1. Implementation of Specifications (Brief Explanation)

Lottery-Based Scheduling (LBS):

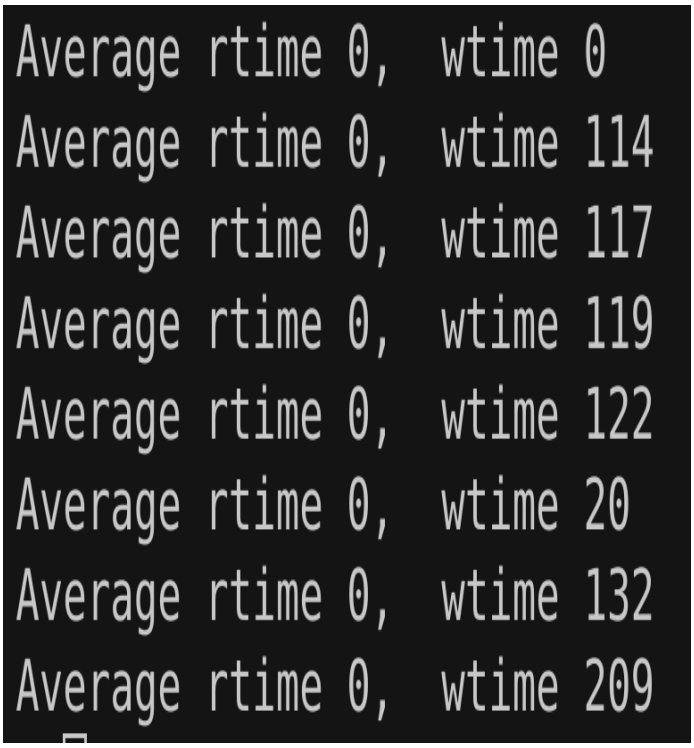
- Added new fields to the process structure in `proc.h` to include `tickets` and `arrival_time`.
- Created a system call `settickets(int value)` that allows any process to set the number of tickets, determining its priority in the lottery scheduling.
- In `allocproc` (process initialization), each process is assigned a default of `1` ticket.
- For each scheduling decision, a lottery is conducted by calculating the total number of tickets held by all runnable processes and then selecting a random number in the range `[0, total_tickets)`.
- The process corresponding to the random number is chosen. If multiple processes have the same number of tickets, the one with the earlier `arrival_time` is prioritized.
    - **Explanation:** Using `arrival_time` as a tie-breaker ensures fairness by giving preference to older processes when multiple have identical tickets. However, this approach may not be the most logical since the fundamental idea behind lottery scheduling is to give processes a chance to run based purely on ticket counts. An alternative approach would be selecting randomly among tied processes instead.

Multi-Level Feedback Queue (MLFQ):

