Analysis of CPU Scheduling Algorithms and Real Time Scheduling

Instructor : Dr Sanjay Chaudhary
Maunil Vyas[1401007], Deep Patel[1401010], Shreyas Patel[1401025], Karan Patel[1401113]
School of Engineering and Applied Science, Ahmedabad University

***Abstract*—CPU Scheduling is one of the basic criteria for system's efficiency and performance. CPU Scheduling algorithms are the basic techniques used by CPU to solve the Scheduling barriers. There exists some traditional CPU Scheduling algorithms like FCFS, RR, SRN, PS, which have some specific scheduling mechanism, still those algorithms are not suitable for Real time systems. This motivates us to use the real time scheduling algorithms. Here in this paper we have simulated and analyzed two sets of algorithms - 1) Traditional CPU Scheduling Schemes, 2) Real time Scheduling Schemes. Here we have referred scheduling as a CPU Scheduling.**

***Keywords*—CPU Scheduling, Real Time Systems, First Come First Serve(FCFS), Shortest Remaining Next(SRN), Round Robin(RR), Priority Scheduling(PS), Rate Monotonic Scheduling(RMS), Earlier Deadline First(EDF).**

I. Introduction

The aim of process scheduling mechanism is to assign processes to the processor for execution, in such a way that it meets system objectives, such as Response time, Throughput, and processor efficiency. In many systems, scheduling mechanisms are break down into three different parts 1)Long term Scheduling 2)Medium term Scheduling 3)Short term Scheduling and based on these categories the scheduler will schedule the processes.

*1) **Long-term scheduling**:* A long term scheduler inserts new processes into the ready queue by deciding which processes should be run on the system.
*2) **Medium-term scheduling**:* A Medium term scheduling mechanism handles the scheduling of processes for three states 1)Ready 2)Suspend 3)Block.
*3) **Short-term scheduling**:* A short term scheduling mechanism only manages the process state from ready to running, also this scheduling mechanism is crucial to achieve the highest efficiency of system.

Fundamentally scheduling is a task to manage queues with different priorities and to achieve higher efficiency for system by emphasizing on parameters like Throughput, CPU utilization rate, Response time, Turnaround time and Waiting time.

II. Scheduling criterias

Scheduling mechanism emphasizes on two criterion - 1) User oriented which favour the user aspects and 2) System oriented which satisfy the system aspects.

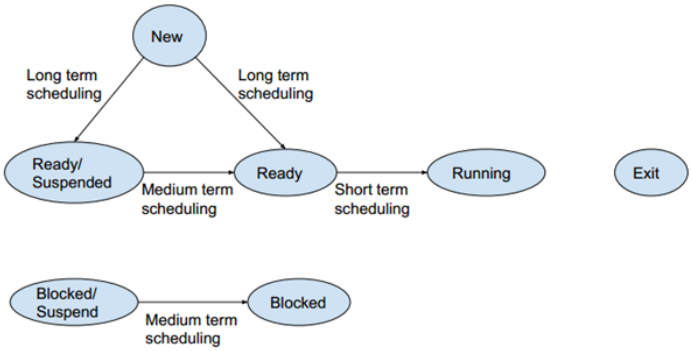

Fig. 1: Scheduling process state transactions

A. *User oriented criterias*

*1) **Waiting time**:* The sum of the time that processes spend in the ready queue/blocked queue.
*2) **Turnaround time**:* Turnaround time is the sum of waiting time and execution time of process.
*3) **Deadline**:* The time for each process by which that process must either be started or completed.

B. *System oriented*

*1) **Throughput**:* The number of processes completed per time unit.
*2) **CPU Utilization**:* It is ratio of total time CPU is busy in running the processes and total time CPU is in running state including idle time.
*3) **Fairness**:* It is the quality of sharing of the CPU fairly by the processes.

