ASSIGNMENT-1

Name- ANJALI
Roll No.- 2301420021
Course- BTech CSE(Data Science)

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:

```
  GNU nano 8.4                                                    task1.py *
import os

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

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

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ python3 task1.py
Enter number of child processes: 3
Child 1
PID: 455
Parent PID: 454
Hello from child 1
Child 2
PID: 456
Parent PID: 454
Hello from child 2
Child 3
PID: 457
Parent PID: 454
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:

```
 GNU nano 8.4                                                    task2.py *
import os

def main():
    n = int(input("Enter number of child processes: "))
    commands = [["ls", "-l"], ["date"], ["ps"]]

    for i in range(n):
        pid = os.fork()
        if pid == 0:
            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:
            os.wait()

    print("\nParent: All child processes have executed commands.")

if __name__ == "__main__":
    main()
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ python3 task2.py
Enter number of child processes: 3

Child 1: PID = 459, Parent PID = 458
Executing command: ['ls', '-l']
total 24
-rw-r--r-- 1 anjali anjali 495 Sep 16 19:39 fork_process.py
-rw-r--r-- 1 anjali anjali 495 Sep 16 19:45 task1.py
-rw-r--r-- 1 anjali anjali 623 Sep 16 19:54 task2.py
-rw-r--r-- 1 anjali anjali 853 Sep 16 20:01 task3.py
-rw-r--r-- 1 anjali anjali 532 Sep 16 20:12 task4.py
-rw-r--r-- 1 anjali anjali 724 Sep 16 20:22 task5.py

Child 2: PID = 460, Parent PID = 458
Executing command: ['date']
Tue Sep 16 08:42:04 PM IST 2025

Child 3: PID = 461, Parent PID = 458
Executing command: ['ps']
    PID TTY          TIME CMD
    325 pts/0    00:00:00 bash
    458 pts/0    00:00:00 python3
    461 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.py *
import os
import time

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

if pid == 0:
    print(f"Child (Zombie) PID = {os.getpid()}")
    os._exit(0)
else:
    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)

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

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

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ python3 task3.py
----- ZOMBIE PROCESS -----
Child (Zombie) PID = 463
Parent PID = 462 | Child PID = 463 created
Sleeping for 10 seconds so child becomes zombie...
Parent done sleeping. You can check zombie with 'ps -el | grep defunct'.

----- ORPHAN PROCESS -----
Orphan Child PID = 466 starting...
Parent PID = 462 exiting immediately, child becomes orphan

┌──(anjali㉿Anjali)-[~]
└─$ Orphan Child PID = 466 done.
```

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:

```
 GNU nano 8.4                                                    task4.py *
import os

pid = input("Enter PID of the process to inspect: ")
proc_path = f"/proc/{pid}"

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}")

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:

```
┌──(anjali㉿Anjali)-[~]
└─$ python3 task4.py
Enter PID of the process to inspect: 233
Executable path for PID 233: /usr/bin/bash
Open file descriptors for PID 233: ['0', '1', '2', '255']
```

Task 5: Process Prioritization

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

Code:

```
 GNU nano 8.4                                                    task5.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:
            os.nice(i * 5)
            print(f"Child {i} started with nice value {i*5}, PID={os.getpid()}, PPID={os.getppid()}")
            cpu_task(f"Child {i}")
            os._exit(0)

    for _ in range(3):
        os.wait()
    print("All child processes completed.")
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ python3 task5.py
-- PROCESS PRIORITIZATION DEMO --
Child 0 started with nice value 0, PID=469, PPID=468
Child 1 started with nice value 5, PID=470, PPID=468
Child 2 started with nice value 10, PID=471, PPID=468
Process Child 0 (PID=469) finished.
Process Child 1 (PID=470) finished.
Process Child 2 (PID=471) finished.
All child processes completed.
```