Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Assignment-2

Vinta Reethu - ES18BTECH11028

March 2, 2021

Question

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

Design of program

- Before running the scheduling algorithms we should store them in a convenient manner, as we keep looking at these parameters during the scheduling. Here I used struct to store the parameters like Process ID, Burst time, Period, K etc

```
typedef struct process  // Struct that as the values related to a process
{
    int ProcessID;      // Store process ID
    int Bursttime;      // Burst time
    int Period;         // Period of process
    int K;              // Number of instances of process
    int K_Backup;       // Backup of number of instances of process
    int NextDeadline;   // Next dead line
    int RemainingBurstTime; // Remaining burst time
    double StartTime;   // Start time
    double WaitingTime; // Overall waiting time
    bool prempted;      // Boolean variable to keep track of premption
}process;
```

- We declare some global variables like n defining number of processes that enter the system, current_time which helps to keep track of the time, missed_deadlines to keep the track of processes that missed deadline.

```
int n;                  // Number of processes
int current_time=0;     // Clock variable
int ProcessID;          // Varibale to store process ID
int missed_deadlines=0; // Variable to count the number of missing deadlines
```

Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

- To implement scheduling we use two queues - **event queue**, **ready queue**. Event queue contains the processes that will enter into ready queue next. Ready queue contains those processes that can be scheduled immediately.

```
// Event queue : It contains the list of processes that will enter the ready queue
// Ready queue : It contains the list of processes that can be scheduled
priority_queue<pair<int,int>,vector<pair<int,int>>,Compare>event_queue,ready_queue;
priority_queue<pair<int,int>,vector<pair<int,int>>,Compare>ready_queue;
```

- Both the priority queues contain pair of integers.
  - **Event queue** : pair.first - Process ID, pair.second - The time when it can enter ready queue
  - **Ready queue** : pair.first - Process ID, pair.second - Period of the process
- We sort the queue using custom **compare** operator. We give priority to the process with lower period in case of RMS and earliest deadline in case of EDF.

```
struct Compare   // Custom operator that does ordering in priority queue
{
    bool operator() (pair<int,int>a, pair<int,int>b)
    {
        if(a.second == b.second) return a.first > b.first;
        return a.second > b.second;
    }
};
```

Main algorithm

- We keep performing scheduling until both the queues are empty, which happens when all the processes are completed executing.
- Initially we add one instance of all the processes to the event queue with pair.second = 0.
- If a process is not in ready queue, then Remaining Burst Time will always be 0.
- If the current_time is equal to that of time at which the top() element in event queue can enter the ready queue, we do the following check :
  - If the Remaining Burst Time of the process is zero, we add this process into the ready queue. Because this process still didn't enter the CPU.
  - If there is a process say P1 that has been preempted by a higher priority process say P2 and later at some other instance if some higher priority process say P3 is currently executing, during this time if the if P2 misses it's deadline, we report that sort of deadlines here.

- If the ready queue is not empty then we can do the following
  - **Start a process :** We start a process only if we know that it's remaining burst time plus the current time is less than it's deadline, otherwise we report it as missing deadline process.
  - **Resume a process :** If there is a process which was earlier preempted by some other process, and now it got chance to acquire CPU again we schedule it. We only do it if we know that it's remaining burst time plus the current time is less than it's deadline, otherwise we report it as missing deadline process.
  - **Finish a process :** This can happen in two ways
    - If event queue is empty and we know that it can completed before the deadline then we finish executing the process.
    - If event queue is not empty then we have to check if the time by which this process completes executing is less than it's deadline and also less than time by which the top() element in event queue will enter ready queue. The process will finish executing.
  - **Preemption :** If the time at which the top() element in event queue will enter ready queue is less than the deadline of this process and it's period is less than the period of current process, we resume the current process.
- If ready queue is empty and event queue is not empty, then CPU is idle till the top() element in event queue will enter ready queue.
- After a process finishes/misses the deadline we set it's remaining burst time to 0.
- For any process we calculate waiting time in the following way
  - If the process completes it's deadline then we use this formula.

```
P[ProcessID-1].WaitingTime+=(current_time-P[ProcessID-1].StartTime
                             +P[ProcessID-1].RemainingBurstTime-P[ProcessID-1].Bursttime);
```

