

ASSIGNMENT -1

Experiment Title: Process Creation and Management Using Python OS Module

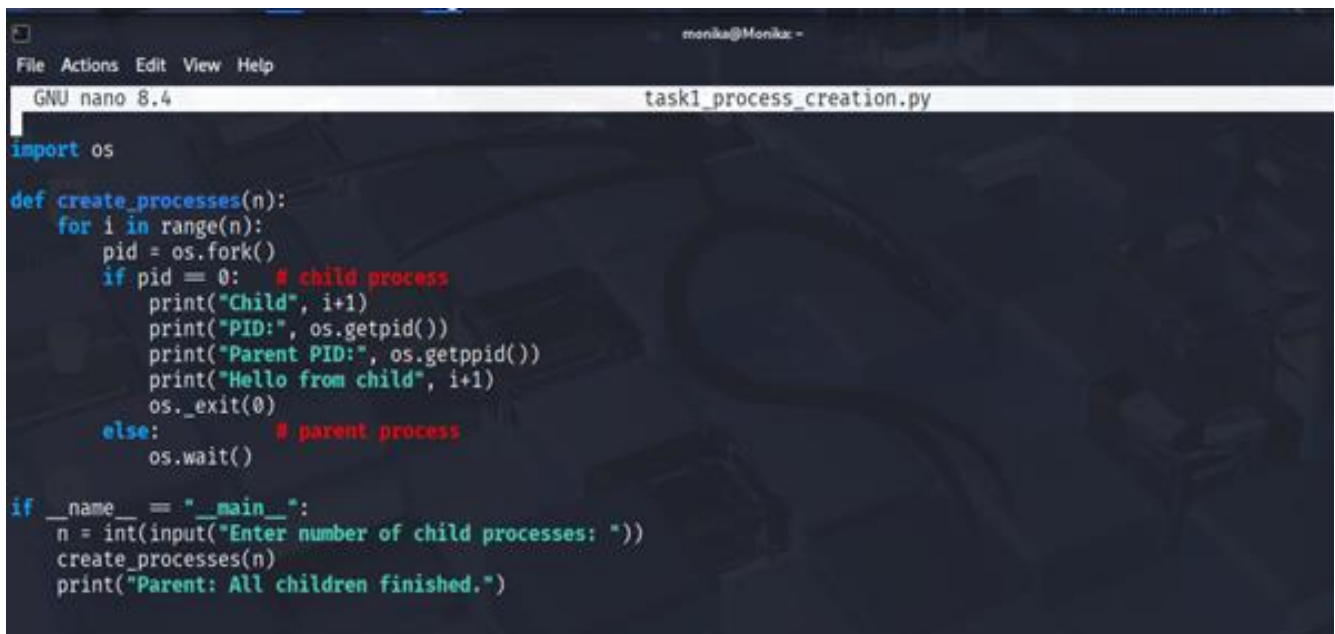
Task 1: Process Creation Utility

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().

Code:



```
import os

def create_processes(n):
    for i in range(n):
        pid = os.fork()
        if pid == 0: # child process
            print("Child", i+1)
            print("PID:", os.getpid())
            print("Parent PID:", os.getppid())
            print("Hello from child", i+1)
            os._exit(0)
        else: # parent process
            os.wait()

if __name__ == "__main__":
    n = int(input("Enter number of child processes: "))
    create_processes(n)
    print("Parent: All children finished.")
```

Output:



```
(monika@Monika)-[~]
$ python3 task1_process_creation.py

Enter number of child processes: 3
Child 1
PID: 22003
Parent PID: 21930
Hello from child 1
Child 2
PID: 22004
Parent PID: 21930
Hello from child 2
Child 3
PID: 22005
Parent PID: 21930
Hello from child 3
Parent: All children finished.
```

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()`.

Code:

```
monika@Monika ~  
File Actions Edit View Help  
GNU nano 8.4 task2_exec_commands.py  
import os  
  
def main():  
    n = int(input("Enter number of child processes: "))  
  
    # List of Linux commands each child will execute  
    commands = [["ls", "-l"], ["date"], ["ps"]]  
  
    for i in range(n):  
        pid = os.fork() # create a child process  
  
        if pid == 0: # Child process  
            print(f"\nChild {i+1}: PID = {os.getpid()}, Parent PID = {os.getppid()}")  
            print(f"Executing command: {commands[i % len(commands)]}")  
            os.execvp(commands[i % len(commands)][0], commands[i % len(commands)])  
        else: # Parent process  
            os.wait() # wait for child to finish  
  
    print("\nParent: All child processes have executed commands.")  
  
if __name__ == "__main__":  
    main()
```

