Advanced Real-Time Systems ECE 5550G

Final Project

Vishnu Kumar Kalidasan (kvishnukumar22)

Sruthi Ayaluru Venkata Krishnan (sruthiavk)

Report on EDF Scheduler

Introduction:

Scheduling is a critical aspect of real-time operating systems that aims to allocate system resources to different tasks in an efficient and fair manner. Earliest Deadline First (EDF), Rate-Monotonic (RM), and Deadline-Monotonic (DM) are the most widely used scheduling policies in real-time systems.

1. **EDF** prioritizes tasks based on their deadlines, assigning the highest priority to the task with the earliest deadline. This approach ensures that the tasks with the closest deadlines are executed first, minimizing the number of missed deadlines.
2. **RM** assigns higher priority to tasks with shorter periods. This approach is simpler than EDF and can be easily implemented, but it is not as flexible in handling a variety of real-time systems.
3. **DM** assigns higher priority to tasks with shorter deadlines, considering both the deadline and the period of the task. If two tasks have the same deadline, the task with the shorter period will be given a higher priority.

The choice of scheduling policy depends on the specific requirements of the real-time system being implemented. EDF, RM, and DM are all effective scheduling policies that have been widely used in various applications, including aerospace, industrial control, and multimedia systems.

Background:

Real-time operating systems are used in a wide range of applications, including industrial control, aerospace, and multimedia systems. These systems differ from conventional operating systems in that they have to respond to events within a guaranteed time frame. This means that tasks in a real-time system must be scheduled in a way that ensures they are completed within their deadlines.

Efficient scheduling policies are essential for the successful implementation of real-time systems. These policies aim to allocate system resources to tasks in a way that meets the system's performance requirements while maintaining fairness. Earliest Deadline First (EDF) is one of the most widely used scheduling policies in real-time systems. EDF assigns the highest priority to the task with the earliest deadline, ensuring that the tasks with the closest deadlines are executed first.

The utilization factor is an essential performance metric for real-time operating systems. It represents the percentage of total CPU time used by the tasks in the system. The utilization factor is particularly important because it determines whether a real-time system can meet its performance requirements or not.

When comparing scheduling policies like EDF, RM, and DM, the utilization factor plays a crucial role in determining which policy is more effective. In general, EDF has a higher utilization factor than RM and DM, making it a more efficient scheduling policy in many cases.

$$U \equiv \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{1/n} - 1)$$

Fig 1a Utilization-Based Analysis for RM and DM

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Fig 1bUtilization-Based Analysis for EDF

EDF has several advantages over other scheduling policies, such as Rate-Monotonic (RM) and Deadline-Monotonic (DM). Unlike RM, which assigns priority based on task period, EDF considers the deadline of each task, making it more effective in handling tasks with varying deadlines. DM, on the other hand, also considers task period but does not provide the same level of flexibility as EDF. The advantages and the limitations of EDF over RM and DM scheduling policies are mentioned below.

Advantages of EDF over RM and DM :

- Better Utilization of System resources :
  EDF scheduling policies do not assign fixed priorities based on their periodicity. EDF prioritizes tasks based on their deadlines, which means that tasks with later deadlines will have lower priority and may be preempted by tasks with earlier deadlines, thus maximizing the number of completed tasks within their respective deadlines.
- Better flexibility in handling Aperiodic tasks:
  RM and DM scheduling policies have limited flexibility in handling aperiodic tasks, as they are designed to handle only periodic tasks. In contrast, EDF scheduling policy can handle both periodic and aperiodic tasks.
- Reduced number of Deadline misses:
  RM and DM scheduling policies do not provide any mechanism for dealing with task deadlines that are missed. In contrast, EDF scheduling policy ensures that tasks with approaching deadlines are executed first, thereby minimizing the number of missed deadlines.

Limitations of EDF over RM and DM:

- Higher scheduling overhead:
  EDF requires periodic sorting of tasks based on their deadlines, which can be computationally expensive and may lead to increased processing time. This causes a higher scheduling overhead.
- Poorer Predictability:
  RM and DM scheduling policies provide better predictability for task execution times. This is because the priority of each task is determined by its period, which is known in advance.

In this project, we implemented the EDF scheduling policy and contrasted it with the performance of RM and DM scheduling policies.

