Implementing and Testing EDF and Comparing Performance Against RMS and DMS

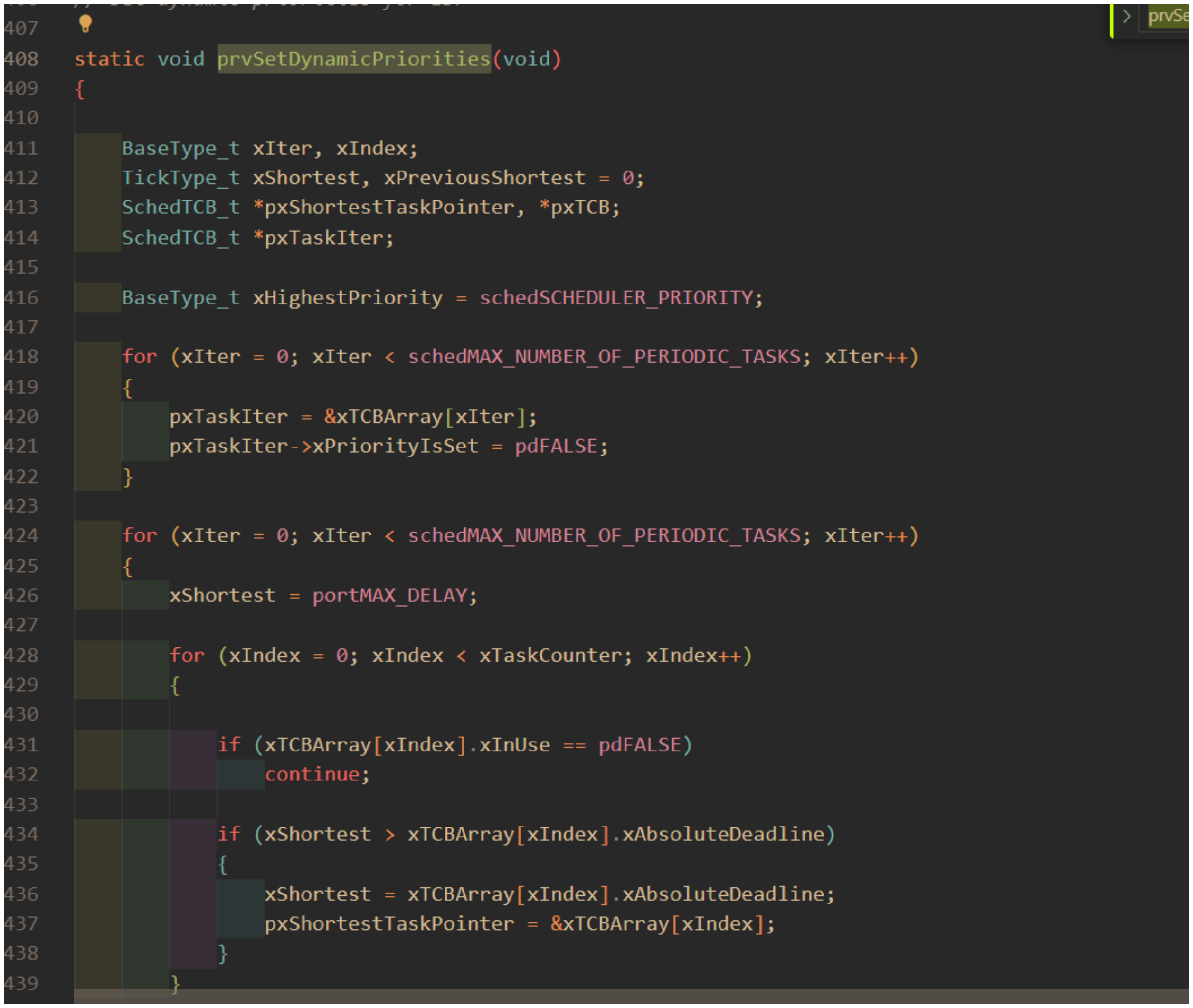
Promit Panja
Dept. of Electrical and Computer Engineering
Virginia Tech

Aim: The aim of this project was to implement EDF scheduling algorithm in FreeRTOS and test its performance on a task set on Arduino MEGA. Compare the performance of EDF against fixed priority algorithms like RMS and DMS.