  - If process misses the deadline, waiting time will be equal to it's period for that instance.

```
P[ProcessID-1].WaitingTime+=P[ProcessID-1].Period;
```

Complications

- Adding the elements into the queue at correct time other wise it will lead to inconsistency.
- Proper updating of waiting time, remaining burst time. If these are not updates properly it will lead to complications.
- Dealing with floating point comparisons in case of context switch.

Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

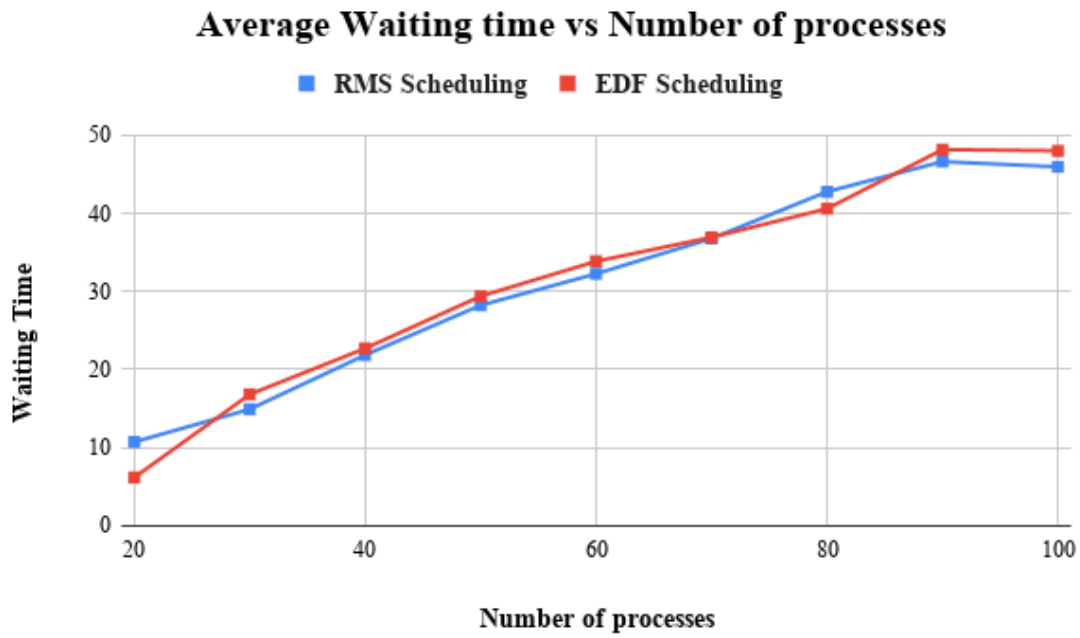
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Comparison of the performance :

Test case - CPU Utilisation >1

```
10
1 27 50 10
2 15 35 10
3 13 40 10
4 20 70 10
5 22 90 10
6 11 78 10
7 11 78 10
8 31 63 10
9 41 85 10
10 12 35 10
```

Graph 1 :

