Lab Experiment Sheet – 1

Name – Yash Sharma

Roll no. - 2301010432

Task 1: Process Creation Utility

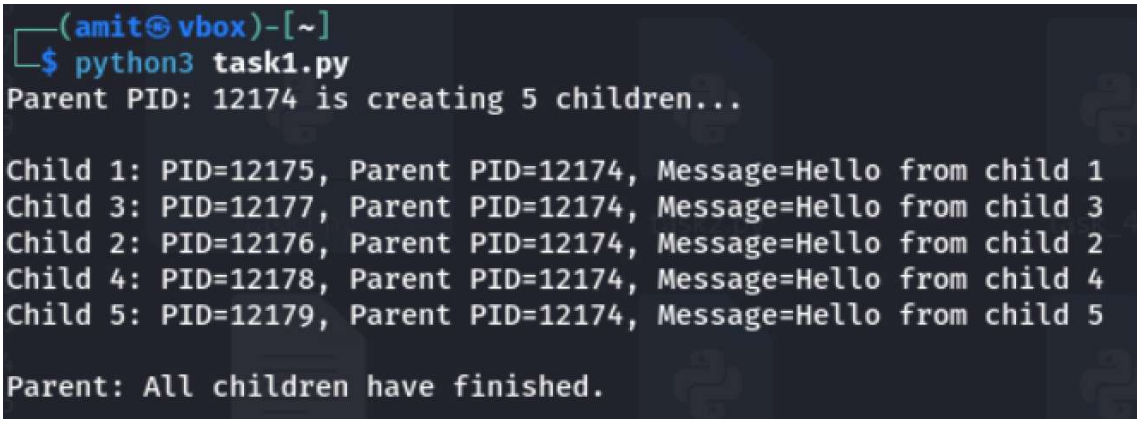
Write a Python program that creates N child processes using os.fork(). Each child prints:
- Its PID
- Its Parent PID
- A custom message

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

Input :

```
import os

def main():
    N = 5
    print(f"Parent PID: {os.getpid()} is creating {N} children...\n")

    for i in range(N):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1}: PID={os.getpid()}, Parent
PID={os.getppid()}, Message=Hello from child {i+1}")
            os._exit(0)
        else:
            continue

    for _ in range(N):
        os.wait()

    print("\nParent: All children have finished.")

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

Ouptut :


(amit㉿vbox)-[~]
$ python3 task1.py
Parent PID: 12174 is creating 5 children...
Child 1: PID=12175, Parent PID=12174, Message=Hello from child 1
Child 3: PID=12177, Parent PID=12174, Message=Hello from child 3
Child 2: PID=12176, Parent PID=12174, Message=Hello from child 2
Child 4: PID=12178, Parent PID=12174, Message=Hello from child 4
Child 5: PID=12179, Parent PID=12174, Message=Hello from child 5
Parent: All children have finished.

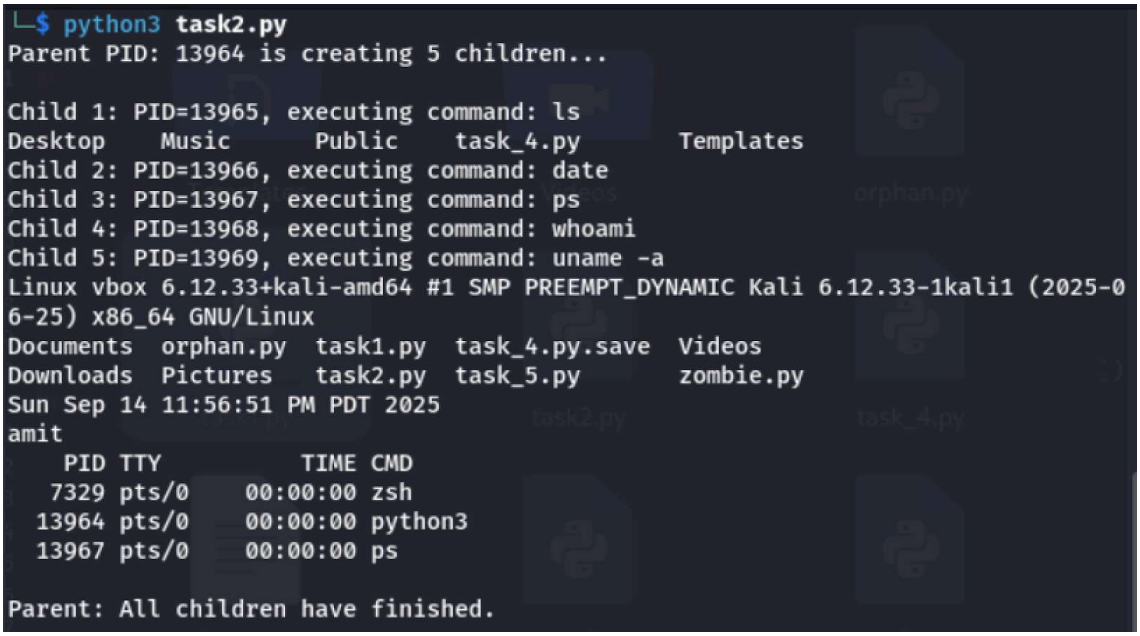
Task 2: Command Execution Using exec()

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

Input :

