**Lab Experiment Sheet – 1**

**Name – Sourabh Sharma**

**Roll no. - 2301010454**

**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):

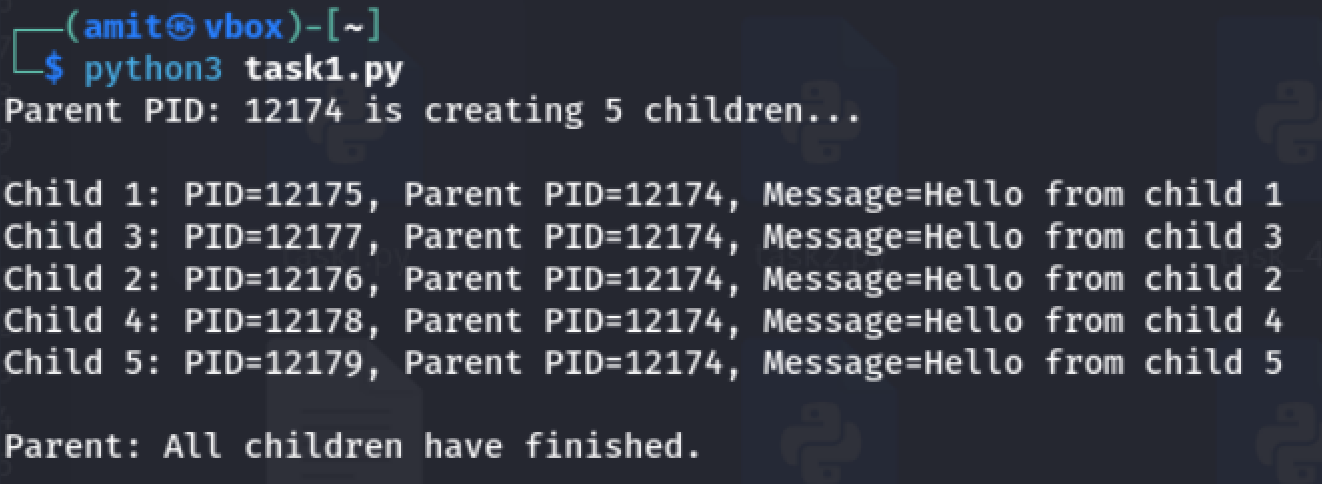
        os.wait()

