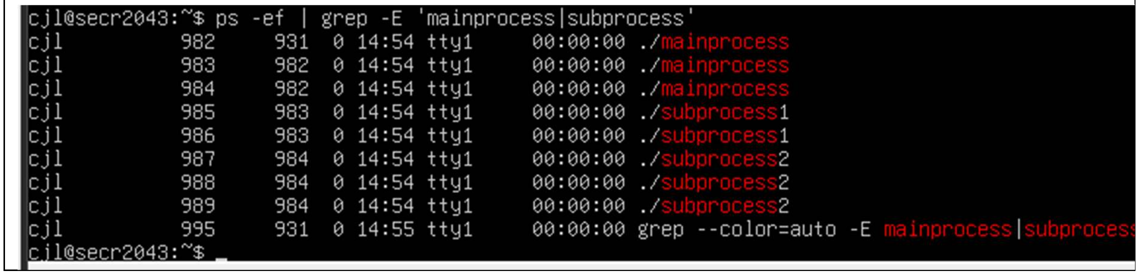
Use the `ps` command with the appropriate option to display a complete list of all running processes within the Linux operating system.

Command
ps -e
Output


```
  982 tty1     00:00:00 mainprocess
  983 tty1     00:00:00 mainprocess
  984 tty1     00:00:00 mainprocess
  985 tty1     00:00:00 subprocess1
  986 tty1     00:00:00 subprocess1
  987 tty1     00:00:00 subprocess2
  988 tty1     00:00:00 subprocess2
  989 tty1     00:00:00 subprocess2
  990 tty1     00:00:00 ps
cjl@secr2043:~$ _
```

Question 2

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

Command
ps -ef | grep -E 'mainprocess|subprocess'
Output


```
cjl@secr2043:~$ ps -ef | grep -E 'mainprocess|subprocess'
cjl         982     931  0 14:54 tty1     00:00:00 ./mainprocess
cjl         983     982  0 14:54 tty1     00:00:00 ./mainprocess
cjl         984     982  0 14:54 tty1     00:00:00 ./mainprocess
cjl         985     983  0 14:54 tty1     00:00:00 ./subprocess1
cjl         986     983  0 14:54 tty1     00:00:00 ./subprocess1
cjl         987     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl         988     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl         989     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl         995     931  0 14:55 tty1     00:00:00 grep --color=auto -E mainprocess|subprocess
cjl@secr2043:~$ _
```

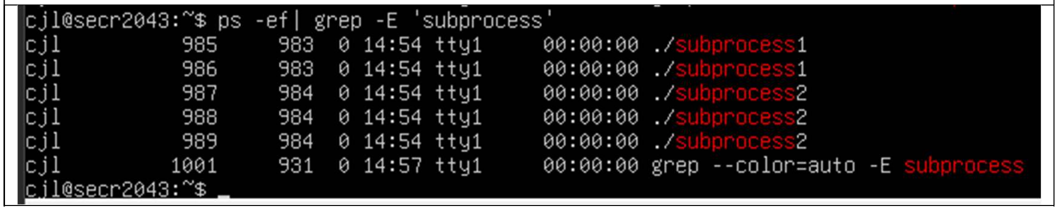
Question 3

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

Command

ps -ef| grep -E ‘subprocess’

Output



```
cjl@secr2043:~$ ps -ef| grep -E 'subprocess'
cjl          985     983  0 14:54 tty1     00:00:00 ./subprocess1
cjl          986     983  0 14:54 tty1     00:00:00 ./subprocess1
cjl          987     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl          988     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl          989     984  0 14:54 tty1     00:00:00 ./subprocess2
cjl         1001     931  0 14:57 tty1     00:00:00 grep --color=auto -E subprocess
cjl@secr2043:~$ _
```

Question 4

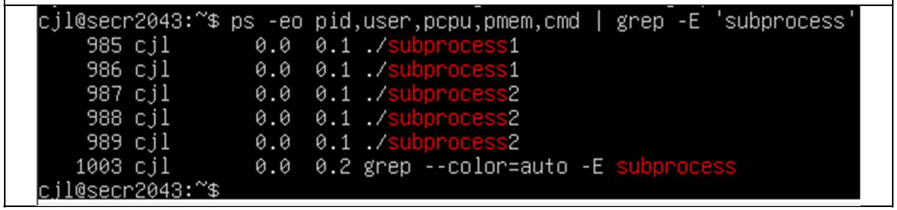
Execute the `ps` command, specifying options that reveal only the following columns:

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

Command

ps -eo pid,user,pcpu,pmem,cmd | grep -E ‘subprocess’

Output



```
cjl@secr2043:~$ ps -eo pid,user,pcpu,pmem,cmd | grep -E 'subprocess'
    985 cjl       0.0  0.1 ./subprocess1
    986 cjl       0.0  0.1 ./subprocess1
    987 cjl       0.0  0.1 ./subprocess2
    988 cjl       0.0  0.1 ./subprocess2
    989 cjl       0.0  0.1 ./subprocess2
   1003 cjl       0.0  0.2 grep --color=auto -E subprocess
cjl@secr2043:~$
```

