**Lab Assignment Sheet-2**

**School of Engineering and Technology**  
**Course Code & Name: ENCS351 /Operating System**  
**Program Name: B.Tech CSE, AI ML, Data Science, Cyber, FSD, UX/UI**

**Submission Guidelines**

***General Requirements***

* **Submission Deadline:** Assignments must be submitted within one week of the assignment's release date.
* **Submission Platform:** All assignments are to be submitted via the Learning Management System (LMS).
* **GitHub Link:** You must provide a link to your GitHub repository with your submission.
* **Individual Submission:** Assignments are to be completed and submitted by each individual student.
* **Formatting:** All assignments must adhere to the specific format shared in class.

***Evaluation***

* **Total Marks:** This assignment is worth a total of 5 marks.
* **Evaluation Metrics:** Assignments will be evaluated based on the following criteria:
* **Originality:** The uniqueness and independent thought demonstrated in the work.
* **Correctness:** The accuracy and validity of the solutions or content.
* **Completeness:** The extent to which all parts of the assignment have been addressed.

# Problem Title:

System Startup, Process Creation, and Termination Simulation in Python

# Problem Statement:

# Modern operating systems are responsible for initializing system components, creating processes, managing execution, and gracefully shutting down. This lab aims to simulate these core concepts using Python, helping students visualize how processes are handled at the OS level. The focus is on creating a simplified startup mechanism that spawns multiple processes and logs their lifecycle using the multiprocessing and logging modules. This hands-on simulation enhances conceptual clarity and promotes coding proficiency in scripting real-world OS behavior.

# Tools/Technology Used:

• Python 3.x  
• multiprocessing module  
• time module  
• logging module

**Learning Objectives:**  
1.Upon completing this lab, students will be able to:

2. Understand the concepts of system booting, process creation, and termination.

3. Develop Python scripts using multiprocessing and logging modules.

4. Simulate system behaviour using programming constructs.

### Assignment Tasks:

1. Write a Python script to simulate a basic system startup sequence.
2. Use the multiprocessing module to create at least two child processes that perform dummy tasks.
3. Implement proper logging to track process start and end times.
4. Generate a log file (process\_log.txt) to reflect system-like behavior.
5. Submit the Python script and log file along with a short report explaining your implementation.

### 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.

#### ****Sub-Task 1: Initialize the logging configuration****

**Objective:** Set up the logging system to log messages with timestamps and process names.

import logging

# Setup logger

logging.basicConfig(

filename='process\_log.txt',

level=logging.INFO,

format='%(asctime)s - %(processName)s - %(message)s'

)

#### ****Sub-Task 2: Define a function that simulates a process task****

**Objective:** Write a function that mimics the work of a system process.

import time

# Dummy function to simulate a task

def system\_process(task\_name):

logging.info(f"{task\_name} started")

time.sleep(2) # Simulate task delay

logging.info(f"{task\_name} ended")

#### ****Sub-Task 3: Create at least two processes and start them concurrently****

**Objective:** Use the multiprocessing module to initiate parallel tasks.

import multiprocessing

if \_\_name\_\_ == '\_\_main\_\_':

print("System Starting...")

# Create processes

p1 = multiprocessing.Process(target=system\_process, args=('Process-1',))

p2 = multiprocessing.Process(target=system\_process, args=('Process-2',))

# Start processes

p1.start()

p2.start()

#### ****Sub-Task 4: Ensure proper termination and verify logs****

**Objective:** Wait for processes to complete and confirm the shutdown.

# Wait for processes to complete

p1.join()

p2.join()

print("System Shutdown.")

# Program Code with Explanation:

The following Python script simulates a simple system startup. It creates a few dummy processes which perform simple tasks. Each process logs its start and end, and the logger records the process flow, simulating system behavior.

# Import required libraries  
import multiprocessing  
import time  
import logging  
  
# Setup logger  
logging.basicConfig(filename='process\_log.txt', level=logging.INFO,  
 format='%(asctime)s - %(processName)s - %(message)s')  
  
# Dummy function to simulate a task  
def system\_process(task\_name):  
 logging.info(f"{task\_name} started")  
 time.sleep(2)  
 logging.info(f"{task\_name} ended")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("System Starting...")  
  
 # Create processes  
 p1 = multiprocessing.Process(target=system\_process, args=('Process-1',))  
 p2 = multiprocessing.Process(target=system\_process, args=('Process-2',))  
  
 # Start processes  
 p1.start()  
 p2.start()  
  
 # Wait for processes to complete  
 p1.join()  
 p2.join()  
  
 print("System Shutdown.")

# Expected Output:

• The terminal displays 'System Starting...' and 'System Shutdown.'  
• A log file 'process\_log.txt' is generated which records the start and end of each process.

# Sample process\_log.txt content:

2025-07-16 12:35:21,005 - Process-1 - Process-1 started  
2025-07-16 12:35:21,006 - Process-2 - Process-2 started  
2025-07-16 12:35:23,007 - Process-1 - Process-1 ended  
2025-07-16 12:35:23,008 - Process-2 - Process-2 ended

**Student Instructions:**

* Submit a .py script file along with a brief report (.pdf/.docx) explaining your implementation.
* Ensure the log file process\_log.txt is included as evidence of execution.
* Maintain proper code comments and structure.
* Submit your files through LMS/Email before the deadline.
* Late submissions will incur grade penalties as per university policy

# Rubrics for Evaluation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **Weight** | **Excellent (5)** | **Good (4)** | **Fair (3)** | **Poor (2 or less)** |
| **Implementation & Execution** | 40% | Correct, complete simulation of all process features | Minor issues in simulation or partial output | Some features missing or errors in output | Incorrect or no execution |
| **Code Quality & Documentation** | 30% | Well-structured, commented, and readable code | Mostly clear code with few comments missing | Basic structure, minimal documentation | Poor structure, unclear or no comments |
| **Logging & Output Accuracy** | 20% | Accurate logging; output matches expected behavior | Minor mismatch in logs or output | Incomplete logs/output | Missing or incorrect output/logs |
| **Report & Submission** | 10% | Report well-organized and submitted on time | Report clear but minor delay or issues | Minimal report or late submission | No report or unacceptable delay |

**Date Assigned:**   
**Last Date of Submission:**

**Contact for Queries:**

Faculty ’s Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_