Uniprocessor Scheduling

Implementation

Raed Albloushy and Jisoo Han
CPR E 458 – Section 1
Department of Electrical and Computer Engineering at Iowa State University
raed@iastate.edu
jhan8@iastate.edu

Abstract

Our Project tackles with the problem of not having a lot of tools that test the different types of scheduling like RMS and EDF. In our project we build a system that schedules both and outputs what the schedule is like for the task set that we give to the program. This tool as well will determine the schedulability analysis for the given task set in terms of EDF and RMS, after that has been determined then only a schedule can be formed. This tool should also not try to schedule RMS tasks that not even schedulable by exact analysis. it should also not have a delay in giving our result and be user friendly, and easy to use. Our Evaluation was based on the idea if our tool successfully gives the schedulability analysis for a given task including exact analysis if needed by RMS and print the logistics for these checks and if possible, schedule the task by RMS and EDF. Our conclusion is that our Tool passes all the HW problem that were given to schedule RMS and EDF, it works instantly with no delay and correctly identifies if the task set is schedulable or not. The user inputs the tasks using the command line in the form of number inputs.

1. Introduction

We have both taken CPRE 308 (Operating Systems Course) in Iowa State, and we have implemented scheduling algorithms like Round Robin in Linux in the labs for that course. We had some knowledge about task scheduling implementation from there and when we were doing Homework for This course, CPRE 458, We realized that it is cumbersome sometimes to schedule some of the task sets and we wished if there were some kind of tool to make our life easier. This is our inspiration for attempting to implement EDF and RMS in this tool. We wanted to design a user-friendly tool that gives off the behavior of both schedulers and the task set analysis. As well as some tool to compare results of schedulers.

2. Problem Formulation

There is not enough and easily found tools that allow you to check schedulability analysis and schedule for a given task set.

2.1. System Model

Task Scheduling presents itself everywhere in RTOS, either in a plane control system or even a vending machine that has a much less strict deadline. Mainly our problem lies in students understanding how these schedulers work and how the behavior is different in what they expect.

2.2. Problem Statement

The problem is that there are not enough readily available tools that are user friendly that show how schedulers work beyond the conceptual understanding. We are trying to build such a tool that easily schedules and implements these schedulers based on a user given task.

2.3. Objectives and Scope

To Identify our own observations of schedulers and their real-life performance based on turnaround and response time of scheduler. Each type of schedulers (RMS, EDF) have different response time and turnaround time. We will figure out the difference among the schedulers. As well as make an easy to use tool to Schedule tasks using these schedulers.

3. Methodology

Our tool will be a C based program that will take inputs of the Number of N tasks in the given task and then the CI (computation time) and PI (period time) of each task. After all these user inputs are given through the command terminal the program will print out if the following task set is schedulable or not and then according to that printout the schedule using RMS and EDF.

3.1. Algorithm / Protocol

There are two Algorithms that we will use for this project, RMS and EDF. Starting with RMS, the way RMS works, it classifies Tasks priority based on the period of the task the smaller the task, the higher the priority of the task is. Based on that it schedules tasks, the behavior would be different up to the LCM of all task periods.

Task Schedulability RMS:

$$\sum_{i=1}^{n} c_i/p_i \leq n(2^{1/n} - 1).$$

Figure 1:RMS Schedulability Check

If a given task set passes this test, the task is Schedulable by RMS but if it fails it, not necessarily schedulable. Then we must check exact analysis for RMS, if it fails that then the task is not schedulable by RMS.

Exact Analysis RMS:

$$\begin{aligned}
&Set \quad t_0 = \sum_{j=1}^{i} c_j \\
&t_1 = W_i(t_0) \\
&t_2 = W_i(t_1) \\
&t_3 = W_i(t_2) \\
&\vdots \\
&t_k = W_i(t_{k-1}) \\
&Stop \quad when \quad W_i(t_k) = t_k
\end{aligned}$$

Figure 2:RMS Exact Analysis

An Example of this RMS process, is the following task Set:

Task T1: c1 = 2; p1 = 8
Task T2: c2 = 3; p2 = 12
Task T3: c3 = 4; p3 = 16

$$\sum_{i=1}^{3} \frac{c_i}{p_i} = \frac{2}{8} + \frac{3}{12} + \frac{4}{16} = 0.25 + 0.25 + 0.25 = 0.75$$

$$3 * (\sqrt[3]{2} - 1) = 0.7797$$

$$0.75 < 0.7797$$

Therefore, this task set is schedulable via RMS.