III. CPU Scheduling algorithms

There are two classes of CPU scheduling algorithms: pre emptive scheduling algorithms and non pre emptive scheduling algorithms.

A. *Pre-emptive Scheduling*

While process is in running state, it is possible that it is temporarily moved into blocked/ready state by the OS to clear CPU for more privileged process. This phenomenon is known as pre-emption.

B. *Non pre-emptive scheduling*

This one is in contrast of pre emptive scheduling. Here the process which is in running state, cannot be interrupted by the OS.

Decision of which process will first start it's execution among all processes, depends on priority of that process. If process has higher priority then other processes then it will be simply moved into running state. But problem with this type of priority assignment is that, processes with lower priority will be starved. Solution for this problem is that processes will be assigned priority such that it will be changed according to execution history also.

IV. Simulated CPU Scheduling Algorithms

Here we will discuss the scheduling algorithms that are simulated in this paper. There exist different types of scheduling algorithms like First Come First Serve(FCFS), Shortest Remaining Next(SRN), Round Robin(RR), Priority Scheduling(PS) etc.

A. *FCFS*

Processes which come first in ready queue which will be dispatched first for execution. It is nothing but FIFO like scheduling algorithms. It is a non-preemptive scheduling algorithm. Implementation of FCFS is very easy. It has very poor performance as average waiting time and turnaround. Processes which are at the end of the queue, are starved even through it has highest priority.

B. *SRN*

Shortest remaining time is the preemptive algorithm. Process which is in ready queue and less remaining execution, will be moved into CPU. For implementation of SRN, priority queue is used. SRN has best avearge waiting and average turnaround time performance compare to other scheduling algorithms.

C. *RR*

This one is pre emptive scheduling algorithm. Each process is provided a fixed execution time, which is known as ***Quantum***. Quantum value is decided by operating system. Other than this, it is same as FCFS scheduling algorithm. Process which comes first, will be provided CPU first, if its execution time is less than the quantum then it will finish its execution, but if its execution time is more than quantum, then it will be executed for given quantum period and then pre-empted and moved into ready or blocked queue again. Context switching is used to save states of preempted processes. RR is greatly depended on value of quantum.

D. *PS*

In this type of scheduling algorithm, each process is assigned priority. Process with highest priority is executed first. Processes with same priority are executed in FCFS manner. Priority can be decided based on memory requirements, time requirements or any other resource requirements.

V. System Design

Here for simulation, C++ programming language is used because it has reach set of libraries and it is directly in touch with operating system.

For simulation, the variables like average waiting time, average turnaround time, average CPU utilization and average throughput, percentage of deadline missed are used. Here the system is assumed to be uniprocessor. Before describing the variables, we show the process states transaction diagram which is considered during the simulation.

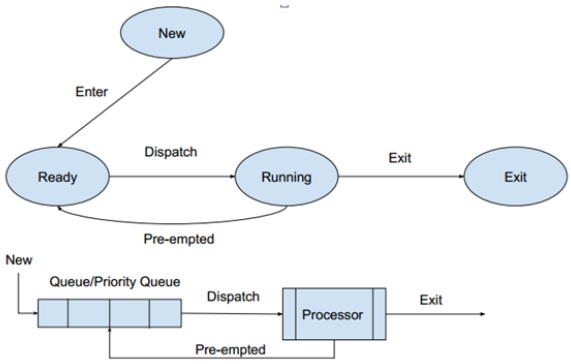

Fig. 2: State transactions diagram

As we can see in figure.2, New processes enter in ready state. Here ready state is nothing but a simple queue or priority queue. If scheduling algorithms is like FCFS then we can use simple queue but if scheduling algorithms is like SRN or PS then we have to use priority queue. Based on priority, process is dispatched from ready state to CPU. If process is pre empted then it is moved to ready state. If process is completed successfully then it is moved to exit state.

