OPERATING SYSTEMS 2

SAI HARSHA KOTTAPALLI
CS17BTECH11036

Design and Working

We simulate the Rate- Monotonic & Earliest Deadline First (EDF) scheduling algorithms in C++ using discrete event simulation.

Rate Monotonic Scheduling

It is a priority assignment algorithm used in real-time operating system with a static-priority scheduling class. The static priorities are assigned according to the cycle duration of the job, so a shorter cycle duration results in a higher job priority. We implement a pre-emptive version of this algo.

Earliest Monotonic Scheduling

Earliest deadline first is a dynamic priority scheduling algorithm used in real-time operating systems to place processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. We implement a pre-emptive version of this algo.

We first ask user whether context switch time is to be included or not in the start, based on which corresponding stats are calculated.
So, whenever context switch occurs, we add a unit of time as its expense.
As we use discrete event simulation we define a unit time as 10 micro-seconds.

Class Process

The class "process" in our program has various attributes such as -
process ID, processing time, period, deadline. It has other properties which would help us in later part of this program, namely –
1. Time remaining for completion of process(so that we can resume a process which was pre-empted previously and for calculation of stats)

2. Start time.(for checking arrival of a new process and for calculation of stats)

3. Repeats(number of times a process is going to repeat)

4. in_ready(whether the process currently(when in execution of algo) is in the ready queue)

We use function – get_input, to process input file and with the help of vector, get process list from the file.

We use vector with process class to implement ready queue, where it is sorted(using stl and "compare" function), whenever a new process is added to its list such that:

1. RMS Algo
    ascending order w.r.t to their period.
2. EDF Algo
    ascending order w.r.t to their deadline.

Every unit time, we first check if there are no processes left to execute.
Last_process var, helps us check if the same process is currently being executed.
Then we check in the ready queue if any process has met its deadline. If it so , then it has met the deadline before its completion(when deadline and its time of completion are equal, then we declare that process has finished executing) and then we correspondingly change the required variables(like in_ready , etc).
We update start time and deadline by its period so that on its arrival these stats are stored in the ready queue.
We delete the processes list item whenever no more repetitions are possible(as defined in input file).
Now, we check if any new process can be added to the read queue.

Now, if current process time of execution is same as its time_left, then it has not yet started executing. That implies, this process is going to start.

If not, then based on last_process, we deduce if process is resuming or if its the same process which is currently running.

Next, we check if time_rem is 0, that if program has finished executing.
We decrease the number of repetitions left in the process list and check if this process again occurs at this time(on which we add it to the queue).
We update the start and deadline of next process (if it occurs again) accordingly.

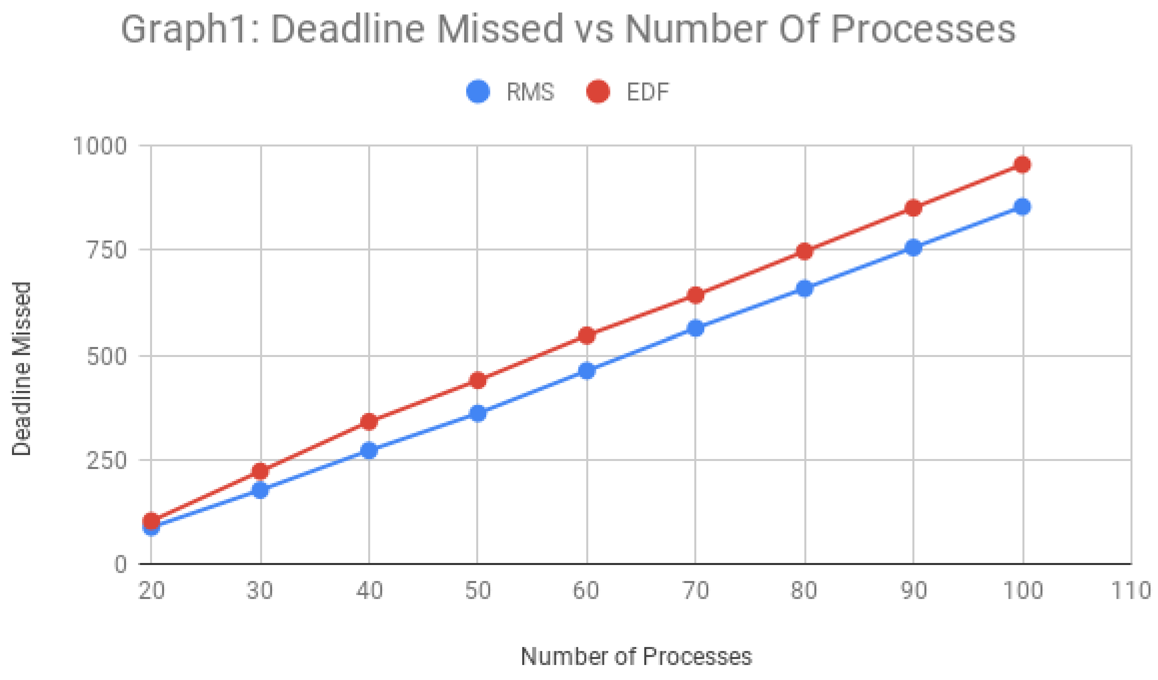
Also, if ready queue was empty then cpu must be idle.
We then on every unit time, check if any process can be added.