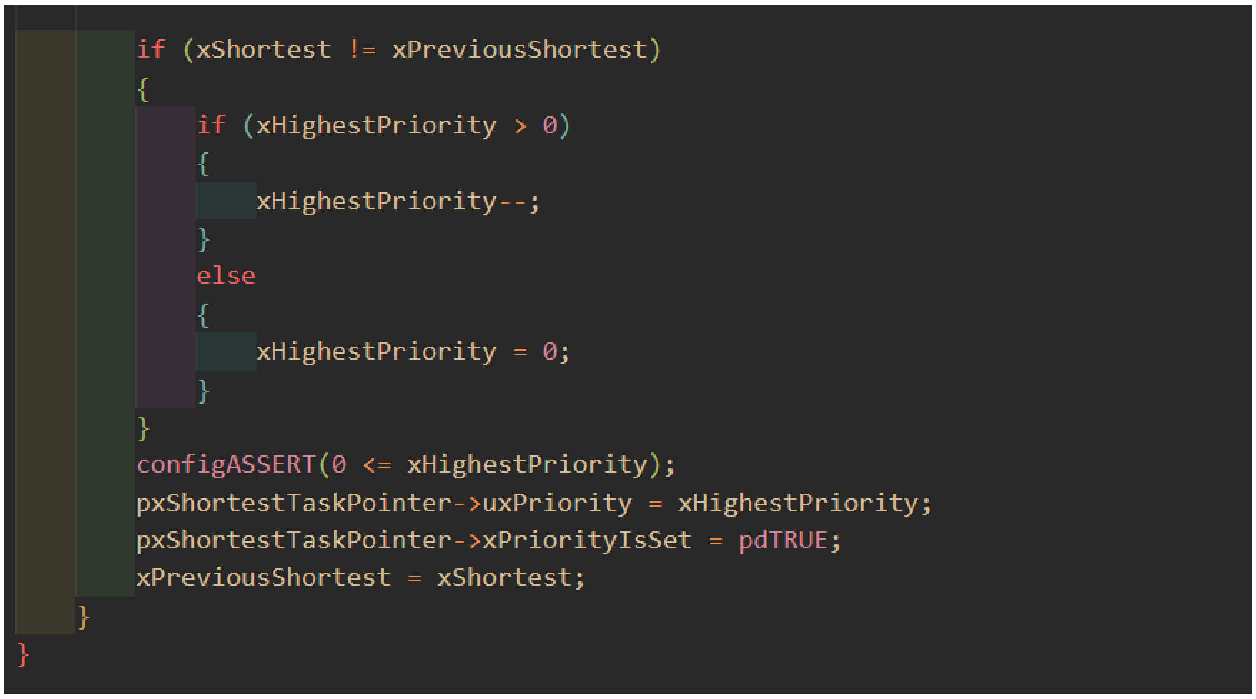
Background: Priority based scheduling is a method of scheduling tasks based on priorities of individual tasks which is decided by a scheduler based on multiple factors. Some of the common priority-based scheduling algorithms are Round-Robin, FIFO, etc. There are mainly two types of priority assignment algorithms— fixed priority scheduling and dynamic priority scheduling.
In fixed priority scheduling the priorities are set by the scheduler ahead of execution of the tasks. This type of scheduling algorithm is pre-emptive in nature and is robust. *Rate Monotonic (RM)* and *Deadline Monotonic Scheduling (DM)* are the most common types of fixed-priority scheduling algorithms. In dynamic priority scheduling the priorities of the tasks are assigned at run-time unlike fixed priority scheduling the priorities in these types of scheduling algorithms are updated on each task completion and release of a new task. *Earliest Deadline First (EDF)* is an example of dynamic priority scheduling algorithm.

Implementation:
For implementing the EDF scheduler the *scheduler.cpp* from project 3 was used as a base.
The following changes were made to the *scheduler.cpp* file to implement EDF scheduling algorithm.

1. The first change that was made over the original *scheduler.cpp* is to declare a new function called **prvSetDynamicPriorities()** which is a function based on the **prvSetFixedPriorities()** but instead of calculating priorities based on relative deadlines this function calculates the priorities based on **absolute deadlines** i.e. In the task set the task with the shortest absolute deadline is assigned the highest priority and the rest are assigned in ascending order of their absolute deadlines. Absolute deadline is chosen because the priorities need to be updated each time a task finishes execution and a new task is released in the next period with new updated absolute deadline.