```
import os

def main():
    commands = [
        ["ls"],
        ["date"],
        ["ps"],
        ["whoami"],
        ["uname", "-a"]
    ]

    print(f"Parent PID: {os.getpid()} is creating {len(commands)}
children...\n")

    for i, cmd in enumerate(commands):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1}: PID={os.getpid()}, executing command: {'
'.join(cmd)}")
            os.execvp(cmd[0], cmd)
        else:
            continue

    for _ in commands:
        os.wait()

    print("\nParent: All children have finished.")

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

Ouptut :


└─$ python3 task2.py
Parent PID: 13964 is creating 5 children...
Child 1: PID=13965, executing command: ls
Desktop Music Public task_4.py Templates
Child 2: PID=13966, executing command: date
Child 3: PID=13967, executing command: ps
Child 4: PID=13968, executing command: whoami
Child 5: PID=13969, executing command: uname -a
Linux vbox 6.12.33+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.33-1kali1 (2025-06-25) x86_64 GNU/Linux
Documents orphan.py task1.py task_4.py.save Videos
Downloads Pictures task2.py task_5.py zombie.py
Sun Sep 14 11:56:51 PM PDT 2025
amit
PID TTY TIME CMD
7329 pts/0 00:00:00 zsh
13964 pts/0 00:00:00 python3
13967 pts/0 00:00:00 ps
Parent: All children have finished.

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.

Input :

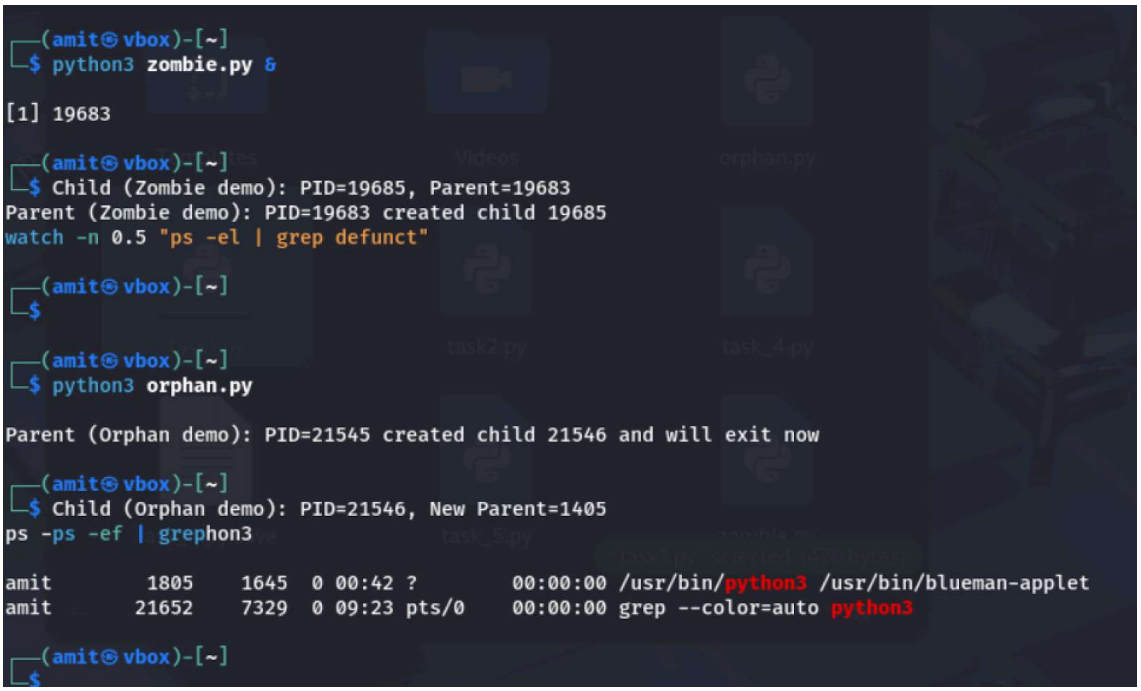
Zombie.py

```
cat > zombie.py << 'EOF'
import os
import time

pid = os.fork()
if pid == 0:
    print(f"Child (Zombie demo): PID={os.getpid()}, Parent={os.getppid()}")
    os._exit(0)
else:
    print(f"Parent (Zombie demo): PID={os.getpid()} created child {pid}")
    time.sleep(20)
    print("Parent exiting...")
EOF
```

