

# Lab Assignment 4: System Calls and OS Simulation Report

---

**Course:** ENCS351 Operating System Lab

**Assignment:** System Calls, VM Detection, and File System Operations

---

## 1. Introduction

This lab focuses on interacting with low-level operating system interfaces. The objective was to implement batch processing, system startup logging, inter-process communication (IPC) using system calls, and virtual machine detection using Python and Shell scripting.

## 2. Implementation Details

### Task 1: Batch Processing Simulation

- **Mechanism:** A Python script was created to manage the sequential execution of other Python files( `batch_script_1.py` , `batch_script_2.py` ).
- **Implementation:** The `subprocess` module was used to spawn new processes for each script. This mimics how an OS manages batch jobs, ensuring one finishes before the next begins.

### Task 2: System Startup and Logging

- **Mechanism:** To simulate an OS boot sequence, `multiprocessing` was used to spawn concurrent tasks.
- **Logging:** The `logging` module was configured to write to `system_log.txt` . Each process logged its start and termination time.
- **Concurrency:** `Process.join()` was used to ensure the main system waited for all services (processes) to initialize and complete before shutting down.

### Task 3: IPC with System Calls (Fork & Pipe)

- **System Calls:** The script utilized `os.pipe()` to create a communication channel and `os.fork()` to create a child process.
- **Communication:**
  - The **Parent** process closed the read end, wrote a message ("Hello from Parent") to the write end, and waited for the child.
  - The **Child** process closed the write end, read the message from the buffer, and printed it to the console.

## Task 4: VM Detection and Shell

- **Shell:** A Bash script was generated to print the kernel version ( `uname -r` ) and user info.
- **Python Detection:** The script utilized the `platform` module to inspect system node names and release strings. It checks for keywords like "vbox", "vmware", or "kvm" to heuristically determine if the environment is virtualized.

## Task 5: CPU Scheduling Algorithms

The classic scheduling algorithms were implemented with hardcoded test data to verify logic:

- **FCFS:** Processes executed in arrival order.
- **SJF:** Processes sorted by Burst Time (BT) minimized average waiting time (6.00s).
- **Priority:** Processes sorted by priority index.
- **Round Robin:** Used a time quantum of 4. This ensured fairness but resulted in a higher average turnaround time compared to SJF.

## 3. Conclusion

The lab successfully demonstrated the use of kernel-level system calls ( `fork` , `exec` ) within a high-level language like Python. We observed how IPC allows coordination between processes and how the OS manages batch jobs and scheduling queues.