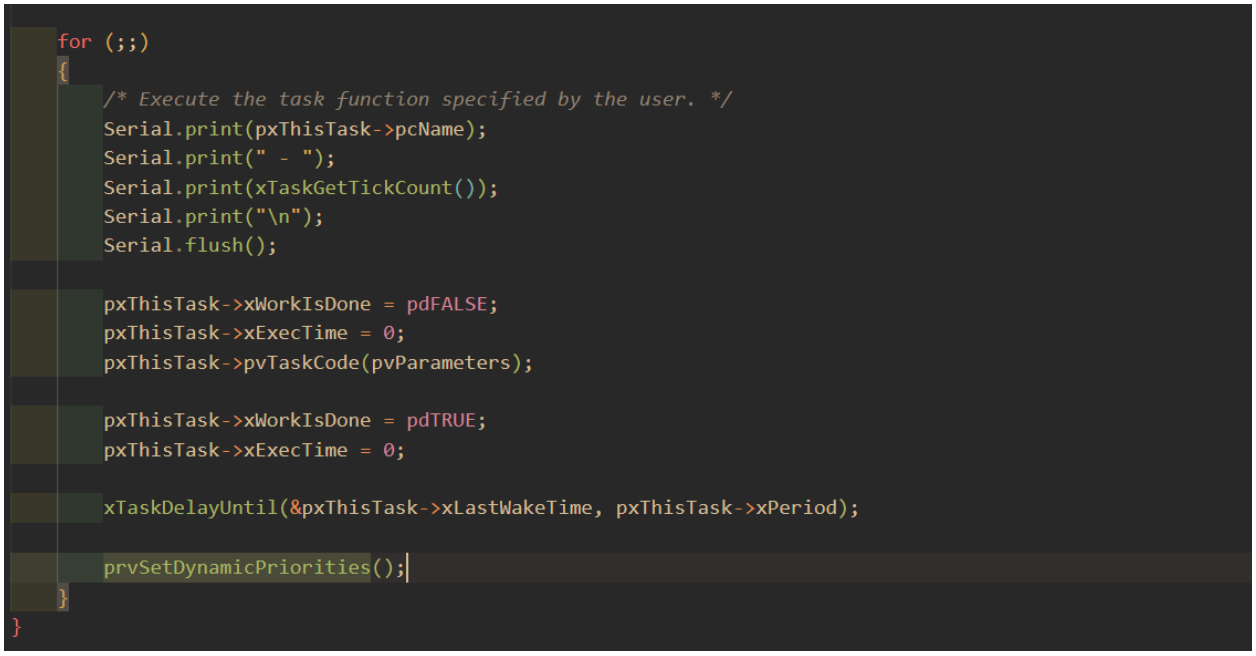


```c
static void prvSetDynamicPriorities(void)
{

    BaseType_t xIter, xIndex;
    TickType_t xShortest, xPreviousShortest = 0;
    SchedTCB_t *pxShortestTaskPointer, *pxTCB;
    SchedTCB_t *pxTaskIter;

    BaseType_t xHighestPriority = schedSCHEDULER_PRIORITY;

    for (xIter = 0; xIter < schedMAX_NUMBER_OF_PERIODIC_TASKS; xIter++)
    {
        pxTaskIter = &xTCBArray[xIter];
        pxTaskIter->xPriorityIsSet = pdFALSE;
    }

    for (xIter = 0; xIter < schedMAX_NUMBER_OF_PERIODIC_TASKS; xIter++)
    {
        xShortest = portMAX_DELAY;

        for (xIndex = 0; xIndex < xTaskCounter; xIndex++)
        {

            if (xTCBArray[xIndex].xInUse == pdFALSE)
                continue;

            if (xShortest > xTCBArray[xIndex].xAbsoluteDeadline)
            {
                xShortest = xTCBArray[xIndex].xAbsoluteDeadline;
                pxShortestTaskPointer = &xTCBArray[xIndex];
            }
        }
```



