Uniprocessor Scheduling Simulator

GUI simulator

Kyle Marek & Griffin Stout
CPR E 458/558 - Section 01
Fall 2020
Department of Electrical and Computer Engineering
Iowa State University

Abstract

The purpose of this Uniprocessor Scheduling Simulator is to implement the RMS, EDF, DMS, and LLF algorithms and display them in an easy to read format. In addition, the goal is to better our understanding not only of the differences of the algorithms but also the situations in which each algorithm works best. This project addresses multiple learning outcomes stated in the syllabus for CPR E 458/558 Real-Time Systems as well as being a coding experience for processor scheduling.

This project is intended to further improve our knowledge of single processor scheduling algorithms that were taught in this course. The main problem is taking the high-level description of each algorithm taught in this course and being able to extend that information into working implementations of each scheduling algorithm. We also were required to determine whether the task set given was able to be scheduled by using schedulability checks and exact analysis. One last addition was to include the number of preemptions for each scheduling algorithm based on the task set.

Using our project, we were able to evaluate not only the performance of each scheduling algorithm based on the number of preemptions, but also the situations where certain algorithms fail to schedule and others succeed.

Based on these evaluations we determined that because the RMS and DMS algorithms priorities are calculated before run time and static that they are less likely to be able to schedule any given task set. However, because EDF and LLF are dynamically assigned every time unit, they generally cause more preemptions, and therefore a longer run time due to the overhead of switching tasks.

1. Introduction

For this project we studied the RMS, EDF, DMS, and LLF scheduling algorithms. Since these algorithms are widely known, our goal was not to perform research about them per say, but to implement them in a way that would deepen our understanding of how they work, as well as display the results in a manner that is easy to understand. We also aimed at producing output that would resolve the problem of determining which algorithm is best suited for a specific task set. A simulator is far quicker and more accurate, when implemented correctly, than attempting to calculate these schedules by hand.

2. Problem Formulation

The underlying problem and motivation behind this research project is to compare the RMS, DMS, EDF, and LLF algorithms and come to a decisive conclusion on whether or not each one is schedulable. This problem is worth studying because processor efficiency is one of the biggest issues limiting the front edge of development on real time systems. While these algorithms are specifically for single processor designs, it is important for young professionals entering the industry to understand the inner workings of processor scheduling.

2.1 System Model

In the real world, knowing which algorithms to use is important to get the best performance out of a processor. It doesn't matter how advanced your architecture is or how fast you can clock your processor if the operating system can't schedule tasks efficiently. The tool that we have created would be useful for learning purposes as well as in the industry where a quick analysis of these algorithms for a specific task set would be needed.

2.2 Problem Statement

The problem that we solved with this simulation is determining which scheduling algorithms (RMS, DMS, EDF, or LLF) pass for a specific task set. The algorithms can also be ranked based on the number of preemptions, where a larger number of preemptions would mean more context switching.

2.3 Objectives and Scope

The objective of this project is to take what we learned in class and apply it in a manner that solves our problem. We wanted to accomplish this with the primary goal of producing accurate results, and the secondary goal of ease of use. To do this, we produced a robust set of algorithms that mock the RMS, DMS, EDF, and LLF scheduling algorithms. To test our algorithms, we used examples from class that had defined solutions, and we also generated our own test cases that showcase the difference between the algorithms. To meet our ease of use goal, we did not want to make a terminal based application, instead opting to create a website where the user can easily decide how many tasks should be in the task set, and easily input three variables for each task. We used material design and google charts to show the schedules side by side where the differences can easily be seen.

3. Methodology

Our solution for programming the four algorithms stated in the Abstract was to take the information provided in class and extrapolate it into separate JavaScript functions for each algorithm. We also made an algorithm for each schedulability check (EDF and LLF) and exact analysis (RMS and DMS). These functions were used to evaluate the differences of each algorithm and will be described in further detail in this section.

3.1 Algorithm / Protocol

First, we will summarize the basic details and functionalities that we used in each algorithm's function. Then, the differences of each algorithm will be explained. Finally, the algorithms for schedulability checks and exact analysis will be discussed.

Each algorithm's function takes in an array of tasks as well as the amount of time units the user wants the algorithm to run. Each task has a computation time, period, and deadline as well as a copy for each to be used when calculating the task's next iteration. The main loop of all of the functions executes at each time unit.

First within the loop, the task that will run at that time unit is determined based on which algorithm is running as well as if that task has computation time left to run. For RMS, the task with the smallest period at that given time unit is selected to be run for one time unit. Using DMS, the task with the smallest deadline is run for one time unit. In EDF, the task with the smallest dynamic deadline is run for one time unit. The dynamic deadline is calculated by taking the tasks' starting period and adding the deadline to it. For LLF, the task with the smallest laxity is run for one time unit. Laxity is the difference between the tasks' dynamic deadline and its remaining computation time. If no task has computation time remaining then none of the tasks are selected.

