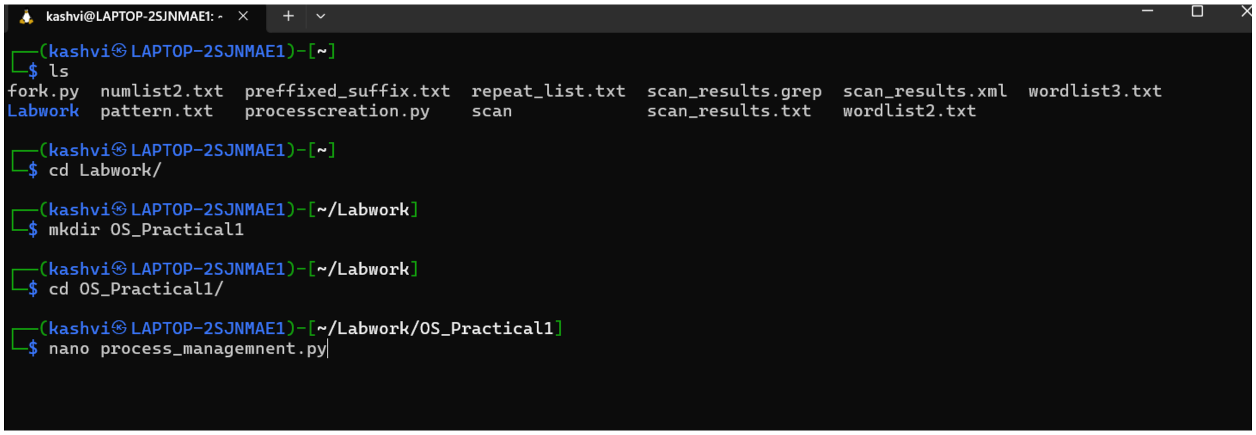
Operating Systems Lab Assignment 1-

Task1- Process Creation Utility

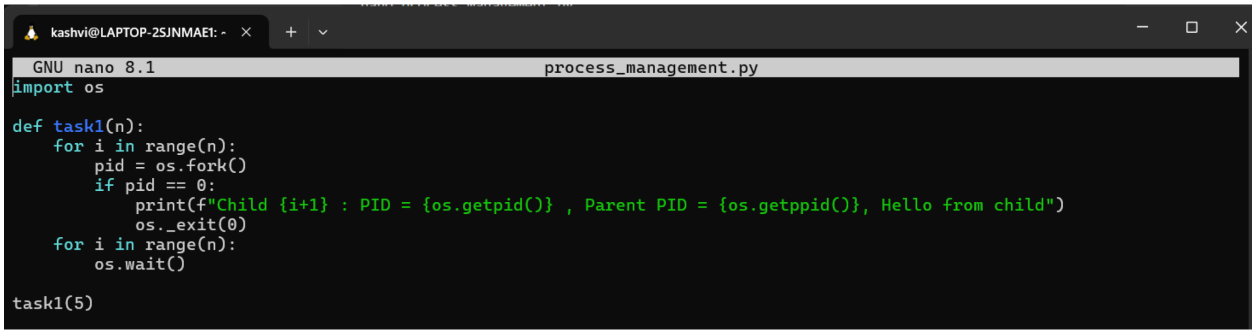
Write a Python program that creates N child processes using os.fork(). Each child prints:
- Its PID
- Its Parent PID
- A custom message

The parent should wait for all children using os.wait().

