Assignment2

NAME - Abdul Waheed Al Faaiz

Roll No. - 2301010468

COURSE - BTech CSE Core - 7

Operating System

Introduction

This lab assignment focuses on simulating the core functions of an operating system during system startup, process creation, execution, and termination. In real-world operating systems, the kernel is responsible for initializing system components, spawning processes, managing their execution, and ensuring proper termination.

To simplify this concept for learning purposes, the task is implemented in Python using the multiprocessing and logging modules. The simulation creates multiple dummy processes that run concurrently, and their activities are recorded in a log file with timestamps.

By completing this lab, students gain practical exposure to:

- How processes are created and executed in parallel.
- How logging can track system-like events.
- The importance of proper process termination and synchronization.

This experiment helps visualize the basic behavior of process management at the operating system level through a hands-on programming approach.

Implementation Details

The simulation was implemented using Python 3.x on a Linux environment. Three main modules were utilized:

1. **multiprocessing** – to create and manage multiple processes that run concurrently.
2. **logging** – to record the activity of each process with timestamps and process names, simulating system logs.

3. **time** – to introduce delays in execution, representing a real task being performed.

Steps Implemented:

- **Logging Setup**: A logger was configured to output messages in the format timestamp – processName – message into a file named process_log.txt.
- **Process Task Function**: A function was defined to simulate system tasks. Each process logs when it starts and ends, with a 2-second delay to represent execution.
- **Process Creation**: At least two processes were created using the multiprocessing.Process class and started concurrently.
- **Process Termination**: The parent process used .join() to ensure that both child processes completed execution before the system shutdown message was displayed.

This design replicates how an operating system initializes processes, tracks their activity, and ensures orderly shutdown.

Python Code

```
  GNU nano 8.6                  system_simulation.py
# Import required libraries
import multiprocessing
import time
import logging

# ------------------ Logging Setup ------------------
logging.basicConfig(
    filename='process_log.txt',    # Log file
    level=logging.INFO,            # Logging level
    format='%(asctime)s - %(processName)s - %(message)s'
)

# ------------------ Process Function ------------------
def system_process(task_name):
    """Simulates a system process by logging start and end, with delay.>
    logging.info(f"{task_name} started")
    time.sleep(2)  # Simulate process execution
    logging.info(f"{task_name} ended")

# ------------------ Main Program ------------------
if __name__ == '__main__':
    print("System Starting...")

    # Create processes
    p1 = multiprocessing.Process(target=system_process, args=('Process->
    p2 = multiprocessing.Process(target=system_process, args=('Process->

    # Start processes
    p1.start()
    p2.start()
```

```python
# ------------------ Main Program ------------------
if __name__ == '__main__':
    print("System Starting...")

    # Create processes
    p1 = multiprocessing.Process(target=system_process, args=('Process-
    p2 = multiprocessing.Process(target=system_process, args=('Process-

    # Start processes
    p1.start()
    p2.start()

    # Wait for processes to complete
    p1.join()
    p2.join()

    print("System Shutdown.")
```

Execution Proof

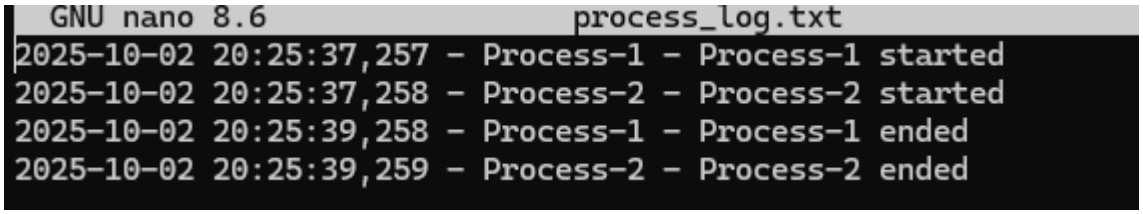
```
┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_Assignment2]
└─$ nano system_simulation.py

┌──(kalilinux㉿DESKTOP-71JCI24)-[~/OS_Lab_Assignment2]
└─$ python3 system_simulation.py
System Starting...
System Shutdown.
```

This shows that the system successfully initialized, executed the processes, and then shut down after all processes had finished.

Log File (process_log.txt)

A log file is automatically generated in the same directory. It contains timestamped entries for each process, recording when they started and ended.

Output from process_log.txt



Conclusion

This lab assignment successfully demonstrated the basic concepts of system startup, process creation, execution, and termination using Python. By leveraging the multiprocessing and logging modules, we simulated how an operating system manages processes and maintains logs.

Key takeaways from this implementation include:

- Understanding how multiple processes can be created and executed concurrently.
- Learning how to track system activities using logging with timestamps.
- Realizing the importance of proper process termination and synchronization (join() function).

Overall, this experiment provided a clear and practical way to visualize fundamental operating system behavior, strengthening both theoretical understanding and programming skills.