Assignment 2

Course Code & Name: ENCS351 /Operating System

Name: Shreya Aggarwal

Roll no: 2301420016

Problem: System Startup, Process Creation, and Termination Simulation in Python

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.

Tasks Implementation

Sub-Task 1: Initialize the logging configuration

os_ass_2301420016.py > ...

```python
import multiprocessing
import time
import logging

#subtask 1: Setup logger
logging.basicConfig(
    filename='process_log.txt',
    level=logging.INFO,
    format='%(asctime)s - %(processName)s - %(message)s'
)
```

This task sets up a basic script that simulates a system boot process. It prints messages like "System Starting" and "System Shutdown," while internally initializing logging. This provides the foundation for mimicking real system startup and shutdown sequences in a controlled environment.

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

```
#subtask 2: 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")
```

This task uses Python's multiprocessing module to create independent child processes. Each process runs concurrently, simulating separate system services. By spawning at least two processes, we mimic multitasking in an operating system, where different components or programs execute simultaneously without interfering with one another.

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

```
# subtask 3: Create two process and run them concurrently
if __name__ == '__main__':
    print("System Starting...")
    logging.info("System startup initiated")

    #create the 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()
```

Logging ensures process execution is tracked with clear, timestamped messages. Each process logs when it starts and ends, providing a record in process_log.txt. This allows us to monitor behavior, debug issues, and verify system-like activities, similar to logs maintained by real operating systems.

Sub-Task 4: Ensure proper termination and verify logs

```
#subtask 4: Ensure proper termination and verify logs
    p1.join()
    p2.join()

    logging.info("System shutdown complete")
    print("System Shutdown.")
```

This task ensures proper process lifecycle management. After starting, all processes are terminated gracefully using .join(), ensuring no orphan processes remain. The final log entries confirm shutdown completion. Verifying logs in process_log.txt validates that tasks executed correctly and system-like behavior was accurately simulated.

Output

```
PS C:\Users\Shreya aggarwal\OneDrive\Desktop\DocumentsOS_Lab> python
 os_ass_2301420016.py
System Starting...
System Shutdown.
PS C:\Users\Shreya aggarwal\OneDrive\Desktop\DocumentsOS_Lab>
```

Process_log.txt file

process_log.txt

```
1   2025-09-26 20:50:24,221 - Process-1 - Process-1 started
2   2025-09-26 20:50:24,227 - Process-2 - Process-2 started
3   2025-09-26 20:50:26,225 - Process-1 - Process-1 ended
4   2025-09-26 20:50:26,229 - Process-2 - Process-2 ended
5
```