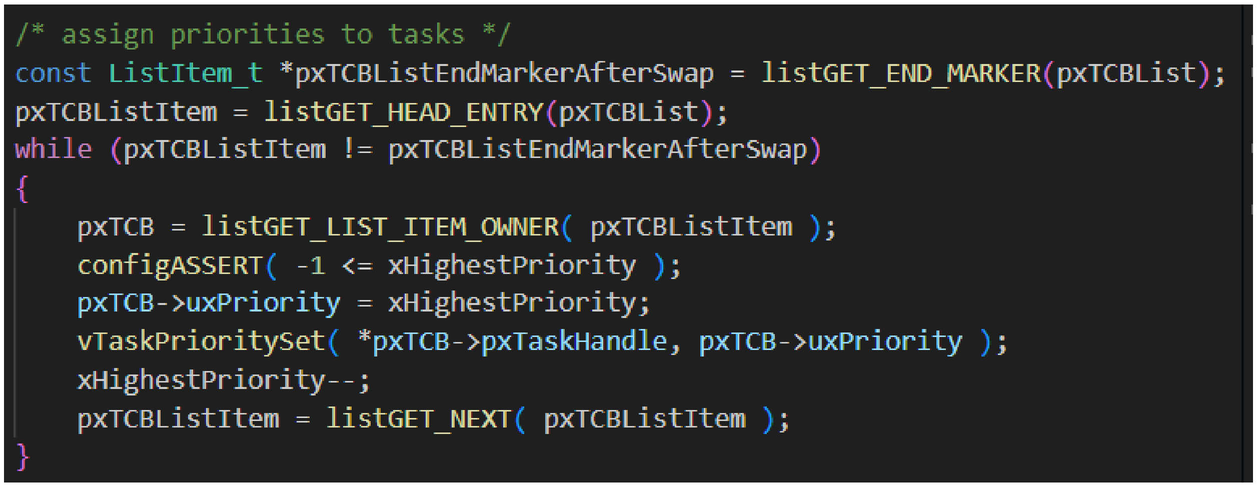
Implementation:

The project uses the RM and DM scheduling policy implementation from the previous project 3 submitted. The implementation of EDF scheduling policy is done in this project.

1. We initialize a few linked lists for sorting the tasks based on their deadlines in order to implement the Earliest Deadline First (EDF) scheduling policy. The four lists defined are:
    - xTCBList - a sorted linked list for all periodic tasks.
    - xTCBTempList - a temporary list used for switching lists.
    - xTCBOverflowedList - a sorted linked list for periodic tasks that have overflowed deadline.
    - pxTCBList, pxTCBTempList, and pxTCBOverflowedList - pointers to the corresponding linked lists.

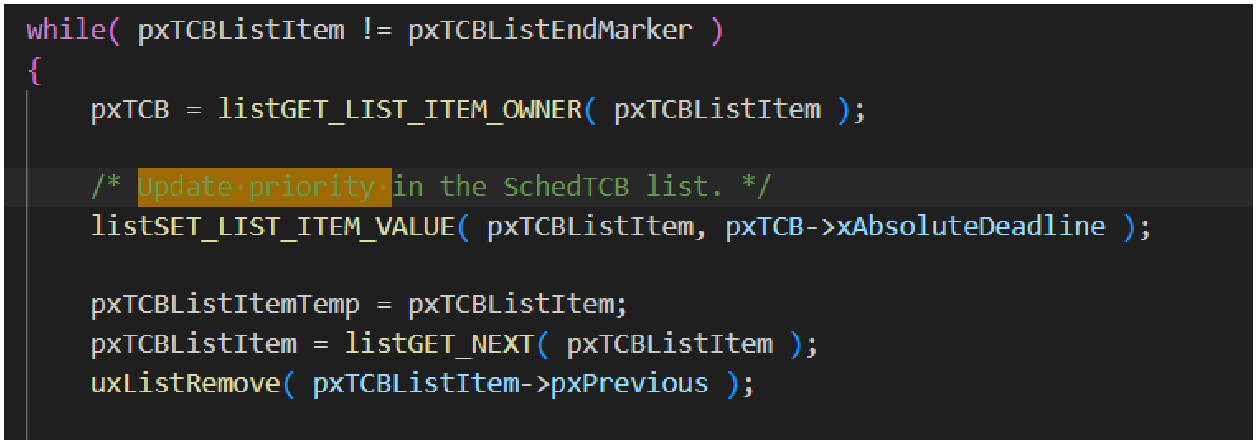
2. We also initialize a function called prvAddTCBToList for the insertion of a TCB into a sorted linked list based on its absolute deadline value. The function first initializes a list item xTCBListItem associated with the TCB using the vListInitialiseItem function. Then, it sets the owner of the list item to the TCB using the listSET_LIST_ITEM_OWNER macro. The listSET_LIST_ITEM_VALUE macro is used to set the value of the list item to the absolute deadline value of the TCB. The vListInsert function is then used to insert the TCB into the linked list in the correct position based on its absolute deadline value.

