Report

1. **Gotta count ‘em all:**

Changes:

- Added getSysCount system call to kernel.
- Implemented tracking of system call counts by checking the provided bitmask for the specified syscall.
- Modified syscall() to count calls to specific syscalls by incrementing the count whenever a system call is called.
- Updated wait () to ensure the counts include calls made by child processes.
- Created a user program syscount to handle mask input and execute the corresponding command and print the output.

2. **Wake me up when my timer ends:**

Changes:

- Created sigalarm() to set an interval and a handler function. The process's context is saved and the handler function is called after the specified number of ticks.
- Added a sigreturn() system call to restore the process's state after the handler finishes, allowing it to resume from where it was interrupted.
- Modified trap.c to handle the periodic check for process tick intervals and trigger the handler when appropriate.

3. Lottery-Based Scheduler (LBS):

Changes:

- Modified proc.c to include a tickets field in the proc struct, initialized with 1 ticket.

- Added settickets(int number) system call to allow processes to set their ticket count.
- Implemented the lottery scheduling logic in scheduler () where the process is chosen based on a random draw and probability is proportional to number of tickets it has.
- Included a tie-breaker mechanism to select the process that arrived earlier when multiple processes have the same number of tickets, where the process is selected with least time of arrival.

4. MLFQ (Multi-Level Feedback Queue) Scheduling:

Changes:

- Added four priority queues in proc.c, with different time slices: 1 tick for priority 0, 4 ticks for priority 1, 8 ticks for priority 2, and 16 ticks for priority 3.
- Modified allocproc() to insert new processes into the highest priority queue (queue 0) but at the end of the.
- Implemented logic to promote processes to lower priority queues if they exhaust their time slice and reset priority through priority boosting every 48 ticks.
- Used round-robin scheduling within the lowest priority queue (queue 3).
- Added logic to preempt lower-priority processes when a higher-priority process becomes available.

Performance Comparison

To compare the performance of the three scheduling policies—Round Robin (RR), Lottery Based Scheduling (LBS), and Multi-Level Feedback Queue (MLFQ)—we measured the **average running time (rtime)** and **average**

waiting time (wtime) for processes. The tests were conducted using the schedulertest command, and the processes were set to run on only 1 CPU for uniform evaluation.

The results are as follows:

Scheduler	Average Running Time (rtime)	Average Waiting Time (wtime)
Round Robin	(9-10) ticks	113 ticks
LBS	(6-7) ticks	115 ticks
MLFQ	(9-10) ticks	109 ticks

Analysis of Lottery-Based Scheduling (LBS):

- Adding arrival time in lottery-based scheduling ensures that if multiple processes have the same number of tickets, the process that arrived earlier is favored. This prevents unfair scenarios where newer processes might win the lottery over older ones with equal tickets, ensuring fairness in CPU allocation.

  Example:

  Consider three processes:

  - **P1** arrives at time **t=0** with 3 tickets,
  - **P2** arrives at **t=1** with 4 tickets,
  - **P3** arrives at **t=2** with 3 tickets.

