Operating Systems–II: CS3523

Programming Assignment 1

Implementing Rate-Monotonic Scheduling & Earliest Deadline First Scheduling through Discrete Event Simulation

Design of the Program :

In this assignment, we use Discrete Event Simulation to implement the two scheduling schemes.

(a) Rate Monotonic Scheduling:

In this program, we have declared a class "process" which contains the variables storing the process id, period, processing time, time remaining before completion, start time of process and arrival time of process each time it is put into ready queue. "current_time" is clock which starts at t=0 and increase with each event.

We have first sorted the arrival of all the instances of each process based on their arrival time and if their arrival time is same then one with smaller period (i.e. higher priority) is placed before. We have created n queues with priority based on the period of process i.e. Q[1] will contain all the process with process id "P1". We have created a "priority" vector in which process are kept in sorted order based on their period , i.e. priority[1] will contain the process with shortest period.

First we have inserted first instances of all processes at time t=0 in their respective ready queue. We keep a pointer to the process which has not been put in queue yet. Now we remove the highest priority process from the queue (one with smallest period) and starts its execution and check all the processes which can come before the completion of this process using the pointer. If any process has higher priority than the running process then the running process will be preempted at the arrival time of this process and put in the ready queue.

Also at the time of removal of process from the queue and while running the process, we need to check if the current process has missed the deadline by comparing the "current_time" and process deadline. We have used the fact that if another instance of a process comes while one instance is in ready queue then the instance in queue will have missed its deadline as period is same as deadline.

Whenever a process is inserted in ready queue then its "arrive_time" is set to "current_time" and then pushed it to queue. Whenever a process is popped from the queue then difference of "current_time" and "arrive_time" will give us the waiting time. Whenever the process starts executing, we check if its remaining time is same as processing time or not. If it is same, then we print "process starts ..", else we print "process resumes .." .

(b)Earliest Deadline First Scheduling:

In this scheduling program, we have the same class as before except on variable "no" which gives the process number like 1 for "P1" which is used in comparator function of set. We have implemented the ready queue using C++ STL, set in which elements are sorted on the basis of their deadlines but if their deadline is same then we check for "no" of each process and process with smaller "no" value is placed at top of queue.

We compute the deadline of each instance of a process by multiplying the instance number with process period. (e.g. 2nd instance of process A with 50 ms period has deadline 100ms) Rest part is similar to the Rate Monotonic Scheduling program. The process with earlier deadline is given higher priority so in place of comparing the periods we now compare the deadlines of two instances. Similarly , for finding missed process we check at the time of removing a process from queue as well as when a new instance of that process comes then also that process misses its deadline. All the waiting time is added which is obtained at time of removing an element from ready queue and then it is divided by total instances of every process to get average waiting time.

Graph:

We have created two graphs using the sample input file provided by taking n=2,3,..,10 :

(i) Graph of "Number of Missed Process" v/s "Total number of process"

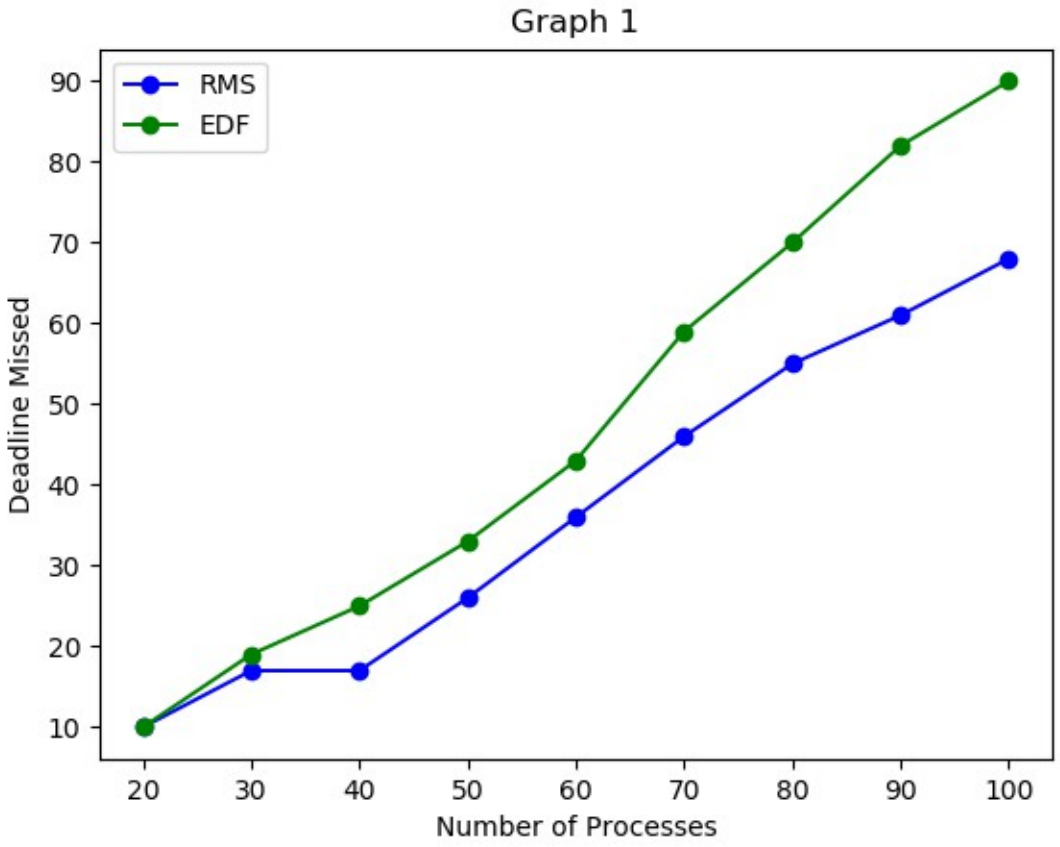
In this graph, for n=2, the number of processes which missed their deadline is same (10) but for n>2 the number of missed processes is greater in the case of EDF scheduling as compared to Rate Monotonic Scheduling.