3. The main function implementing the EDF scheduling policy is the prvUpdatePrioritiesEDF() function. It is called if the scheduling policy macro is defined as EDF. It updates the priorities of all periodic tasks in the xTCBList based on their absolute deadlines. It first checks if the list is empty and if the overflowed list is not. If so, it swaps the two lists.
   Next, as shown in Fig3, the function loops through all the periodic tasks in the xTCBList, updates their priorities based on their absolute deadlines, and removes them from the list. If a task's absolute deadline has already passed (i.e., it has missed its deadline), it is inserted into the xTCBOverflowedList. Otherwise, it is inserted into the xTCBTempList.
   After all the periodic tasks have been processed, the function swaps the xTCBList with the xTCBTempList. Finally, using the code in Fig 2, the function assigns priorities to each periodic task based on their position in the sorted list, starting from the highest priority. The priorities are set using the vTaskPrioritySet() function, and the pcName, uxPriority, and xAbsoluteDeadline values of each task are printed to the serial monitor for debugging purposes.



```
/* assign priorities to tasks */
const ListItem_t *pxTCBListEndMarkerAfterSwap = listGET_END_MARKER(pxTCBList);
pxTCBListItem = listGET_HEAD_ENTRY(pxTCBList);
while (pxTCBListItem != pxTCBListEndMarkerAfterSwap)
{
    pxTCB = listGET_LIST_ITEM_OWNER( pxTCBListItem );
    configASSERT( -1 <= xHighestPriority );
    pxTCB->uxPriority = xHighestPriority;
    vTaskPrioritySet( *pxTCB->pxTaskHandle, pxTCB->uxPriority );
    xHighestPriority--;
    pxTCBListItem = listGET_NEXT( pxTCBListItem );
}
```

Fig 2 priority assigning policy for EDF scheduler



```
while( pxTCBListItem != pxTCBListEndMarker )
{
    pxTCB = listGET_LIST_ITEM_OWNER( pxTCBListItem );

    /* Update priority in the SchedTCB list. */
    listSET_LIST_ITEM_VALUE( pxTCBListItem, pxTCB->xAbsoluteDeadline );

    pxTCBListItemTemp = pxTCBListItem;
    pxTCBListItem = listGET_NEXT( pxTCBListItem );
    uxListRemove( pxTCBListItem->pxPrevious );
```

Fig 3 xAbsoluteDeadline becomes the key value for sorting

4. The prvPeriodicTaskCode function has been modified to update the priorities based on the nearest deadline every time before and after the task execution. This ensures that the scheduler is woken up before and after the task to allow for priority updates.
   Inside the function, a pointer to the SchedTCB_t structure is declared and initialized to the value of the task's thread local storage pointer using pvTaskGetThreadLocalStoragePointer() function. Then, the current task handle is obtained using xTaskGetCurrentTaskHandle().The function proceeds with checking if the handle is not NULL and obtaining further information about the task. If the schedUSE_TCB_ARRAY macro is set to 1, it retrieves the index of the task's TCB structure using prvGetTCBIndexFromHandle() function and sets the pxThisTask pointer to the corresponding TCB structure from the TCB array xTCBArray. Otherwise, it searches for the TCB structure in the task list pxTCBList using listGET_HEAD_ENTRY() and listGET_NEXT() macros. Finally, depending on the scheduling policy, the function wakes up the scheduler task again and delays until the next release time using xTaskDelayUntil() function.

5. The vApplicationTickHook is disabled as the prvPeriodicTaskCode function is modified to wake up the scheduler as it can cause a conflict in the scheduler wake up time from the task code.

6. The vSchedulerPeriodicTaskDelete function is modified to delete a TCB associated with a given task handle from a linked list of TCBs as well.
   The loop iterates through the list by getting the head entry and end marker of the list and comparing them. Within the loop, the TCB is extracted from the current list item using the listGET_LIST_ITEM_OWNER() macro. If the TCB associated with the current list item matches the input task handle, the loop is terminated. Finally, the function prvDeleteTCBFromList() is called with the TCB as the argument to remove it from the list.

7. In the prvCreateAllTasks function update, the configuration macro schedUSE_TCB_SORTED_LIST is checked to be if defined as 1. If it is, the code initializes a pointer pxTCBListItem to the head of a task control block (TCB) sorted list pxTCBList. The code then iterates through each TCB in the list, creating a new periodic task for each TCB using the xTaskCreate function.

8. Also in the prvCreateAllTasks function we assign the priorities to the tasks. It initializes the priorities of all periodic tasks with respect to the Earliest Deadline First (EDF) policy. First, it determines the highest priority for periodic tasks by checking the value of schedUSE_SCHEDULER_TASK. If schedUSE_SCHEDULER_TASK is defined as 1, uxHighestPriority is set to schedSCHEDULER_PRIORITY - 1, otherwise uxHighestPriority is set to configMAX_PRIORITIES - 1. Next, the function uses a while loop to iterate through all the periodic tasks in pxTCBList. Inside the loop, it assigns

priorities to the sorted tasks by setting the uxPriority field of the pxTCB to uxHighestPriority and assigns xReleaseTime to 0. It then prints the name of the task and its assigned priority using the Serial.print() function. Finally, it decrements uxHighestPriority and moves on to the next task until all tasks have been assigned a priority.