A. *Average waiting time*

n = number of processes, w_i = waiting time of i^{th} process

$$Average\ waiting\ time = \frac{\sum_{i=1}^{n} w_i}{n} \tag{1}$$

B. *Average turnaround time*

n = number of processes, w_i = waiting time of i^{th} process
e_i = execution time of i^{th} process

$$Average\ turnaround\ time = \frac{\sum_{i=1}^{n} (w_i + e_i)}{n} \tag{2}$$

C. *Average CPU-utilization*

T = Time taken for completion of n processes
e_i = execution time of i^{th} process

$$Average\ CPU - Utilization = \frac{\sum_{i=1}^{n} e_i}{T} \tag{3}$$

D. *Average Throughput*

The number of processes completed successfully per time unit.
n = number of processes
T = Time taken for completion of n processes
t = time value where t < T
$n^{'}$ = number of processes with finish time < t

$$Average\ Throughput = \frac{n^{'}}{T} \tag{4}$$

Here, average through put means number of processes completed successfully before given time.

E. *Deadline missed*

Here we are going to analyze that out of all processes, how many processes missed their fixed deadline.

VI. Simulation Result and Analysis

In this section, simulated results and analysis on results are presented.

A. *Simulation results*

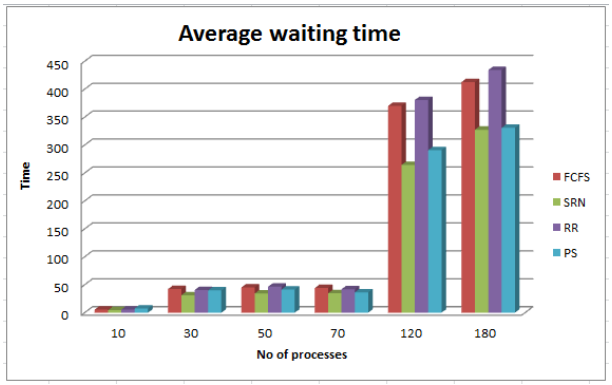

Fig. 3: Average waiting time vs No. of processes

In uniprocessor, with fewer than 70 tasks/processes, the average waiting time was less than 44 time units. However, this average waiting time increased with the number of processes or tasks. But surely if there would be multi-core system then for given number of processes, average waiting time will decrease because of parallel execution of processes at a time. If number of processes increased in multi-core, average waiting time also increased. For RR, average waiting time was very high compare to other processes because RR depends on value of ***quantum***.

We can do same analysis for average turnaround time because we know turnaround time is waiting time + execution time. RR has bad performance compare to other scheduling algorithms in terms of turnaround time because of value of the quantum.

The average throughput decreased when we had more tasks

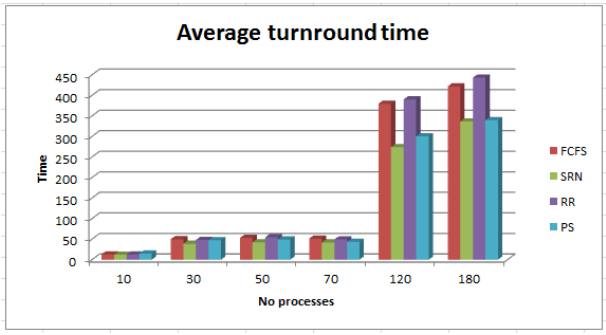

Fig. 4: Average turnaround time vs No. of processes

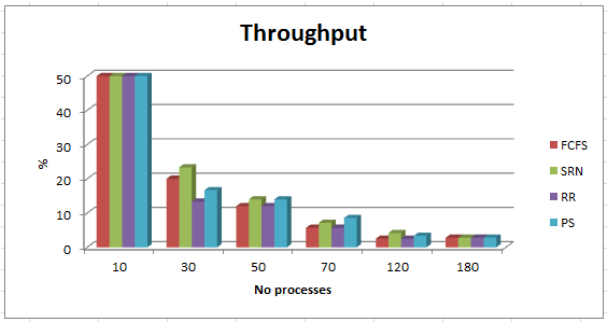

Fig. 5: Average Throughput vs No. of processes

to execute. When executing 70 tasks or more, the average throughput for different scheduling algorithms converged, and their percentage was minimized.

