Operating Systems-2
CS 3510 Spring 2019
Programming Assignment 1:

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

INSTRUCTOR: DR. SATHYA PERI

Report By:
Vijay Tadikamalla
CS17BTECH11040

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Task

The goal of this assignment is to implement a program to simulate the Rate-Monotonic & Earliest Deadline First (EDF) scheduling algorithms. Then compare the average waiting time and deadlines missed of both algorithms. Implement both the algos in C++ using discrete event simulation.

Algorithm

Rate Monotonic Scheduling

Rate Monotonic Scheduling is characterised by giving priority to processes which have the least period. This is static-priority scheduling as period of a process is constant throughout execution.

Earliest Deadline First (EDF):

Earliest Deadline First Scheduling is characterised by giving priority to processes which have the earliest deadline. This is a dynamic-priority scheduling as the deadline of a processes is increased by its period after every repetition.

Approach and Implementation

1. To achieve our above mentioned goal we define a class PCB which contains some standard parameters of Process Control Block like process id , period , CPU burst time , deadline , number of repetitions.

```
class PCB{
public:
      PCB(){}
      ~PCB(){}
      int id;
      lli time;
      lli period;
      lli next_deadline;
      int k;
      bool flag=false;
      bool operator() (Pair a, Pair b){
        return a.second > b.second;
    }
};
```

2. We will implement the algorithms using two priority queues which will help us in our discrete event simulation.
3. Events to be considered are when a process:
    a. starts its execution.
    b. finishes its execution.
    c. is preempted by another process.
    d. resumes its execution.
4. But the algorithm which has been implemented has additional features like deadline miss. This feature will help us to implement even more powerful and real schedulers where time taken by context-switch can also be considered.
5. The first priority queue is used as a ready-queue(a queue where the PCBs are waiting to get executed on the CPU) and the second priority queue(next-process-queue) is used to know which PCB will enter the ready queue in the near future.

```
priority_queue<Pair,vector<Pair>,PCB> ready_queue;
priority_queue<Pair,vector<Pair>,PCB> next_process_queue;
```

6. Each priority_queue contains a pair of Integers.
    a. In ready queue:
        i. Pair.first represents the process id
        ii. Pair.second represents the priority of the process.
    b. In next_process_queue:
        i. Pair.first represents the process id
        ii. Pair.second represents arrival time of the process in the ready queue.
7. We will run a while loop till all the processes are over, that is till both the priority queues become empty
8. At any given time t, if a process enters the ready_queue it will
    a. Complete its CPU BURST
    b. Get preempted by another process
    c. Miss its deadline
9. The above three conditions are mutually exclusive and exhaustive. So they can be implemented easily.
10. Implementation of the above three conditions very slightly with change in algorithm.

Complications

1. We can't decide the behaviour of priority queue when two processes of same priority are compared.
2. So this sometimes leads to complications when a particular behaviour is dezired.

3. Floating point operations get involved if we try to include very small time (time required for Context Switch). So unpredictable behaviour may be observed.

Design Decision

1. Priority queue reduces the time complexity of the algorithm to a greater extent.
    a. Choosing Highest priority element - O(1)
    b. Inserting a new element in queue -O(log(n))
    c. Next process which will enter ready queue - O(log(n))
    d. Helps in Discrete Event simulation
2. Data type for priority queue - Using a pair of integers. This helps us by providing a direct mapping to the elements in the priority queue.
3. Logging extra information- All logs related to missing of deadlines are logged.
4. Using double data type for preventing overflow.

Output analysis

Input 1

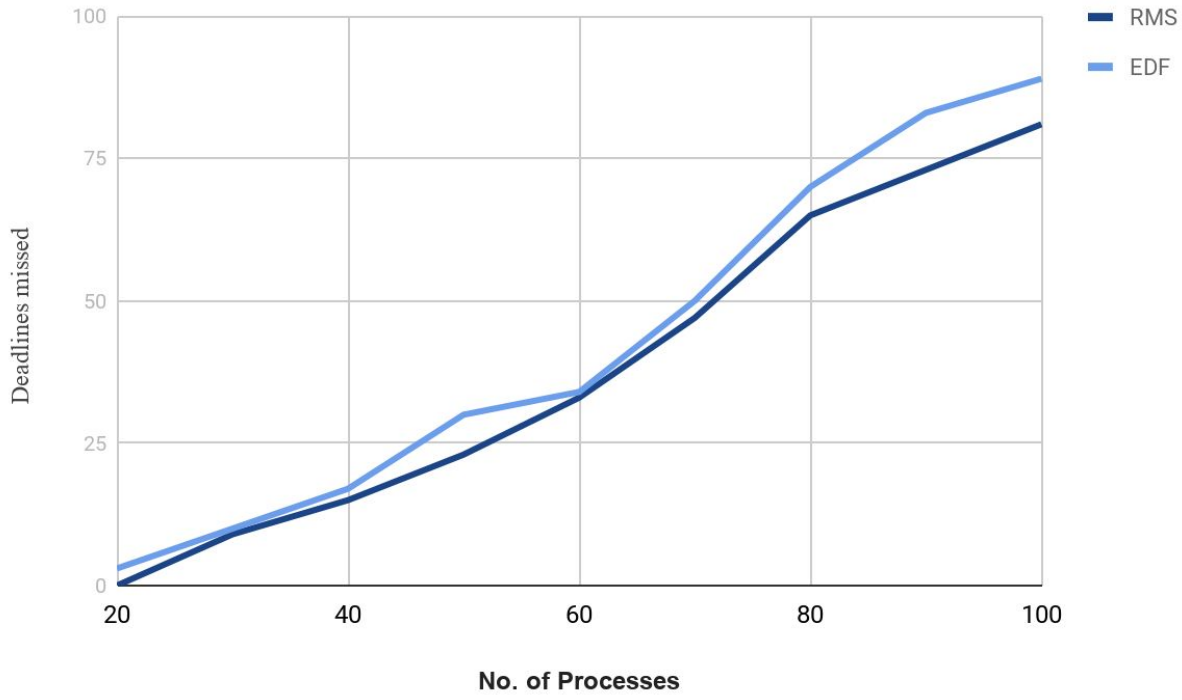
```
10
1 13 74 5
2 10 65 8
3 15 82 8
4 13 102 5
5 15 87 5
6 13 89 9
7 18 88 9
8 23 81 9
9 10 81 9
10 11 77 8
```