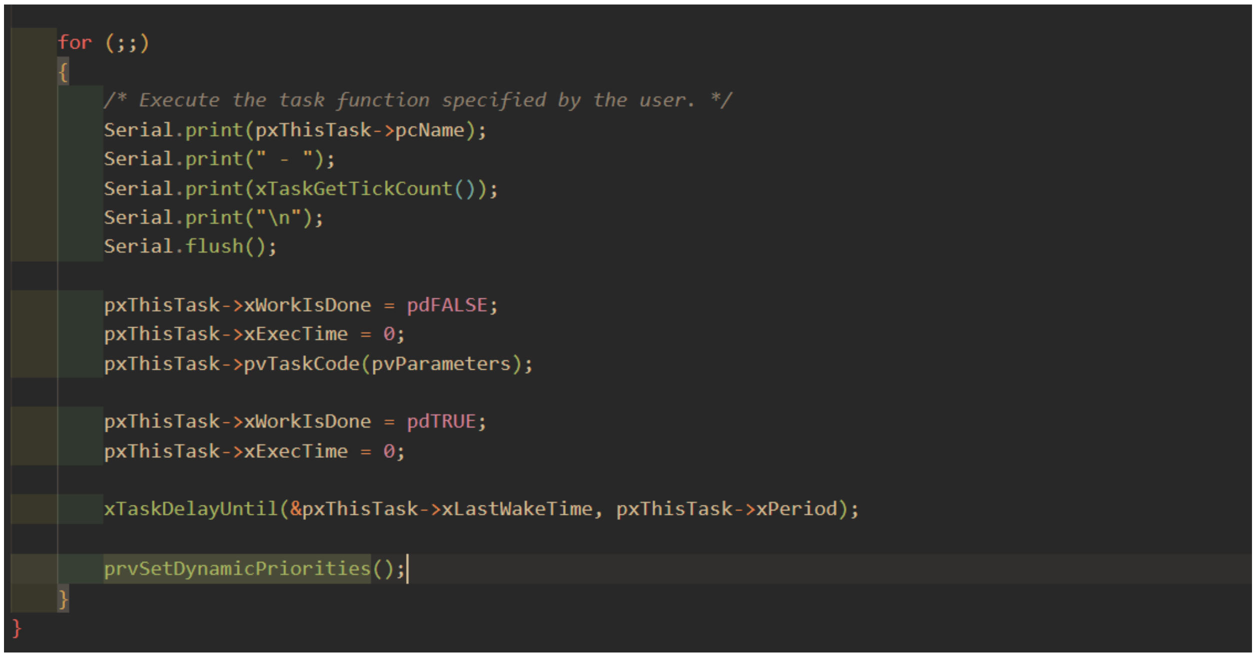
```c
        if (xShortest != xPreviousShortest)
        {
            if (xHighestPriority > 0)
            {
                xHighestPriority--;
            }
            else
            {
                xHighestPriority = 0;
            }
        }
        configASSERT(0 <= xHighestPriority);
        pxShortestTaskPointer->uxPriority = xHighestPriority;
        pxShortestTaskPointer->xPriorityIsSet = pdTRUE;
        xPreviousShortest = xShortest;
    }
}
```

2. prvSetDynamicPriorities() function is called in **prvPeriodicTaskCode()** such that new priorities are calculated and assigned after the execution of a task.



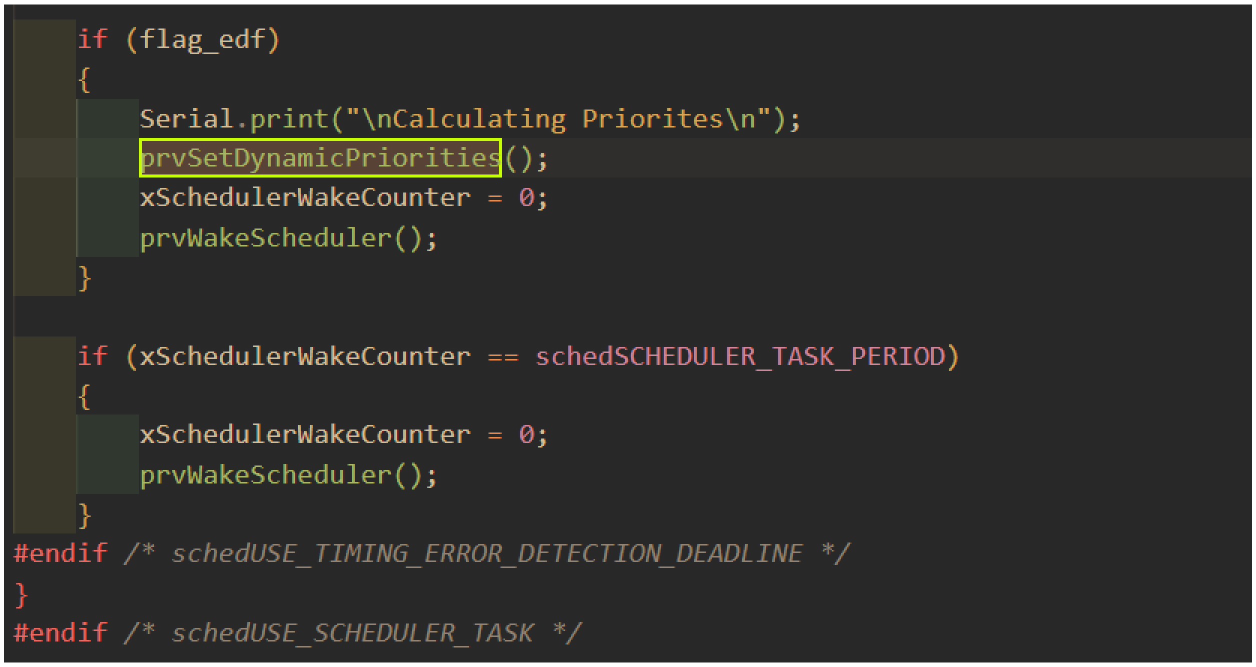
```
for (;;)
{
    /* Execute the task function specified by the user. */
    Serial.print(pxThisTask->pcName);
    Serial.print(" - ");
    Serial.print(xTaskGetTickCount());
    Serial.print("\n");
    Serial.flush();

    pxThisTask->xWorkIsDone = pdFALSE;
    pxThisTask->xExecTime = 0;
    pxThisTask->pvTaskCode(pvParameters);

    pxThisTask->xWorkIsDone = pdTRUE;
    pxThisTask->xExecTime = 0;

    xTaskDelayUntil(&pxThisTask->xLastWakeTime, pxThisTask->xPeriod);

    prvSetDynamicPriorities();
}
}
```