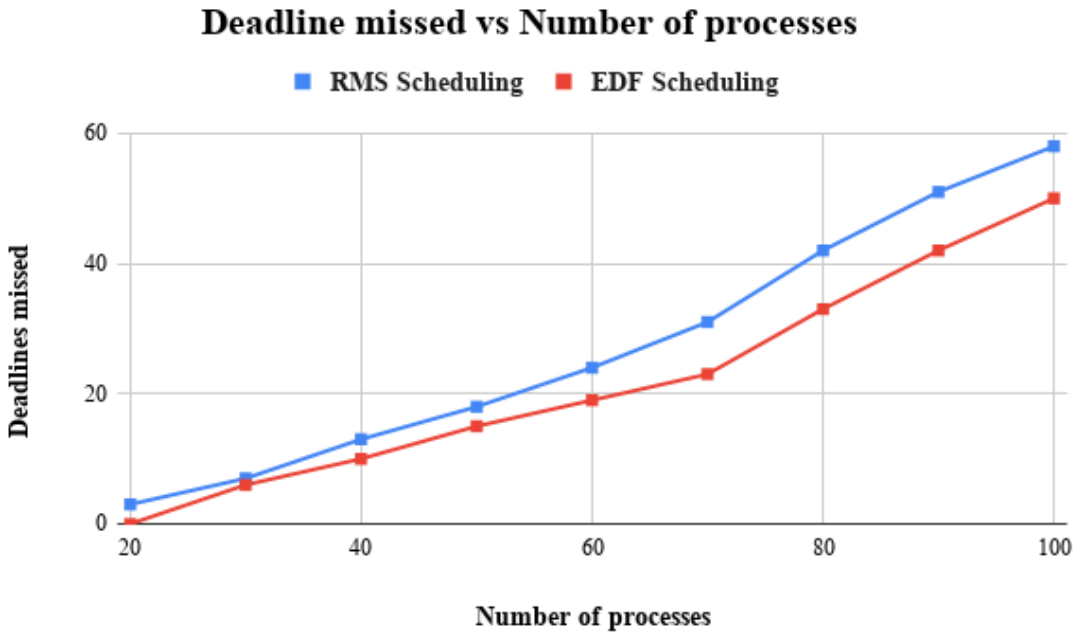


We can see from the above graph that as number of processes increases the average waiting time increases. **RMS** performs better in terms of average waiting time.

Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Graph 2 :



We can see from the above graph that as number of processes increases number of deadlines missed increases. **EDF** performs better in terms of missed deadlines.

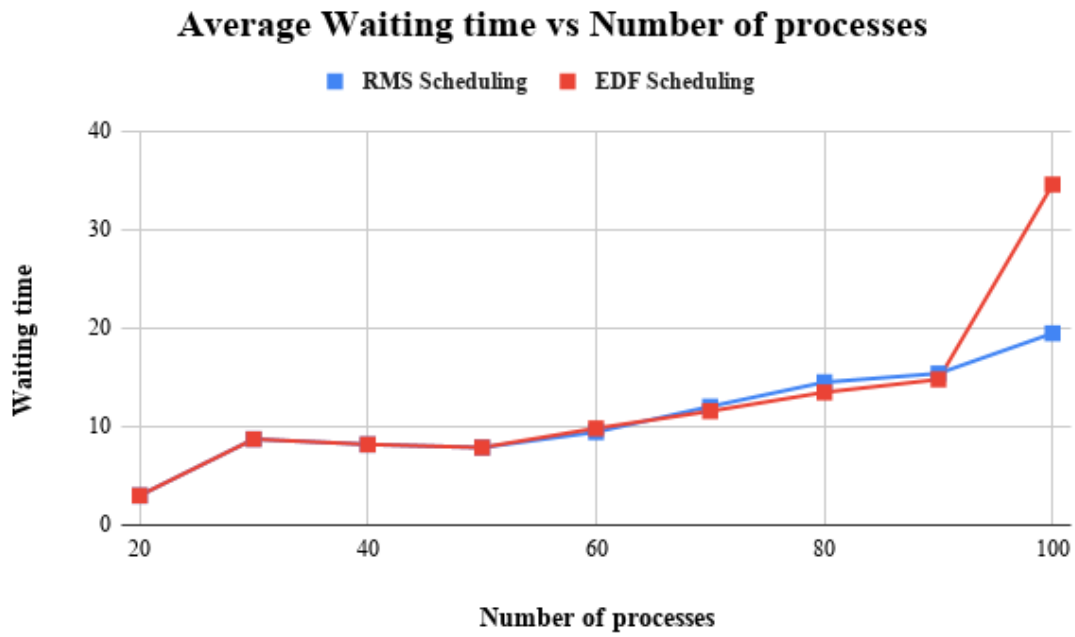
Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Test case - CPU Utilisation <1

```
10
1 19 138 10
2 15 94 10
3 13 94 10
4 10 91 10
5 6 90 10
6 11 101 10
7 11 111 10
8 9 128  10
9 2 109 10
10 8 105 10
```