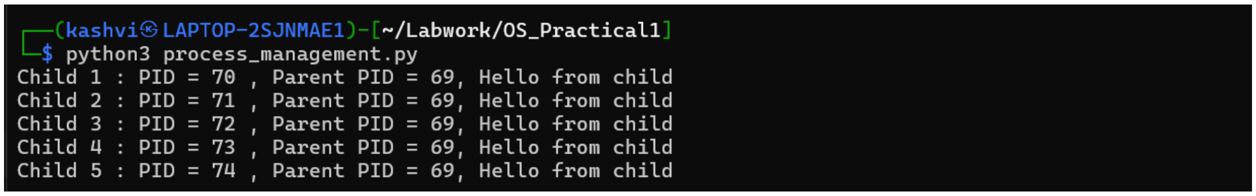
Sol.-



```
kashvi@LAPTOP-2SJNMAE1: ~

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~]
└─$ ls
fork.py   numlist2.txt  preffixed_suffix.txt  repeat_list.txt  scan_results.grep  scan_results.xml  wordlist3.txt
Labwork   pattern.txt   processcreation.py    scan             scan_results.txt   wordlist2.txt

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~]
└─$ cd Labwork/

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork]
└─$ mkdir OS_Practical1

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork]
└─$ cd OS_Practical1/

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ nano process_managemnent.py
```



```
kashvi@LAPTOP-2SJNMAE1: ~

  GNU nano 8.1                              process_management.py
import os

def task1(n):
    for i in range(n):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1} : PID = {os.getpid()} , Parent PID = {os.getppid()}, Hello from child")
            os._exit(0)
    for i in range(n):
        os.wait()

task1(5)
```

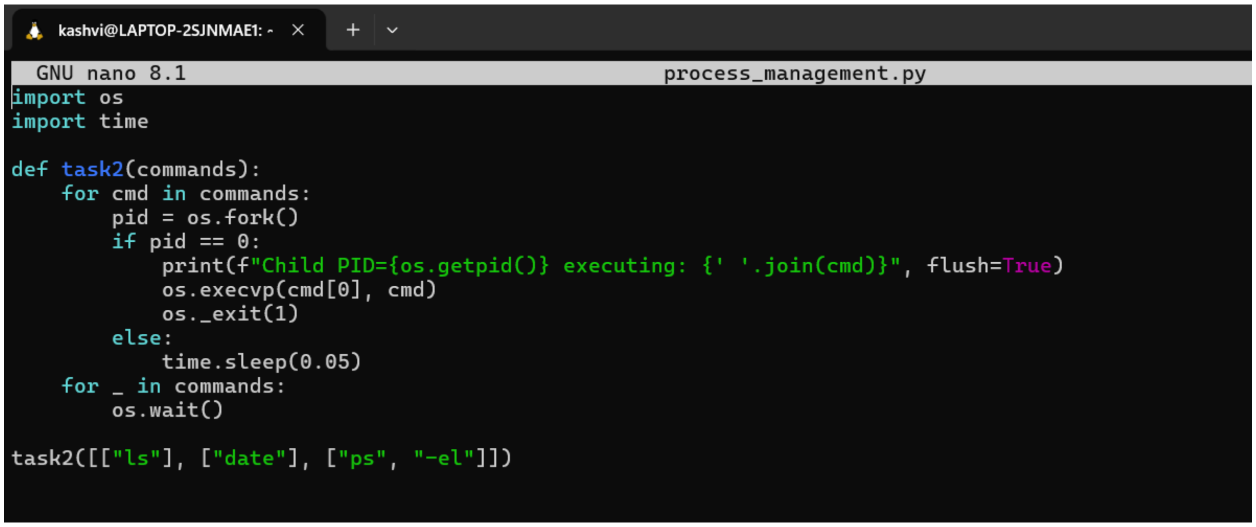


```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ python3 process_management.py
Child 1 : PID = 70 , Parent PID = 69, Hello from child
Child 2 : PID = 71 , Parent PID = 69, Hello from child
Child 3 : PID = 72 , Parent PID = 69, Hello from child
Child 4 : PID = 73 , Parent PID = 69, Hello from child
Child 5 : PID = 74 , Parent PID = 69, Hello from child
```