Orphan.py

```
cat > orphan.py << 'EOF'
import os
import time

pid = os.fork()
if pid == 0:
    time.sleep(10)
    print(f"Child (Orphan demo): PID={os.getpid()}, New
Parent={os.getppid()}")
else:
    print(f"Parent (Orphan demo): PID={os.getpid()} created child {pid} and
will exit now")
    os._exit(0)
EOF
```

Output :


┌──(amit㉿vbox)-[~]
└─$ python3 zombie.py &
[1] 19683
┌──(amit㉿vbox)-[~]
└─$ Child (Zombie demo): PID=19685, Parent=19683
Parent (Zombie demo): PID=19683 created child 19685
watch -n 0.5 "ps -el | grep defunct"
┌──(amit㉿vbox)-[~]
└─$
┌──(amit㉿vbox)-[~]
└─$ python3 orphan.py
Parent (Orphan demo): PID=21545 created child 21546 and will exit now
┌──(amit㉿vbox)-[~]
└─$ Child (Orphan demo): PID=21546, New Parent=1405
ps -ps -ef | grephon3
amit 1805 1645 0 00:42 ? 00:00:00 /usr/bin/python3 /usr/bin/blueman-applet
amit 21652 7329 0 09:23 pts/0 00:00:00 grep --color=auto python3
┌──(amit㉿vbox)-[~]
└─$

Task 4: Inspecting Process Info from /proc

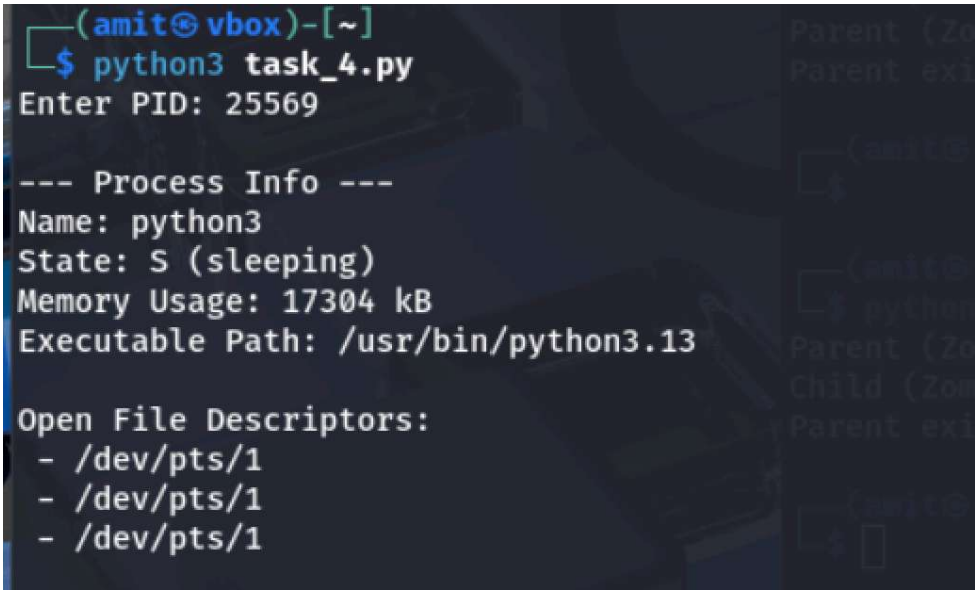
Take a PID as input. Read and print:
- Process name, state, memory usage from /proc/[pid]/status
- Executable path from /proc/[pid]/exe
- Open file descriptors from /proc/[pid]/fd

