Simarpreet Singh
s244sing@uwaterloo.ca | 20434209

ECE 455: Lab 4
EDF Scheduler

Objective: In this lab we were asked to implement an EDF scheduler that handles the tasks in an Earliest Deadline first manner.

The scheduler works as follows:

1) The scheduler picks up the next process with the earliest deadline first and then picks up the ones that have a later deadline later.
2) In the case of similar deadlines, the scheduler picks up the next process with its task ID. In our case we have this condition set to: Task 1 > Task 2 > Task 3 (in terms of priority)
3)

In my implementation of the EDF scheduler I dynamically compute the deadlines for each process depending on how long they have run for. Then after each run of each task, the deadlines are recomputed again so that the next order can be calculated. This is done using the following logic:

```
/*

        Type 1 Conditions (Based on deadline order):

        1 > 2 > 3

        2 > 1 > 3

        3 > 2 < 1

*/
```

In the case of Type 1 conditions, all conditions have a strictly greater inequality and have 3 possible combinations.

```
/*

        Type 2 conditions (Based on deadline order):

        2 > 3 > 1

        1 > 3 > 2

        3 > 1 > 2

*/
```

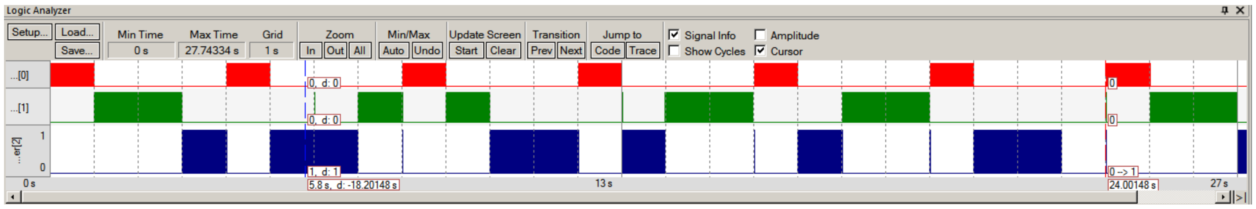
In the case of Type 2 conditions, all conditions have a strictly greater inequality and have 3 possible combinations. But are now in out of order.

Simarpreet Singh
s244sing@uwaterloo.ca | 20434209

```
/*
        Type 3 conditions (equality conditions) (Based on deadline order):
        1 = 2 , 1 > 3
        1 = 2 , 1 < 3
        1 = 3 , 1 > 2
        1 = 3 , 1 < 2
        2 = 3 , 2 > 1
        2 = 3 , 2 > 1
        1 = 2 = 3
*/
```

In the case of Type 3 conditions, the first check is done on equality of two conditions then the next two are evaluated as nested conditions.

Screenshot obtained:



Difficulties faced:

1) Coming with up all the possible cases that can happen when you have three tasks running was hard and I had to think of all the possible situations and cases that the code might take.
2) I initially solved the problem by hand as described by the TA and then later proceeded to code it.
3) I initially didn't take into account that I had to recalculate priorities both times, before and after running the task for the required amount of time. This resulted in some tasks missing their deadlines and as a result I was getting a wrong schedule.
4) I also while writing my initial implementation, missed the fact that in the case of no precedence case being taken, I had to set the priorities according the rule that T1 > T2 > T3 in terms of priority of running the tasks.
5) I also thought initially of using the timer API to design my implementation and get a period of seconds as close to real life as possible. But I ran into linker issues linking against the timers API and as a result had to change my approach.