Exact Analysis

$$W_i(t) = \sum_{j=1}^{i} c_j * \left\lceil \frac{t}{p_j} \right\rceil$$

Is T3 Schedulable?

$$t_0 = C_1 + C_2 + C_3 = 2 + 3 + 4 = 9$$

$$t_1 = W_3(t_0) = 2C_1 + C_2 + C_3 = 4 + 3 + 4 = 11$$

$$t_2 = W_3(t_1) = 2C_1 + C_2 + C_3 = 4 + 3 + 4 = 11$$

Since $t_1 = W_3(t_1)$ and $W_3(t_1) < 16$, T3 is schedulable.

Figure 3:RMS Schedulability & Exact Analysis example

Using figure 3 we can see that according to the formula from figure 1 our task set is schedulable, exact analysis was not needed here but for the sake of clarification, we took the lowest priority task to show how to use figure 2.

The main difference between RMS and EDF, is that Tasks are scheduled in EDF by the task with earliest deadline as well as the deadline for each Task is a dynamic deadline.

Task Schedulability EDF:

$$\sum_{i=1}^{n} c_i/p_i \leq 1$$

Figure 4:EDF Schedulability Check

Using the previous given task set with figure 4:

2/8+3/12+4/16 = 0.75 <= 1 ➔ Schedulable

EDF Schedulability test is necessary and sufficient, there is no exact analysis like RMS. If a given task set fails this check it is not schedulable by EDF.

3.2. Illustrative Example

Our tool resolves the problem that many Learners might face in understanding EDF and RMS schedulers, as it will schedule the given tasks for them if possible as well as print the schedulability analysis for both RMS and EDF.

Figure 5 below, will show the general structure of our program and how inputs and outputs are taken and displayed.

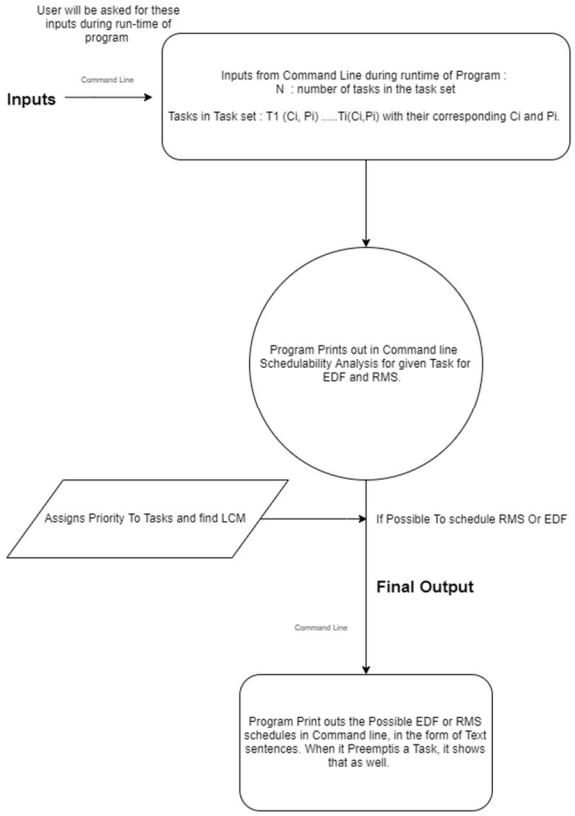

Figure 5:Design Diagram

4. Implementation/Simulation Architecture

We Used C Code to simulate the schedulers and we take our inputs and display our outputs using the command line that is used to launch the tool. The libraries that we use in our C code is the standard C library stdio.h and the string.h library as well the math.h library.

We Implemented Tasks as a C struct as follows:

```c
struct task
{
    /* Values initialized for each task */
    int period;   /*the interval of the the task*/
    int runtime;  /* Time task requires to complete job */
    int priority; /* Priority of the task */
    int arrivaltime; /*exact arrival time for next time*/

    int arrivaltimes[100];/*Possible Arrival times*/
    int instancesP;    /*according to LCM how many instances are possible*/
    int instancesleft; /*how many instances left*/
    /* Values algorithm may use to track tasks */
    int starttime;
    int endtime;
    int flag;
    int remainingtime;
    int orignalindex;
};
```

