Indian Institute of Technology, Jodhpur

Department of Computer Science and Engineering

CSL3030: Operating Systems Lab

LAB ASSIGNMENT IV

Lab Assignment IV: Dynamic and Multi-Level CPU Scheduling

Objective: This assignment moves beyond basic scheduling algorithms. You will implement two advanced schedulers that adapt to system conditions and manage different types of processes. The goal is to understand how schedulers solve critical problems like process starvation and the need to prioritize interactive tasks over batch jobs.

The simulation must account for **I/O bursts** and a non-zero **context switching overhead**.

Task 1: The Anti-Starvation Priority Scheduler

In a simple preemptive priority system, low-priority processes can be starved by a continuous stream of high-priority tasks. Your first task is to solve this by implementing **aging**.

Algorithm to Implement:

- **Preemptive Priority Scheduling with Aging**

Requirements:

1. Implement a preemptive priority scheduler where a process's priority dynamically increases the longer it waits in the ready queue.
2. **Aging Mechanism:** For every **X** milliseconds a process spends waiting in the ready queue, its priority value should decrease by 1 (numerically lower values represent higher priority). X will be an input parameter.
3. Your simulation must manage a **ready queue** and a **blocked queue** for processes performing I/O.
4. Display a detailed **Gantt chart** showing process execution and context switches.
5. Calculate and display the **Waiting Time (WT)**, **Turnaround Time (TAT)**, and **Response Time (RT)** for each process, along with the **CPU Utilization**.

6. **Bonus:** Display a table showing how the priority of each process changed over its lifetime in the system.

Input Format:

1. Number of processes.
2. For each process: Arrival Time, Initial Priority, CPU Burst 1, I/O Burst Time, CPU Burst 2.
3. Aging Interval X.
4. Context Switch Time.

Task 2: The Multi-Level Queue Scheduler

Different types of processes have different scheduling needs. Interactive (foreground) jobs need a fast response time, while batch (background) jobs just need to finish eventually. Your second task is to build a scheduler that handles both.

Algorithm to Implement:

- **Multi-Level Queue Scheduling**

Requirements:

1. Implement a scheduler with two queues:
    - **Queue 1 (Foreground):** Uses **Round-Robin (RR)** scheduling.
    - **Queue 2 (Background):** Uses **First-Come, First-Serve (FCFS)** scheduling.
2. **Inter-Queue Scheduling:** The Foreground queue has **absolute priority** over the Background queue. The FCFS queue will only run when the RR queue is completely empty.
3. Processes are permanently assigned to a queue based on their type (given in the input).
4. A process in the FCFS queue that is executing will be **preempted** if a new process arrives in the high-priority RR queue.
5. Display a single, unified **Gantt chart** showing execution from both queues.
6. Calculate and display the **WT, TAT, RT,** and **CPU Utilization** for all processes combined.

Input Format:

1. Number of processes.
2. For each process: Arrival Time, Process Type (0 for Foreground, 1 for Background), CPU Burst 1, I/O Burst Time, CPU Burst 2.
3. Time Quantum for the RR queue.
4. Context Switch Time.

Final Submission

- Submit a single program that allows the user to choose which scheduler (Task 1 or Task 2) to run.
- Include a `README.md` file containing a brief **comparative analysis**. Discuss the problem that each scheduler solves and describe a scenario where you would choose one over the other.