Operating Systems 2

Tanmay Garg CS20BTECH11063

Programming Assignment 3

Program Design

- The program take input from the file “inp_params.txt” and reads the Number of Processes given in the first line
- In a loop all the processes are read and put into the list of all processes with the following as initialized
    - Task ID
    - Task Time
    - Task Time Period
    - Task Arrival Time (periodic processes)
    - Task Priority (Depends on EDF or RMS)
    - Task Deadline (Next periodic arrival)
- A *PrevRunningProcess* is used to keep track of previous process that was pre-empted or was running before the new task was executed
- A loop is run till *ListTaskProcesses* is empty
- *Current_time* is updated by taking the arrival time of the first element from *ListTaskProcesses*
- First, we check if any of the processes in the *ReadyQueue* have missed a deadline or not
    - For this we use an iterator to go through the entire list
    - If the Deadline of the task is before the *current_time* then process has missed its deadline and it is then terminated and removed from *ReadyQueue*
- Next, we check if the previous running process has missed its deadline or not
    - If so, then it is terminated and *PrevRunningProcess* is initialized to its default value
- Now, we add all the processes that have arrived in according to *current_time* from *ListTaskProcesses* to *ReadyQueue*
- *Next_arrival_time* is initialized as the arrival time of first process given in *ListTaskProcesses*
- If both the *ReadyQueue* is empty and *PrevRunningProcess* has been terminated, then the *current_time* is updated to the *next_arrival_time*
- If not, then we move on to run the processes given in the *ReadyQueue* and *PrevRunningProcess*
- If *ReadyQueue* is not empty, then highest priority process is taken from it and is made the running process
    - Else the *PrevRunningProcess* is made the running process as it hasn’t missed its deadline
- If uptill now the CPU was in an Idle State, then it is logged as Idle
- If the *runningProcess* has higher priority than *PrevRunningProcess* then *PrevRunningProcess* will be pre-empted
    - Else *runningProcess* is pushed back to *ReadyQueue* and *PrevRunningProcess* becomes the current *runningProcess*
- If the *runningProcess* was interrupted before (i.e., it was preempted before) then it resumes its execution
    - Else it starts executing for the first time

- Now we check if the current *runningProcess* can be completed in time
    - If not then it will be either be preempted by another process in the next iteration or will be missing its deadline, so it is made into *PrevRunningProcess*
- If it can be completed in time, then it will be completed in the following loop
- In the loop after completing the process, if a process is available in the *ReadyQueue* then it will remove that process and run it
    - If that particular process was preempted, then it will resume
    - Else it will be executed for the first time
- If the *ReadyQueue* becomes empty but there is still some time left between next arrival and current time, then CPU will become idle, and it will be flagged as one
- If no time is left, then it breaks out of this loop and goes back to the first loop to continue the entire process
- After getting out of the main loop with No processes left to run in *ListTaskProcesses*, we will check if the *PrevRunningProcess* is still running then we will end the process
- We will check if *ReadyQueue* is empty or not
    - If not, then we will continue to run those processes and terminate them based on their priority
    - Highest priority process would be found and would be run in the end
- Wait time has been taken as

$$Waiting\ Time = Current\ Time - Arrival\ Time - Burst\ Time$$

- For processes that have missed deadline, the wait time Is taken as

$$Waiting\ Time = Deadline\ Time - Arrival\ Time$$

Structure of Program

- We have two classes
    - *TaskProcess*
        - *ID*
        - *Task_time*
        - *Task_period*
        - *Task_deadline* (initialized as task_arrivaltime + task_period)
        - *Task_arrivaltime*
        - *Task_burst_time*
        - *Task_priority* (inverse of period in RMS & inverse of deadline in EDF)
        - *isInterrupted* (initialized as false)
    - *ListProcess*
        - Inherited from *std::vector< TaskProcess>*
        - *SortList()* : sort the vector in order of arrival time using a lambda function
        - *Push (TaskProcess &p)*: push at end of vector then heapify it based on lambda function
        - *Pop()* : using standard heap pop function and remove last element
- *DeadlineMiss*
    - This function is for processes that have missed deadline
    - It outputs in the log file

