Operating Systems Lab Assignment - 1 Report

Name: Mayank Jha
Roll no. 2301730165
Course Code: ENCS351
Program : B.Tech CSE(Ai/Ml)

Experiment: Process Creation and Management Using Python

Objective:

To simulate Linux process management operations using Python to understand process creation, execution, and control, including fork(), exec(), zombie/orphan behavior, and scheduling via nice values.

Tools Used:

Python 3.x

Linux Environment (Ubuntu/WSL via Terminal)

Tasks Performed:

- Task 1: Created multiple child processes using os.fork() and displayed Parent/Child PIDs.
- Task 2: Executed system commands (ls, date, ps) using os.execvp() to replace process images.
- Task 3: Simulated Zombie (parent sleeps, child exits) and Orphan (parent exits, child sleeps) processes.
- Task 4: Inspected process details using the Linux /proc filesystem (specifically /proc/[pid]/status, /exe, and /fd).
- Task 5: Created CPU-intensive tasks with different nice() values (-5, 0, 5) to observe process prioritization logic.

Outputs:

- Successfully displayed parent-child relationships and distinct PIDs.
- Verified valid execution of external shell commands within Python.
- Observed how the OS adopts Orphan processes (re-parented to init/systemd).
- Retrieved internal process status and file descriptors from /proc.
- Demonstrated priority scheduling (noting that increasing priority requires root privileges).

```
===== Operating Systems Assignment: Process Management =====

--- TASK 1: Creating Child Processes ---

[Child] PID: 11660, Parent PID: 11658, Message: Child 0 running
[Child] PID: 11661, Parent PID: 11658, Message: Child 1 running
[Child] PID: 11662, Parent PID: 11658, Message: Child 2 running

[Parent] All children finished.


--- TASK 2: Executing Commands in Child Processes ---


[Child] PID 11663 executing command: ['ls']
process_management.py  projects

[Child] PID 11664 executing command: ['date']
Tue Nov 18 15:36:50 UTC 2025

[Child] PID 11665 executing command: ['ps']
    PID TTY          TIME CMD
  11657 pts/7    00:00:00 sudo
  11658 pts/7    00:00:00 python3
  11665 pts/7    00:00:00 ps

--- TASK 3: Zombie & Orphan Simulation ---


[Zombie Simulation]

[Parent] Not calling wait() → Child becomes zombie.
[Child-Zombie] PID 11666 exiting...
Run: ps -el | grep defunct

[Orphan Simulation]

[Parent] Exiting immediately → Child becomes orphan.

--- TASK 4: /proc Inspection ---
```

```
--- TASK 4: /proc Inspection ---

Enter a PID to inspect: [Child-Orphan] PID 11716 sleeping...
[Child-Orphan] New Parent PID after orphaning: 11658
11658

[PROCESS STATUS]
Name:   python3
State:  R (running)
VmSize:    17000 kB

[EXECUTABLE PATH]
/usr/bin/python3.12

[OPEN FILE DESCRIPTORS]
['0', '1', '2', '3']

--- TASK 5: Priority Scheduling ---

[Child] High Priority → nice value: -5
[Child] Normal Priority → nice value: 0
[Child] Low Priority → nice value: 5
[Child High Priority] Completed
[Child Low Priority] Completed
[Child Normal Priority] Completed

[Parent] Priority execution complete.


===== All Tasks Completed Successfully =====
```

Learning Outcomes:

- Process Life-cycle: Understood how fork() duplicates a process and how the OS manages Parent-Child relationships.
- Context Switching: Demonstrated how exec() replaces the current process memory with a new program.
- State Management: Observed the creation of Zombies (terminated but not waited for) and Orphans (running without original parent).
- System Internals: Learned to extract raw process data from the /proc directory.

- Scheduling: Learned that negative nice values increase priority and require sudo privileges.

Complexity:

- Time Complexity: $O(n)$ for creating n child processes.
- Space Complexity: $O(n)$ for maintaining process IDs and context in memory.

Conclusion:

This experiment deepened the understanding of process management concepts in Linux. By practically implementing the Python os module, we visualized how the Operating System handles process creation, resource allocation (/proc), and scheduling priorities. We successfully identified that Orphan processes are adopted by the init process (PID 1 or systemd) to prevent resource leaks.