Input 2

```
15
1 11 69 5
2 26 104 7
3 18 87 7
4 29 66 8
5 22 69 7
```

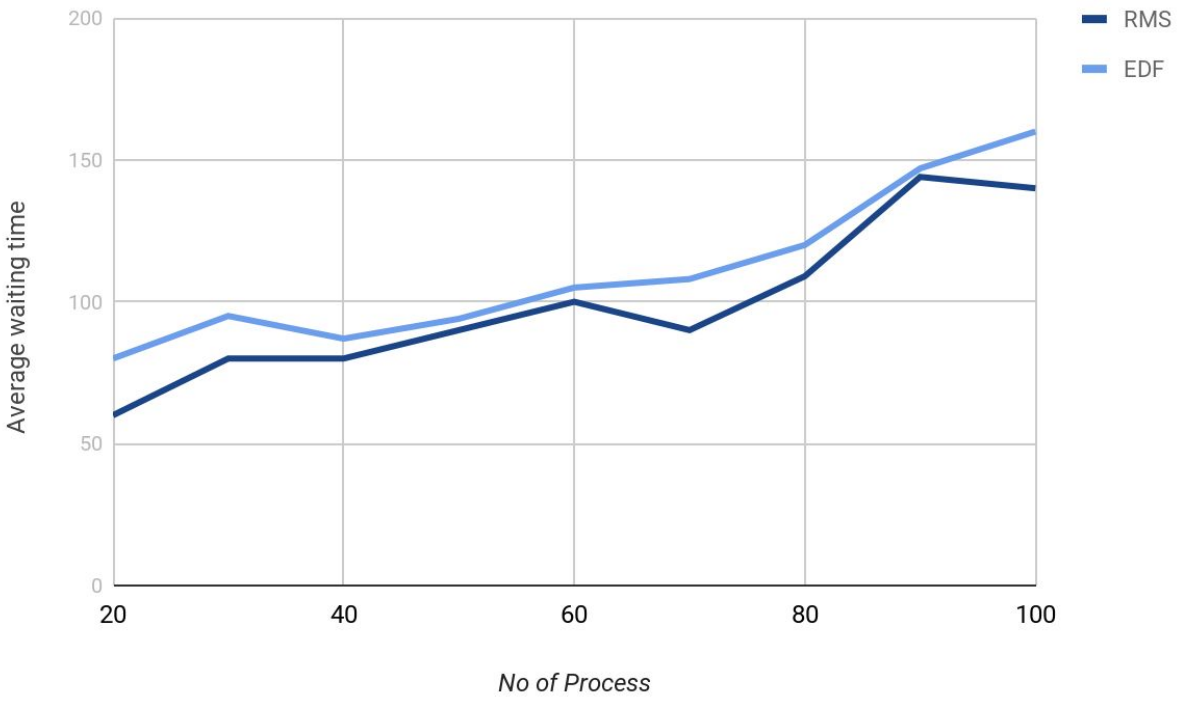
```
6 21 93 6
7 20 61 7
8 28 106 7
9 16 86 5
10 13 75 5
11 18 106 5
12 13 64 6
13 11 92 8
14 20 94 5
15 23 66 9
```

Graph 1 : Deadline missed vs Number of Process



As the no of processes increase, the no of Deadline misses increases.
In general RMS performs better than the EDF.
Still EDF can perform better in some cases.

Graph 2 : Average Waiting Time vs Number of Process



We can deduce from the graph that as N approach very large value ,the rate of increase of waiting time decrease with increase in no of process.
RMS shows better performance in terms of average waiting time also.