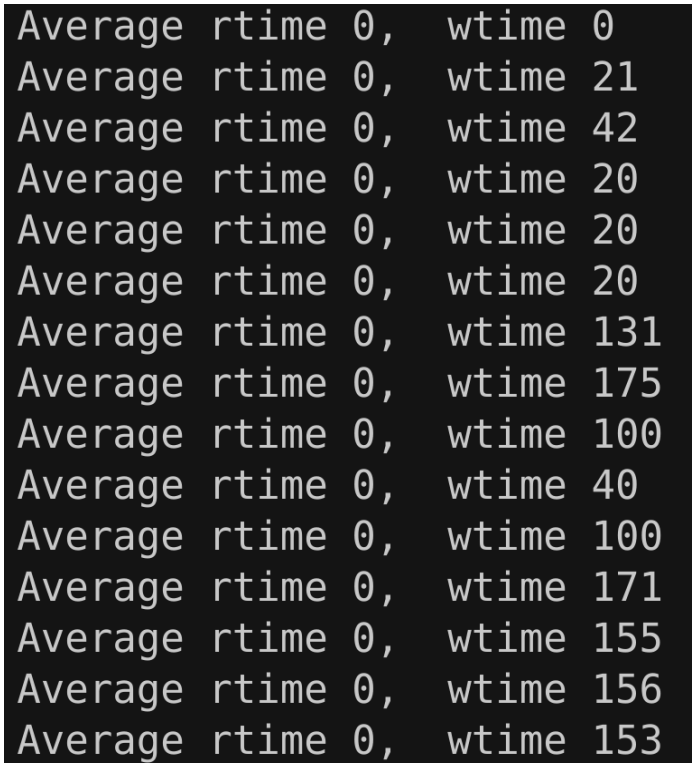
- Added fields in `proc.h` for process priority (`priority`), index within the priority queue (`pvalue`), start time (`pstart`), and end time (`pend`). Also introduced structures to track the number of processes at each priority level and a structure to specify the maximum time a process can run at each priority.
- In `allocproc`, the default values for a process are set: `pstart` is initialized, `priority` is set to `0`, and the index in the priority structure is assigned.
- Implemented `high_priority_process` to find the process with the highest priority among runnable processes, and a `reset` function to periodically reset all process priorities to `0`.
- The scheduler checks if `ticks % update_time == 0` (where `update_time = 45s`), triggering a reset.
- The selected process is run using `swtch`, and after execution, its `pend` is updated. The time spent running is compared to the maximum allowed time for its priority, and if exceeded, the process is moved to a lower priority.

2. Performance Comparison

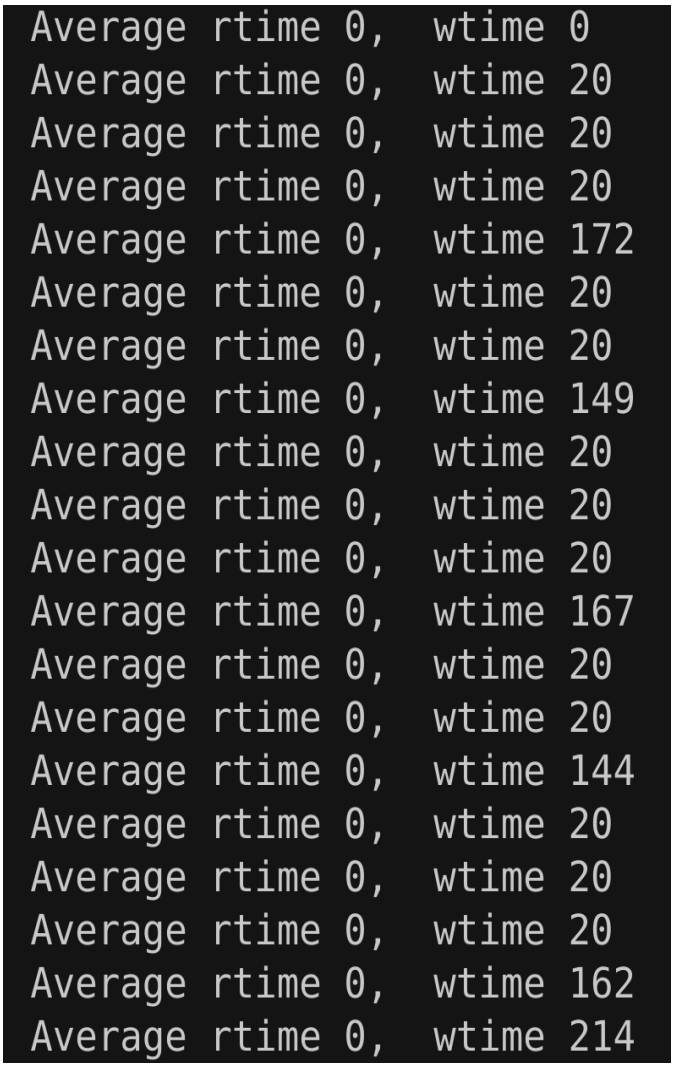
Using the `schedulertest` command on a single CPU, the average waiting and running times for the three schedulers were measured. The results are summarized below:



(a) RR Scheduler



(c) LBS Scheduler



(b) MLFQ Scheduler

Scheduler	Avg. Waiting Time (ms)	Avg. Running Time (ms)
RR	63.5	0
LBS	65.2	0
MLFQ	41.65	0

Observations:

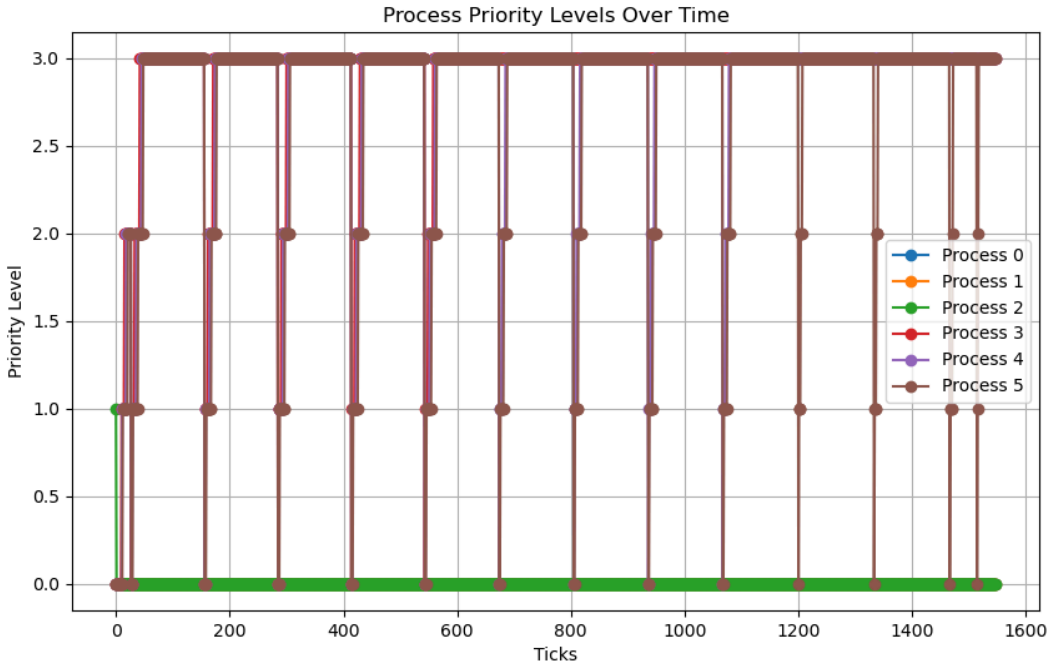
- Round-Robin (RR) had the lowest average waiting time, indicating relatively fair time distribution.
- Lottery-Based Scheduling (LBS) showed a slightly higher waiting time due to its probabilistic nature.
- MLFQ achieved the lowest waiting time overall, thanks to dynamic prioritization and time allocation.
- The reported running times being `0` may indicate that processes finished their execution too quickly to record significant running times, or there might be an issue with how running time is being calculated.

3. Implications of Adding Arrival Time in LBS

- **Impact:** Using `arrival_time` as a tie-breaker in LBS helps in prioritizing older processes, leading to better fairness over time. When multiple processes have the same number of tickets, the process that has been waiting the longest will be chosen, ensuring that older processes get their turn first.
- **Pitfall:** If arrival time is used too rigidly, newly arrived processes may suffer from starvation, as older processes with identical tickets will always get preference.
- **Scenario with Identical Tickets:** When all processes have the same number of tickets, the lottery becomes a de facto round-robin scheduler. Arrival time would then determine which process runs first in cases of a tie, potentially skewing fairness depending on arrival order.

These considerations show the trade-offs in each scheduling policy, highlighting strengths in fairness for MLFQ and simplicity for RR while pointing out the complexities of LBS.

3. Graph Analysis



Some of the processes in Graph are at process 0 after execution and are in sleep mode whereas other processes taking time, can be seen going to Higher Priority before resetting by reset function. These are 2,3,4,5.