9. The prvSchedulerFunction is modified to update the priorities of the periodic tasks if the scheduling policy is EDF. It also checks for timing errors by calling the prvSchedulerCheckTimingError function for each periodic task in the system, either by iterating through an array of TCBs or a sorted list of TCBs, depending on the configuration.

10. The vSchedulerInit function is modified to include a check if the sorted TCB list is used, and if so, it initializes three lists using vListInitialise(): xTCBList, xTCBTempList, and xTCBOverflowedList. It also initializes three pointers pxTCBList, pxTCBTempList, and pxTCBOverflowedList to point to their corresponding lists.

Evaluation :

The task set taken to show the results of EDF , RM and DM scheduling policy is as follows.

Task	C	T
τ_1	28	70
τ_2	12	40
τ_3	4	20

Task set 1

The timeline diagram for task set 1 as expected are shown below. The Rm and DM scheduling policy will show the same timeline as the time period is equal to the deadline. That is shown as figure a. The EDF scheduling policy is shown as figure b.

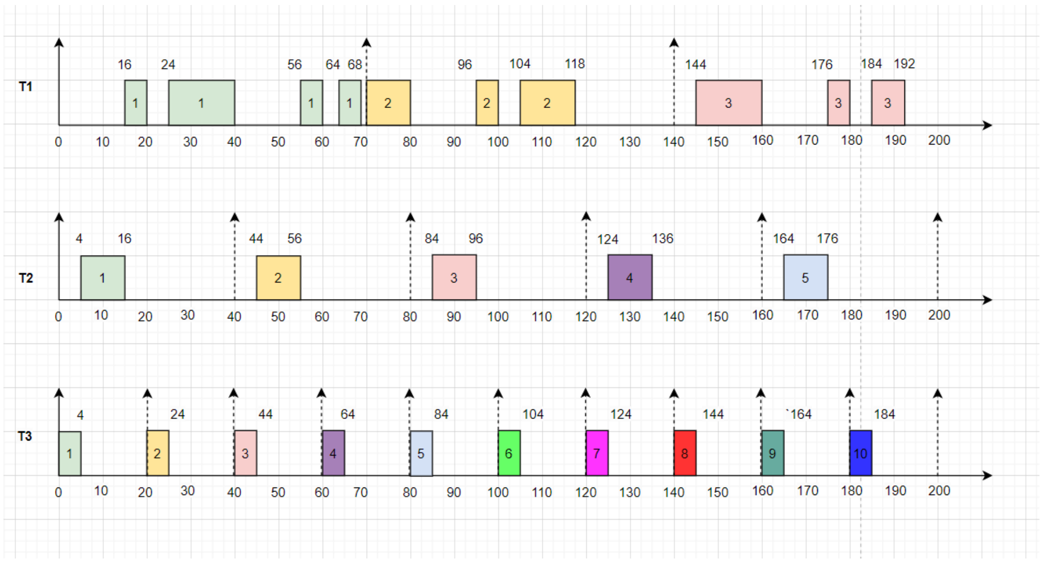


Figure 4a. RM/DM Timeline – Task set 1

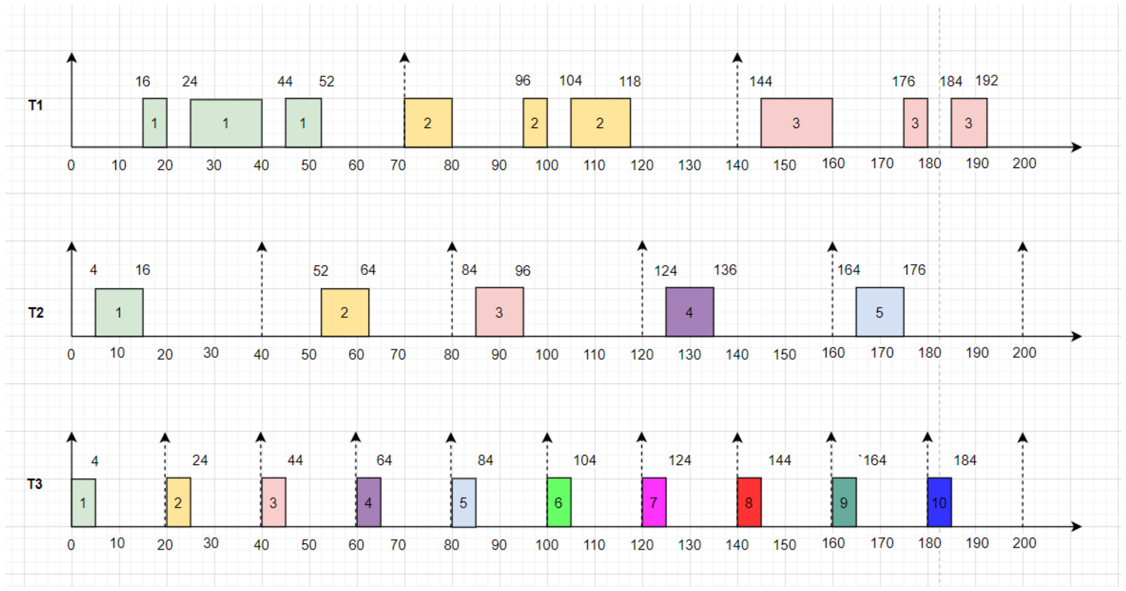
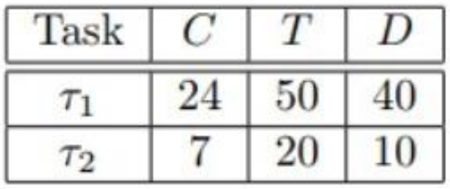