Next, we calculated the preemptions of the algorithm by determining if the task that was selected to be executed was different from the task run at the last time unit. If the task run at the previous time unit was not finished computing, then a preemption is added to the total. After, the selected task is added to an array of tasks to be output at the end of the algorithm. The array contains the task number that runs at each time unit or a -1 if no task is run at that time unit. Then one computation time is taken from the selected task.

At the end of the main loop, each task is checked to see if it failed to execute its computation before its deadline. If that occurs, then the algorithm failed and that is passed back to the user interface. Finally, each task is checked to see if its end of period is at the current time unit. If it is, then the new period, deadline, and computation time is calculated based on the copies of the original task information. Lastly, the array of tasks in order of being run is returned along with the number of preemptions and whether the algorithm succeeded in its execution or not.

The functions for the schedulability of EDF and LLF are the same and are both necessary and sufficient, meaning that the condition has to pass in order for the task set to be schedulable and that if the condition passes it is known that the task set will be able to be scheduled by the algorithm [2]. The function is shown below

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

where c_i is the computation of task i and p_i is the period of task i [4].

The exact analysis of RMS and DMS also share the same algorithm. It is basic on the Critical Zone Theorem which states that if a task meets its deadline while all other higher priority tasks are also ready, then it will meet all of its future deadlines. This is specifically for independent periodic task sets. The functions are shown below

$$W_i(t) = \sum_{j=1}^{i} c_j(ceiling\left[\frac{t}{p_j}\right])$$

$$t_0 = \sum_{j=1}^{i} c_j$$

$$t_k = W_i(t_{k-1})$$

$$Stop\ when\ W_i(t_k) = t_k$$

and each task T_i is schedulable if $W_i \leq d_i$ for EDF and if $W_i \leq p_i$ for RMS [3].

Each time before the function for RMS, DMS, EDF, or LLF is run, its respective schedulability check or exact analysis is also run to determine whether the algorithm will pass or fail given a task set.

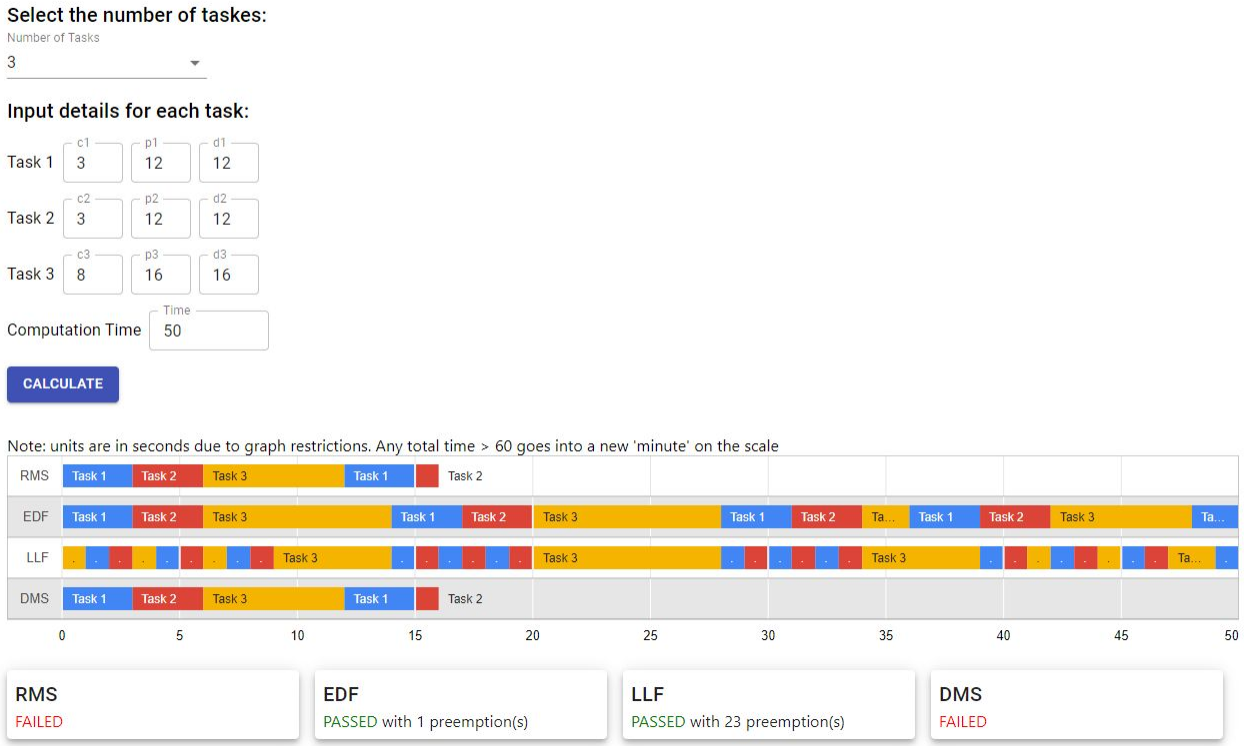
The functions testing with real task set examples will be shown in the following section.