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

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Ouptut :**

****

**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:

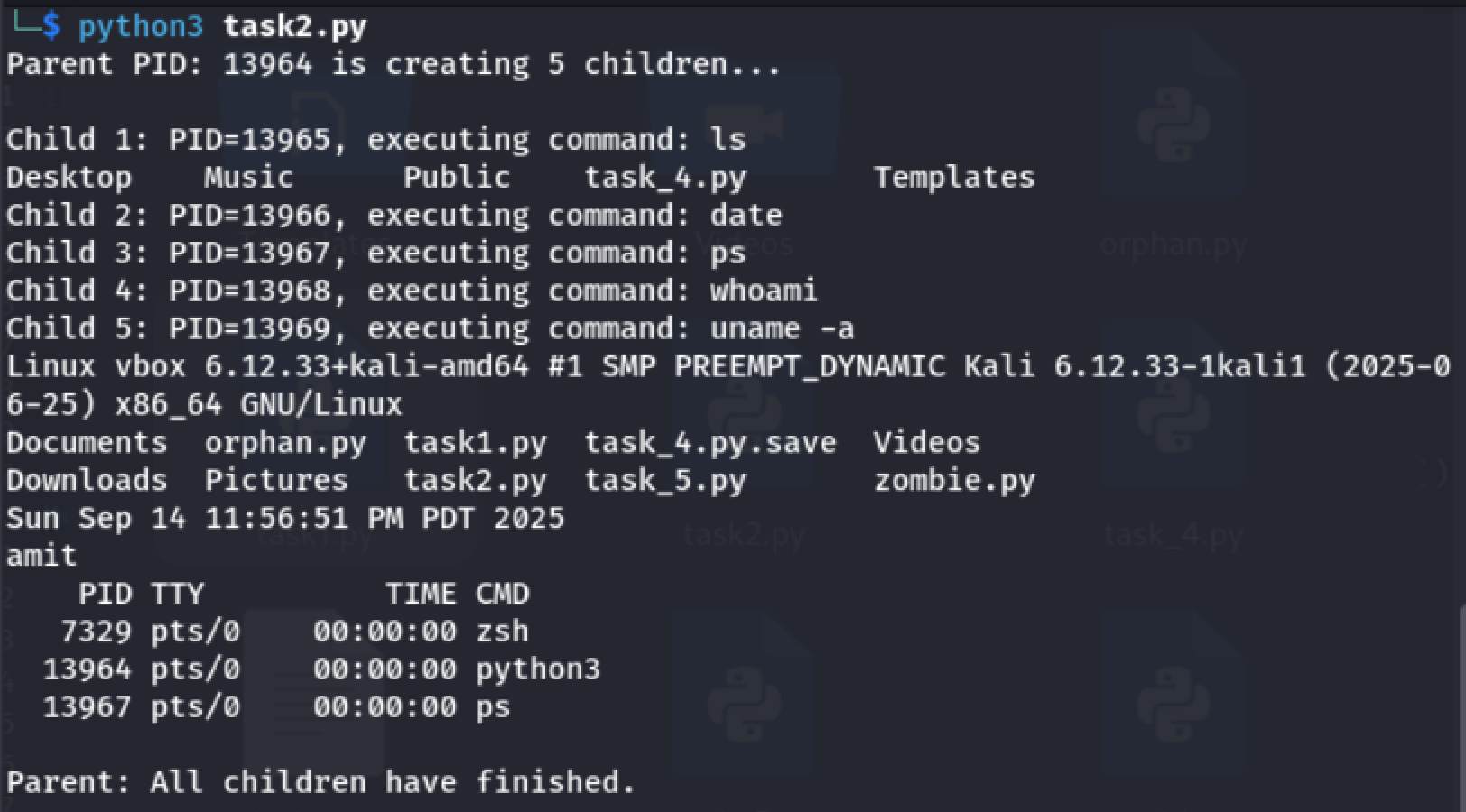
        os.wait()