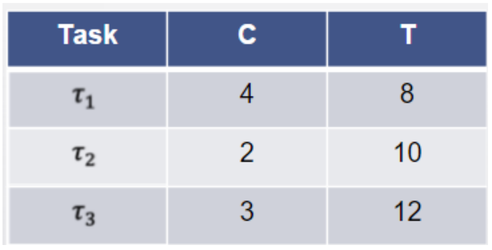
Figure 4b. EDF Timeline – Task set 1

Another set of tasks, task set 2 and task set 3, is also taken to show the results of EDF scheduling policy implementation is effective and works as expected. It is as shown below.



Task	C	T	D
τ_1	24	50	40
τ_2	7	20	10

Task set 2

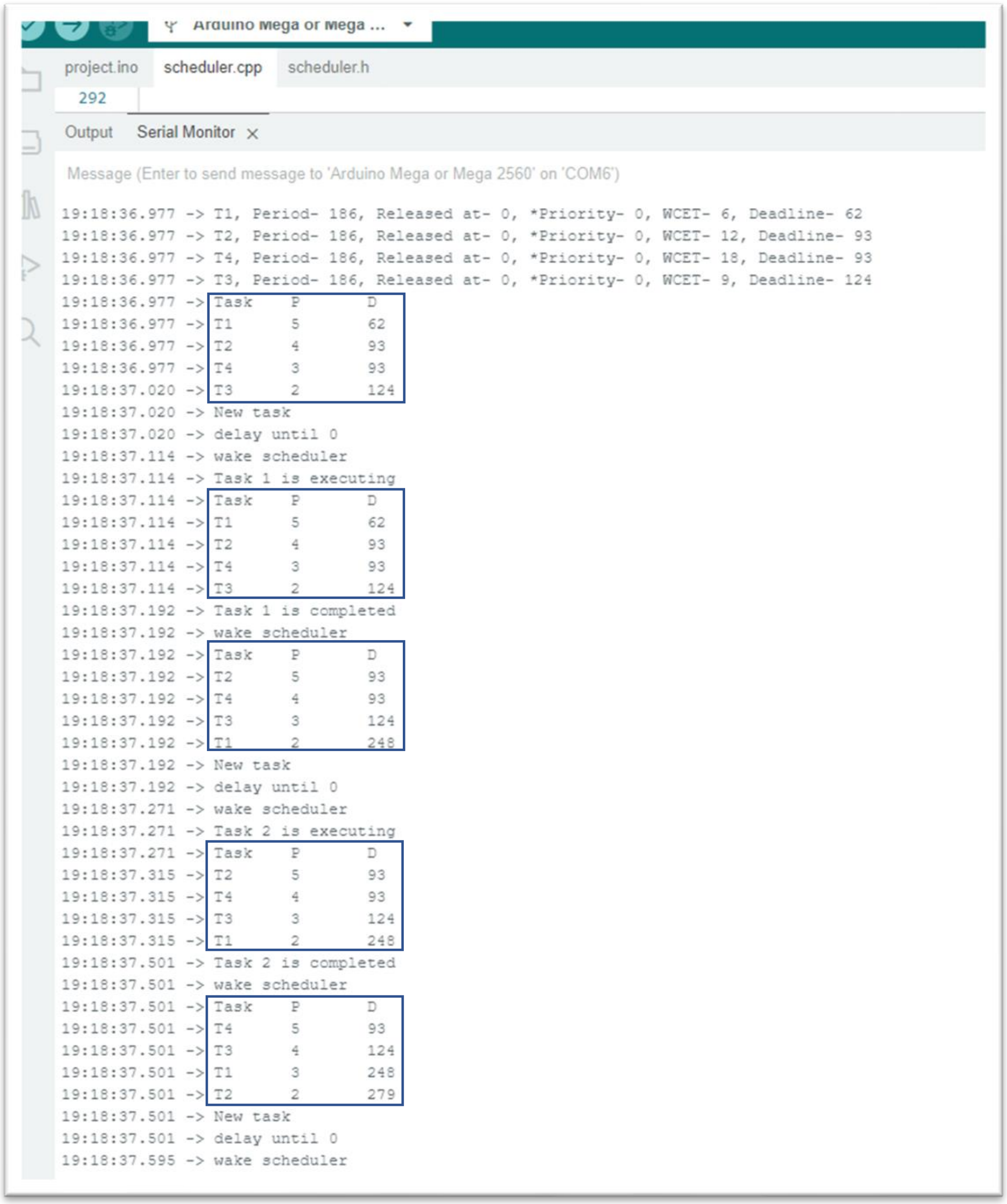


Task	C	T
τ_1	4	8
τ_2	2	10
τ_3	3	12

Task set 3 (not schedulable by RM/DM)