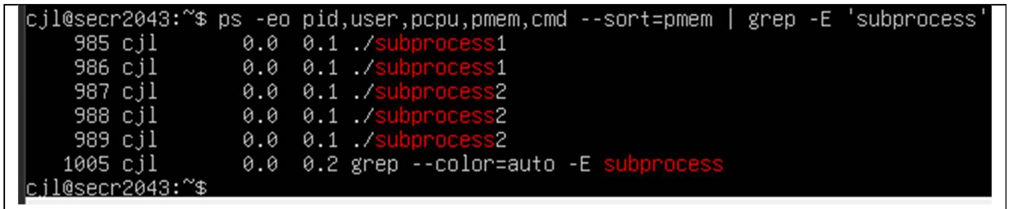
Question 5

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 | grep -E ‘subprocess’

Output

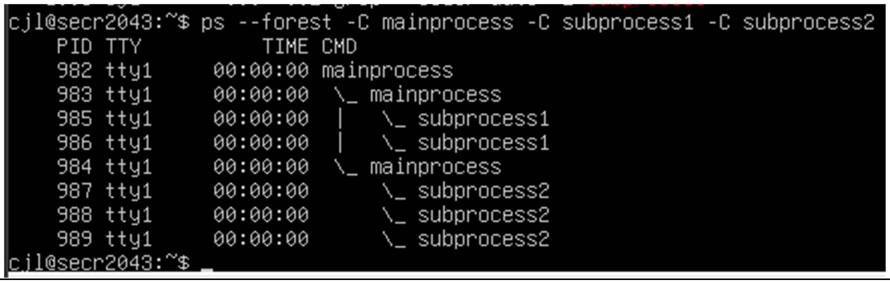


```
cjl@secr2043:~$ ps -eo pid,user,pcpu,pmem,cmd --sort=pmem | grep -E 'subprocess'
    985 cjl       0.0  0.1 ./subprocess1
    986 cjl       0.0  0.1 ./subprocess1
    987 cjl       0.0  0.1 ./subprocess2
    988 cjl       0.0  0.1 ./subprocess2
    989 cjl       0.0  0.1 ./subprocess2
   1005 cjl       0.0  0.2 grep --color=auto -E subprocess
cjl@secr2043:~$
```

Question 6

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

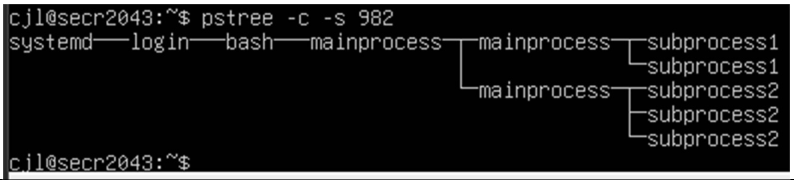
- "mainprocess"
- "subprocess1"
- "subprocess2"

Command
ps --forest -C mainprocess -C subprocess1 -C subprocess2
Output


```
cjl@secr2043:~$ ps --forest -C mainprocess -C subprocess1 -C subprocess2
    PID TTY          TIME CMD
    982 tty1     00:00:00 mainprocess
    983 tty1     00:00:00  \_ mainprocess
    985 tty1     00:00:00  |   \_ subprocess1
    986 tty1     00:00:00  |   \_ subprocess1
    984 tty1     00:00:00  \_ mainprocess
    987 tty1     00:00:00      \_ subprocess2
    988 tty1     00:00:00      \_ subprocess2
    989 tty1     00:00:00      \_ subprocess2
cjl@secr2043:~$ _
```

Question 7

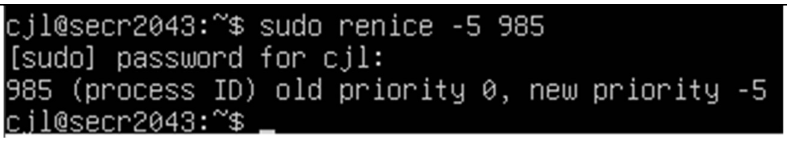
Use pstree command with option that show the number of threads to each process.

Command
pstree -c -s 982
Output


```
cjl@secr2043:~$ pstree -c -s 982
systemd───login───bash───mainprocess─┬─mainprocess─┬─subprocess1
                                     │             └─subprocess1
                                     └─mainprocess─┬─subprocess2
                                                   ├─subprocess2
                                                   └─subprocess2
cjl@secr2043:~$
```

Question 8

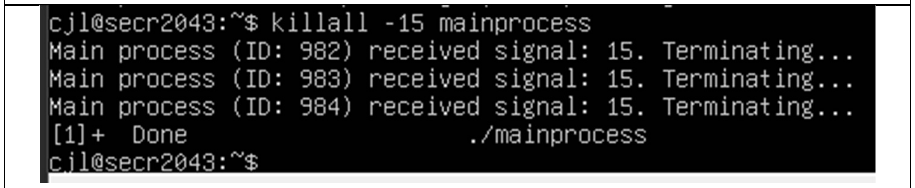
Use `renice` command to change priority level of one of process “subprocess1”.