Figure 6 : Task Struct Data Structure

Using the Task struct data structure we initialized all the tasks in an array that takes each's Task Struct.

```c
void checkfeasibility(int numoftasks, struct task *proc)
{
    int edfv = checkedf(numoftasks, proc);
    int rmsv = checkrms(numoftasks, proc);
    struct task proc_copy[numoftasks];
     memcpy(proc_copy, proc, numoftasks * sizeof(struct task));
    if (edfv == 1)
    {
        printf("task set is schedulable by edf\n");
        edf(numoftasks, proc);
    }
    else
    {
        printf("task is not schedulable by edf\n");
    }

    if (rmsv == 1)
    {

        printf("task set is schedulable by rms\n");
        rms(numoftasks, (proc_copy));
    }
    else
    {
        printf("task is not schedulable by rms\n");
    }
}
```

Figure 7: Test Check and scheduling coding example

Once we have all the tasks initiated we take them into a function called *checkfeasiblitly()* , where we have helper functions and we use the previously mentioned schedulability checks in figures 1, 2 and 4 to check our tasks. If they are schedulable we do schedule them using two function one called simply *rms()* and the other *edf()* both of these functions take the number of tasks and the tasks struct array as well as parameters. This is what figure 7 basically shows.

The actual RMS scheduling logic is a clutter of code that has multiple of helper functions that check do things like checking the LCM and initializing the tasks arrival times as well as sorting them by priority.

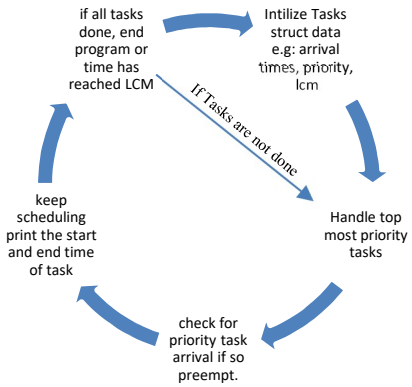

Figure 8: RMS Implementation logic

As figure 8 shows, we basically start with high priority tasks and record their start time and move on time as well as checking if any new instance of a high priority task arriving, if so, we preempt. If not, we keep on moving time and checking tasks until everything is done or time has reached the LCM. Which should not happen without everything being scheduled if the task is feasible to schedule

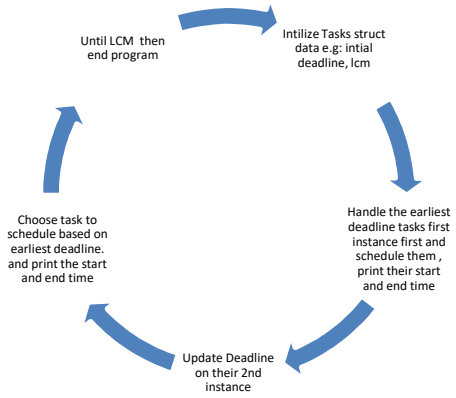

Figure 9: EDF Implementation logic

As for EDF our logic in figure 9, It is simpler to implement as we just keep updating the deadline for each task and keep on scheduling tasks based on earliest deadline and end program once we have reached LCM. It is much simpler to handle as there are no gaps in the schedule this way as well as the priority is directly handled by the deadline.

Our testing was done on mainly HW problems to make sure that it follows most of the and is accurate with the answers that we have.

This is for example a test case of our whole application running we are testing a case where the exact analysis should show that it is not possible to schedule via RMS and schedule using EDF.

```
PS C:\Users\Raed\Documents\GitHub\Cpre458-project> ./a.exe
enter the number of tasks needed 3

 Enter Execution Time for task 1
3

 Enter Period for task 1
12

 Enter Execution Time for task 2
3

 Enter Period for task 2
12

 Enter Execution Time for task 3
8

 Enter Period for task 3
16
EDF: 1.000000 <= 1
```

```
EDF: 1.000000 <= 1
Exact Analysis: WC=0
Exact Analysis WC= 14
Exact Analysis WC= 20
Exact Analysis: WC=20
>Pitask set is schedulable by edf
 LCM = 48.000000
EDF Schedule:::::::
Task 0 Started from 0 till time 3
Task 1 Started from 3 till time 6
Task 2 Started from 6 till time 14
Task 0 Started from 14 till time 17
Task 1 Started from 17 till time 20
Task 2 Started from 20 till time 28
Task 0 Started from 28 till time 31
Task 1 Started from 31 till time 34
Task 2 Started from 34 till time 42
Task 0 Started from 42 till time 45
Task 1 Started from 45 till time 48
--------------------------------
task is not schedulable by rms


 Task Set:
  Task 1  (3,12)  Task 2  (3,12)  Task 3  (8,16)
```