3. The **prvSetDynamicPriorities()** function is called in **prvDeadlineMissedHook()** such that new priorities are calculated and assigned whenever a task misses its deadline and is recreated with new updated absolute deadline.



```
for (;;)
{
    /* Execute the task function specified by the user. */
    Serial.print(pxThisTask->pcName);
    Serial.print(" - ");
    Serial.print(xTaskGetTickCount());
    Serial.print("\n");
    Serial.flush();

    pxThisTask->xWorkIsDone = pdFALSE;
    pxThisTask->xExecTime = 0;
    pxThisTask->pvTaskCode(pvParameters);

    pxThisTask->xWorkIsDone = pdTRUE;
    pxThisTask->xExecTime = 0;

    xTaskDelayUntil(&pxThisTask->xLastWakeTime, pxThisTask->xPeriod);

    prvSetDynamicPriorities();
}
}
```

4. Another place where the **prvSetDynamicPriorities()** is called is in **vApplicationTickHook()** as the scheduler needs to calculate and assign new priorities whenever the scheduler task is woken.



```
    if (flag_edf)
    {
        Serial.print("\nCalculating Priorites\n");
        prvSetDynamicPriorities();
        xSchedulerWakeCounter = 0;
        prvWakeScheduler();
    }

    if (xSchedulerWakeCounter == schedSCHEDULER_TASK_PERIOD)
    {
        xSchedulerWakeCounter = 0;
        prvWakeScheduler();
    }
#endif /* schedUSE_TIMING_ERROR_DETECTION_DEADLINE */
}
#endif /* schedUSE_SCHEDULER_TASK */
```

The implementation of EDF includes calculating the priorities based on absolute deadlines and then, calculating and assigning the priorities each time a task finishes execution and each time a task is released, which is achieved by calling the **prvSetDynamicPriorities()** function at appropriate places.

Experimentation and Results:

The following results were obtained when the following task set was run using EDF, RM, and DM.