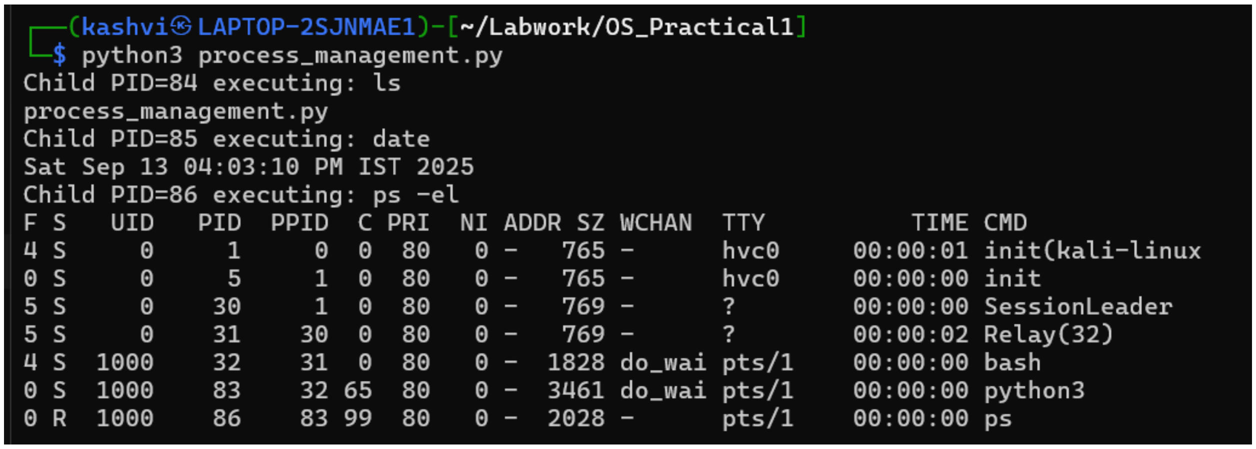
Task 2- Command Execution Using exec()

Modify Task 1 so that each child process executes a Linux command (ls, date, ps, etc.) using os.execvp() or subprocess.run().

Sol.-



```
 GNU nano 8.1                                           process_management.py
import os
import time

def task2(commands):
    for cmd in commands:
        pid = os.fork()
        if pid == 0:
            print(f"Child PID={os.getpid()} executing: {' '.join(cmd)}", flush=True)
            os.execvp(cmd[0], cmd)
            os._exit(1)
        else:
            time.sleep(0.05)
    for _ in commands:
        os.wait()

task2([["ls"], ["date"], ["ps", "-el"]])
```

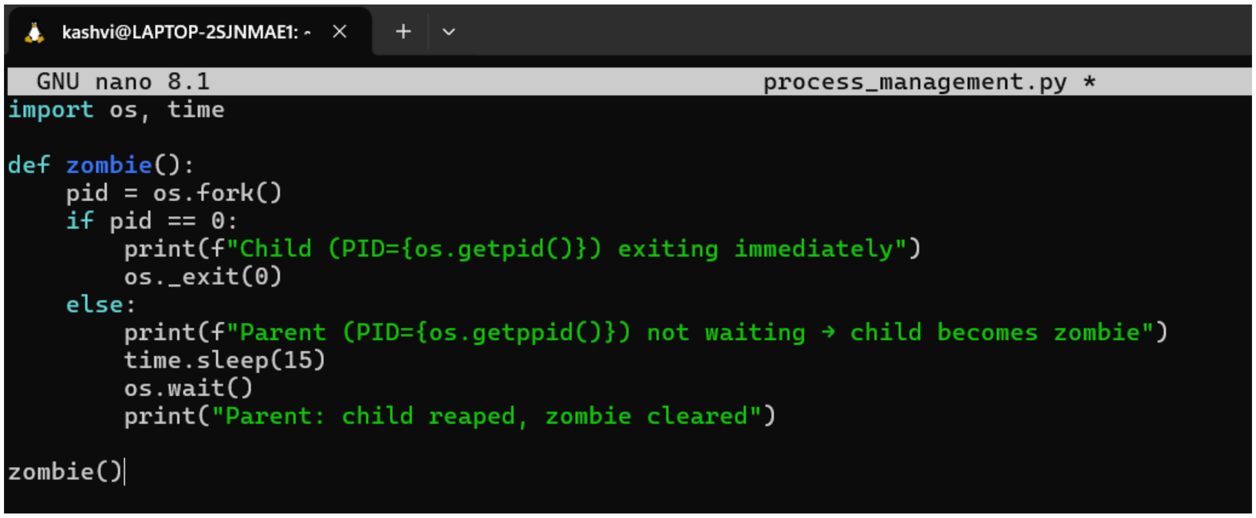


```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ python3 process_management.py
Child PID=84 executing: ls
process_management.py
Child PID=85 executing: date
Sat Sep 13 04:03:10 PM IST 2025
Child PID=86 executing: ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   765 -      hvc0     00:00:01 init(kali-linux
0 S     0     5     1  0  80   0 -   765 -      hvc0     00:00:00 init
5 S     0    30     1  0  80   0 -   769 -      ?        00:00:00 SessionLeader
5 S     0    31    30  0  80   0 -   769 -      ?        00:00:02 Relay(32)
4 S  1000    32    31  0  80   0 -  1828 do_wai pts/1    00:00:00 bash
0 S  1000    83    32 65  80   0 -  3461 do_wai pts/1    00:00:00 python3
0 R  1000    86    83 99  80   0 -  2028 -      pts/1    00:00:00 ps
```