Results :

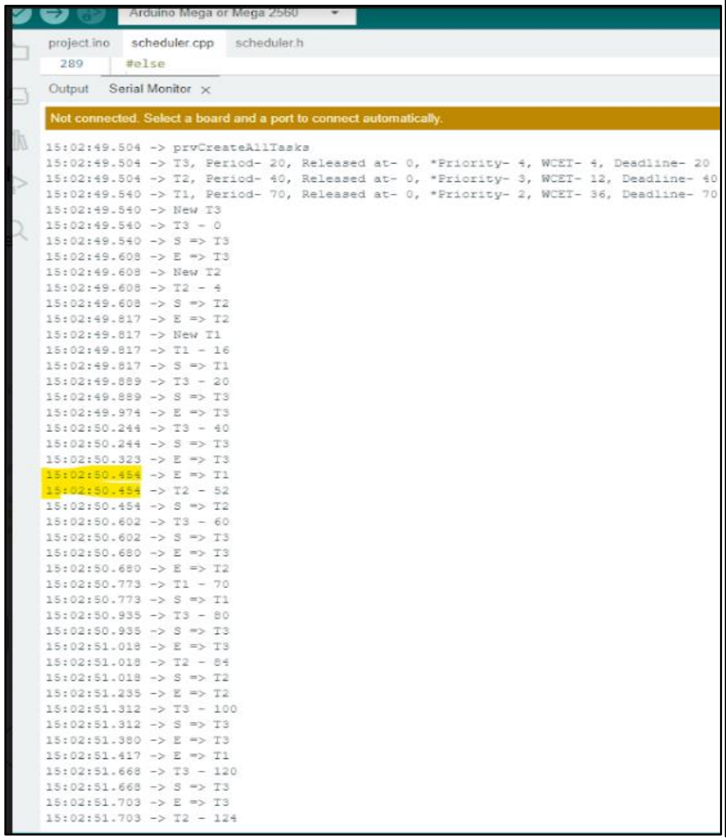
For a basic functionality test of our EDF scheduler implementation we tested with custom task set and tracked the dynamically changing priority of the scheduler. As can be seen in Fig 5, the priority of the tasks gets changed over time based on the closest deadline. The initial priorities of the tasks T1, T2, T3 and T4 are 5, 4, 2, 3 respectively. But over the time as the deadline of task T3 get's closer the priority increases and the priority of the tasks T1 and T2 decreases over time on each scheduler wakeup period. This effectively proves the working functionality of our EDF scheduler.



```
19:18:36.977 -> T1, Period- 186, Released at- 0, *Priority- 0, WCET- 6, Deadline- 62
19:18:36.977 -> T2, Period- 186, Released at- 0, *Priority- 0, WCET- 12, Deadline- 93
19:18:36.977 -> T4, Period- 186, Released at- 0, *Priority- 0, WCET- 18, Deadline- 93
19:18:36.977 -> T3, Period- 186, Released at- 0, *Priority- 0, WCET- 9, Deadline- 124
19:18:36.977 -> Task     P         D
19:18:36.977 -> T1       5        62
19:18:36.977 -> T2       4        93
19:18:36.977 -> T4       3        93
19:18:37.020 -> T3       2       124
19:18:37.020 -> New task
19:18:37.020 -> delay until 0
19:18:37.114 -> wake scheduler
19:18:37.114 -> Task 1 is executing
19:18:37.114 -> Task     P         D
19:18:37.114 -> T1       5        62
19:18:37.114 -> T2       4        93
19:18:37.114 -> T4       3        93
19:18:37.114 -> T3       2       124
19:18:37.192 -> Task 1 is completed
19:18:37.192 -> wake scheduler
19:18:37.192 -> Task     P         D
19:18:37.192 -> T2       5        93
19:18:37.192 -> T4       4        93
19:18:37.192 -> T3       3       124
19:18:37.192 -> T1       2       248
19:18:37.192 -> New task
19:18:37.192 -> delay until 0
19:18:37.271 -> wake scheduler
19:18:37.271 -> Task 2 is executing
19:18:37.271 -> Task     P         D
19:18:37.315 -> T2       5        93
19:18:37.315 -> T4       4        93
19:18:37.315 -> T3       3       124
19:18:37.315 -> T1       2       248
19:18:37.501 -> Task 2 is completed
19:18:37.501 -> wake scheduler
19:18:37.501 -> Task     P         D
19:18:37.501 -> T4       5        93
19:18:37.501 -> T3       4       124
19:18:37.501 -> T1       3       248
19:18:37.501 -> T2       2       279
19:18:37.501 -> New task
19:18:37.501 -> delay until 0
19:18:37.595 -> wake scheduler
```

Fig 5 EDF scheduler effectiveness test

The results for both task sets 1 and 2 are shown below on running the code in Arduino. The results are exactly as expected and same as those were obtained using the theoretical timeline diagram. The screenshots are attached below.

