REPORT
ROLL NUMBER: CS20BTECH11028

The program first takes the input and then initiates the processes. Each process stores information regarding entrytime, leaving time,number of iterations, period,deadline, ispreempted or not, current iteration etc, in a class “process”.

Rate Montonic Scheduling design:

Process with least period is of high priority and vice-versa in Rate Monotonic Scheduling.

The program runs a while loop which breaks after all the processes are processed. We store the processes entered in a vector , say “rmqueue”, and we always take the highest priority element i.e., process with least period.

First, we start a process of highest priority and run it, while running, at every millisecond we check if any process is incoming and if it has missed its deadline. If a process ‘A’ missed its deadline, then ‘A’ is terminated and resetted (reset its remaining time, waiting time etc). As ‘A’ missed its deadline it hadn’t removed from the rmqueue before, so just ‘A’ will be left in the rmqueue after reset, we need not again add it as it was not removed before. If a new process which has already completed its previous iteration is incoming, we reset and add it to the rmqueue as it was previously removed after it’s completion.

Then, if the process currently running is same as the highest priority process, after the new processes were added, then the process is continued else it is pre-empted and the process with high priority is started for execution.

If the process completes, then we remove it from the rmqueue and then start the higher priority process and repeat as above.

If no process is left in the rmqueue, then the CPU remains idle until any other process comes into the rmqueue. If a process has entered, then again it repeats as above, If all the iterations of all processes are completed, then the loop exits.

Earliest Deadline First Scheduling design:

Process with earliest deadline is of high priority and vice-versa in Earliest Deadline First Scheduling.

The design is almost same as that of Rate Monotonic Scheduling design, except that the priority is different.

We store all the processes in a vector, say “edfqueue”, and we always take the process with nearest deadline and start running.

While running we check at every millisecond, if any process is coming, and if any of the already entered processes missed their deadline. If any of the processes missed their deadline, we then reset the process, meaning we calculate its waitting time, reset its remaining time,ispreempted etc, to be ready for next iteration. If any new process is incoming, then we add it to the edfqueue. If the newly added processes pre-empts the currently running process, (we check whtther seeing the id of the current process is same as high priority process in the edfqueue), then we preempt and store the remaining time of the process which is pre-empted. We start the next process now. Else if no pre-emption occurs we run until the current process finishes.

If there is no process entered into the edfqueue, then we wait until any new process is added, CPU remains idle. If all the iterations of all processses are completed then we break out of the while loop.

Note: If the processes are about to cross the deadline, then processes are runned till their deadline and terminated.

Complications:

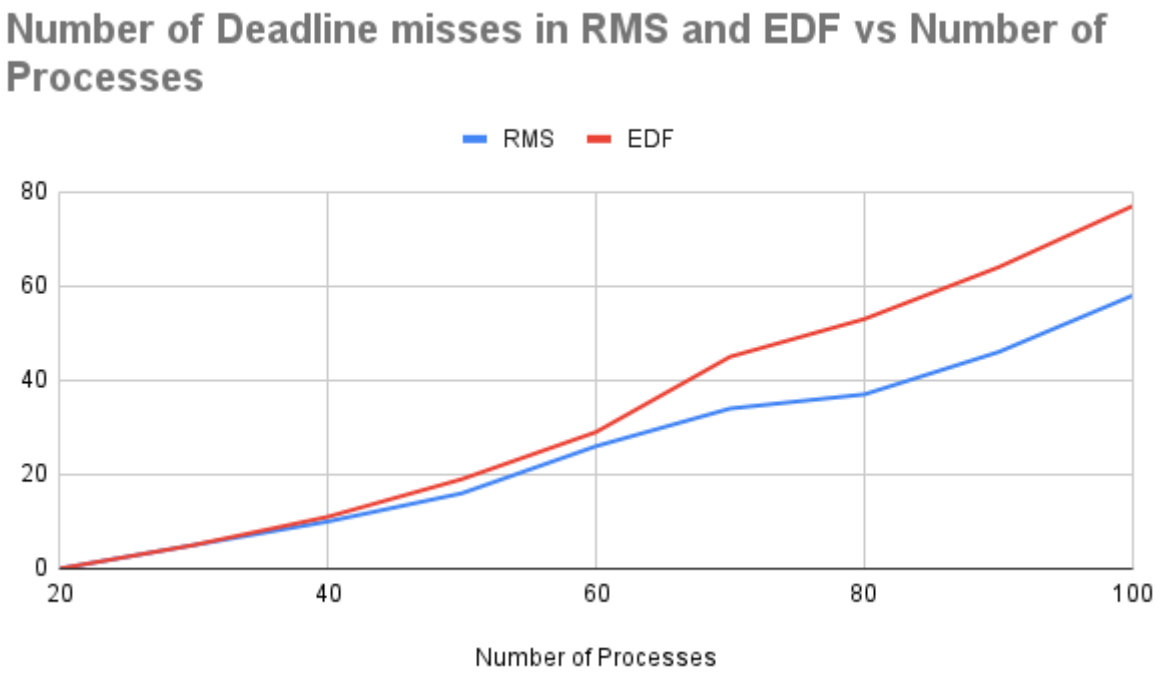
- Not considering the case when the process is about to finishes ‘at’ its deadline. Initially for the code i have written, it would show that the process missed its deadline for this case. Finding where the problem was a bit tough.