(ii) Graph of "Average Waiting Time" v/s "Total number of process"

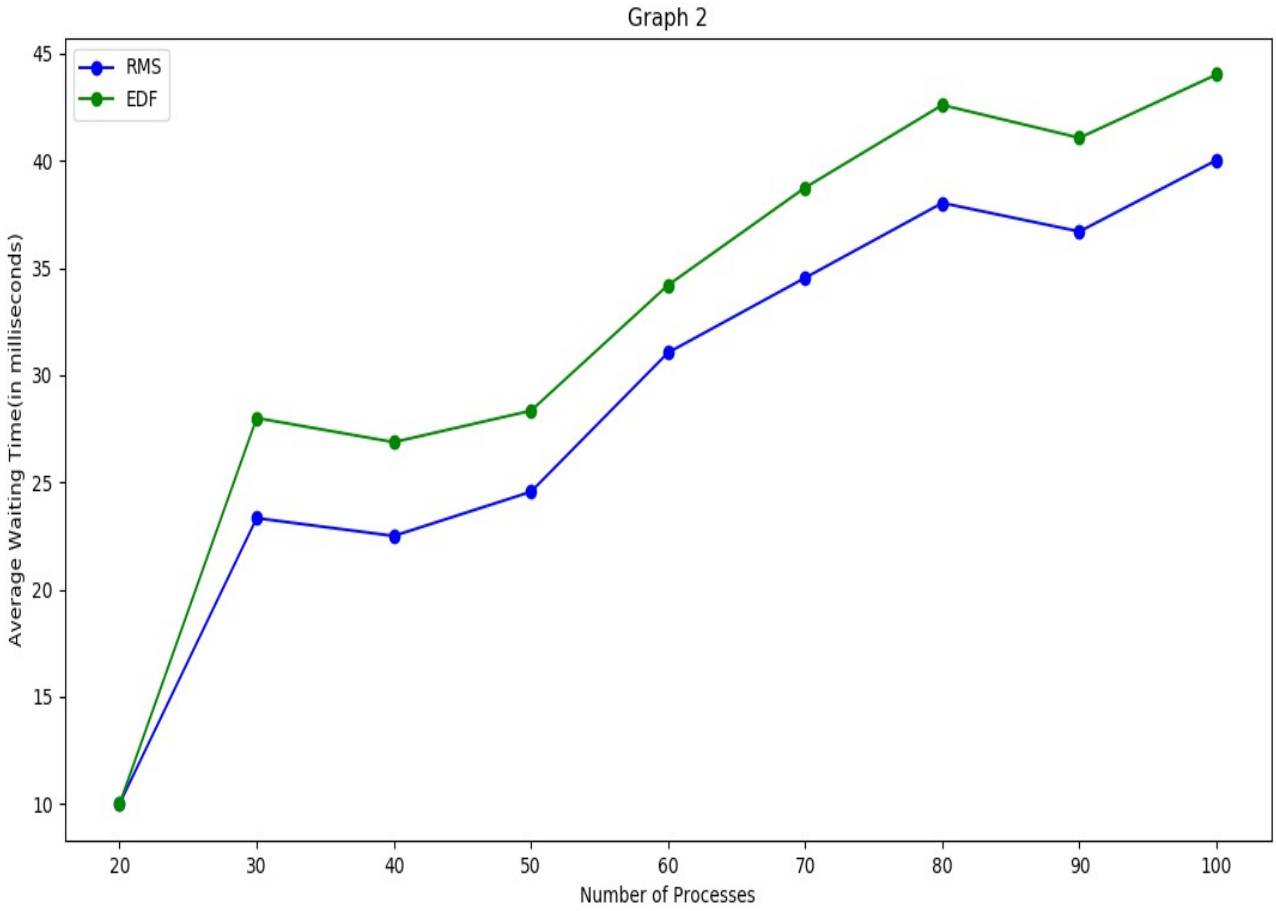
In this graph, for n=2, the average waiting time of all the processes for Rate Monotonic Scheduling(10 ms) is same as that for EDF scheduling(10ms).

For n>2 the average waiting time of all the processes in Rate Monotonic Scheduling is smaller than that in EDF scheduling for given n.

Hence, Rate Monotonic Scheduling shows better performance than EDF scheduling both in terms of average waiting time as well as fewer number of processes missing their deadline.



Graph of "Deadline Missed" v/s "Number of processes"



Graph of "Average Waiting time in milliseconds" v/s "Number of processes"