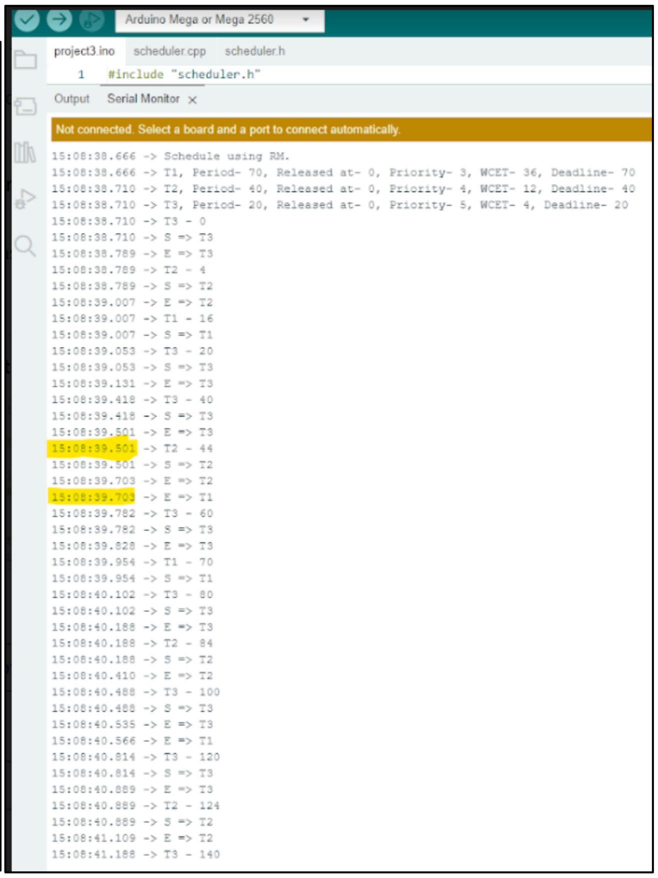


```
289    #else
```

Not connected. Select a board and a port to connect automatically.

```
15:02:49.504 -> prvCreateAllTasks
15:02:49.504 -> T3, Period- 20, Released at- 0, *Priority- 4, WCET- 4, Deadline- 20
15:02:49.504 -> T2, Period- 40, Released at- 0, *Priority- 3, WCET- 12, Deadline- 40
15:02:49.540 -> T1, Period- 70, Released at- 0, *Priority- 2, WCET- 36, Deadline- 70
15:02:49.540 -> New T3
15:02:49.540 -> T3 - 0
15:02:49.540 -> S => T3
15:02:49.608 -> E => T3
15:02:49.608 -> New T2
15:02:49.608 -> T2 - 4
15:02:49.608 -> S => T2
15:02:49.817 -> E => T2
15:02:49.817 -> New T1
15:02:49.817 -> T1 - 16
15:02:49.817 -> S => T1
15:02:49.889 -> T3 - 20
15:02:49.889 -> S => T3
15:02:49.974 -> E => T3
15:02:50.244 -> T3 - 40
15:02:50.244 -> S => T3
15:02:50.323 -> E => T3
15:02:50.454 -> E => T1
15:02:50.454 -> T2 - 52
15:02:50.454 -> S => T2
15:02:50.602 -> T3 - 60
15:02:50.602 -> S => T3
15:02:50.680 -> E => T3
15:02:50.680 -> E => T2
15:02:50.773 -> T1 - 70
15:02:50.773 -> S => T1
15:02:50.935 -> T3 - 80
15:02:50.935 -> S => T3
15:02:51.018 -> E => T3
15:02:51.018 -> T2 - 84
15:02:51.018 -> S => T2
15:02:51.235 -> E => T2
15:02:51.312 -> T3 - 100
15:02:51.312 -> S => T3
15:02:51.380 -> E => T3
15:02:51.417 -> E => T1
15:02:51.668 -> T3 - 120
15:02:51.668 -> S => T3
15:02:51.703 -> E => T3
15:02:51.703 -> T2 - 124
```

EDF Scheduling policy – **Task set 1** ref Fig 4b



```
#include "scheduler.h"
```

