Rate Monotonic Scheduling

1. First read input from inp-params.txt.
2. Then calculate the total time for execution, which is the maximum value among the values of k*period of all processes.
3. Next for each unit time, scheduling the processes based on their priority.
4. Priority is calculated based on period lengths, if the process with smaller period length has higher priority.
5. Now find process with minimum period and assign its ID to next_process variable.
6. We have an array which stores all the process IDs in the execution order.
7. After assigning value to next_process, we check if this process has its execution time > 0 and k > 0, if both these satisfy then add this process to the execQueue array, and decrement exectime by 1.
8. Then finding waiting time for processes that are not currently executing.
9. And then incrementing the misses count if the process has not completed its execution in its period time.
10. Also, if time is multiple of period, k is > 1 and remaining period is 0, allocate the remaining period, remaining exectime the original values and decrement k by 1.
11. This scheduling is done in a for loop which iterates for total_time. If the CPU is idle, then 0 is added to the execQueue.
12. Once this is done, write all the required information into stats file.
13. Now we iterate over the execQueue array and based on the ID values (start, end, pre-empted), write into log file.
14. Close all the files, and we are done with RMS scheduling.

Earliest Deadline First

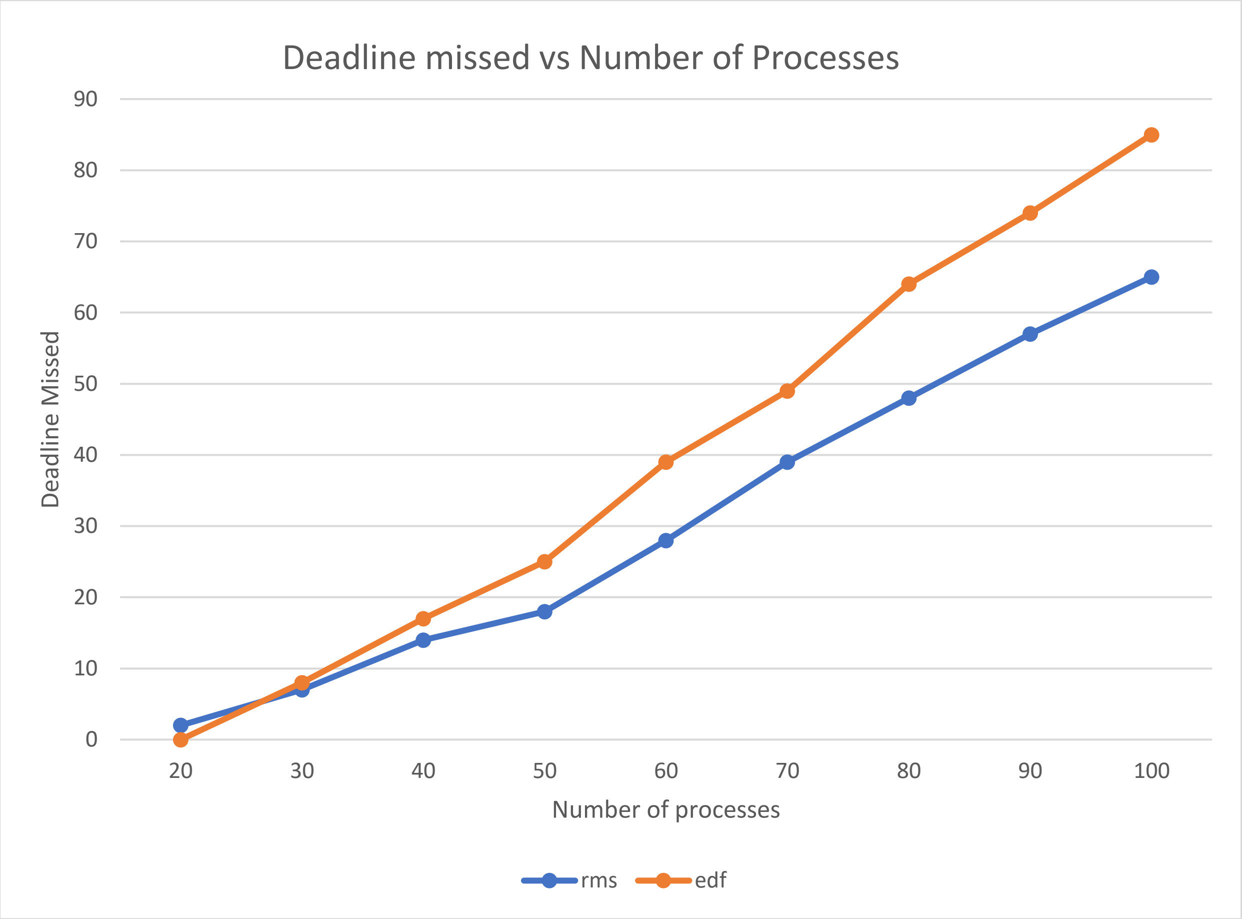
1. First read input from inp-params.txt.
2. Then calculate the total time for execution, which is the maximum value among the values of k*period of all processes.
3. Next for each unit time, scheduling the processes based on their priority.
4. Priority is calculated based on deadline, the process with earliest deadline has higher priority.
5. Now find process with earliest deadline and assign its ID to next_process variable.
6. Here I have made a vector execQueue that stores all the process IDs in the execution order.
7. Now push_back the next_process into execQueue and decrease the reaminingExecTime of this process by one. (remainingExecTime is initialized with execTime in the starting)
8. And now we increase waiting time of all the other processes by one which aren't executing, has k > 0, and remainingExecTime not equal to zero.
9. Now if the remainingExecTime of next_process is equal to zero, then decrease its k by one unit.
10. Then check if there is a miss or not.
11. Now this loop iterates for total time of execution and in each loop a process ID is pushed to execQueue vector. If the CPU is idle, then -1 is pushed to the vector.
12. Now we iterate over the execQueue array and based on the ID values (start, end, pre-empted), write into log file.
13. Then we write the required information of the processes into stats file.
14. Close all the files and we are done with EDF scheduling.