Command
sudo renice -5 985
Output


```
cjl@secr2043:~$ sudo renice -5 985
[sudo] password for cjl:
985 (process ID) old priority 0, new priority -5
cjl@secr2043:~$
```

Question 9

Terminate all running processes with the name “mainprocess”.

Command
killall -15 mainprocess
Output


```
cjl@secr2043:~$ killall -15 mainprocess
Main process (ID: 982) received signal: 15. Terminating...
Main process (ID: 983) received signal: 15. Terminating...
Main process (ID: 984) received signal: 15. Terminating...
[1]+  Done                    ./mainprocess
cjl@secr2043:~$
```

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.

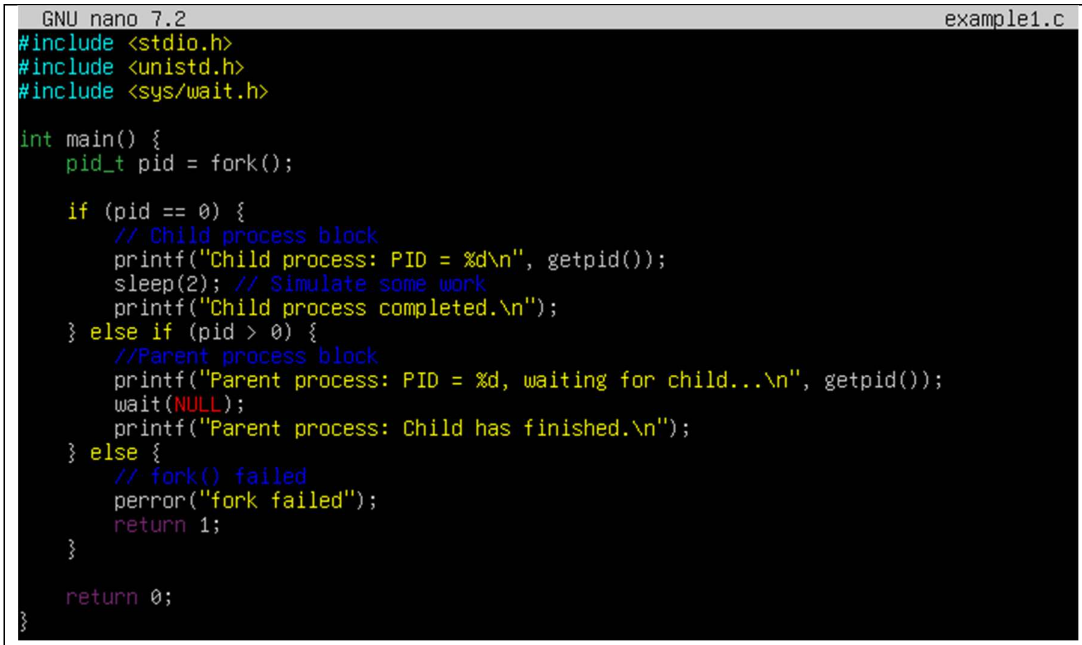
Source Code:

```
nano example1.c
gcc example1.c –o example1
./example1
```

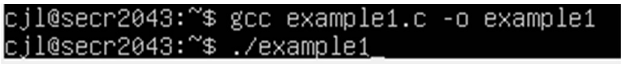
example1.c

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork(); // Create a new process

    if (pid == 0) {
        // Child process block
        printf("Child process: PID = %d\n", getpid());
        sleep(2); // Simulate some work
        printf("Child process completed.\n");
    } else if (pid > 0) {
        // Parent process block
        printf("Parent process: PID = %d, waiting for child...\n", getpid());
        wait(NULL); // Wait for child to finish
        printf("Parent process: Child has finished.\n");
    } else {
        // fork() failed
        perror("fork failed");
        return 1;
    }

    return 0;
}
```

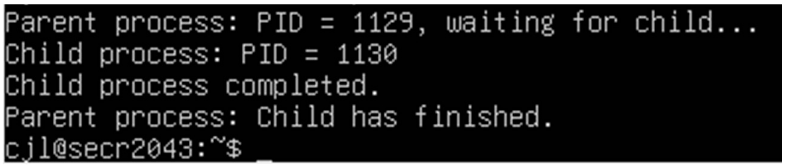
Output:



```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process block
        printf("Child process: PID = %d\n", getpid());
        sleep(2); // Simulate some work
        printf("Child process completed.\n");
    } else if (pid > 0) {
        //Parent process block
        printf("Parent process: PID = %d, waiting for child...\n", getpid());
        wait(NULL);
        printf("Parent process: Child has finished.\n");
    } else {
        // fork() failed
        perror("fork failed");
        return 1;
    }

    return 0;
}
```



```
cjl@secr2043:~$ gcc example1.c -o example1
cjl@secr2043:~$ ./example1_
```



```
Parent process: PID = 1129, waiting for child...
Child process: PID = 1130
Child process completed.
Parent process: Child has finished.
cjl@secr2043:~$ _
```