We analyzed that for different scheduling algorithms, average CPU-utilization did not change much more. But if we analyzed same for multi-core then it might give some useful results.

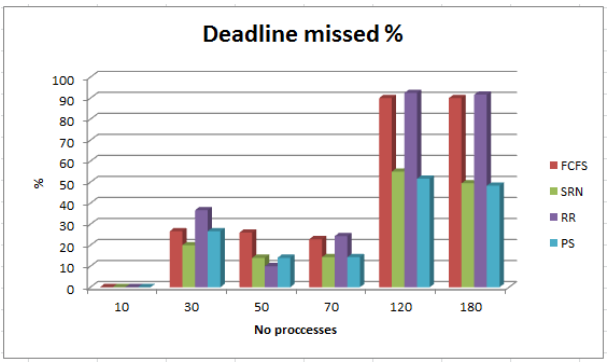

Fig. 6: Deadline missed vs No. of processes

As we can see, RR had more deadline missed percentage than other because it depended on value of quantum value. FCFS also had very bad performance compare to SRN and PS. Number of processes increased, deadline missed percentages significantly increased. From this result we can say that this type of scheduling algorithms, may break down performance of Real time systems very badly because those systems depended on fixed deadlines.

B. *Analysis*

- The SRN algorithm shows the lowest average waiting and turnaround time compare to other algorithms.
- RR scheduling algorithm is significantly dependant on value of the quantum.
- If we use this scheduling algorithms in real time systems which depend on deadlines, then systems will fail. So for real time systems, we have to come up with other scheduling algorithms which consider this deadlines as scheduling parameter and these types of scheduling algorithms are known as Real time scheduling algorithms.

VII. Real Time scheduling

Real Time scheduling is a technique in which we schedule the processes that are generated at run time in the real time systems.

The Real Time scheduling algorithms require the process deadline as the main parameter for process scheduling.

A. *Deadline Consequences*

Mainly two type of deadline consequences are there in real time systems.

*1) **Hard Deadlines**:*
- System fails if deadline missed
- Goal: Guarantee no deadline miss

*2) **Soft Deadlines**:*
- User may miss but the system will not fail
- Goal: Meet most deadline most of the time

So, real time scheduling will take this deadline in consideration and try to come up with good scheduling algorithms and techniques.

B. *Deadline Characteristics*

There are mainly two class of process in real time systems as shown below:-

*1) **Periodic Processes**:* Appears frequently in the system.

*2) **Aperiodic Processes**:* They don't have specific appearance period.

External events triggers the activation of aperiodic process and these events are random in nature. Their execution should be finished within the given deadline. This is the only timing constraint associated with them.

The deadline is time before which the process must be completed. Process requires some time for execution after admission in the system and before exiting from it. The computation time may be static or dynamic depending on the nature of the process.

Following are the constraints that must be met by a process before scheduling it in RTS:-

$$C \leq D \tag{5}$$

where

C = Computation time
D = Deadline

For each periodic processes, its period must be at least equal to its deadline. Hence, for a periodic processes, the following relationship must hold:

$$C \leq D \leq T \tag{6}$$

where

T = Period of process

In contrast to the periodic processes, the non periodic processes can be invoked at any time.

C. *Static and Dynamic Algorithms*

There are mainly two types of scheduling algorithms :
- Static priority based algorithms
- Dynamic priority based algorithms

In static priority based algorithms, the scheduling decisions are made at compile time and schedule is generated offline based on the prior knowledge of task parameters. Where as in dynamic priority based algorithms, the scheduling decision are made at run time.

D. *System Model*

As a basis for discussing scheduling a system model is required. A common model in the literature is given as follows.