- When two or more processes have same priority, it implied that my code was wrong when i checked on paper, but finding that, it was due to some process running among the equal prior processes, became difficult, especially in case of EDF, where there is dynamic priority.

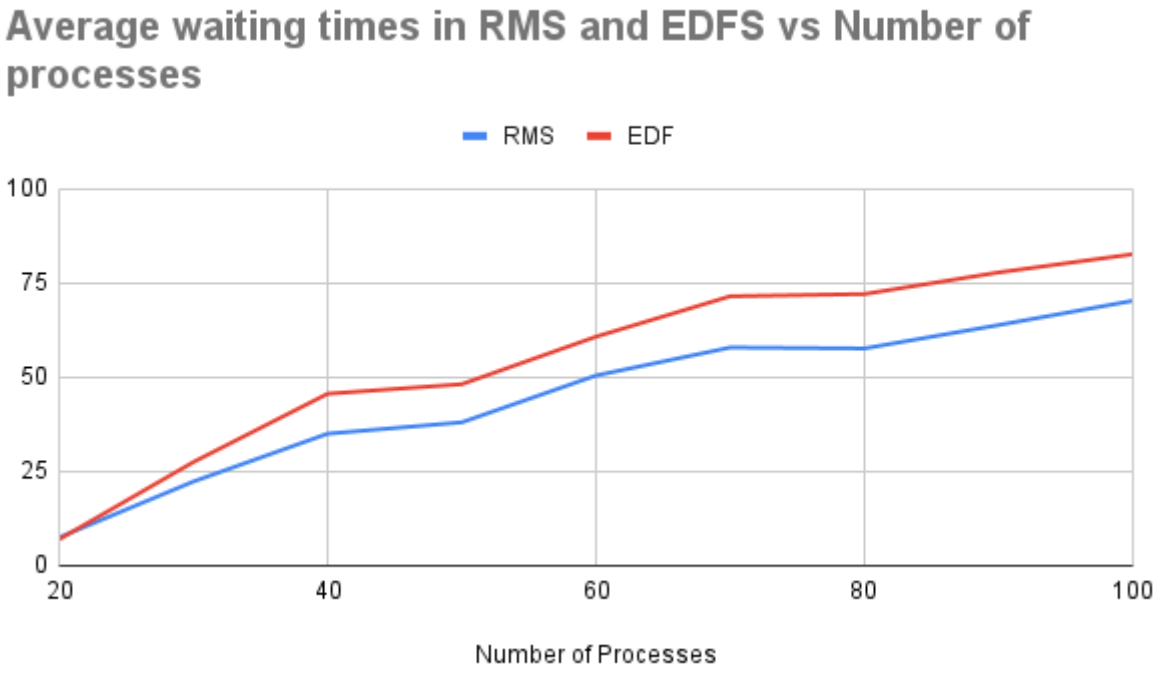
COMPARISON USING GRAPHS:
(Here CPU Utilization >100%)

(a) Deadline Misses vs Number of processes

X- axis: Number of processes
Y-axis: Number of Deadline misses



(b) Average waiting time vs Number of processes
X-axis: Number of processes
Y-axis: Average waiting times in ms



Analysis:

From the graphs we can see that,

- Number of Deadline misses are more in case of EDFS than RMS.
- Average waiting time is more in case of EDFS than RMS
- So when CPU utiliation is greater than 100%, EDF may not be that efficient when compared with RMS as both deadline misses and waiting time is high for EDF. (Though it might differ based on input data set)
- But when CPU utiliztion is <100%, Number of deadline misses would be more for RMS than EDF. In that cases EDF might be better choice, (but in general, there would be a tradeoff between number of context switches and deadline misses.)

Data set used for comparison in the graphs

10
1 20 50 10
2 40 90 10
3 30 100 10
4 25 120 10
5 20 60 10
6 45 120 10
7 35 110 10
8 15 75 10
9 55 125 10
10 60 120 10

Extra Credit:(Idea only, didn't implement)
Context switch time
Every time we pre-empt a process or if a process finishes or if a process missed it's deadline i.e., if another process is starting we do a context switch i.e., we add the context switch time for the time counter.

Selection time
The time taken for selecting the next process can be taken can be calculating by using the gethighprior function where we can check the number of comparisons it had done to get the process with high priority.