Operating System Lab

(ENCS351)

Lab File Submitted to

K. R. Mangalam University

for

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Shami Ahmad (2301010030) (Sem-05)

Course Teacher

Mrs. Suman

School of Engineering & Technology

K. R. MANGALAM UNIVERSITY

Sohna, Haryana 122103, India

LAB ASSIGNMENT -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-

```java
package org.example.Lab_OS;

import java.io.*;
import java.util.*;

public class ProcessManagement {

    // Task 1: Create N child processes (Windows compatible)
    public static void task1_createProcesses(int n) throws Exception {  1 usage
        System.out.println("=== Task 1: Process Creation ===");
        for (int i = 0; i < n; i++) {
            // Run "echo" command through Windows CMD
            ProcessBuilder pb = new ProcessBuilder( ...command: "cmd.exe", "/c", "echo Child " + (i + 1) + " is running!");
            Process p = pb.start();
            System.out.println("Started Child Process: PID=" + p.pid() + " | Parent PID=" + ProcessHandle.current().pid());

            // Read output of the child
            BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
            p.waitFor();
        }
        System.out.println("All child processes completed.\n");
    }
```

Output-

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagen
=== Task 1: Process Creation ===
Started Child Process: PID=25728 | Parent PID=21972
Child 1 is running!
Started Child Process: PID=10460 | Parent PID=21972
Child 2 is running!
Started Child Process: PID=3704 | Parent PID=21972
Child 3 is running!
All child processes completed.
```

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().

```
// Task 2: Execute system commands (Windows)
public static void task2_execCommands(String[] commands) throws Exception {  1 usage
    System.out.println("=== Task 2: Execute Commands ===");
    for (String cmd : commands) {
        System.out.println("Executing command: " + cmd);
        ProcessBuilder pb = new ProcessBuilder( ...command: "cmd.exe", "/c", cmd);
        pb.redirectErrorStream(true);
        Process p = pb.start();

        BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        p.waitFor();
        System.out.println("Command '" + cmd + "' completed.\n");
    }
}
```

Output –

```
=== Task 2: Execute Commands ===
Executing command: date /t
06-10-2025
Command 'date /t' completed.

Executing command: time /t
10:13
Command 'time /t' completed.

Executing command: whoami
jungkook\aditi
Command 'whoami' completed.
```

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 –

```java
// Task 3: Simulate Zombie and Orphan-like behavior (conceptually)
public static void task3_simulateProcesses() throws Exception {  1 usage
    System.out.println("=== Task 3: Simulating Zombie/Orphan ===");

    // Simulate "Zombie" - parent doesn't wait
    new Thread(() -> {
        try {
            Process zombie = new ProcessBuilder( ...command: "cmd.exe", "/c", "timeout /t 2").start();
            System.out.println("Zombie simulated: Child PID=" + zombie.pid() + " (Parent didn't wait)");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }).start();

    // Simulate "Orphan" - parent exits before child finishes
    Thread orphanParent = new Thread(() -> {
        try {
            Process orphan = new ProcessBuilder( ...command: "cmd.exe", "/c", "timeout /t 5").start();
            System.out.println("Orphan simulated: Child PID=" + orphan.pid());
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
    orphanParent.start();
    System.out.println("Parent thread exiting early (Orphan created)\n");
}
```

Output-

```
=== Task 3: Simulating Zombie/Orphan ===
Parent thread exiting early (Orphan created)
```

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 –**

```
}
// Task 4: Inspect process info (Windows version)
public static void task4_processInfo(long pid) throws Exception {  1 usage
    System.out.println("=== Task 4: Inspect Process Info ===");
    System.out.println("Process ID: " + pid);
    ProcessHandle process = ProcessHandle.of(pid).orElse( other: null);
    if (process != null) {
        ProcessHandle.Info info = process.info();
        System.out.println("Command: " + info.command().orElse( other: "N/A"));
        System.out.println("Start Time: " + info.startInstant().orElse( other: null));
        System.out.println("CPU Duration: " + info.totalCpuDuration().orElse( other: null));
        System.out.println("User: " + System.getProperty("user.name"));
    } else {
        System.out.println("Process not found!");
    }
}
```

Output-

```
=== Task 4: Inspect Process Info ===
Process ID: 21972
Command: C:\Program Files\Java\jdk-21\bin\java.exe
Start Time: 2025-10-06T04:43:14.811Z
CPU Duration: PT1.09375S
User: aditi
```

Task 5: Process Prioritization

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

Sol -

```
    // Task 5: Simulate process priority (Thread priority)
    public static void task5_prioritySimulation() {  1 usage
        System.out.println("=== Task 5: Priority Simulation ===");

        Runnable cpuTask = () -> {
            long start = System.currentTimeMillis();
            long sum = 0;
            for (long i = 0; i < 1e7; i++) sum += i;
            long end = System.currentTimeMillis();
            System.out.println(Thread.currentThread().getName() + " completed in " + (end - start) + "ms");
        };

        Thread low = new Thread(cpuTask,  name: "Low Priority");
        Thread high = new Thread(cpuTask,  name: "High Priority");

        low.setPriority(Thread.MIN_PRIORITY);   // Low = 1
        high.setPriority(Thread.MAX_PRIORITY);  // High = 10

        low.start();
        high.start();
    }

    public static void main(String[] args) throws Exception {
        task1_createProcesses( n: 3);
        task2_execCommands(new String[]{"date /t", "time /t", "whoami"});
        task3_simulateProcesses();
        task4_processInfo(ProcessHandle.current().pid());
        task5_prioritySimulation();
}
}
```

Output-

```
=== Task 5: Priority Simulation ===
Zombie simulated: Child PID=4188 (Parent didn't wait)
Orphan simulated: Child PID=468
High Priority completed in 20ms
Low Priority completed in 22ms

Process finished with exit code 0
```