# Lab Experiment Sheet-1

School of Engineering and Technology  
Course Code & Name: ENCS351 Operating System

Program Name: B.Tech CSE, AI ML, Data Science, Cyber, FSD, UX/UI

**Name – Harshit Chauhan Course – BTECH CSE DS -Sem 5**

**Roll No -230142002230**

**Summary of objectives**   
  
Task 1: Process Creation Utility  
For this task, I implemented a Python function named task1. Using a for loop, I invoked os.fork() three times to create three separate child processes. In each child process (pid == 0), I printed its own PID and its parent’s PID using os.getpid() and os.getppid(), along with a custom message. On the parent’s side, I ensured it didn’t terminate early by calling os.waitpid() for each child, so it properly waited until all children finished execution.

Task 2: Command Execution with exec()  
In task2, I created a single child process via os.fork(). Within the child block, I replaced its code with the command ls -l by using os.execvp("ls", ["ls", "-l"]). This system call overwrote the child process with the new program. The parent process simply waited until the command execution completed before continuing.

Task 3: Zombie & Orphan Processes  
I wrote two separate functions to illustrate these special states:

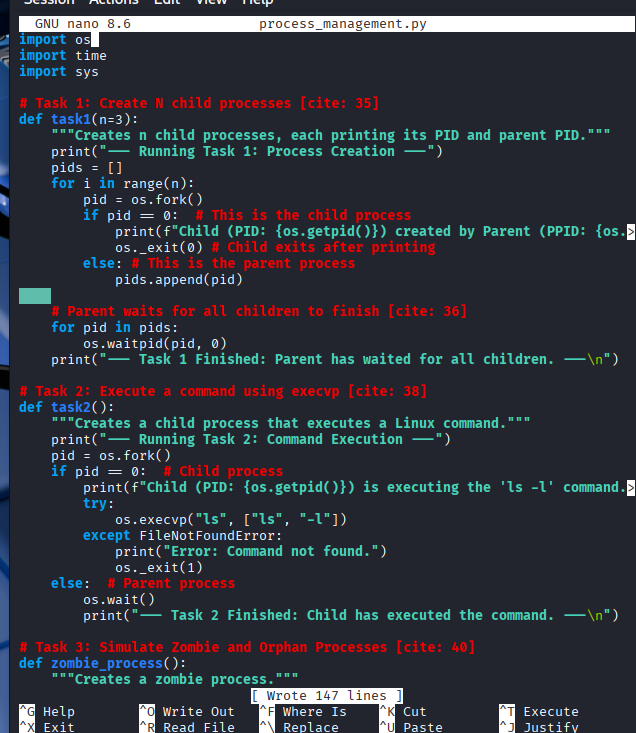
* Zombie Process: The child process printed a message and exited immediately using os.\_exit(0). The parent deliberately skipped os.wait(), instead sleeping for 10 seconds. During this period, the child stayed in the process table as a zombie (defunct) until the parent eventually reaped it.
* Orphan Process: Here, the parent exited right after forking, while the child was set to sleep for 5 seconds. When the child resumed, its parent no longer existed, and its new parent became PID 1 (the init process). I verified this by printing the child’s os.getppid().

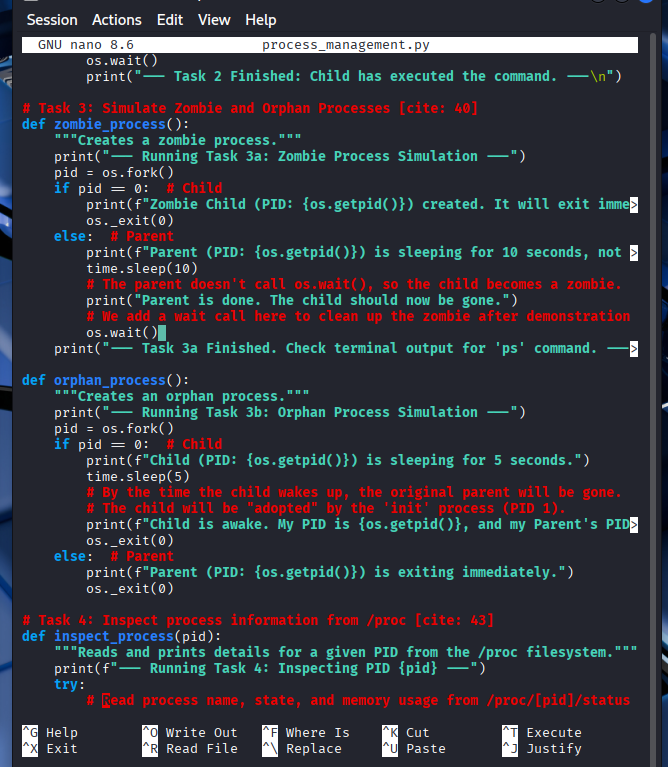
Task 4: Inspecting Process Information via /proc  
I defined a function inspect\_process that takes a PID as input and extracts information directly from the /proc filesystem:

* From /proc/[pid]/status, I parsed and displayed the process Name, State, and VmSize.
* Using os.readlink(), I obtained the executable’s absolute path from /proc/[pid]/exe.
* Finally, I listed all open file descriptors by reading /proc/[pid]/fd.

Task 5: Process Prioritization  
In task5, I forked multiple child processes and assigned each a different priority level using os.nice() with values 0, 5, and 10. Since a lower nice value means higher priority, the child with nice(0) received more CPU time. Each child executed the same CPU-intensive workload (a large summation loop). Observing the completion order confirmed that the higher-priority process consistently finished earlier, demonstrating how process scheduling is influenced by priority**.**

**Code snippets**

****

****