3.2 Illustrative Example

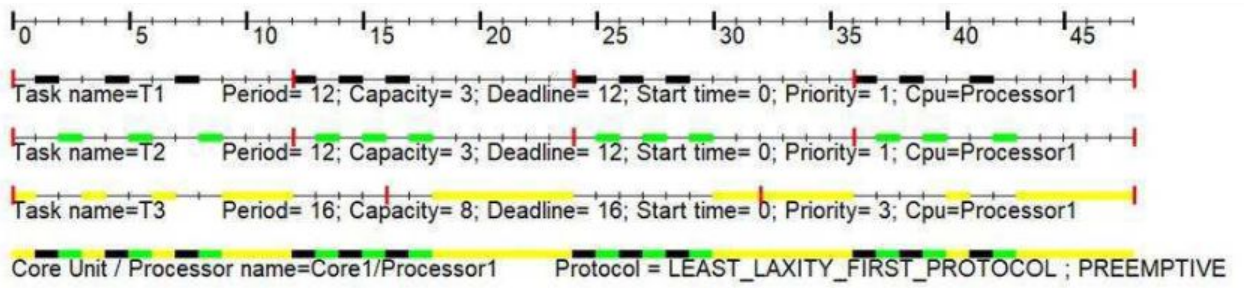
The first example isn't schedulable by RMS and DMS but can be scheduled by EDF and LLF. The second example isn't schedulable by RMS but is by DMS, EDF, and LLF; and the third example is schedulable by all algorithms.

Algorithm Scheduling

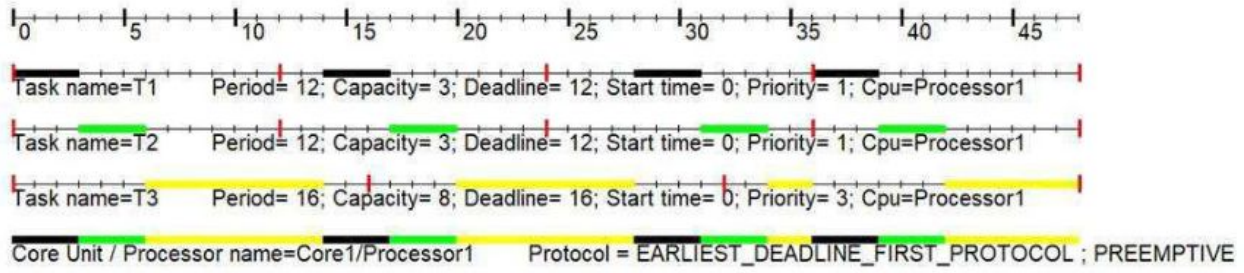
By: Kyle Marek and Griffin Stout



LLF [1]



EDF [1]



Algorithm Scheduling

By: Kyle Marek and Griffin Stout

Select the number of taskes:

Number of Tasks

2

Input details for each task:

	c	p	d
Task 1	1	5	4
Task 2	3	6	3

Computation Time: 12

CALCULATE

Note: units are in seconds due to graph restrictions. Any total time > 60 goes into a new 'minute' on the scale

