Assignment-1

Name - Abdul Waheed Al Faaiz

Roll No.-2301010468

Couse - Btech CSE Core

Section G (7)

Operating System

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:

```
┌──(kalilinux㊚DESKTOP-71JCI24)-[~/OS_Lab_1]
└─$ python3 process_management.py --task 1 --n 3
[TASK1] Parent PID 730 creating 3 children
Parent: created child 731
Parent: created child 732
Child 1: PID=731, PPID=730, msg='Hello from child 1'
Child 2: PID=732, PPID=730, msg='Hello from child 2'
Parent: created child 733
Parent: waiting for children to finish...
Parent: reaped child 731 (status 0)
Parent: reaped child 732 (status 0)
Child 3: PID=733, PPID=730, msg='Hello from child 3'
Parent: reaped child 733 (status 0)
[TASK1] Done.
```

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:

```
┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_1]
└─$ python3 process_management.py --task 2 --n 2 --cmd date --no-exec
[TASK2] Parent PID 737 creating 2 children to run ['date'] (use_exec=Fal
se)
Parent: forked child 738
[child 738] will run: ['date']
Parent: forked child 739
[child 739] will run: ['date']
Thu Oct  2 06:49:18 PM IST 2025
Thu Oct  2 06:49:18 PM IST 2025
Parent: reaped child 738 (status 0)
Parent: reaped child 739 (status 0)
[TASK2] Done.
```

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:

```
┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_1]
└─$ python3 -c "import process_management as pm; pm.task3_orphan_demo(20
)"
[TASK3-ORPHAN] Forking child; parent will exit immediately so child beco
mes orphan.
[parent 766] exiting immediately, child 767 will be orphaned.
[child 767] started; sleeping 20s. Initial PPID=766
```

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:

```
┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_1]
└─$ python3 process_management.py --task 4 --pid 1234
[TASK4] Inspecting /proc/1234
Could not read status: [Errno 2] No such file or directory: '/proc/1234/
status'
Could not read exe link: [Errno 2] No such file or directory: '/proc/123
4/exe'
Could not list fd: [Errno 2] No such file or directory: '/proc/1234/fd'
[TASK4] Done.
```

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:

```
┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_1]
└─$ python3 process_management.py --task 5 --children 3 --duration 8
[TASK5] Parent 778 creating 3 CPU-bound children with nice values [0, 5,
 10]
Parent: forked child 779 with target nice 0
[child 779 nice=0] start at 1759412303.795914
Parent: forked child 780 with target nice 5
Parent: forked child 781 with target nice 10
[child 780 nice=5] start at 1759412303.7973323
[child 781 nice=10] start at 1759412303.7975657
```