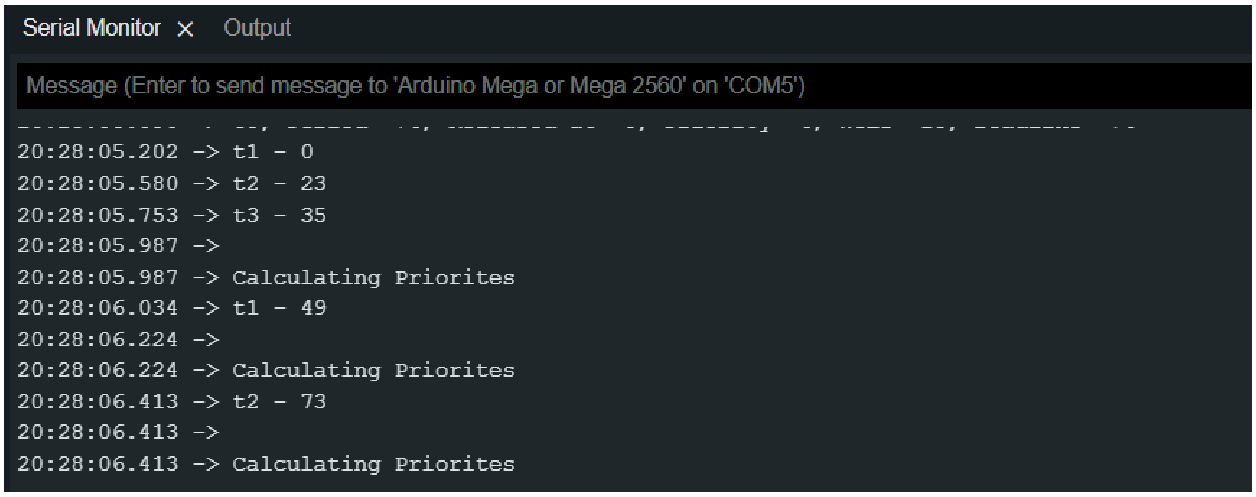
Task	C	T
t1	4	8
t2	2	10
t3	3	12

In order to simulate the above execution times a nested for loop like below was implemented:

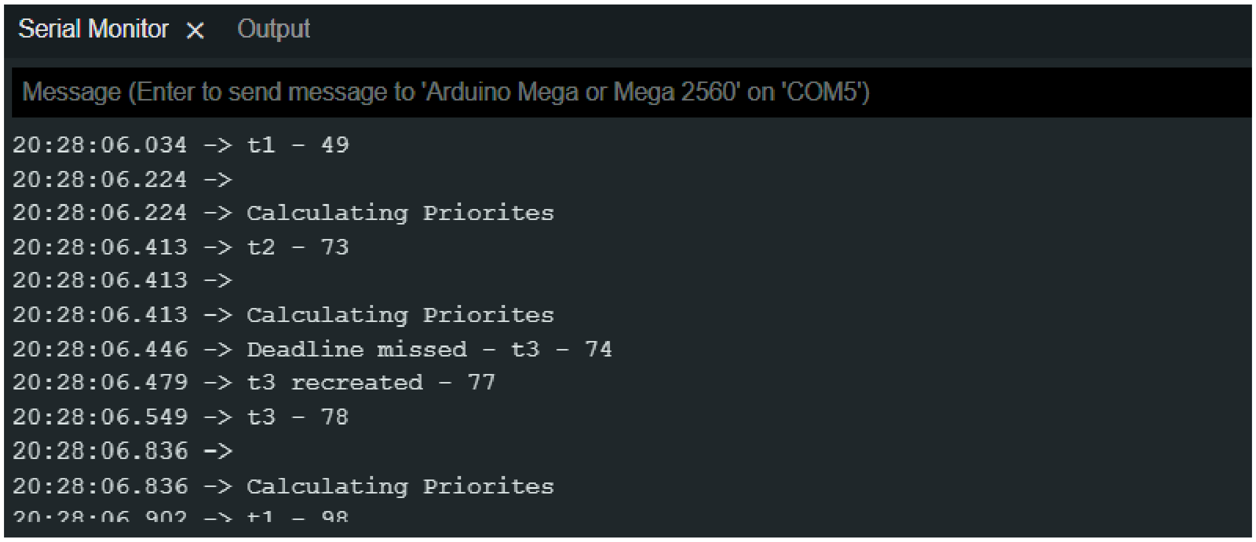
for(i = 0; i < 80; i++) { for(j = 0; j < 1000; j++) { }}

The above shown for loop takes about 100ms to execute on an Arduino MEGA, the outer loop iteration was multiplied accordingly to simulate each task's WCET.