I faced difficulty in writing log files. I was able to generate the execution order of the processes but then counting the processes, matching with its execution time, checking if it has finished its execution or got pre-empted became a bit difficult.

Graph1: Deadline missed vs Number of processes:

X-axis is the number of processes

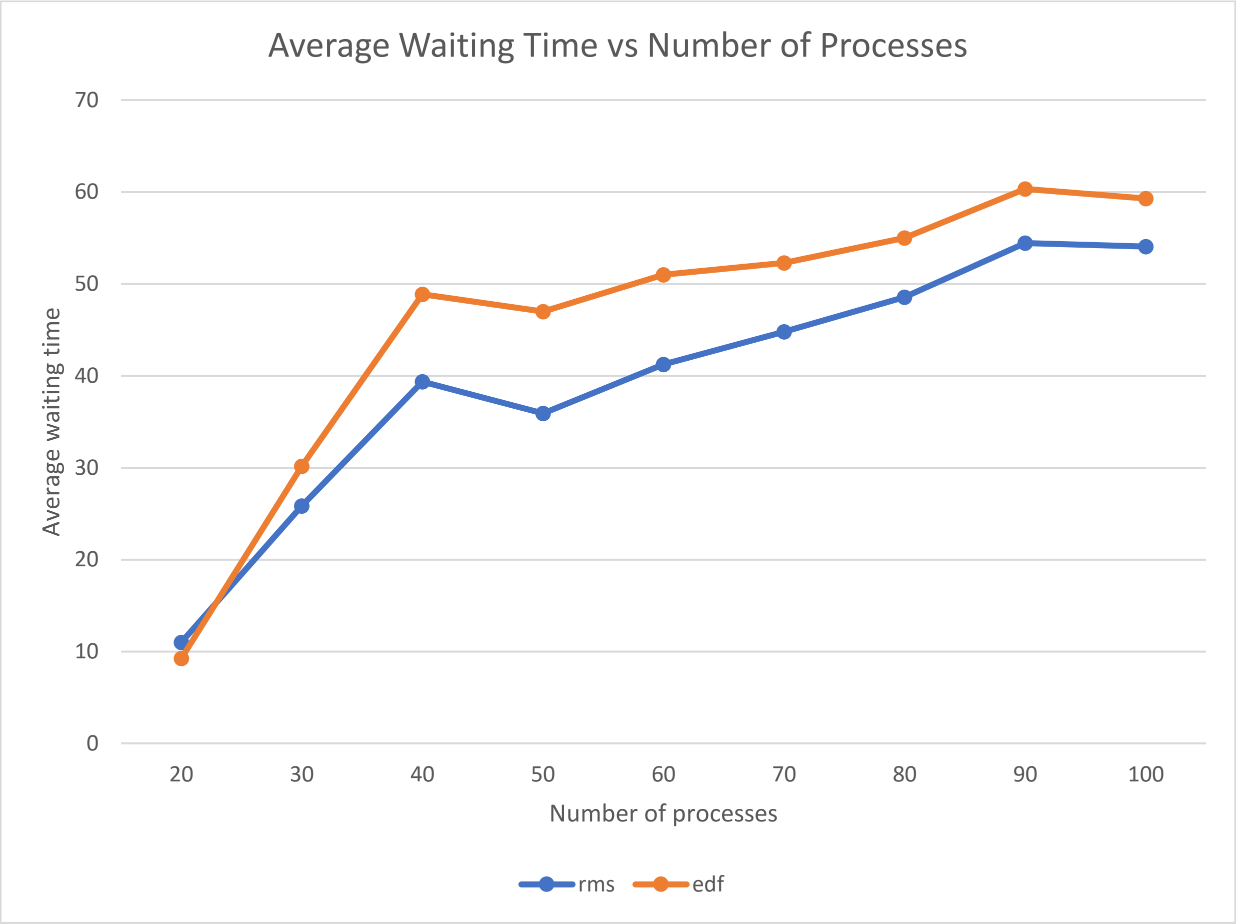
Y-axis is the number of processes that miss the deadline



Graph2: Average Waiting time vs Number of processes:

X-axis is the number of processes

Y-axis is the average waiting time


Average Waiting Time vs Number of Processes
Average waiting time
Number of processes
rms
edf