Output:

```
(monika@Monika)~  
$ python3 task2_exec_commands.py  
Enter number of child processes: 3  
  
Child 1: PID = 57296, Parent PID = 57199  
Executing command: ['ls', '-l']  
total 52  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Desktop  
drwxr-xr-x 3 monika monika 4096 Sep  7 11:36 Documents  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Downloads  
-rw-rw-r-- 1 monika monika 12 Sep  4 16:55 fork.py.save  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Music  
drwxr-xr-x 2 monika monika 4096 Sep  7 11:31 Pictures  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Public  
-rw-rw-r-- 1 monika monika 524 Sep  5 20:07 task1_process_creation.py  
-rw-rw-r-- 1 monika monika 522 Sep  5 20:13 task1_process_creation.py.save  
-rw-rw-r-- 1 monika monika 521 Sep  5 20:13 task1_process_creation.py.save.1  
-rw-rw-r-- 1 monika monika 723 Sep  7 13:11 task2_exec_commands.py  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Templates  
drwxr-xr-x 2 monika monika 4096 Sep  3 22:31 Videos  
  
Child 2: PID = 57297, Parent PID = 57199  
Executing command: ['date']  
Sunday 07 September 2025 01:15:57 PM IST  
  
Child 3: PID = 57298, Parent PID = 57199  
Executing command: ['ps']  
    PID TTY          TIME CMD  
  5787 pts/0    00:00:00 zsh  
  57199 pts/0    00:00:00 python3  
  57298 pts/0    00:00:00 ps  
  
Parent: All child processes have executed commands.
```

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.

Code:

```
GNU nano 8.4 task3_zombie_orphan.py *
import os
import time

# ZOMBIE PROCESS
print("—— ZOMBIE PROCESS ——")
pid = os.fork()

if pid == 0:
    # Child process
    print(f"Child (Zombie) PID = {os.getpid()}")
    os._exit(0) # Child exits immediately
else:
    # Parent does NOT call wait(), so child becomes zombie
    print(f"Parent PID = {os.getpid()} - Child PID = {pid} created")
    print("Sleeping for 10 seconds so child becomes zombie...")
    time.sleep(10)
    print("Parent done sleeping. You can check zombie with 'ps -el | grep defunct'.")

time.sleep(2) # small delay

# ORPHAN PROCESS
print("\n—— ORPHAN PROCESS ——")
pid2 = os.fork()

if pid2 == 0:
    # Child process
    print(f"Orphan Child PID = {os.getpid()} starting...")
    time.sleep(5)
    print(f"Orphan Child PID = {os.getpid()} done.")
else:
    # Parent exits immediately
    print(f"Parent PID = {os.getpid()} exiting immediately, child becomes orphan")
```

Output:

```
(monika@Monika)-[~]
$ python3 task3_zombie_orphan.py

—— ZOMBIE PROCESS ——
Child (Zombie) PID = 109149
Parent PID = 109148 - Child PID = 109149 created
Sleeping for 10 seconds so child becomes zombie...
Parent done sleeping. You can check zombie with 'ps -el | grep defunct'.

—— ORPHAN PROCESS ——
Parent PID = 109148 exiting immediately, child becomes orphan
Orphan Child PID = 109246 starting...

(monika@Monika)-[~]
$ Orphan Child PID = 109246 done.
ps -ps -el | grepunct
```

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

Code:

```
monika@Monika: -
File Actions Edit View Help
GNU nano 8.4 task4_proc_info.py
import os

# Get PID from user
pid = input("Enter PID of the process to inspect: ")

proc_path = f"/proc/{pid}"

# Executable path
try:
    exe_path = os.readlink(f"{proc_path}/exe")
    print(f"Executable path for PID {pid}: {exe_path}")
except Exception as e:
    print(f"Could not read executable path: {e}")

# Open file descriptors
fd_path = f"{proc_path}/fd"
try:
    fds = os.listdir(fd_path)
    print(f"Open file descriptors for PID {pid}: {fds}")
except Exception as e:
    print(f"Could not list file descriptors: {e}")
```

Output:

```
(monika@Monika)-[~]
$ python3 task4_proc_info.py

Enter PID of the process to inspect: 1407
Executable path for PID 1407: /usr/libexec/gvfs-afc-volume-monitor
Open file descriptors for PID 1407: ['0', '1', '2', '3', '4', '5', '6', '7']
```


Task 5: Process Prioritization

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

Code:

```
monika@Monika: ~  
File Actions Edit View Help  
GNU nano 8.4 task5_priority.py  
import os  
import time  
  
def cpu_task(name, duration=5):  
    start = time.time()  
    while time.time() - start < duration:  
        = 0  
        for i in range(100000):  
            += i*i  
    print(f"Process {name} (PID={os.getpid()}) finished.")  
  
if __name__ == "__main__":  
    print("—— PROCESS PRIORITIZATION DEMO ——")  
  
    for i in range(3):  
        pid = os.fork()  
        if pid == 0: # child  
            # Set different nice values  
            os.nice(i * 5) # 0, 5, 10  
            print(f"Child {i} started with nice value {i*5}, PID={os.getpid()}, PPID={os.getppid()}")  
            cpu_task(f"Child {i}")  
            os._exit(0)  
  
    # Parent waits  
    for _ in range(3):  
        os.wait()  
    print("All child processes completed.")
```

Output:

```
(monika@Monika)-[~]  
$ python3 task5_priority.py  
  
—— PROCESS PRIORITIZATION DEMO ——  
Child 0 started with nice value 0, PID=11169, PPID=11168  
Child 1 started with nice value 5, PID=11170, PPID=11168  
Child 2 started with nice value 10, PID=11171, PPID=11168  
Process Child 1 (PID=11170) finished.  
Process Child 0 (PID=11169) finished.  
Process Child 2 (PID=11171) finished.  
All child processes completed.
```