Figure 10:Test Case Example

Figure 10 shows one test case, where it attempts the exact analysis and basically shows why it cannot be scheduled as the WC for the lowest priority task exceeds its Pi and therefore means can't be scheduled using RMS.

```
EDF: 0.625000 <= 1
RMS: 0.625000 <= 0.779763
task set is schedulable by edf
 LCM = 24.000000
EDF Schedule:::::::
Task 0 Started from 0 till time 2
Task 1 Started from 2 till time 3
Task 2 Started from 3 till time 7
Task 0 Started from 7 till time 9
Task 1 Started from 9 till time 10
Task 2 Started from 10 till time 10 , actually got premptied
Task 1 Started from 10 till time 10 , actually got premptied
Task 0 Started from 10 till time 12
Task 1 Started from 12 till time 12 , actually got premptied
Task 0 Started from 12 till time 14
Task 1 Started from 14 till time 15
Task 2 Started from 15 till time 15 , actually got premptied
Task 1 Started from 15 till time 15 , actually got premptied
Task 0 Started from 15 till time 17
Task 1 Started from 17 till time 17 , actually got premptied
Task 0 Started from 17 till time 19
Task 1 Started from 19 till time 20
Task 2 Started from 20 till time 20 , actually got premptied
Task 0 Started from 20 till time 22
Task 1 Started from 22 till time 22 , actually got premptied
Task 2 Started from 22 till time 24 , actually got premptied
--------------------------------
task set is schedulable by rms
 LCM = 24.000000
RMS Schedule:::::::
Task 0 Started from 0 till time 2
Task 1 Started from 2 till time 3
Task 2 Started from 3 till time 6 , actually got premptied
Task 0 Started from 6 till time 8
Task 1 Started from 8 till time 9
Task 2 Started from 9 till time 10
Task 0 Started from 12 till time 14
Task 1 Started from 16 till time 17
Task 0 Started from 18 till time 20
--------------------------------
```

Figure 11:Test Case 2 example

Figure 11 shows another example we use for a task set that should be able to be scheduled using both schedules and exact analysis is not needed here. We don't show the part of inputting data here to save space.

5. Evaluation

Testing our schedulers, we relied on our HW1 problem solutions as well as one task set from Exam 1, Our Evaluations are done by running our tool on Windows and inputting the given Task sets:

1. {T1,T2,T3} = {(1, 8), (2, 6), (4, 24)}.
2. {T1,T2,T3} = {(3, 12), (3, 12), (8, 16)}.
3. {T1,T2,T3} = {(2, 8), (3, 12), (4, 16)}.

(Ci,Pi)

Exam 1; Question A2: Task set

4. {T1,T2,T3}={(4, 12), (2, 6), (3, 9)}.

Tasks Sets made by us for testing:

5. {T1,T2,T3}={(1, 2), (1, 4), (1, 4)}.
6. {T1,T2,T3}={(2, 6), (1, 4), (1, 4)}.
7. {T1,T2}={(3, 8), (3, 9)}.
8. {T1,T2}={(4, 10), (3, 7)}.
9. {T1,T2}={(5, 10), (2, 10)}.
10. {T1,T2}={(11, 20), (1, 4)}.
11. {T1,T2}={(7, 10), (1, 4)}.
12. {T1,T2}={(9, 10), (1, 10)}.
13. {T1,T2,T3}={(1, 10), (2, 10), (3, 10)}.
14. {T1,T2,T3}={(10, 30), (1, 6), (30, 60)}.
15. {T1,T2,T3}={(5, 20), (6, 30), (2, 10)}.
16. {T1,T2,T3}={(5, 20), (2, 30), (3, 10)}.

Our Performance Metric of Both schedulers relied on the following values:

- Number of preemptions of task set per schedule
- Ability to schedule (RMS, EDF).
- Finish Time of whole Task set up to LCM.