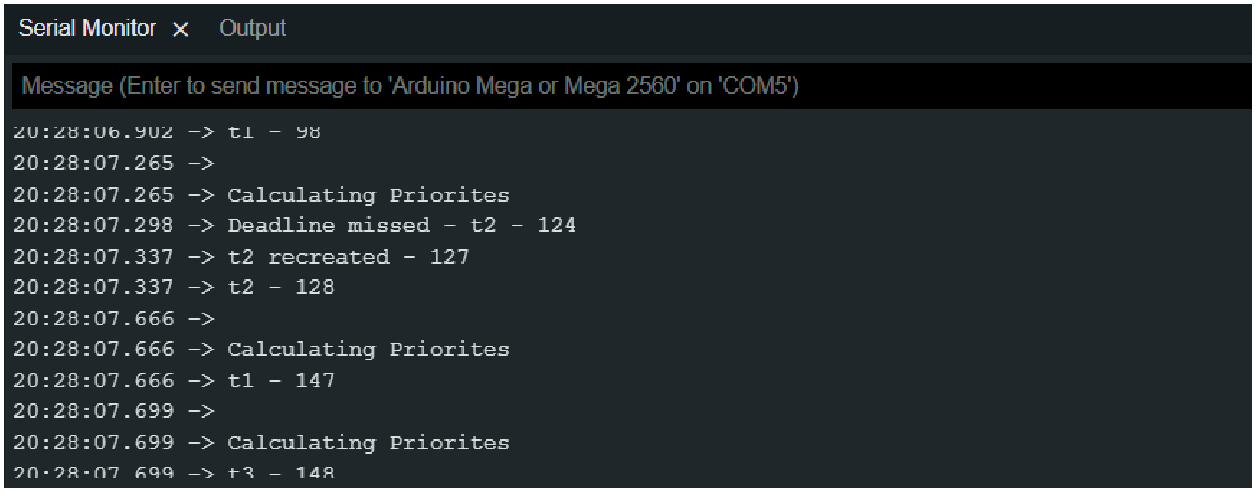
EDF:


Serial Monitor
Output
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM5')

```
[illegible]
20:28:05.202 -> t1 - 0
20:28:05.580 -> t2 - 23
20:28:05.753 -> t3 - 35
20:28:05.987 ->
20:28:05.987 -> Calculating Priorites
20:28:06.034 -> t1 - 49
20:28:06.224 ->
20:28:06.224 -> Calculating Priorites
20:28:06.413 -> t2 - 73
20:28:06.413 ->
20:28:06.413 -> Calculating Priorites
```


Serial Monitor
Output
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM5')

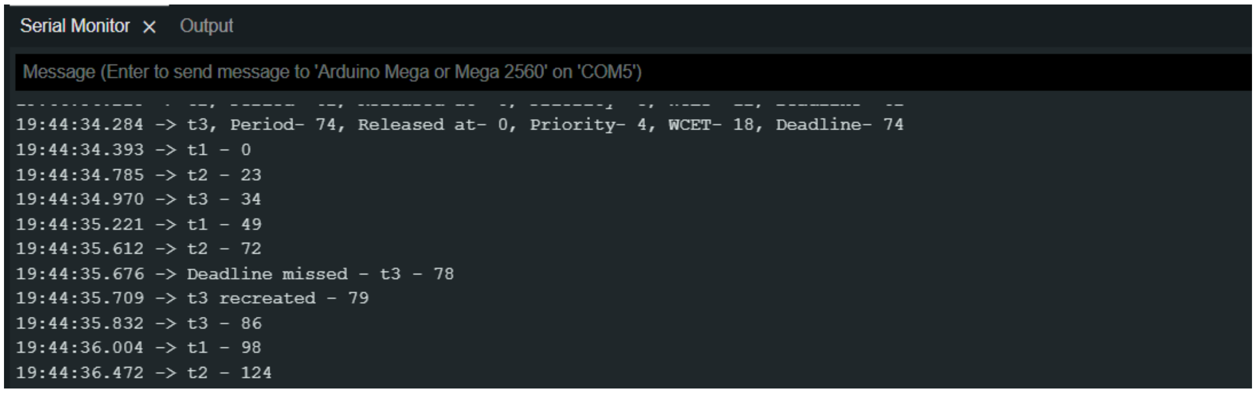
```
20:28:06.034 -> t1 - 49
20:28:06.224 ->
20:28:06.224 -> Calculating Priorites
20:28:06.413 -> t2 - 73
20:28:06.413 ->
20:28:06.413 -> Calculating Priorites
20:28:06.446 -> Deadline missed - t3 - 74
20:28:06.479 -> t3 recreated - 77
20:28:06.549 -> t3 - 78
20:28:06.836 ->
20:28:06.836 -> Calculating Priorites
20:28:06.902 -> t1 - 98
```


Serial Monitor
Output
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM5')

```
20:28:06.902 -> t1 - 98
20:28:07.265 ->
20:28:07.265 -> Calculating Priorites
20:28:07.298 -> Deadline missed - t2 - 124
20:28:07.337 -> t2 recreated - 127
20:28:07.337 -> t2 - 128
20:28:07.666 ->
20:28:07.666 -> Calculating Priorites
20:28:07.666 -> t1 - 147
20:28:07.699 ->
20:28:07.699 -> Calculating Priorites
20:28:07.699 -> t3 - 148
```