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

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Ouptut :**

****

**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()}")

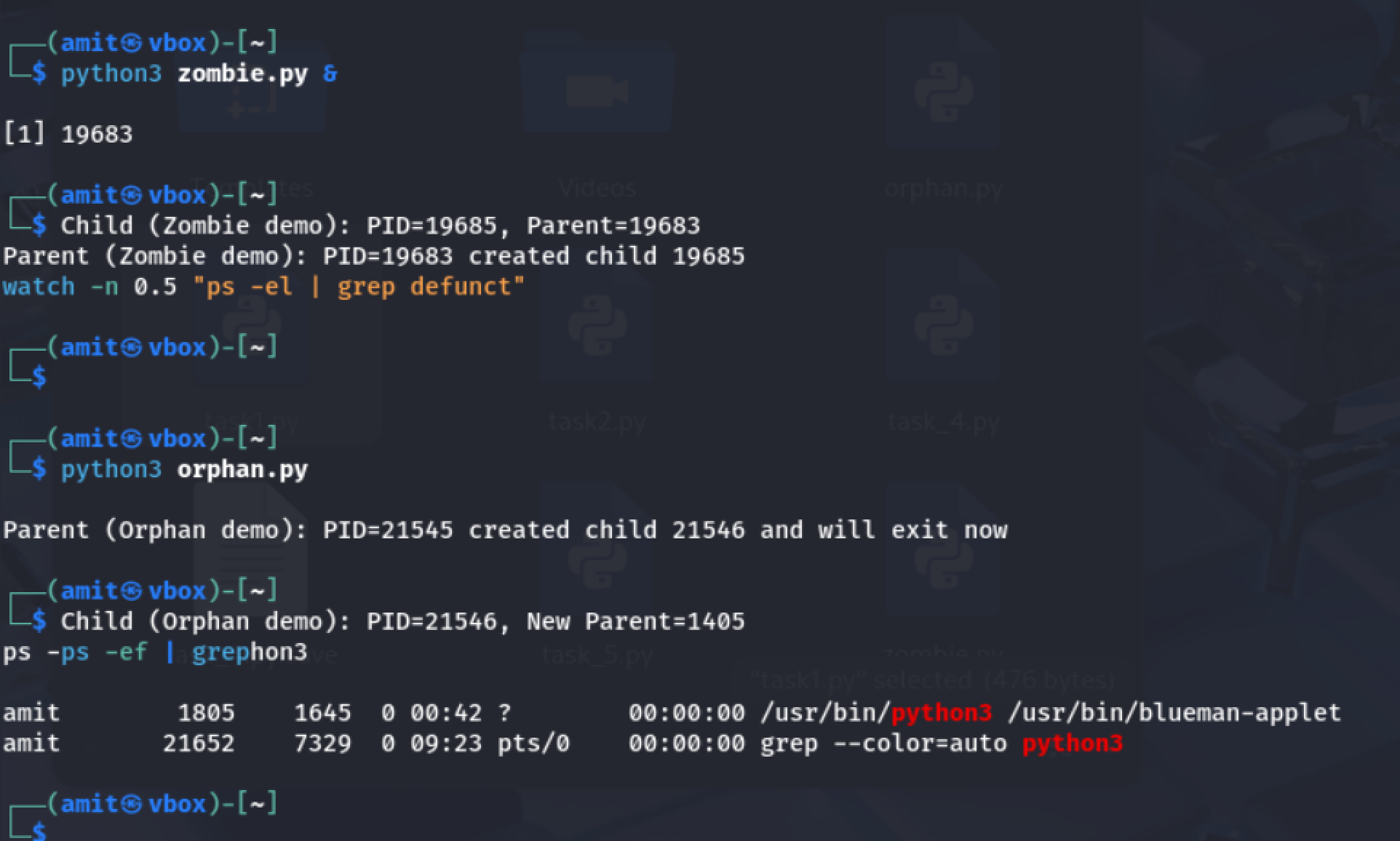
else:

    print(f"Parent (Orphan demo): PID={os.getpid()} created child {pid} and will exit now")

    os.\_exit(0)

EOF

**Output :**

****

**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:")

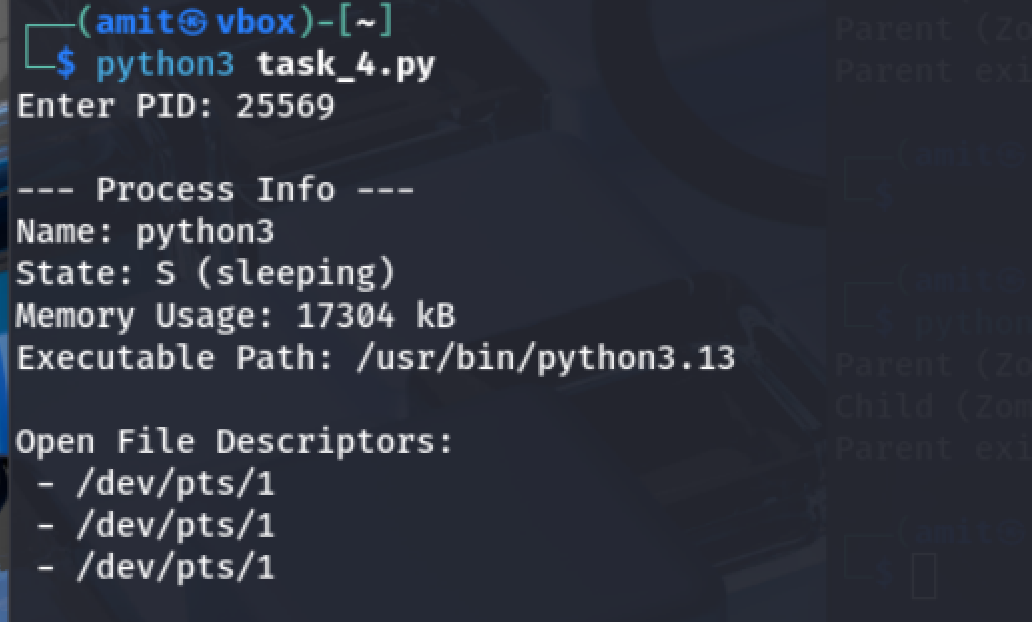
    for fd in fds:

        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:

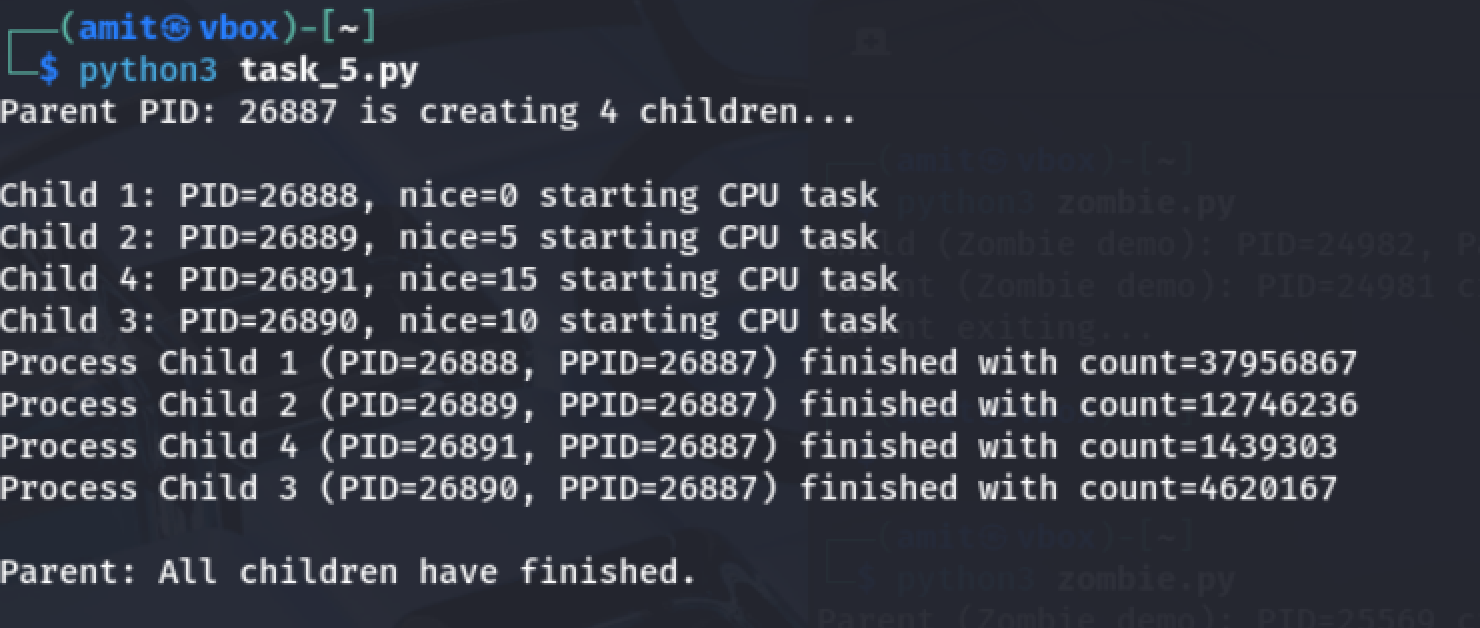
        os.wait()

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

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output :**

****