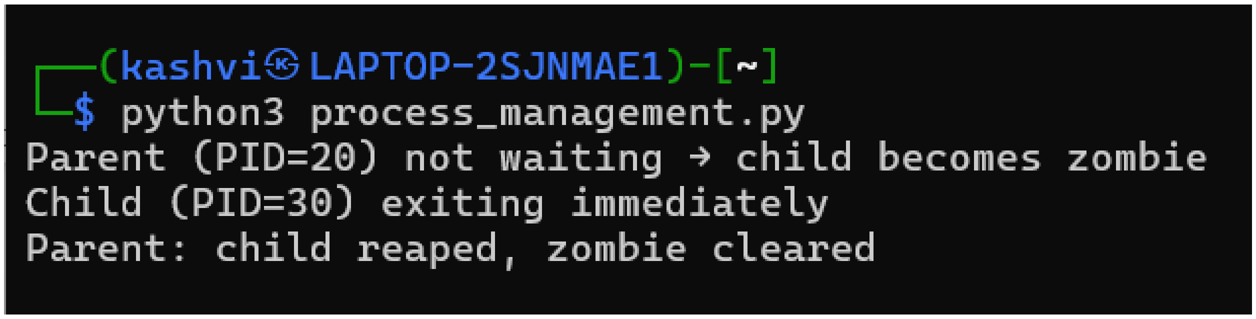
Task 3 - Zombie & Orphan Processes

Zombie: Fork a child and skip wait() in the parent.
Orphan: Parent exits before the child finishes.
Use ps -el | grep defunct to identify zombies.

