Operating Systems–2
Report - Assignment 1

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

Name: Naitik Malav Roll No. CS19BTECH11026

Rate Monotonic Scheduling & Earliest Deadline First Scheduling

1. I have defined a class *ProcessInfo* which has parameters – *id, k, kctr, period, execTime, deadline*.

   i) id -> process id
   ii) k -> number of times process is going to execute
   iii) kctr -> same as k
   iv) period -> period of process
   v) execTime -> execution time or processing time
   vi) *deadline -> next deadline of a process*

```
class ProcessInfo{
    public:
        int pid, k, kctr;       //process id, number of time process gonna execute
        ui period, execTime, deadline;  //period, execution time, next deadline of a process
        double waitingTime = 0;     //total waiting time of a process
        double startTime = 0;   //time when it's execution started

        bool Flag=false;
        bool operator() (PAIR, PAIR);   //comparison

        ProcessInfo(){}
        ~ProcessInfo(){}

        void getInput(int, int, ui, ui);    //function for input
};
```

Description of *getInput function:* It initializes all the parameters of the class which we have scanned from *inp-params.txt.*

2. I have implemented the algorithm using two priority queues which will help us in our discrete event simulation. The two priority queues are –

a) activeQueue - a queue where the Processes are waiting to get executed on the CPU.
b) eventQueue – it has the information about the process that will enter activeQueue in future.

3. activeQueue contains a pair of integer and unsigned int in which -

a) PAIR.first represents the process id,
b) PAIR.second represents the priority of the process.

4. eventQueue also contains a pair of integer and unsigned int in which –

a)PAIR.first represents the process id, and
b) PAIR.second represents the arrival time of process in activeQueue.

5. I have run a while loop till both queues become empty, i.e. until all the processes are over.

6. Inside this while there are few if-else conditions for a process i.e. for –

a) execution Time calculation, like arrival and finishing of process, or you can say CPU burst calculation
b)Preemption, if a process gets pre-empted by another then printing and related stuff.
c)If any process miss it's deadline then printing time and related stuff.

7. All of the above time calculations are saved/printed into RMS-Log.txt with the help of log pointer.

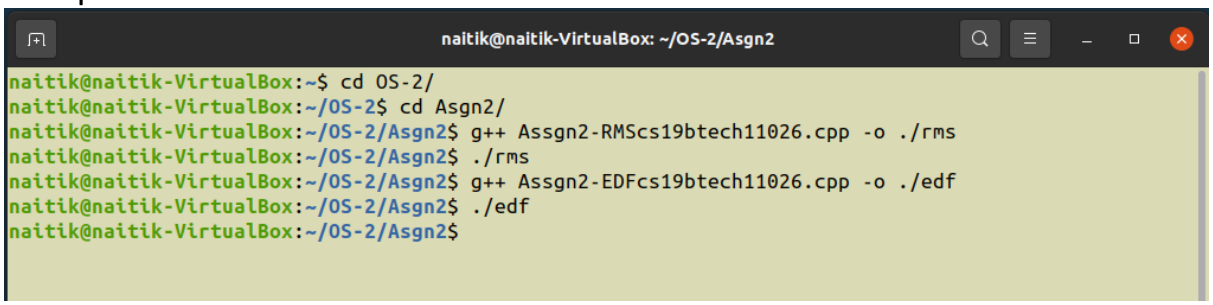
8. *TotalWaitingTime* stores the total of average waiting time of each process.

9. According to the question –

- Average Waiting time of a process = (Sum of waiting time of all the rounds) / (the number of times process repeats)
- Average Waiting time of all the process = (Sum of Average Waiting time of all the Processes)/(Total number of Processes)

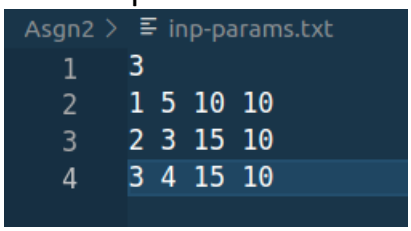
So I have printed average waiting time of all process in the file RMS-Stats.txt with the help of stats pointer. It is basically equally to TotalWaitingTime divided by n.

Compilation Screenshot:



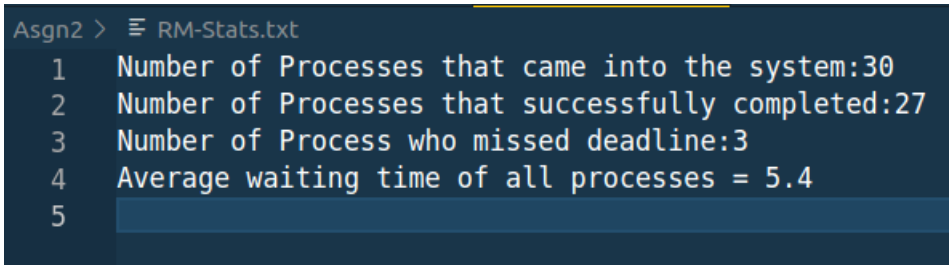
Example:

- Input1 -



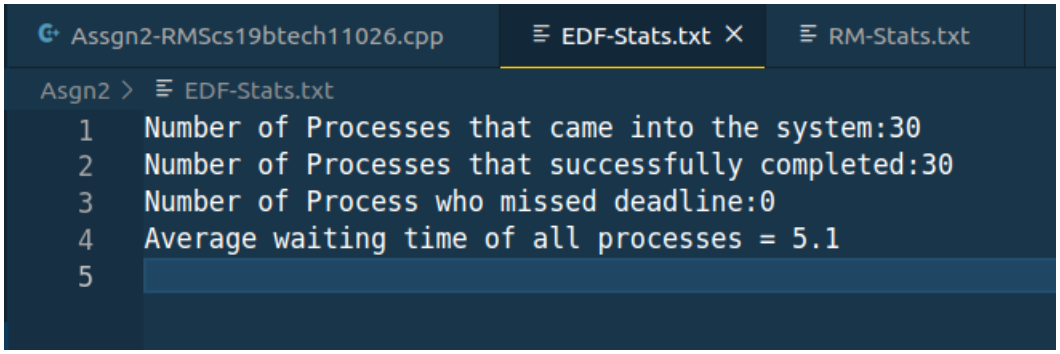
Output1-

RMS-Stats screenshot



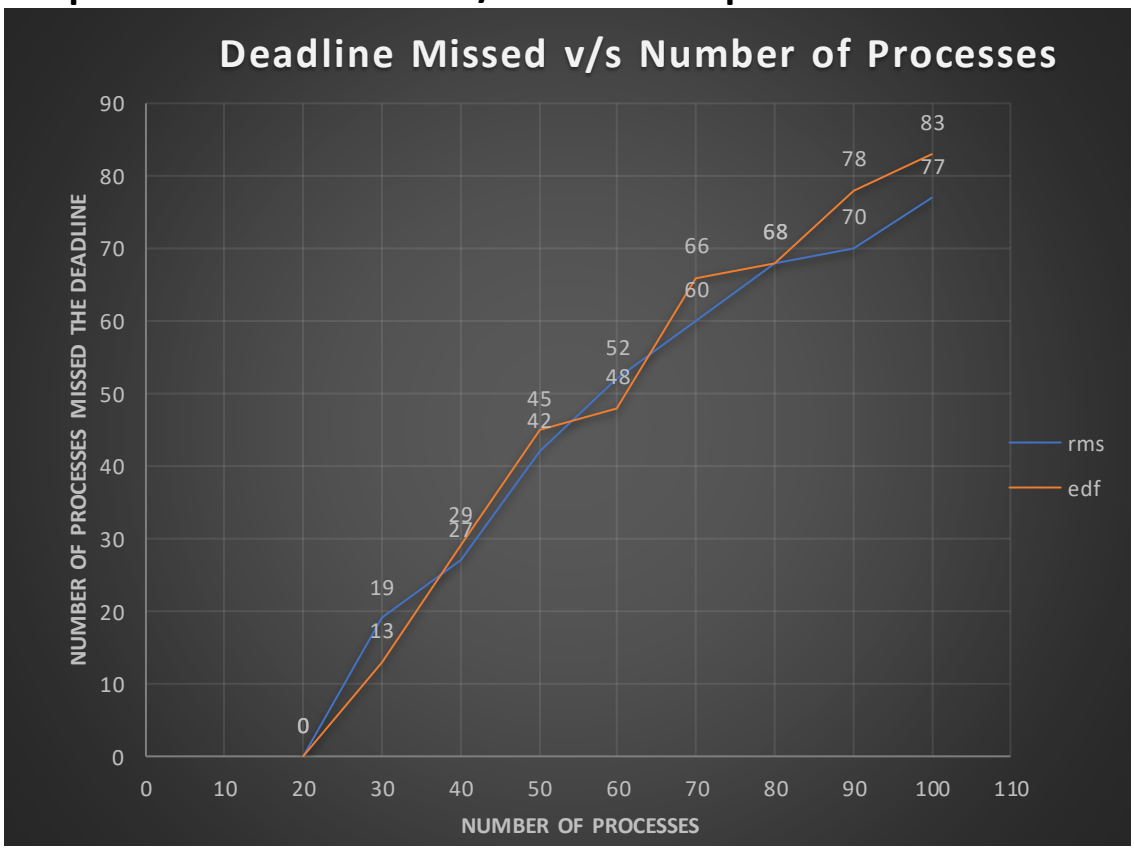
For RMS-Log please refer RMS-Log.txt

EDF-Stats screenshot



For EDF-Log please refer EDF-Log.txt

Graph1: Deadline missed v/s Number of processes



Graph2: Average Waiting Time v/s Number of processe