Assume that there exist n number processes with following characteristics:-

D_i - Deadline
P_i - Period
C_i - Computation time

Following are the assumptions that we make for simulation:-
- $C_i \leq D_i = P_i$
- Only periodic processes are taken in consideration
- Constant computation(execution) time
- Inter process communication not permitted
- Uniprocessor environment

But in actual processes following situations may be there:-
- Process may be either periodic or aperiodic in nature.
- Processes may share their data with each other.
- There may exists the multicore real time systems with static or dynamic process allocation.

VIII. Real time Scheduling Algorithms and Implementation

Basically, there two main real time scheduling algorithms - 1) RMS(Rate Monotonic Scheduling) 2) EDF(Earliest deadline first). RMS is of type static priority driven pre-emptive approach whereas EDF is of type dynamic pre-emptive approach. Here it is assumed that both the scheduling algorithms are implemented on the uniprocessor.

A. *RMS-Rate Monotonic Scheduling*

RMS requires the periodic processes for the scheduling. It is a pre-emptive scheduling algorithm. All the processes are allocated priorities according to their period. Shorter the period, higher their priority. RMS' Performance is acceptable.

It is divided into two parts, the first part checks for the constraint satisfiability by the process for RMS scheduling, while the other assigns the priority to the processes according to their period.

The constraint which is checked by the first part of the algorithm which is also known as processor :-
$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} + ... + \frac{C_n}{P_n} \leq n(2^{\frac{1}{n}} - 1)$

where n = number of processes

If this constraint is satisfied then this set of processes can be schedulable with rate monotonic scheduling algorithm. Here, this constraint is a necessary, but not a sufficient condition for being scheduled.
The following describes the sufficient condition. The period of each process i is P_j. The workload over $[0, t]$ (for any t>0) due to all processes of equal or higher priority than process j is given by

$$W_l(t) = \sum_{i=1}^{j} (C_i \lceil \frac{t}{P_i} \rceil) \tag{7}$$

Task j is schedulable for all process types if

$$min_{0<t\leq P_j} (\frac{W_j(t)}{t}) \leq 1 \tag{8}$$

The entire given process set is schedulable if the both above conditions holds for given processes.

B. *EDF-Earliest Deadline First*

Earliest deadline first scheduling uses the simple system model given earlier. It is a pre-emptive and priority based scheduling algorithm like the RMS. The condition used for scheduling is: ***the process with the closest deadline is assigned the highest priority in the system and therefore it is scheduled first.***

The schedulability constraint is thus given as:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{9}$$

Hence, 100 % processor utilization is possible.

C. *RMS-Implementation*

For implementation, we assume that it's done in the uniprocessor system. Process state diagram is same as shown in figure 2. In this type of scenario, mutual exclusion is guaranted by disabling all interrupts because uniprocessor only allows interleaving of processes. We had disabled all interrupts. We implemented priority queue for allowing lowest period process first in queue. Following are some screens shot of console output, we got for RMS.

```
int Period[] = {10,5,30,15};
int Execution[] = {2,1,5,2};
process *Array[N];
```

Fig. 7: Process set for RMS

```
// Register signal and signal handler
signal(SIGINT, signal_callback_handler);
```

Fig. 8: Signal handler

```
// Define the function to be called when ctrl-c (SIGINT) signal is sent to process
void signal_callback_handler(int signum)
{
   outputfile<<" Interrupt is Caught :- signal "<<signum<<endl;
   // Cleanup and close up stuff here
   outputfile<<" Interrupt is disabled. "<<endl;
   // Terminate program
   cout<<" Interrupt is Caught :- signal "<<signum<<endl;
   // Cleanup and close up stuff here
   cout<<" Interrupt is disabled. "<<endl;
}
```