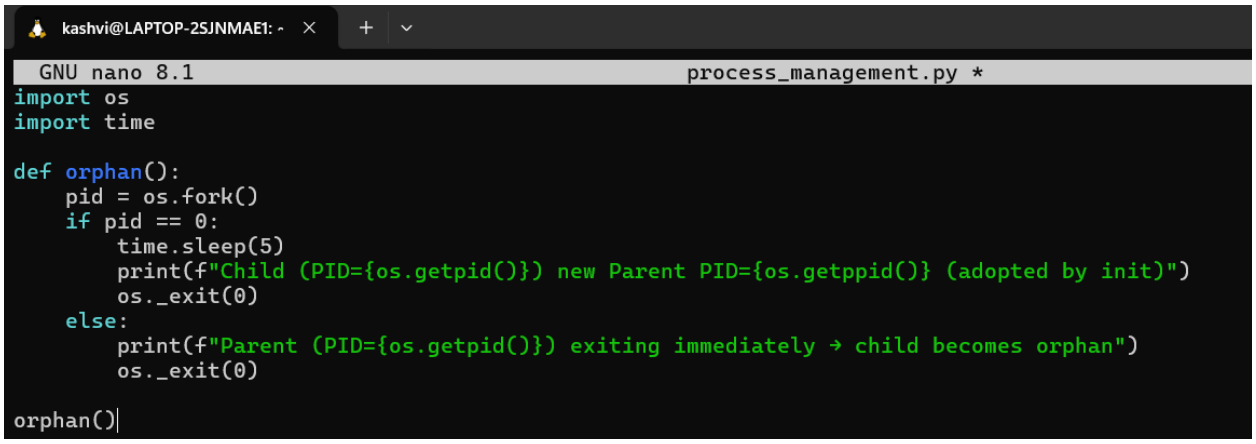
Sol.-



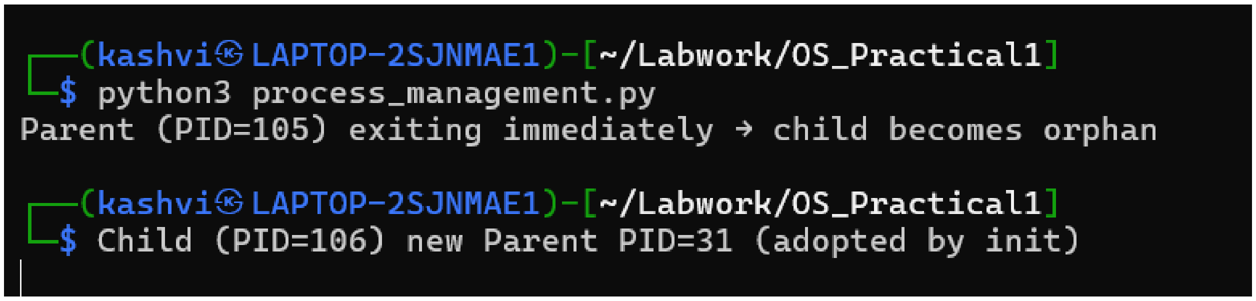
```
kashvi@LAPTOP-2SJNMAE1: ~
 GNU nano 8.1                                                process_management.py *
import os, time

def zombie():
    pid = os.fork()
    if pid == 0:
        print(f"Child (PID={os.getpid()}) exiting immediately")
        os._exit(0)
    else:
        print(f"Parent (PID={os.getppid()}) not waiting → child becomes zombie")
        time.sleep(15)
        os.wait()
        print("Parent: child reaped, zombie cleared")

zombie()
```



```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~]
└─$ python3 process_management.py
Parent (PID=20) not waiting → child becomes zombie
Child (PID=30) exiting immediately
Parent: child reaped, zombie cleared
```



```
kashvi@LAPTOP-2SJNMAE1: ~
 GNU nano 8.1                                             process_management.py *
import os
import time

def orphan():
    pid = os.fork()
    if pid == 0:
        time.sleep(5)
        print(f"Child (PID={os.getpid()}) new Parent PID={os.getppid()} (adopted by init)")
        os._exit(0)
    else:
        print(f"Parent (PID={os.getpid()}) exiting immediately → child becomes orphan")
        os._exit(0)

orphan()
```



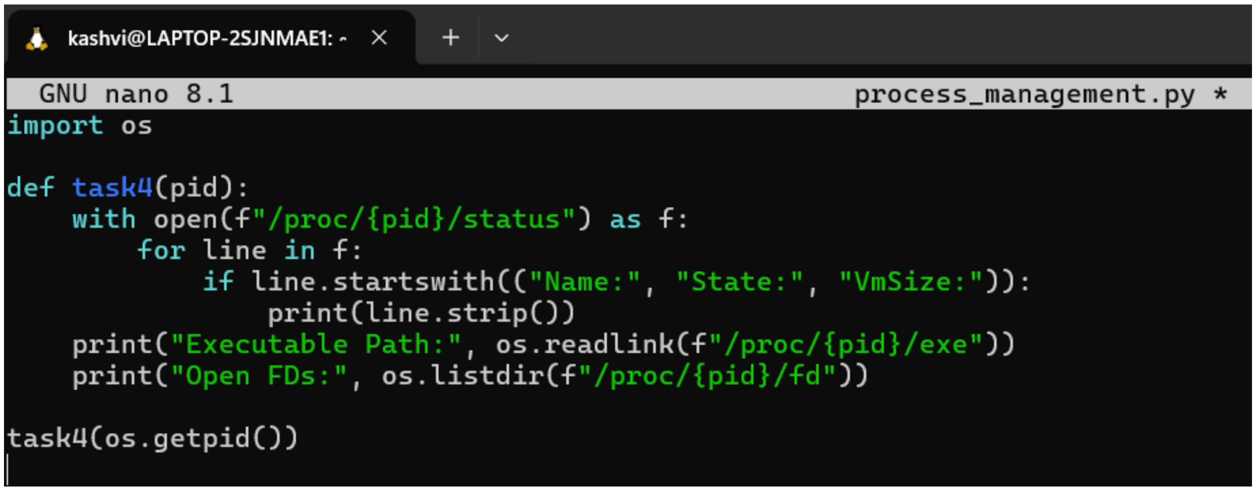
```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ python3 process_management.py
Parent (PID=105) exiting immediately → child becomes orphan

┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ Child (PID=106) new Parent PID=31 (adopted by init)
```