In our testing we noticed that only 2 of the tasks out of 16 were not schedulable by RMS giving us a percentage of 87.5% of the ability to schedule using both schedulers. These sets are shown in figure 12

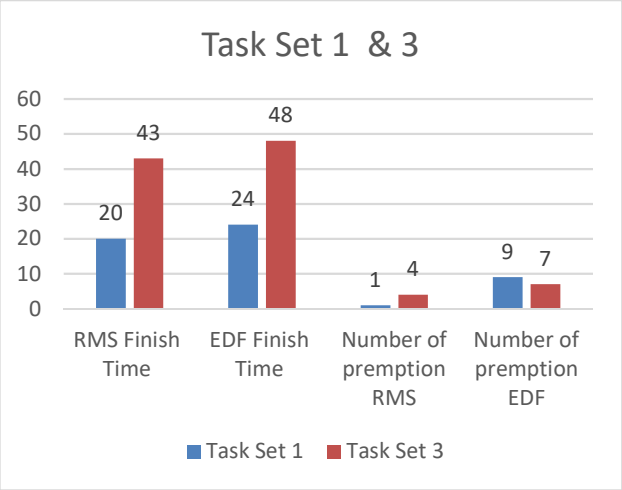

Figure 12:Only Tasks not schedulable by RMS.

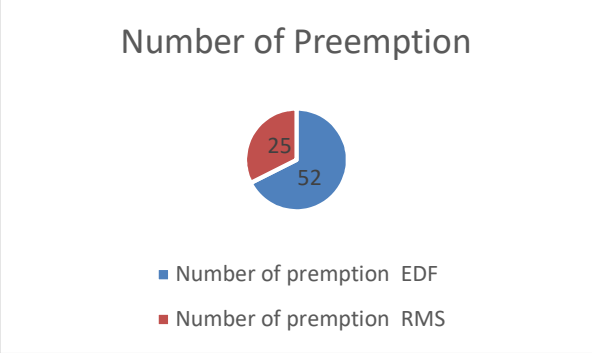

Figure 13:All Tasks Preemption

If you look at our figure 13, in general we can see that our EDF algorithm tends to preempt often which will increase overhead when scheduling Tasks. Our tool almost took no time to form these schedules which is why time of forming the schedule is the same for both schedulers. The percentage of preemption of EDF was higher by about 35% percent than RMS. I believe this could mean that our EDF could be much more Improved.

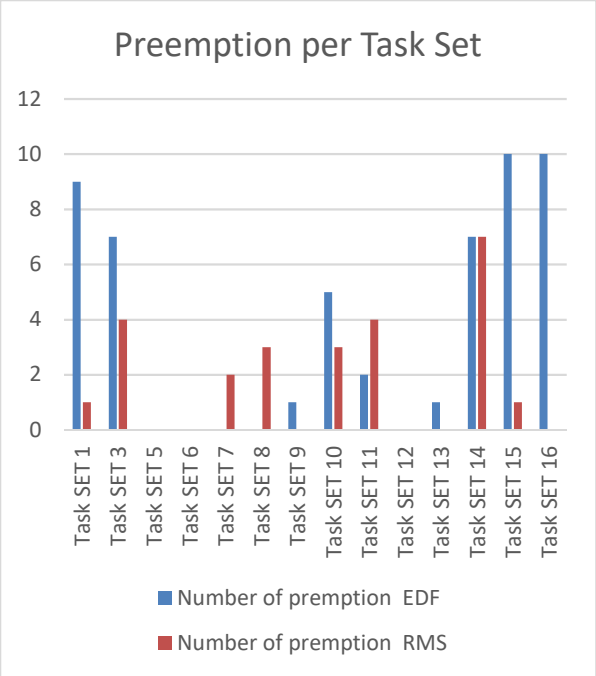

Figure 14:Number of Preemption per Task Set

Figure 14 simply shows the number of times our scheduler preempted during each Task set that we tested that worked both schedulers.

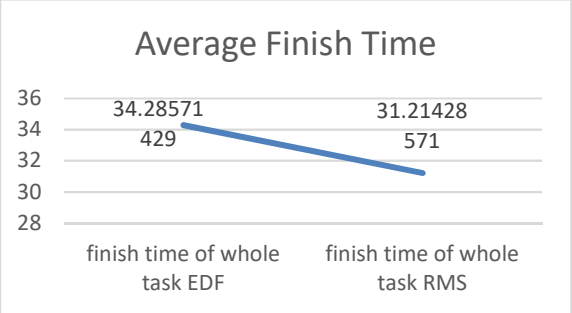

Figure 15:Average finish time

According to our figure 15, we can also see that the average finish time of the EDF schedules is about 9% percent which is to be expected as the way that our EDF implementation is, it should stop only when time reaches the LCM while RMS can actually stop if the number of instances of tasks up to the period of the LCM are done before the LCM.