As we can see the above task set is schedulable under EDF, the tasks do not miss any deadlines at least in their first two periods, but after some time few tasks miss their deadlines and some exceed their WCET. This behavior is observed because of scheduler overheads, due to calculating priorities each time a task finishes its execution and each time a new task is released, it causes overheads and hence few tasks miss their deadlines.
From the above output of the serial monitor we can see new priorities are calculated and assigned after each task finishes its execution and each task is released, even after deadline misses whenever a deleted task is recreated new priorities are calculated and assigned based on new updated absolute deadline.

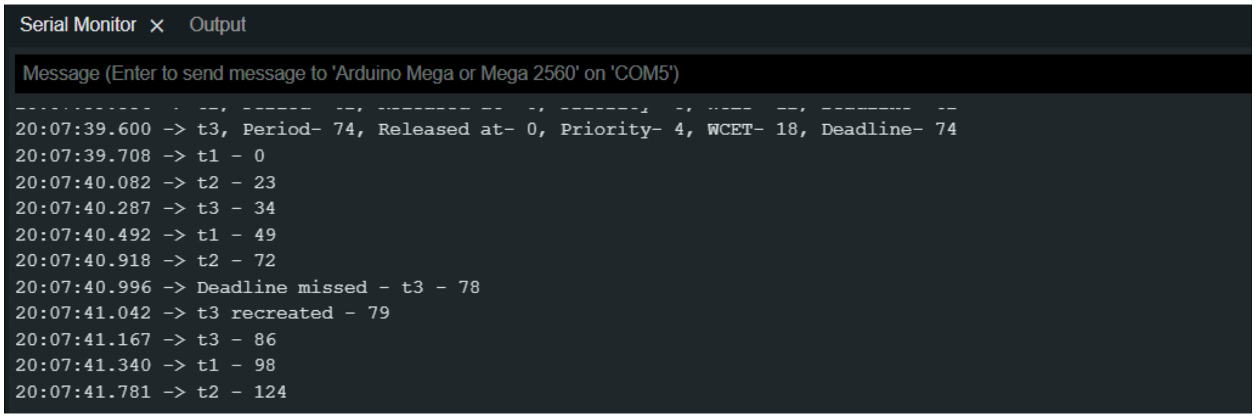
RMS:


Serial Monitor ×
Output
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM5')

```
19:44:34.284 -> t3, Period- 74, Released at- 0, Priority- 4, WCET- 18, Deadline- 74
19:44:34.393 -> t1 - 0
19:44:34.785 -> t2 - 23
19:44:34.970 -> t3 - 34
19:44:35.221 -> t1 - 49
19:44:35.612 -> t2 - 72
19:44:35.676 -> Deadline missed - t3 - 78
19:44:35.709 -> t3 recreated - 79
19:44:35.832 -> t3 - 86
19:44:36.004 -> t1 - 98
19:44:36.472 -> t2 - 124
```

As we can see the same task set is not schedulable under RMS, and hence second instance of t3 misses its deadline.

DMS:


Serial Monitor ×
Output
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM5')

```
20:07:39.600 -> t3, Period- 74, Released at- 0, Priority- 4, WCET- 18, Deadline- 74
20:07:39.708 -> t1 - 0
20:07:40.082 -> t2 - 23
20:07:40.287 -> t3 - 34
20:07:40.492 -> t1 - 49
20:07:40.918 -> t2 - 72
20:07:40.996 -> Deadline missed - t3 - 78
20:07:41.042 -> t3 recreated - 79
20:07:41.167 -> t3 - 86
20:07:41.340 -> t1 - 98
20:07:41.781 -> t2 - 124
```

From the above output we can see that like RMS second instance of t3 misses its deadline and hence the task set is not schedulable under DMS.

Conclusion: In this project we implemented EDF scheduling algorithm in FreeRTOS and tested its performance on a task set running on an Arduino MEGA and comparing the results with that of RMS and DMS. From this result we learned that the task set on which the algorithms were tested is schedulable only under EDF and is not schedulable under RMS and DMS. For future work the algorithm can be further improved to reduce the scheduler overhead of calculating and assigning dynamic priorities, such that the tasks can execute as close to the theoretical time instance as possible.