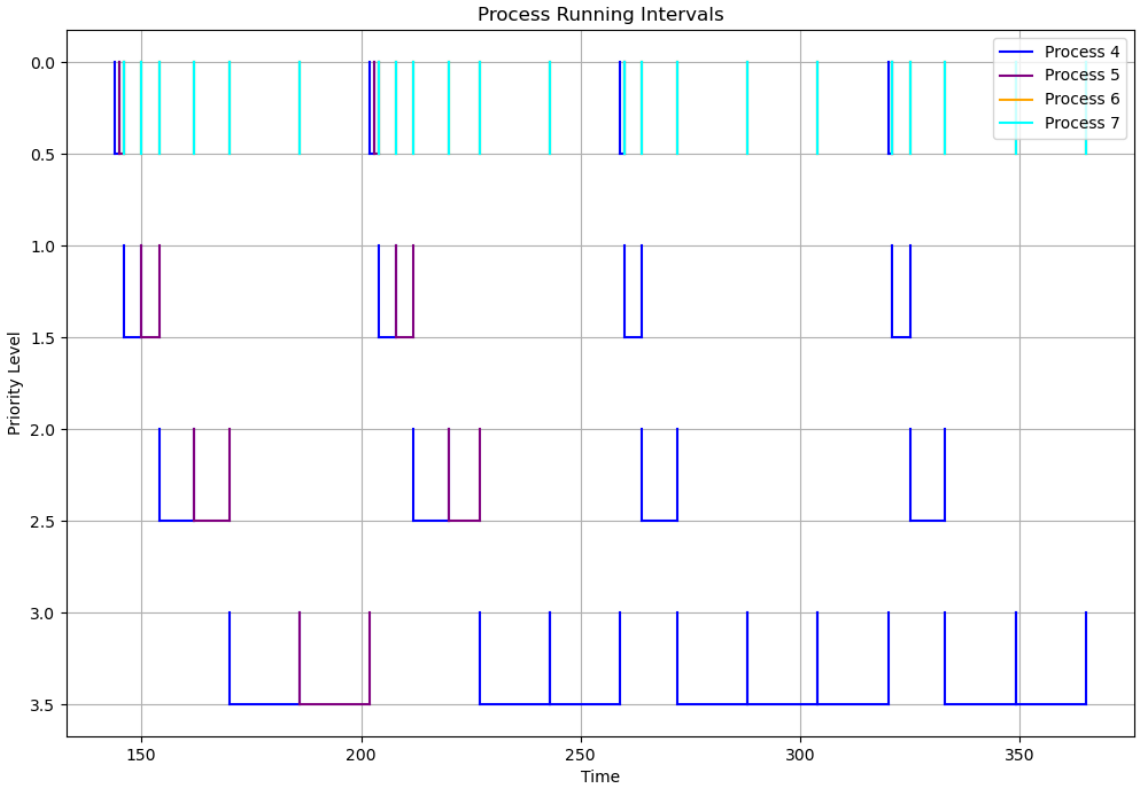
  If the lottery selects P3 as the winner at **t=3**, P1 will run instead because it arrived earlier and has the same

number of tickets. If P2 is chosen, it will run since it has more tickets than both P1 and P3.

- Pitfalls to watch out for include potential starvation of later-arriving processes if earlier ones continuously win the lottery. If all processes have the same number of tickets, the scheduler will prioritize the one that arrived earliest, which could lead to some processes waiting indefinitely if there's no mechanism to rotate fairness over time. Example: If P1 continuously wins due to earlier arrival, P2 and P3 may suffer starvation, especially if no mechanism is in place to eventually prioritize newer processes.

- If All Processes Have the Same Number of Tickets: When all processes have the same number of tickets, the scheduler essentially behaves like a First-Come, First-Serve (FCFS) algorithm. For example, if P1, P2, and P3 all have 3 tickets and arrive at times t=0, t=1, and t=2, respectively, the process that arrived first (P1) will always be prioritized when a tie occurs, leading to a behavior that mimics FCFS.

MLFQ Analysis:



The graph illustrates the priority changes of various processes over time within a Multi-Level Feedback Queue (MLFQ) scheduling system. On the x-axis, time steps represent event occurrences, while the y-axis displays priority levels, with higher priorities at the top. Each process of interest (Processes 4, 5, 6, and 7) is assigned a distinct color, while other processes are shown in black.

In an MLFQ system, processes can move between different priority levels based on their behavior and resource usage. A process may be boosted to a higher priority when it has to wait for too long, or its priority may decrease if it consumes a

lot of CPU time. The vertical lines in the graph indicate priority boosts (increased color) and decreases (decreased color) for the respective processes, providing a visual representation of how MLFQ dynamically adjusts process priorities to optimize CPU scheduling and improve responsiveness.