

Operating System

Lab File Submitted to

K. R. Mangalam University

for

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Riya Singh (2301010041) (Sem-05)

Course Teacher

Mrs. Suman

School of Engineering & Technology

K. R. MANGALAM UNIVERSITY

Sohna, Haryana 122103, India

**Lab Sheet 1**

**Summary of Objectives**

The main objective of this experiment is to understand how operating systems manage and control processes. Through this practical, we aim to simulate the process lifecycle — including creation, execution, and termination — using Java on a Windows system. The experiment helps visualize the relationship between parent and child processes and demonstrates key OS concepts like process creation, command execution, priority scheduling, and orphan/zombie process behavior.

By implementing these operations using Java’s ProcessBuilder, ProcessHandle, and thread management, students gain hands-on experience with how real-world operating systems handle multitasking and process coordination. The experiment also explores process information retrieval and priority adjustment, which are crucial for performance optimization and system-level programming.

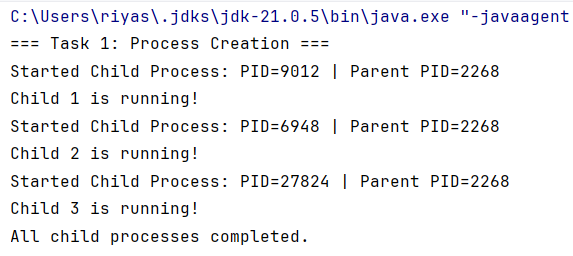
Overall, this experiment bridges theoretical understanding of process management with practical implementation, strengthening our grasp of how concurrent and parallel tasks are handled at the operating system level.

**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().

**Sol- **

**Output-**

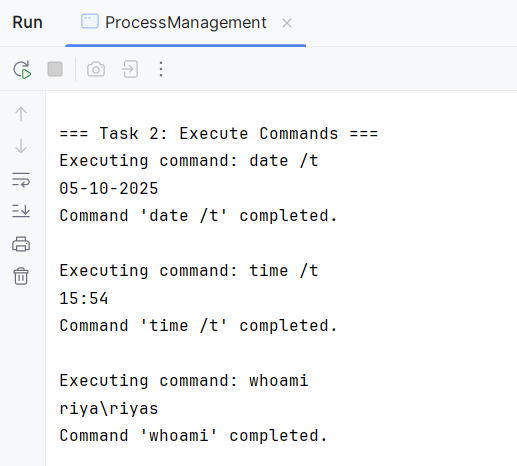
****

**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().



**Output –**

****

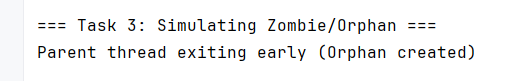
**Task 3 Task 3: Zombie & Orphan Processes**

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.

**Sol –**

****

**Output-**

****

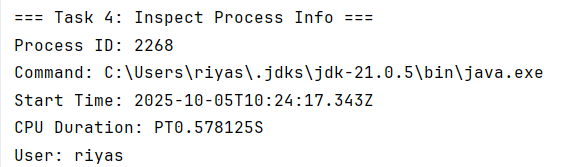
**Task 4: Inspecting Process Info from /proc**

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

**Sol –**

****

**Output-**

****

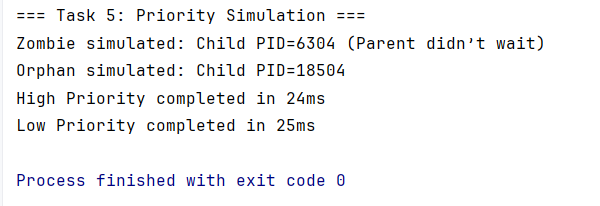
**Task 5: Process Prioritization**

Create multiple CPU-intensive child processes. Assign different nice() values. Observe and log execution order to show scheduler impact.

**Sol -**



**Output-**

****

**Lab Sheet 2**

**Summary of Objectives**

The objective of this experiment is to simulate how an operating system manages startup, process creation, and shutdown. Through this experiment, we understand how multiple processes can be created, executed concurrently, and terminated in a controlled manner.

