ASSIGNMENT – 1

Name: Payal Rathore

Roll No. : 2301410022

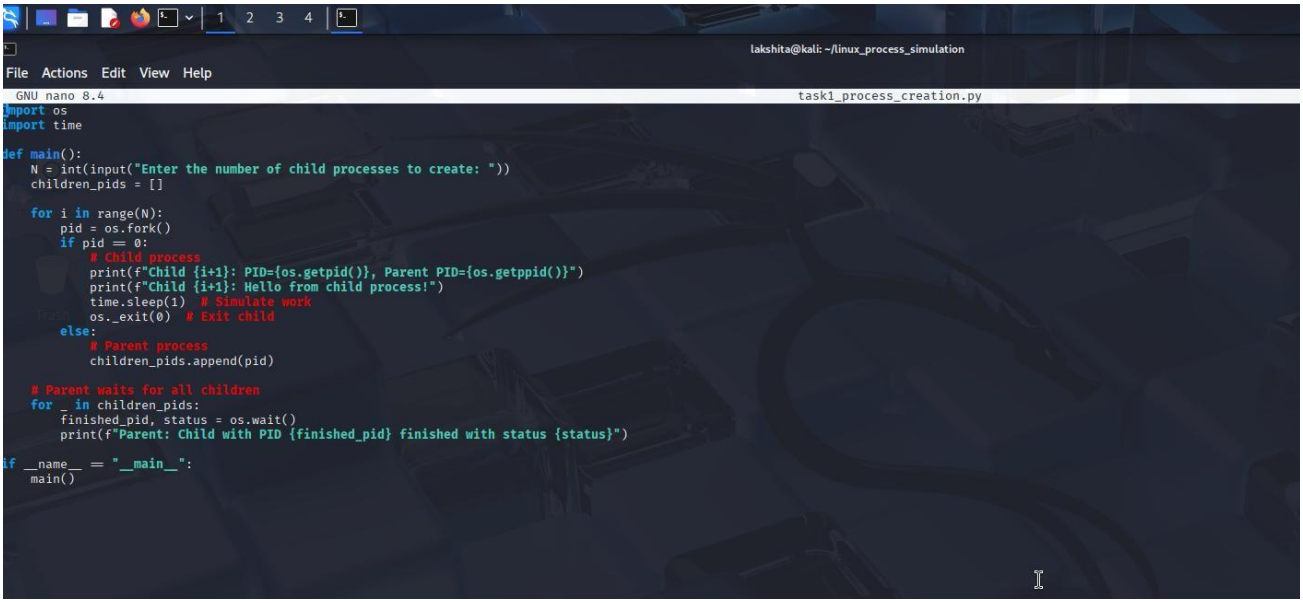
Course: BTech CSE (Cyber Security)

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:



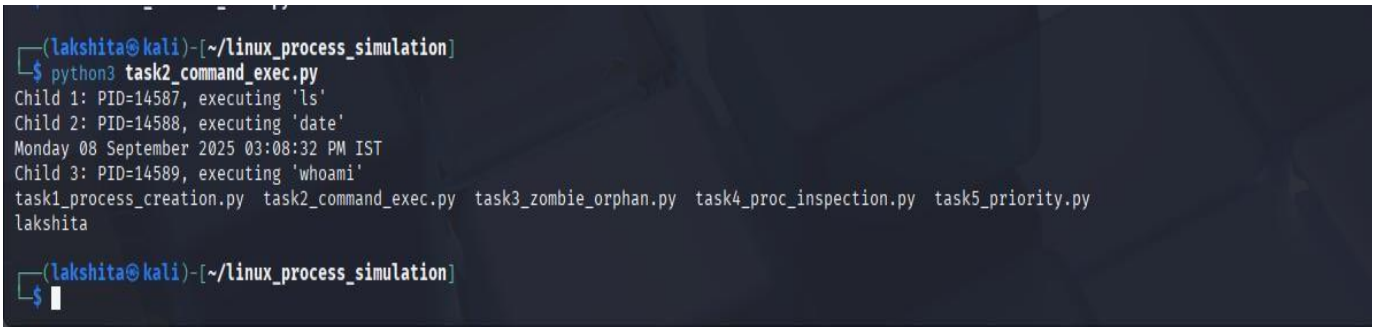
OUTPUT

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:

OUTPUT



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

lakshita@kali: ~/linux_process_simulation

File Actions Edit View Help

GNU nano 8.4 task3_zombie_orphan.py

```
import os
import time

def create_zombie():
    pid = os.fork()
    if pid == 0:
        # Child sleeps briefly and exits
        print(f"Zombie Child: PID={os.getpid()} exiting...")
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} not waiting for child {pid}")
        time.sleep(10)  # Gives time to check zombie with `ps -el | grep defunct`

def create_orphan():
    pid = os.fork()
    if pid == 0:
        time.sleep(5)
        print(f"Orphan Child: PID={os.getpid()}, new Parent PID={os.getppid()}")
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} exiting immediately")
        os._exit(0)

if __name__ == "__main__":
    print("Creating zombie process...")
    create_zombie()
    time.sleep(2)
    print("\nCreating orphan process...")
    create_orphan()
```

OUTPUT

```
┌──(lakshita㉿kali)-[~/linux_process_simulation]
└─$ python3 task3_zombie_orphan.py
Creating zombie process...
Parent PID=18060 not waiting for child 18061
Zombie Child: PID=18061 exiting...

Creating orphan process...
Parent PID=18060 exiting immediately

┌──(lakshita㉿kali)-[~/linux_process_simulation]
└─$ Orphan Child: PID=18159, new Parent PID=1
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

lakshita@kali: ~/linux_process_simulation

File Actions Edit View Help

GNU nano 8.4 task4_proc_inspection.py

```
import os

def main():
    pid = input("Enter PID to inspect: ")
    status_file = f"/proc/{pid}/status"
    exe_file = f"/proc/{pid}/exe"
    fd_folder = f"/proc/{pid}/fd"

    try:
        # Read status
        with open(status_file) as f:
            for line in f:
                if line.startswith(("Name", "State", "VmRSS")):
                    print(line.strip())

        # Executable path
        exe_path = os.readlink(exe_file)
        print(f"Executable Path: {exe_path}")

        # Open file descriptors
        fds = os.listdir(fd_folder)
        print(f"Open File Descriptors: {fds}")

    except FileNotFoundError:
        print(f"No process with PID {pid} exists.")

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

OUTPUT

```
┌──(lakshita㉿kali)-[~/linux_process_simulation]
└─$ python3 task4_proc_inspection.py
Enter PID to inspect: 1310
Name:   gvfs-afc-volume
State:  S (sleeping)
VmRSS:      8792 kB
Executable Path: /usr/libexec/gvfs-afc-volume-monitor
Open File Descriptors: ['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

lakshita@kali: ~/linux_process_simulation

File Actions Edit View Help

GNU nano 8.4 task5_priority.py

```
import os
import time

def cpu_intensive_task():
    count = 0
    for i in range(10**7):
        count += i
    print(f"Process PID={os.getpid()} finished counting.")

def main():
    nice_values = [0, 5, 10]  # Different priorities
    children_pids = []

    for nice_val in nice_values:
        pid = os.fork()
        if pid == 0:
            os.nice(nice_val)  # Set process priority
            print(f"Child PID={os.getpid()} with nice={nice_val} starting task...")
            cpu_intensive_task()
            os._exit(0)
        else:
            children_pids.append(pid)

    # Parent waits
    for _ in children_pids:
        os.wait()

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

OUTPUT

```
┌──(lakshita㉿kali)-[~/linux_process_simulation]
└─$ python task5_priority.py
Child PID=27411 with nice=0 starting task...
Child PID=27412 with nice=5 starting task...
Child PID=27413 with nice=10 starting task...
Process PID=27411 finished counting.
Process PID=27412 finished counting.
Process PID=27413 finished counting.
```