

Lab Report: Process Management in Python

Course: ENCS351 Operating System

Student: Arman

Course : B.Tech CSE

Roll no. 2301010037

1. Objective

The objective of this assignment was to simulate a basic operating system startup, process creation, and shutdown sequence. The goal was to use Python's multiprocessing module to create concurrent processes and the logging module to track their lifecycles, thereby gaining a practical understanding of how an OS manages multiple tasks.

2. Implementation Details

The simulation was implemented in a single Python script, `system_startup_simulation.py`, which performs the following steps:

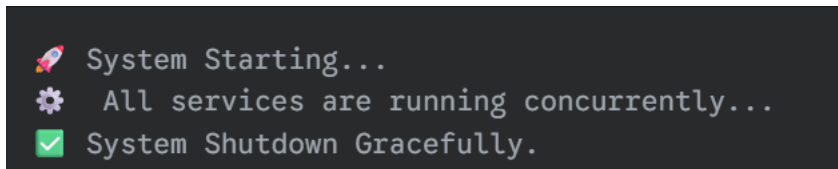
- **Logging Configuration:** The logging module was configured at the start of the script. It was set to write to a file named `process_log.txt` with `filemode='w'` to ensure a clean log for each run. The log format `%(asctime)s - %(processName)s - %(message)s` was chosen to clearly record the timestamp, the name of the process generating the log, and the log message itself.
- **Simulated Service Function:** A function named `system_service` was created to represent a typical system process (e.g., a network manager or a database service). This function accepts a `service_name`, logs its start, simulates work by pausing for a random duration between 2 to 4 seconds using `time.sleep()`, and finally logs its completion.
- **Main Execution Block** (`if __name__ == '__main__':`): This standard Python construct ensures the process creation code runs only when the script is executed directly. The main block performs the "boot sequence":

1. It first logs that the system boot has been initiated.
2. It then creates multiple multiprocessing.Process objects. Each process is assigned the system_service function as its target and given a descriptive name (e.g., 'NetworkService').
3. Each process is started using the .start() method, which allows them to run concurrently in parallel.
4. The main process then calls the .join() method on each child process. This forces the main process to wait until all child processes have completed their execution.
5. Once all join() calls return, it signifies that all services have finished, and a final shutdown message is logged.

3. Results and Verification

The script ran successfully, producing two outputs: console messages and a detailed log file.

- Console Output:



```
🚀 System Starting...
⚙️ All services are running concurrently...
✅ System Shutdown Gracefully.
```

- Log File (process_log.txt): The log file provides clear evidence of the simulation. The timestamps confirm that the services (NetworkService, DatabaseService, etc.) started at nearly the same time and ran in parallel. The processName field in each log entry correctly identifies whether the message came from the MainProcess or one of the child processes, perfectly capturing the entire system lifecycle from boot to shutdown.