Experiment Objectives:

In this assignment, students will simulate Linux process management operations using Python. The experiment focuses on replicating the behaviors of fork(), exec(), and process state inspections using the os and subprocess modules in Python. It provides an understanding of process creation, child-parent relationship, and zombie/orphan process scenarios.

Concepts Used:

- os.fork(), os.getpid(), os.getppid()
- os._exit(), os.wait(), os.nice()
- subprocess.run(), os.execvp()
- Reading /proc/[pid]/status, /exe, and /fd

- Python script process_management.py that:
- Implements process creation with os.fork() .
- Executes Linux commands from child processes using os.execvp().
- Simulates zombie and orphan processes with and without proper parent waiting.
- Inspects process information from /proc/[pid] directories.

- Demonstrates process priority effect using os.nice() on CPU-intensive child processes.
- Ensured output is correctly captured and flushed by adding flush=True in all print statements.

Code of process_management.py

File: **process_management.py**

```python
import os
import time

def task1_process_creation(num_children=3):
    print("\n--- Task 1: Process Creation Utility ---", flush=True)
    children = []
    for i in range(num_children):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1}: PID={os.getpid()}, PPID={os.getppid()}, Msg=Hello from child process.", flush=True)
            os._exit(0)
        else:
            children.append(pid)
    for _ in range(num_children):
        os.wait()
    print(f"Parent PID={os.getpid()}: All children have finished.\n", flush=True)

def task2_exec_command(commands=["ls", "date", "ps"]):
    print("--- Task 2: Command Execution Using exec() ---", flush=True)
    for idx, cmd in enumerate(commands):
        pid = os.fork()
        if pid == 0:
            # Print message and flush before execvp replaces the process image
            print(f"Child {idx+1} PID={os.getpid()} executing: {cmd}", flush=True)
            os.execvp(cmd, [cmd])
            # If execvp fails:
            print(f"Exec failed in child {idx+1}", flush=True)
            os._exit(1)
        else:
            os.wait()
    print("Finished executing commands in child processes.\n", flush=True)

def run_in_subprocess(func):
    # Use fork to run a function in its own subprocess, allowing parent to continue main script
    pid = os.fork()
```

```
    pid = os.fork()
    if pid == 0:
        func()
        os._exit(0)
    else:
        os.wait()

def task3_zombie():
    print("--- Task 3a: Zombie Process Demonstration ---", flush=True)
    pid = os.fork()
    if pid == 0:
        print(f"Zombie Child: PID={os.getpid()} (about to exit)", flush=True)
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} (not waiting for child PID={pid} -- child will be zombie)", flush=True)
        time.sleep(5)
        print("(Check zombie by running: ps -el | grep defunct)\n", flush=True)
        os.wait()

def task3_orphan():
    print("--- Task 3b: Orphan Process Demonstration ---", flush=True)
    pid = os.fork()
    if pid == 0:
        print(f"Orphan Child: PID={os.getpid()} sleeping, parent will exit.", flush=True)
        time.sleep(5)
        print(f"Orphan Child: PID={os.getpid()} finished (parent exited, now adopted by init).", flush=True)
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} exiting before child PID={pid} finishes.", flush=True)
        # Instead of os._exit, just return to allow main program to continue
        return

def task4_proc_inspection(pid=None):
    print("--- Task 4: Process Info from /proc ---", flush=True)
    if pid is None:
        pid = os.getpid()
    try:
        with open(f"/proc/{pid}/status") as f:
            status_lines = f.readlines()
        exe_path = os.readlink(f"/proc/{pid}/exe")
```

```
        exe_path = os.readlink(f"/proc/{pid}/exe")
        open_fds = os.listdir(f"/proc/{pid}/fd")
        name = next((l for l in status_lines if l.startswith("Name:")), "").strip()
        state = next((l for l in status_lines if l.startswith("State:")), "").strip()
        mem = next((l for l in status_lines if l.startswith("VmSize:")), "").strip()
        print(f"Process Info for PID {pid}:", flush=True)
        print(f"  {name}\n  {state}\n  {mem}", flush=True)
        print(f"  Executable Path: {exe_path}", flush=True)
        print(f"  Open File Descriptors: {open_fds}\n", flush=True)
    except Exception as e:
        print(f"Error reading /proc info for PID={pid}: {e}\n", flush=True)

def cpu_work(duration=2):
    # Simple CPU-bound work
    end = time.time() + duration
    while time.time() < end:
        pass

def task5_priority_demo():
    print("--- Task 5: Process Prioritization (nice values) ---", flush=True)
    nice_values = [0, 5, 10]
    for idx, nv in enumerate(nice_values):
        pid = os.fork()
        if pid == 0:
            os.nice(nv)
            print(f"Child {idx+1}: PID={os.getpid()} Nice={nv} starting CPU-bound work.", flush=True)
            cpu_work()
            print(f"Child {idx+1}: PID={os.getpid()} Nice={nv} finished CPU-bound work.", flush=True)
            os._exit(0)
    for _ in range(len(nice_values)):
        os.wait()
    print("Priority demo finished.\n", flush=True)

def print_header():
    print("\nLab Experiment Sheet-1", flush=True)
    print("School of Engineering and Technology", flush=True)
    print("Course Code & Name: ENCS351 Operating System", flush=True)
    print("Name: Adhvay Banerjee", flush=True)
    print("Roll Number: 2301410002", flush=True)
    print("Date/Time: {}".format(time.strftime("%A %d %B %Y %I:%M:%S %p %Z")), flush=True)
```

```
def print_header():
    print("\nLab Experiment Sheet-1", flush=True)
    print("School of Engineering and Technology", flush=True)
    print("Course Code & Name: ENCS351 Operating System", flush=True)
    print("Name: Adhvay Banerjee", flush=True)
    print("Roll Number: 2301410002", flush=True)
    print("Date/Time: {}".format(time.strftime("%A %d %B %Y %I:%M:%S %p %Z")), flush=True)

if __name__ == "__main__":
    print_header()
    task1_process_creation()
    task2_exec_command()
    run_in_subprocess(task3_zombie)
    run_in_subprocess(task3_orphan)
    time.sleep(6)  # Wait for orphan child to finish
    task4_proc_inspection()
    task5_priority_demo()