Fig. 9: Signal call back function

```
Process 1 created successfully.
Process 2 created successfully.
Process 3 created successfully.
Process 4 created successfully.

CPU Utilization :-0.700000 Theoritical :- 0.756828
Process No. -> Waiting time -> Turn around time

ProcessID : 1 state : ready Tue Dec  6 23:38:29 2016

ProcessID : 2 state : ready Tue Dec  6 23:38:29 2016

ProcessID : 3 state : ready Tue Dec  6 23:38:29 2016

ProcessID : 4 state : ready Tue Dec  6 23:38:29 2016


ProcessID : 2 state : running Tue Dec  6 23:38:29 2016

ProcessID : 2 state : exit Tue Dec  6 23:38:30 2016

ProcessID : 2 -> 0 -> 1
ProcessID : 1 state : running Tue Dec  6 23:38:30 2016

ProcessID : 1 state : exit Tue Dec  6 23:38:32 2016

ProcessID : 1 -> 1 -> 3
ProcessID : 4 state : running Tue Dec  6 23:38:32 2016

ProcessID : 4 state : exit Tue Dec  6 23:38:34 2016
```

Fig. 10: Output on console for RMS

D. *EDF-Implementation*

For implementation, again we assumed that it's uniprocessor system.

```
int deadline[] = {15,12,9,8};
int Execution[] = {4,3,5,2};
int Arrivaltime[] = {0,0,2,5};
```

Fig. 11: Process set for EDF

```
Process 1 created successfully.
Process 2 created successfully.
Process 3 created successfully.
Process 4 created successfully.
ProcessID : 1 state : ready Tue Dec  6 23:40:14 2016

ProcessID : 2 state : ready Tue Dec  6 23:40:14 2016

ProcessID : 2 state : running Tue Dec  6 23:40:14 2016

ProcessID : 2 state : ready Tue Dec  6 23:40:16 2016

ProcessID : 3 state : ready Tue Dec  6 23:40:16 2016

ProcessID : 3 state : running Tue Dec  6 23:40:16 2016

ProcessID : 3 state : ready Tue Dec  6 23:40:19 2016

ProcessID : 4 state : ready Tue Dec  6 23:40:19 2016

ProcessID : 4 state : running Tue Dec  6 23:40:19 2016

ProcessID : 4 state : exit Tue Dec  6 23:40:21 2016

ProcessID : 4 -> 2 -> 0 -> 2
ProcessID : 3 state : running Tue Dec  6 23:40:21 2016

ProcessID : 3 state : exit Tue Dec  6 23:40:23 2016

ProcessID : 3 -> 5 -> 2 -> 7
ProcessID : 2 state : running Tue Dec  6 23:40:23 2016
```

Fig. 12: Output on console for EDF

Process state diagram is same as shown in figure 2. Mutual exclusion is guaranted by disabling all interrupts.. With the help signals, We disabled all interrupts. We implemented priority queue for allowing process with lowest deadline first in queue. figure 13 and figure 14 are the output of our simulation of EDF.

Same signal handler and function as showed in fig.10 and 11 are used in EDF also.

IX. Comparison with Real time Scheduling algorithm - EDF

We have already seen simulation result for FCFS, SRN, RR, and PS for given criterias. After implementation of RMS and EDF , Now we are going to compare EDF with those general CPU scheduling algorithms.

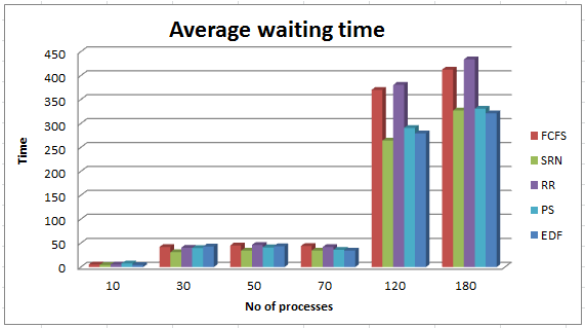

Fig. 13: Average waiting time vs No. of processes

As we can see that EDF has better better avearge waiting time compare to other CPU scheduling algorithms. SRN and EDF, both have lowest waiting time compare to other algorithms because both consider criteria like execution time and deadline respetively.

