



K. R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Operating System Lab

(ENCS351)

Lab File Submitted to

K. R. Mangalam University

for

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Aditi Saraswat (2301010020) (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-

```
1 package org.example.Lab_OS;
2
3 import java.io.*;
4 import java.util.*;
5
6 public class ProcessManagement {
7
8     // Task 1: Create N child processes (Windows compatible)
9     public static void task1_createProcesses(int n) throws Exception {
10         System.out.println("== Task 1: Process Creation ==");
11         for (int i = 0; i < n; i++) {
12             // Run "echo" command through Windows CMD
13             ProcessBuilder pb = new ProcessBuilder(...command: "cmd.exe", "/c", "echo Child " + (i + 1) + " is running!");
14             Process p = pb.start();
15             System.out.println("Started Child Process: PID=" + p.pid() + " | Parent PID=" + ProcessHandle.current().pid());
16
17             // Read output of the child
18             BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
19             String line;
20             while ((line = br.readLine()) != null) {
21                 System.out.println(line);
22             }
23             p.waitFor();
24         }
25         System.out.println("All child processes completed.\n");
26     }
27 }
```

Output-

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:  
== 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 {  
    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 –

```

// Task 3: Simulate Zombie and Orphan-like behavior (conceptually)
public static void task3_simulateProcesses() throws Exception {
    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 { 1usage
    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
```

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-

```

1 import java.io.IOException;
2 import java.util.logging.*;
3 import java.util.*;
4
5 > public class LabSheet2 {
6
7     // Logger setup
8     private static final Logger logger = Logger.getLogger(LabSheet2.class.getName());  8 usages
9
10    static {
11        try {
12            FileHandler fileHandler = new FileHandler( pattern: "process_log.txt", append: true);
13            fileHandler.setFormatter(new SimpleFormatter());
14            logger.addHandler(fileHandler);
15            logger.setUseParentHandlers(false); // avoid console duplication
16        } catch (IOException e) {
17            System.err.println("Error setting up logger: " + e.getMessage());
18        }
19    }
20
21    // Simulated system process (task)
22    static class SystemProcess extends Thread { no usages
23        private String taskName;  4 usages
24
25        public SystemProcess(String taskName) { no usages
26            super(taskName);
27            this.taskName = taskName;
28        }
29
30        @Override
31        public void run() {
32            try {
33                logger.info( msg: taskName + " started");
34                Thread.sleep( millis: 2000); // Simulate process execution
35                logger.info( msg: taskName + " ended");
36            } catch (InterruptedException e) {
37                logger.warning( msg: taskName + " interrupted");
38            }
39        }
40    }
41

```

```

public static void main(String[] args) {
    System.out.println("System Starting...");
    logger.info( msg: "System Boot Sequence Initiated");

    // Create simulated processes
    SystemProcess p1 = new SystemProcess( taskName: "Process-1");
    SystemProcess p2 = new SystemProcess( taskName: "Process-2");

    // Start processes concurrently
    p1.start();
    p2.start();

    // Wait for both to finish
    try {
        p1.join();
        p2.join();
    } catch (InterruptedException e) {
        logger.warning( msg: "Main system thread interrupted");
    }

    System.out.println("System Shutdown.");
    logger.info( msg: "System Shutdown Completed");
}

```

Output-

```
"C:\Program Files\Java\jdk-21\bin\java.  
System Starting...  
System Shutdown.  
  
Process finished with exit code 0
```

Process Log

```
os\OS_Lab ① Main.java ② pom.xml (OS_Lab) ③ ProcessManagement.java ④ LabSheet2.java ⑤ process_log.txt  
1 Oct 07, 2025 10:45:30 PM LabSheet2 main  
2 INFO: System Boot Sequence Initiated  
3 Oct 07, 2025 10:45:30 PM LabSheet2$SystemProcess run  
4 INFO: Process-1 started  
5 Oct 07, 2025 10:45:30 PM LabSheet2$SystemProcess run  
6 INFO: Process-2 started  
7 Oct 07, 2025 10:45:32 PM LabSheet2$SystemProcess run  
8 INFO: Process-2 ended  
9 Oct 07, 2025 10:45:32 PM LabSheet2$SystemProcess run  
10 INFO: Process-1 ended  
11 Oct 07, 2025 10:45:32 PM LabSheet2 main  
12 INFO: System Shutdown Completed  
13 Oct 07, 2025 10:46:22 PM LabSheet2 main  
14 INFO: System Boot Sequence Initiated  
15 Oct 07, 2025 10:46:22 PM LabSheet2$SystemProcess run  
16 INFO: Process-1 started  
17 Oct 07, 2025 10:46:22 PM LabSheet2$SystemProcess run  
18 INFO: Process-2 started  
19 Oct 07, 2025 10:46:24 PM LabSheet2$SystemProcess run  
20 INFO: Process-1 ended  
21 Oct 07, 2025 10:46:24 PM LabSheet2$SystemProcess run  
22 INFO: Process-2 ended  
23 Oct 07, 2025 10:46:24 PM LabSheet2 main  
24 INFO: System Shutdown Completed  
25
```