Task 4 - Inspecting Process Info from /proc

Take a PID as input. Read and print:

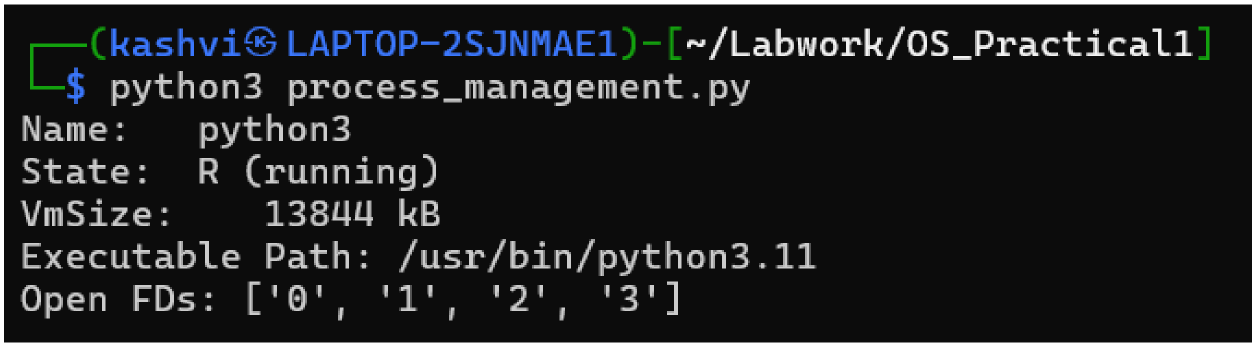
- Process name, state, memory usage from /proc/[pid]/status
- Executable path from /proc/[pid]/exe
- Open file descriptors from /proc/[pid]/fd

Sol.- -



kashvi@LAPTOP-2SJNMAE1: ~

```
  GNU nano 8.1                                          process_management.py *
import os

def task4(pid):
    with open(f"/proc/{pid}/status") as f:
        for line in f:
            if line.startswith(("Name:", "State:", "VmSize:")):
                print(line.strip())
    print("Executable Path:", os.readlink(f"/proc/{pid}/exe"))
    print("Open FDs:", os.listdir(f"/proc/{pid}/fd"))

task4(os.getpid())
```

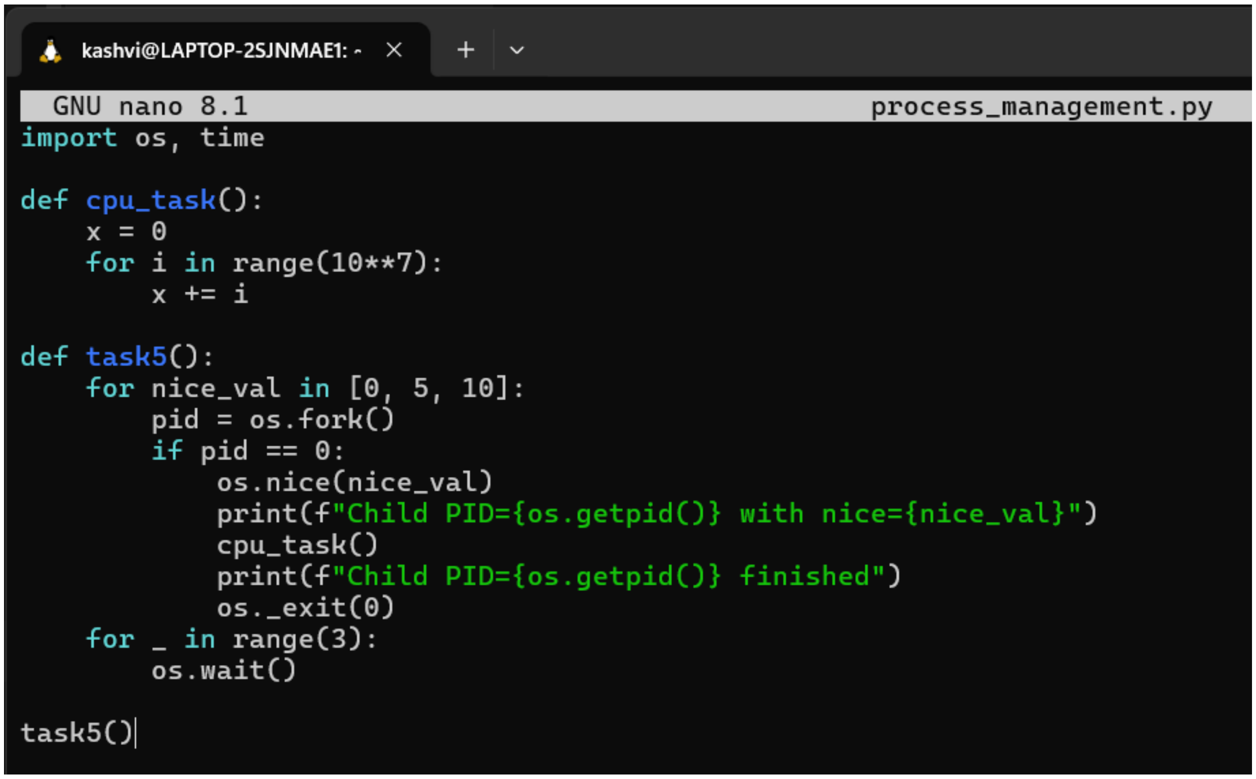


```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ python3 process_management.py
Name:   python3
State:  R (running)
VmSize:    13844 kB
Executable Path: /usr/bin/python3.11
Open FDs: ['0', '1', '2', '3']
```