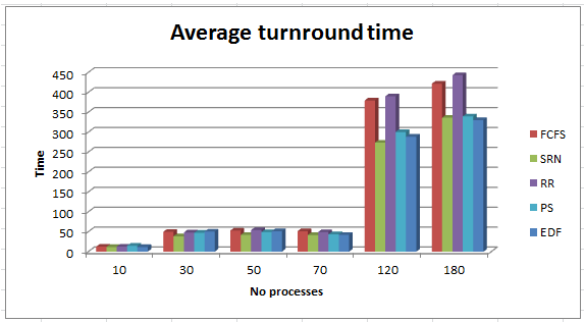

Fig. 14: Average turnaround time vs No. of processes

Again SRN and EDF have lowest turnaround time compare to other scheduling algorithms. As number of processes increase, average turnaround time also increases.

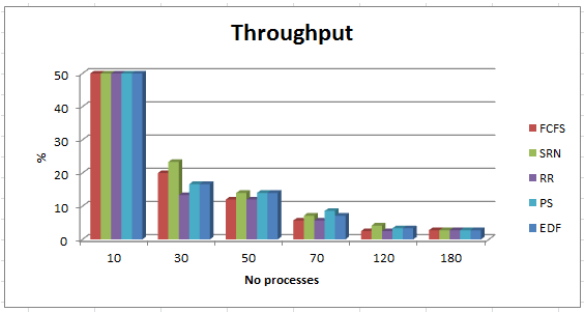

Fig. 15: Average Throughput vs No. of processes

As we can see in figure.15, the average throughput for EDF decrease as well, when we have more tasks to execute. When executing 70 tasks or more, the average throughput for different scheduling algorithms converged, and their percentage was minimized.

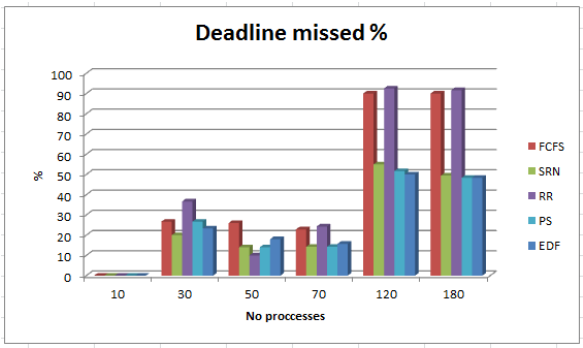

Fig. 16: Deadline missed vs No. of processes

As we know that EDF mainly focus on deadline so it will have

better responce then other scheduling algorithms and that what we can see in figure.16. As we can see EDF has comparatively less deadline missed percentage then other algorithms.

X. Conclusion

We compared various CPU Scheduling Algorithms, both static and real time and did their simulation in C++, after analyzing their results, we concluded that traditional scheduling algorithms like FCFS, RR, SRN, PS etc. missed lots of deadlines. Therefore we conclude that, these type of scheduling algortihms cannot be used for scheduling in Real time systems and the specialized real time algorithms like RMS and EDF are needed for process scheduling in Real time systems. SRN and EDF has less waiting and turnaround time compare to other scheduling algorithms.

References

[1] Sultan Almakdi, Mohammed Aleisa, Mohammed Alshehri, Simulation and Performance Evaluation of CPU Scheduling Algorithms, Alghat, KSA:2015.
[2] William Stallings, Operating systems, Internals And Design principles, Eighth edition.
[3] www.tutorialspoint.com, Operating System Scheduling algorithms
[4] en.wikipedia.org, Earlier Deadline First Scheduling
[5] en.wikipedia.org, Rate Monotonic Scheduling