```

Result/Output of output.txt

File: **output.txt**

```
Lab Experiment Sheet-1
School of Engineering and Technology
Course Code & Name: ENCS351 Operating System
Name: Adhvay Banerjee
Roll Number: 2301410002
Date/Time: Monday 15 September 2025 03:09:03 PM IST

--- Task 1: Process Creation Utility ---
Child 1: PID=3442, PPID=3441, Msg=Hello from child process.
Child 2: PID=3443, PPID=3441, Msg=Hello from child process.
Child 3: PID=3444, PPID=3441, Msg=Hello from child process.
Parent PID=3441: All children have finished.

--- Task 2: Command Execution Using exec() ---
Child 1 PID=3445 executing: ls
output.txt
process_management.py
Child 2 PID=3446 executing: date
Monday 15 September 2025 03:09:03 PM IST
Child 3 PID=3447 executing: ps
    PID TTY          TIME CMD
   2661 pts/0    00:00:02 zsh
   3441 pts/0    00:00:00 python3
   3447 pts/0    00:00:00 ps
Finished executing commands in child processes.

--- Task 3a: Zombie Process Demonstration ---
Parent PID=3448 (not waiting for child PID=3449 -- child will be zombie)
Zombie Child: PID=3449 (about to exit)
(Check zombie by running: ps -el | grep defunct)

--- Task 3b: Orphan Process Demonstration ---
Parent PID=3452 exiting before child PID=3453 finishes.
Orphan Child: PID=3453 sleeping, parent will exit.
Orphan Child: PID=3453 finished (parent exited, now adopted by init).
--- Task 4: Process Info from /proc ---
Process Info for PID 3441:
  Name: python3
  State:    R (running)
  VmSize:      17700 kB
  Executable Path: /usr/bin/python3.13
  Open File Descriptors: ['0', '1', '2', '3']

--- Task 5: Process Prioritization (nice values) ---
Child 1: PID=3454 Nice=0 starting CPU-bound work.
Child 2: PID=3455 Nice=5 starting CPU-bound work.
Child 3: PID=3456 Nice=10 starting CPU-bound work.
Child 1: PID=3454 Nice=0 finished CPU-bound work.
Child 2: PID=3455 Nice=5 finished CPU-bound work.
Child 3: PID=3456 Nice=10 finished CPU-bound work.
Priority demo finished.
```