By using Java’s Thread class and Logger, we replicate the functionality of the Python multiprocessing and logging modules, demonstrating system-level behavior in a simplified manner. Each simulated process logs its lifecycle — from start to end — and the system records these events in a log file to reflect real OS operations.

This simulation enhances our understanding of process management, concurrency, and logging mechanisms in modern operating systems.

**Sub-Tasks:**

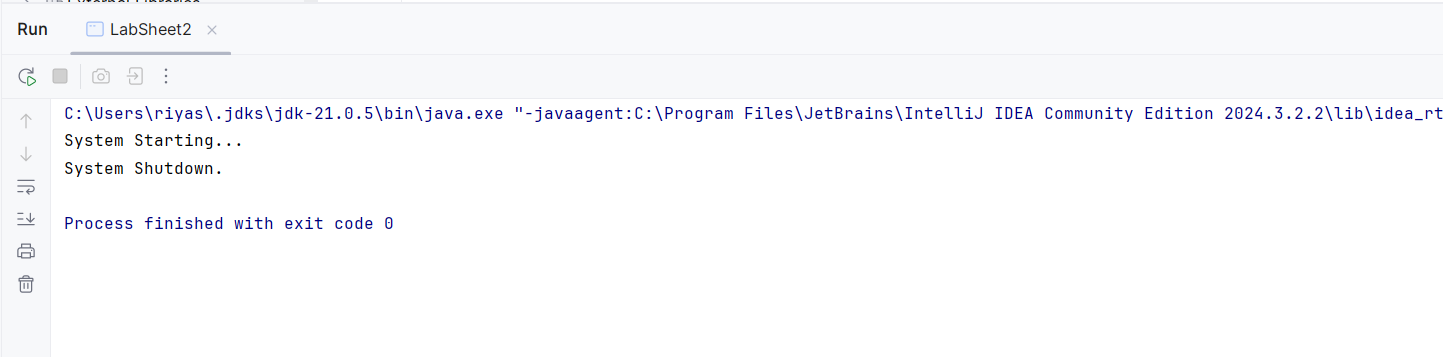
1. **Sub-Task 1:** Initialize the logging configuration to capture timestamped messages.
2. **Sub-Task 2:** Define a function that simulates a process task (e.g., sleep for 2 seconds).
3. **Sub-Task 3:** Create at least two processes and start them concurrently.
4. **Sub-Task 4:** Ensure proper termination and joining of processes, and verify the output in the log file.

**Sol-**

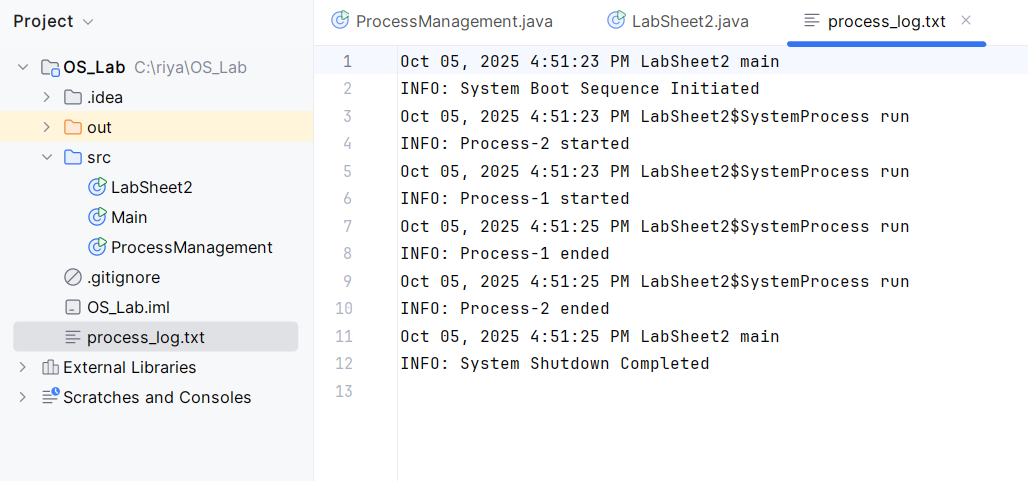
****

****

**Output-**

****

**Process Log**

****