Title: System Startup, Process Creation, and Termination Simulation in Python
Course Code: ENCS351 – Operating System
Program: B.Tech CSE
Name: Lavya Kumar Beriwal
Roll no.: 2301010012

Lab Sheet 2

Objective

To simulate the startup, process creation, and termination mechanisms of an operating system using Python's `multiprocessing` and `logging` modules.

Tools and Technologies

- Python 3.x
- `multiprocessing` module
- `logging` module
- `time` module

Concepts Implemented

- System initialization simulation
- Process creation using `multiprocessing.Process()`
- Logging process lifecycle with timestamps
- Controlled shutdown after all processes complete

Code Explanation

1. Logging Configuration:
The `logging` module is initialized to store timestamped logs in `process_log.txt`.

```
logging.basicConfig(
    filename='process_log.txt',
    level=logging.INFO,
    format='%(asctime)s - %(processName)s - %(message)s'
)
```

2. Dummy Process Function:
A function `system_process()` simulates a system process by logging its start and end.

```
def system_process(task_name):
    logging.info(f"{task_name} started")
    time.sleep(2)
    logging.info(f"{task_name} ended")
```

3. Process Creation and Management:
Two processes are created and executed concurrently. Each simulates a separate system process.

```
p1 = multiprocessing.Process(target=system_process, args=('Process-1',))
p2 = multiprocessing.Process(target=system_process, args=('Process-2',))
p1.start()
p2.start()
p1.join()
p2.join()
```

Expected Output

Terminal Output:

```
System Starting...
System Shutdown.
```

Generated Log File (process_log.txt):

```
2025-10-06 21:25:42,110 - MainProcess - System Booting...
2025-10-06 21:25:42,111 - Process-1 - Process-1 started
2025-10-06 21:25:42,112 - Process-2 - Process-2 started
2025-10-06 21:25:44,115 - Process-1 - Process-1 ended
2025-10-06 21:25:44,116 - Process-2 - Process-2 ended
2025-10-06 21:25:44,117 - MainProcess - All processes completed successfully.
```

Learning Outcomes

✅ Gained understanding of OS-level process creation and management.
✅ Learned how to use `multiprocessing` for concurrent task execution.
✅ Understood how system logging tracks process activity.
✅ Observed synchronization through `join()` method.

Complexity Analysis

- **Time Complexity:** O(n), where n = number of processes.

- **Space Complexity:** O(n), storing process logs and references.

Conclusion

This experiment successfully simulates system startup and termination, showing how real OS components initiate, run, and shut down gracefully. It provides a practical view of process lifecycle and interprocess synchronization using Python.

Screenshot