Input :

```
import os

def read_status(pid):
    status_file = f"/proc/{pid}/status"
    info = {}
    with open(status_file, "r") as f:
        for line in f:
            if line.startswith("Name:") or line.startswith("State:") or
line.startswith("VmSize:"):
                key, value = line.split(":", 1)
                info[key.strip()] = value.strip()
    return info

def read_exe(pid):
    try:
        return os.readlink(f"/proc/{pid}/exe")
    except FileNotFoundError:
        return "Executable path not found"
    except PermissionError:
        return "Permission denied"

def read_fds(pid):
    fd_dir = f"/proc/{pid}/fd"
    try:
        return [os.readlink(os.path.join(fd_dir, fd)) for fd in
os.listdir(fd_dir)]
    except FileNotFoundError:
        return ["No FD info (process might have ended)"]
    except PermissionError:
        return ["Permission denied"]

def main():
    pid = input("Enter PID: ").strip()
    if not pid.isdigit():
        print("Invalid PID")
        return

    status = read_status(pid)
    exe = read_exe(pid)
    fds = read_fds(pid)
```

```
    print("\n--- Process Info ---")
    print(f"Name: {status.get('Name', 'N/A')}")
    print(f"State: {status.get('State', 'N/A')}")
    print(f"Memory Usage: {status.get('VmSize', 'N/A')}")
    print(f"Executable Path: {exe}")
    print("\nOpen File Descriptors:")
    for fd in fds:
        print(f" - {fd}")

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

Output :



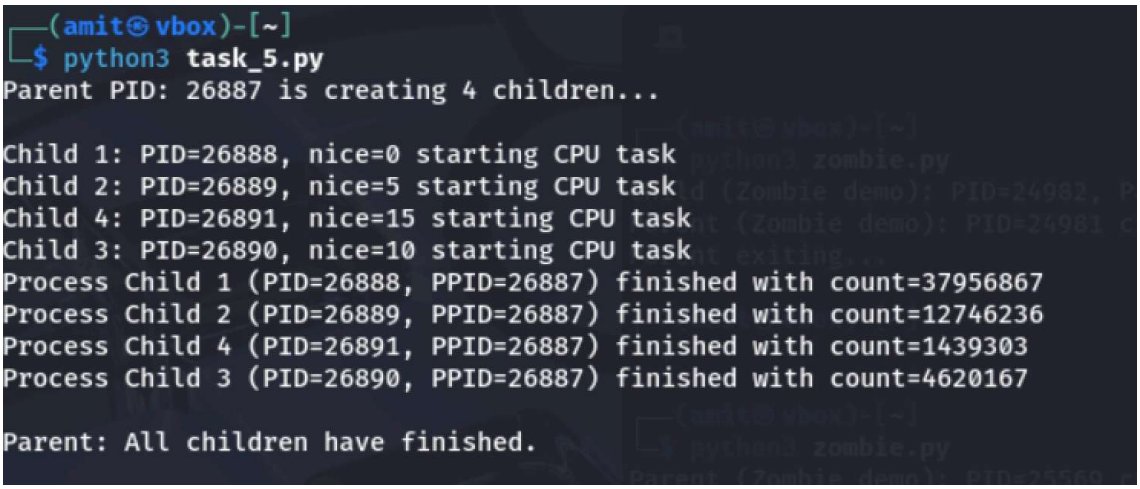
Task 5: Process Prioritization

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

Input :

```
import os
import time

def cpu_task(name, duration=5):
    start = time.time()
    count = 0
    while time.time() - start < duration:
        count += 1  # CPU intensive loop
    print(f"Process {name} (PID={os.getpid()}, PPID={os.getppid()})
finished with count={count}")

def main():
    priorities = [0, 5, 10, 15]  # lower = higher priority
    print(f"Parent PID: {os.getpid()} is creating {len(priorities)}
children...\n")

    for i, prio in enumerate(priorities):
        pid = os.fork()
        if pid == 0:
            try:
                os.nice(prio)  # adjust priority
            except PermissionError:
                print(f"Child {i+1}: cannot decrease nice value (need
root), using default priority.")
            print(f"Child {i+1}: PID={os.getpid()}, nice={os.nice(0)}
starting CPU task")
            cpu_task(f"Child {i+1}")
            os._exit(0)
        else:
            continue

    for _ in priorities:
        os.wait()

    print("\nParent: All children have finished.")

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

Output :


┌──(amit㉿vbox)-[~]
└─$ python3 task_5.py
Parent PID: 26887 is creating 4 children...
Child 1: PID=26888, nice=0 starting CPU task
Child 2: PID=26889, nice=5 starting CPU task
Child 4: PID=26891, nice=15 starting CPU task
Child 3: PID=26890, nice=10 starting CPU task
Process Child 1 (PID=26888, PPID=26887) finished with count=37956867
Process Child 2 (PID=26889, PPID=26887) finished with count=12746236
Process Child 4 (PID=26891, PPID=26887) finished with count=1439303
Process Child 3 (PID=26890, PPID=26887) finished with count=4620167
Parent: All children have finished.