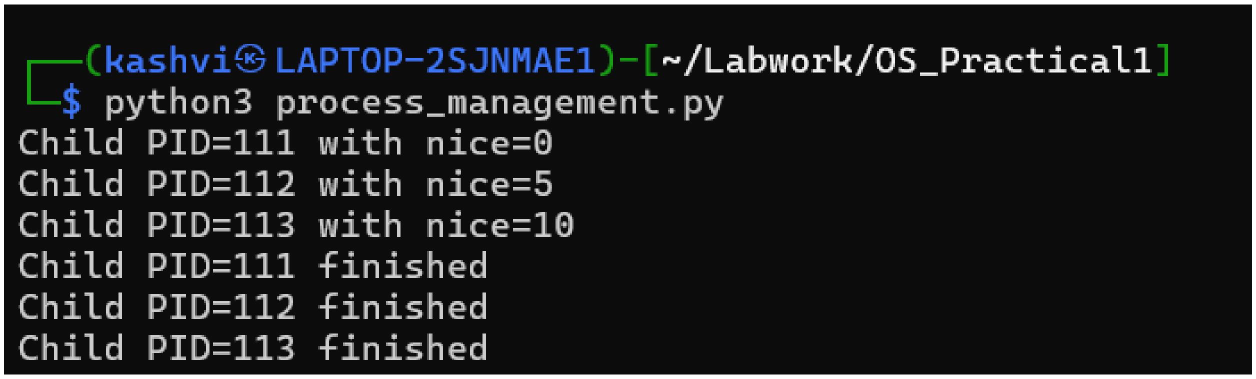
Task 5 - Process Prioritization

Create multiple CPU-intensive child processes. Assign different nice() values. Observe and log execution order to show scheduler impact.

Sol.-



```
  GNU nano 8.1                                        process_management.py
import os, time

def cpu_task():
    x = 0
    for i in range(10**7):
        x += i

def task5():
    for nice_val in [0, 5, 10]:
        pid = os.fork()
        if pid == 0:
            os.nice(nice_val)
            print(f"Child PID={os.getpid()} with nice={nice_val}")
            cpu_task()
            print(f"Child PID={os.getpid()} finished")
            os._exit(0)
    for _ in range(3):
        os.wait()

task5()
```



```
┌──(kashvi㉿LAPTOP-2SJNMAE1)-[~/Labwork/OS_Practical1]
└─$ python3 process_management.py
Child PID=111 with nice=0
Child PID=112 with nice=5
Child PID=113 with nice=10
Child PID=111 finished
Child PID=112 finished
Child PID=113 finished
```