The Screenshots below in figure 16 taken from our program, show the process of scheduling task set 1 using EDF and RMS, it interesting to see that the reason that preemption rate of our EDF algorithm is high is because it seems like, our EDF implementation get confused on what Task to schedules often but it still correctly follows the dynamic deadline of each task while ignoring Period of task sometimes.

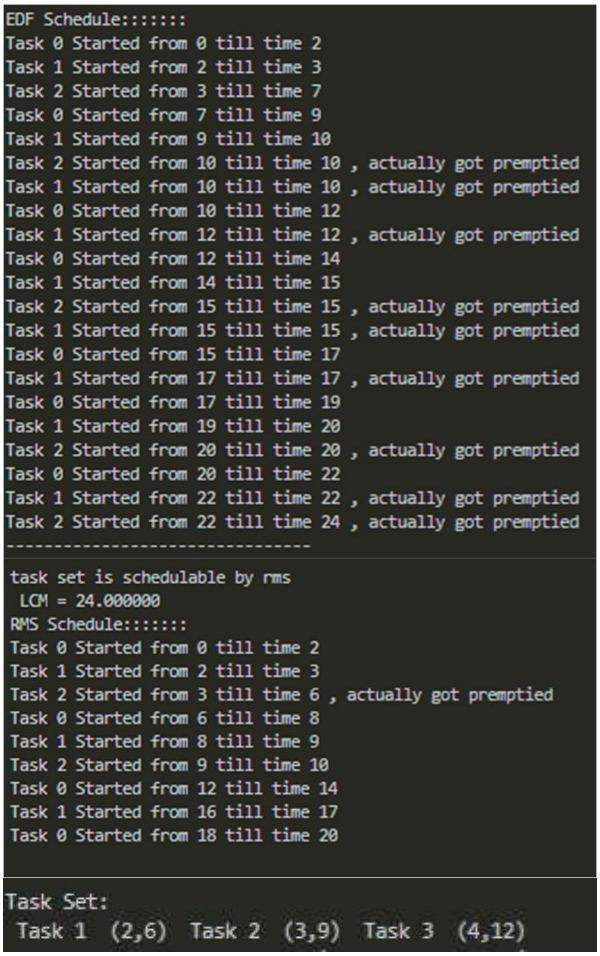

Figure 16:Program Screenshot EDF & RMS Example

6. Conclusions

Learning from this Experiment we realize that RMS is hard to implement and while according to our reference the famous Judgment Day Paper that compares EDF and RMS [4] , EDF should have less preemptions but because I think from our observation, Our EDF implementation might not have been the most efficient as in figure 16 we can see that it tried to schedule a certain tasks multiple times when it should have not and it realized that and preempted them.

From our experiment we have also realized why, it is may be harder to implement RMS as there are gaps in the schedule sometimes and that lead our first attempt of making the scheduler fail. Because the Scheduler would get confused and start scheduling even though there is nothing to schedule at the moment.

We also learned that EDF have a lot of room to be flexibly implemented by that I mean that it does not have a strict rule on preemption as a fixed priority like

RMS so you can make an EDF Implementation that is less than efficient that is should be in terms of preemption. While RMS has a strict implementation scheme.

Our Advice to those who attempt to do such an experiment is to pay special attention to implementing EDF scheduler in terms of number of task preemption for EDF to reach the standard provided by the judgment paper [4].

7. References

[1]Deter, Morgan, et al. *Rate-Monotonic Analysis in the C Type System* . https://www.cse.wustl.edu/~cdgill/RTAS03/published/workshop.pdf.

[2]"Rate Monotonic Scheduling." *Rate Monotonic Scheduling - an Overview | ScienceDirect Topics*, https://www.sciencedirect.com/topics/computer-science/rate-monotonic-scheduling.

[3]"Program for Rate Monotonic Scheduling Algorithm." *C*, http://c-programmingguide.blogspot.com/2012/09/c-program-for-rate-monotonic-scheduling.html.

[4] *Rate Monotonic vs. EDF: Judgment Day*. 26 May 2005, http://retis.sssup.it/~giorgio/paps/2005/rtsj05-rmedf.pdf.

[5]Lecture Notes, Exam and Homework Material