- Returns the wait time of the process

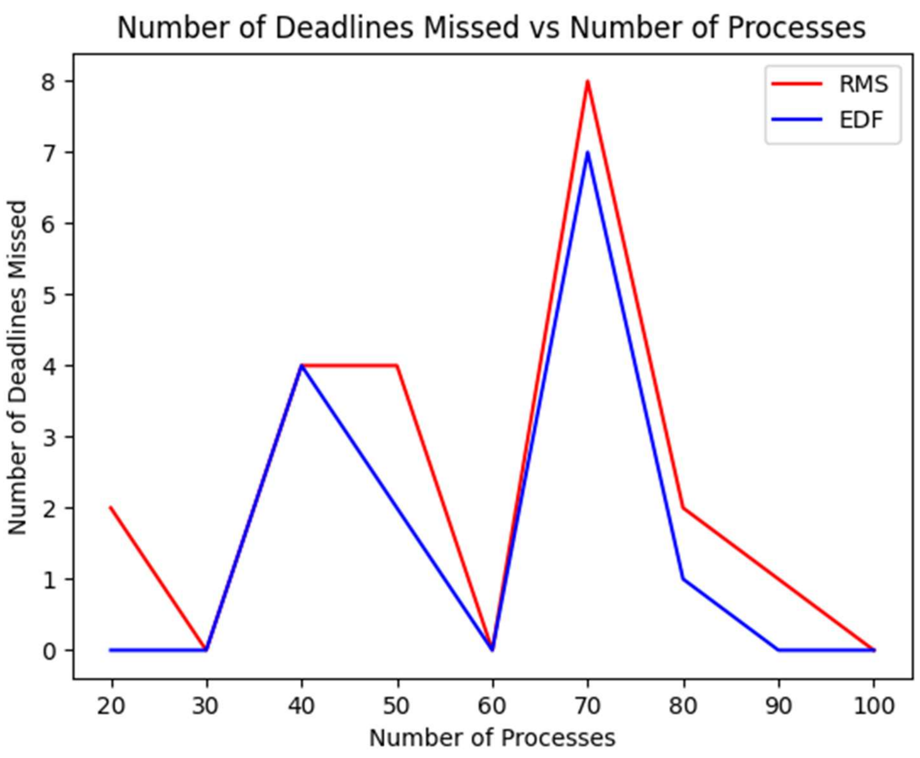
$$Waiting\ Time = Deadline\ Time - Arrival\ Time$$

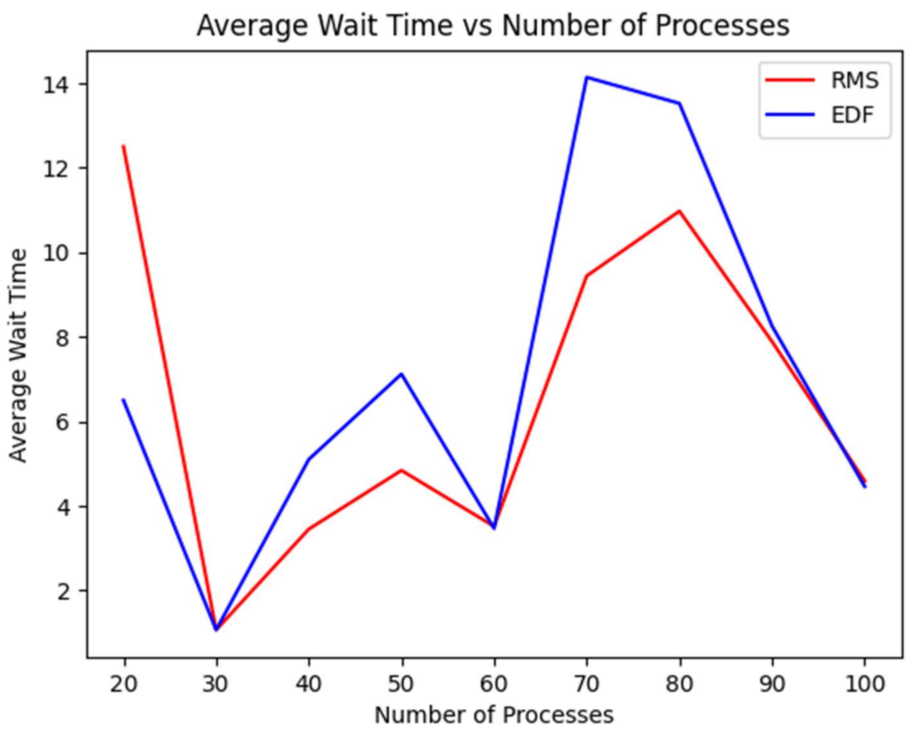
- *findHighestPriority*
  - To find the highest Priority function from *ReadyQueue*
  - By iterating through the entire vector

Complications in the program

- Making a priority-based data structure using vectors
- Handling the previously running process and current running process based
- Handling task deadlines in cases where they have been preempted
- Terminating process after they have been completed and printed in the log file
- Checking for cases when the *ReadyQueue* is not empty, and program exits the main loop

Comparison between EDF and RMS





Note:

- The processes have been generated at random
- The processes start with 2 processes repeating 10 time each to 10 processes repeating 10 times each
- Behavior of the program changes when the processes in comparison have same priority, hence the results might differ in different situations and on different systems as we are not using any other metric to compare two processes with same priority
- Our main point of comparison is between RMS and EDF for a particular situation i.e., for running for a particular number of processes in both the algorithms and then comparing them
- The same file has been used to compare for both the scheduling methods in order to avoid ambiguity with respect to algorithm working differently on different types of processes with different parameters
- The results have been separately added to a python file to generate the graph and have been added to the report

Conclusion

- Since all the processes and situations were randomly generated and have different burst times, time periods and CPU utilizations here is the conclusion that we can draw based on the plots above
- According to the graph and various situations, it can be observed that in general EDF has performed better in running the processes by having lesser missed deadlines
- It can be observed that EDF performs with higher average waiting time because it is scheduling all the processes and running them completely, so in general the average wait time is higher than RMS