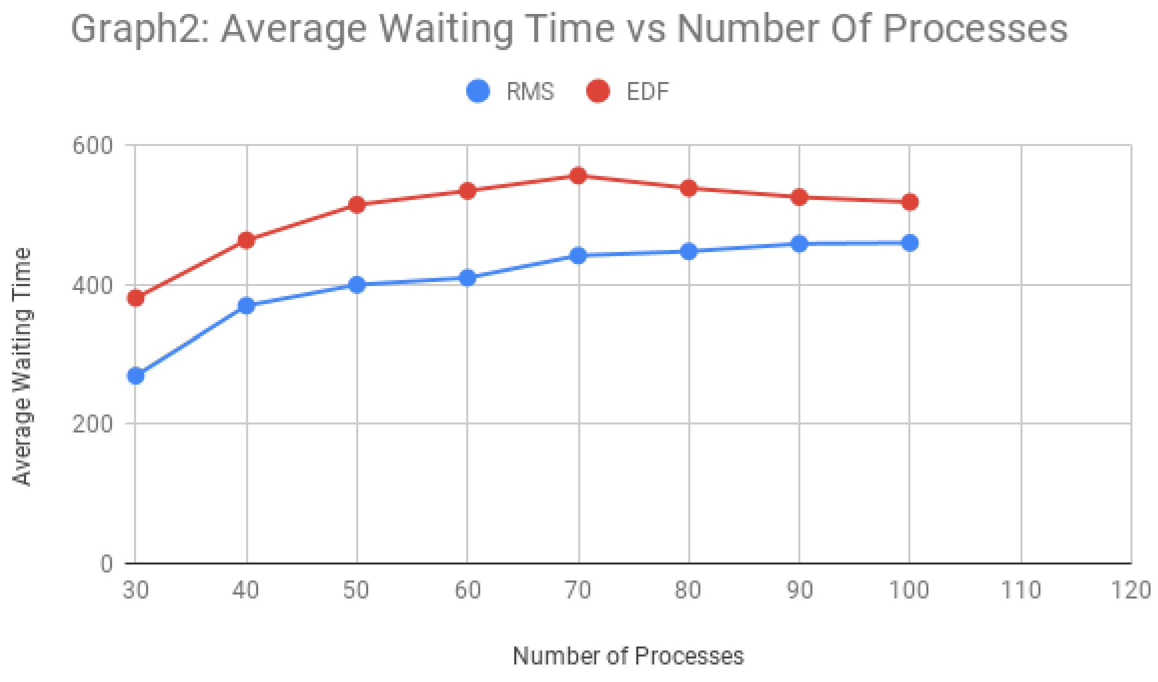
Finally, all the processes will be iterated and that stats calculated will be processed into certain files(defined in ReadMe).

Comparision

We use following parameters:
1. Waiting time:
    It is defined as the total time process spends in ready queue.
2. Deadline misses:
    total number of processes whose deadline was met before completion.

Graph


Graph1: Deadline Missed vs Number Of Processes
RMS
EDF
Deadline Missed
0
250
500
750
1000
20
30
40
50
60
70
80
90
100
110
Number of Processes


Graph2: Average Waiting Time vs Number Of Processes
RMS
EDF
Average Waiting Time
0
200
400
600
30
40
50
60
70
80
90
100
110
120
Number of Processes