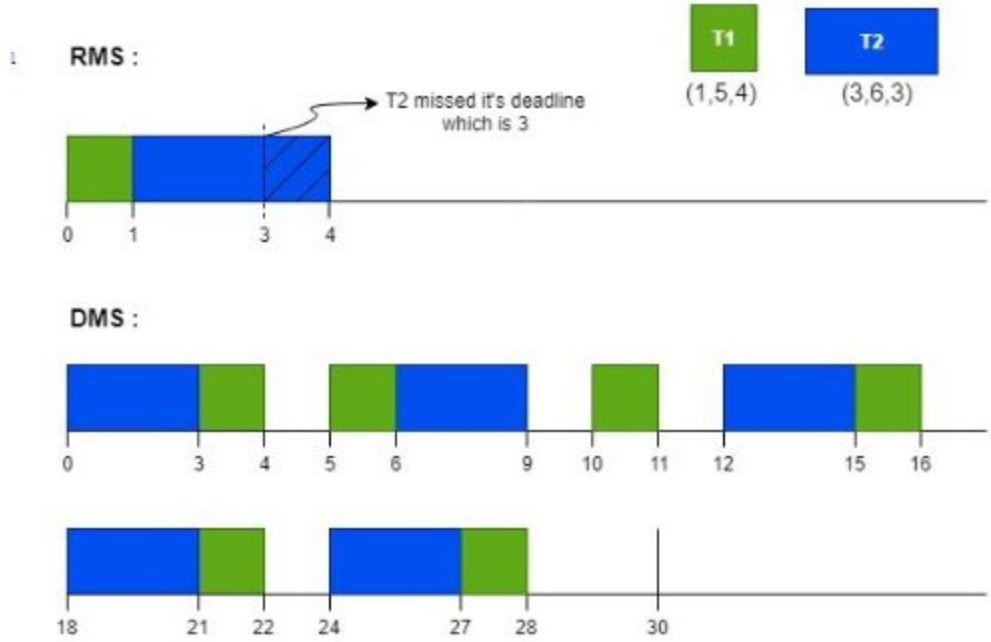
RMS
FAILED

EDF
PASSED with 0 preemption(s)

LLF
PASSED with 0 preemption(s)

DMS
with 0 preemption(s)

RMS and DMS [1]



From time 3 to time 4, T2 missed its period. Thus it is not schedulable on RMS (No Good).

There is not a problem for DMS scheduling (Good).

Algorithm Scheduling

By: Kyle Marek and Griffin Stout

Select the number of taskes:

Number of Tasks

5

Input details for each task:

	c	p	d
Task 1	1	5	5
Task 2	1	4	4
Task 3	1	10	10
Task 4	1	8	8
Task 5	2	10	10

Computation Time: Time 40

CALCULATE

Note: units are in seconds due to graph restrictions. Any total time > 60 goes into a new 'minute' on the scale

RMS
PASSED with 2 preemption(s)

EDF
PASSED with 2 preemption(s)

LLF
PASSED with 4 preemption(s)

DMS
with 2 preemption(s)

4. Implementation/Simulation Architecture

To develop the software for this project, we used the following tools, frameworks, languages, and libraries. All of the code was written in the Visual Studio Code editor. We chose this because it is free, fast, and has many extensions that can make it more powerful. We decided to use the ReactJS framework for our website because of its reusable components, scalability, and efficiency. Normally a language like C would be used to write algorithms like this because it is much faster, but Javascript was used in this case because it is the native language of the React framework. We did not see any meaningful processing delay with reasonable task sets and computation time inputs. Material-ui, a user interface library, and Google charts, a graph generation library, were both used to help make the GUI better.

5. Evaluation

To make sure that our algorithms were correct (our primary goal), we used multiple examples from the class that had the solutions published. You can see some of these examples in the section 3 screenshots. We also made our own test cases with the goal of testing edge cases for our own code as well as the scheduling algorithms themselves. Our algorithms passed every test. Our secondary goal of making the end product easy to use is slightly harder to quantify and evaluate because it is subjective, but we believe that we have exceeded expectations in this matter. The GUI that we created is simple, yet effective, and our usage of google charts to show the schedules is the best way to display this type of information.

6. Conclusions

In conclusion, we have fully met both our primary and secondary goals for this project. The RMS, DMS, EDF, and LLF scheduling algorithms can be compared for a specific task set, and with our easy to use interface, a conclusion about which algorithm is best can easily be drawn. We found that the static algorithms, RMS and DMS, were able to schedule less task sets, but when they were able to, there were less preemptions. EDF and LLF, the dynamic algorithms, were able to schedule more total task sets, with the side effect of causing many preemptions. LLF was the worst offender in this regard.

Individual contributions by team members can be divided as follows. Kyle Marek wrote the algorithms that took a task set as input, and output a correct schedule, pass/fail result, and number of preemptions. Griffin Stout created the user interface for input of task sets and output of schedules. Both team members worked together to test the software.

We came to understand the nuances of these algorithms much better than we had before. We would recommend implementing these scheduling algorithms to anyone that wants a deeper understanding of why they produce the results that they do. In the future, experiments like this should include a more dynamic input set, and implement other scheduling algorithms to compare with.

Project Learning Objectives	**Status**	**Location in Report**
Self contained description of the project goal, scope, and relevant requirements	Fully completed	Section 2
Self contained	Fully	Section 4,

description of the solutions	completed	Pages
Adequate description of the implementation details	Fully Completed	Section 3.1
Testing and evaluation	Fully Completed	Section 3.2, Section 5
Overall project success assessment	100% successful	n/a

7. References

[1] Govindarasu, Manimaran. "Homework 1 Solutions". 2020.

[2] Govindarasu, Manimaran. "Real-Time Task Scheduling, Part 1". 2020.

[3] Govindarasu, Manimaran. "Real-Time Task Scheduling, Part 2". 2020.

[4] Yi, Wang. "Scheduling Periodic Tasks". 2010.