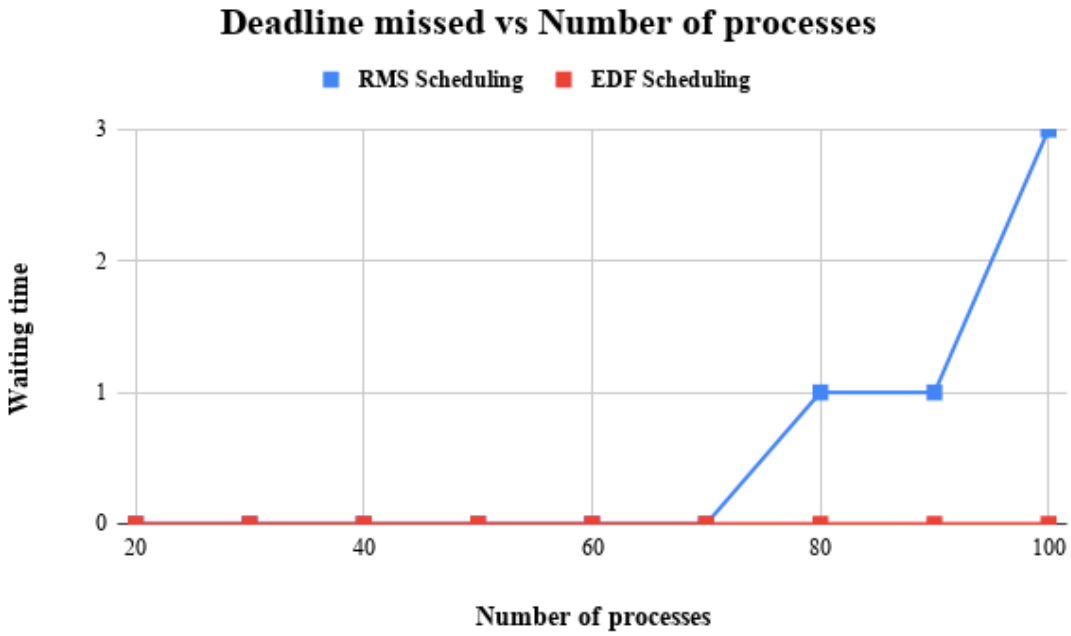
Graph 1 :



From the above graph we can see that, initially both the scheduling algorithms are having same average waiting time because the CPU Utilisation being less than 1. As the number of processes increased we can see a slight increase in waiting time of EDF than RMS. Under this setting **RMS** performs better wrt average waiting time.

Vinta Reethu
Operating systems - II
Indian Institute of Technology Hyderabad
Assignment-2

Graph 2 :



From the above graph we can see that as till 70 process there are 0 deadlines in both scheduling algorithm as the CPU Utilisation is less than 1. Towards the end we can see that deadlines missed by RMS is greater than EDF. RMS starts missing it's deadlines as the CPU Utilisation exceeds the upper bound $N(2^N - 1)$. Under this setting **EDF** performs better wrt number of deadlines missed.

Extra credit

- Switching of CPU to another process occurs when
  - Process completes it's CPU time.
  - Process is preempted by another process.
  - Process missed it's deadline.
- To add the context switching we should just perform some minor tweaks.
- I added a global variable named context_switch. Whenever a process starts/resumes executing we add context switch to the current time. This takes care of adding context switch for above mentioned 3 tasks.
- Here the context switch we take is in double. So all comparisons operations are replaced accordingly.
- The codes for context switch are the files named **Assgn2-RMS-CS-ES18BTECH11028.cpp** and **Assgn2-EDF-CS-ES18BTECH11028.cpp**