Not connected. Select a board and a port to connect automatically.

```
15:08:38.666 -> Schedule using RM.
15:08:38.666 -> T1, Period- 70, Released at- 0, Priority- 3, WCET- 36, Deadline- 70
15:08:38.710 -> T2, Period- 40, Released at- 0, Priority- 4, WCET- 12, Deadline- 40
15:08:38.710 -> T3, Period- 20, Released at- 0, Priority- 5, WCET- 4, Deadline- 20
15:08:38.710 -> T3 - 0
15:08:38.710 -> S => T3
15:08:38.789 -> E => T3
15:08:38.789 -> T2 - 4
15:08:38.789 -> S => T2
15:08:39.007 -> E => T2
15:08:39.007 -> T1 - 16
15:08:39.007 -> S => T1
15:08:39.053 -> T3 - 20
15:08:39.053 -> S => T3
15:08:39.131 -> E => T3
15:08:39.418 -> T3 - 40
15:08:39.418 -> S => T3
15:08:39.501 -> E => T3
15:08:39.501 -> T2 - 44
15:08:39.501 -> S => T2
15:08:39.703 -> E => T2
15:08:39.703 -> E => T1
15:08:39.782 -> T3 - 60
15:08:39.782 -> S => T3
15:08:39.828 -> E => T3
15:08:39.954 -> T1 - 70
15:08:39.954 -> S => T1
15:08:40.102 -> T3 - 80
15:08:40.102 -> S => T3
15:08:40.188 -> E => T3
15:08:40.188 -> T2 - 84
15:08:40.188 -> S => T2
15:08:40.410 -> E => T2
15:08:40.488 -> T3 - 100
15:08:40.488 -> S => T3
15:08:40.535 -> E => T3
15:08:40.566 -> E => T1
15:08:40.814 -> T3 - 120
15:08:40.814 -> S => T3
15:08:40.889 -> E => T3
15:08:40.889 -> T2 - 124
15:08:40.889 -> S => T2
15:08:41.109 -> E => T2
15:08:41.188 -> T3 - 140
```

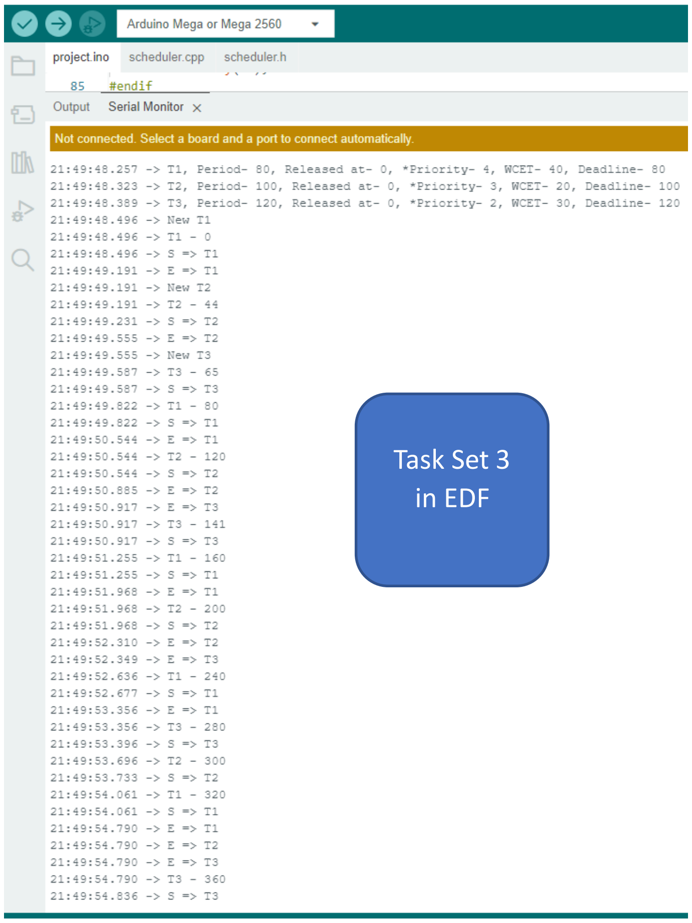
RM/DM Scheduling policy – **Task set 1** ref Fig 4a

For comparing the task set1 for the different scheduling policies, we can see that for RM, the priorities are assigned based on the period of the task, with the shortest period receiving the highest priority. Therefore, the priorities for T1, T2, and T3 would be 1, 2, and 3, respectively. Similarly , for DM, priorities are assigned based on the deadline of the task, with the shortest deadline receiving the highest priority. In this case, T3 will receive the highest priority, followed by T2 and T1.

In contrast, EDF prioritizes tasks based dynamically on their earliest absolute deadline. The task with the earliest deadline gets the highest priority. In this case, the priorities will be dynamically allocated. This order ensures that all tasks meet their respective deadlines, making EDF a better choice for this task set compared to RM or DM.

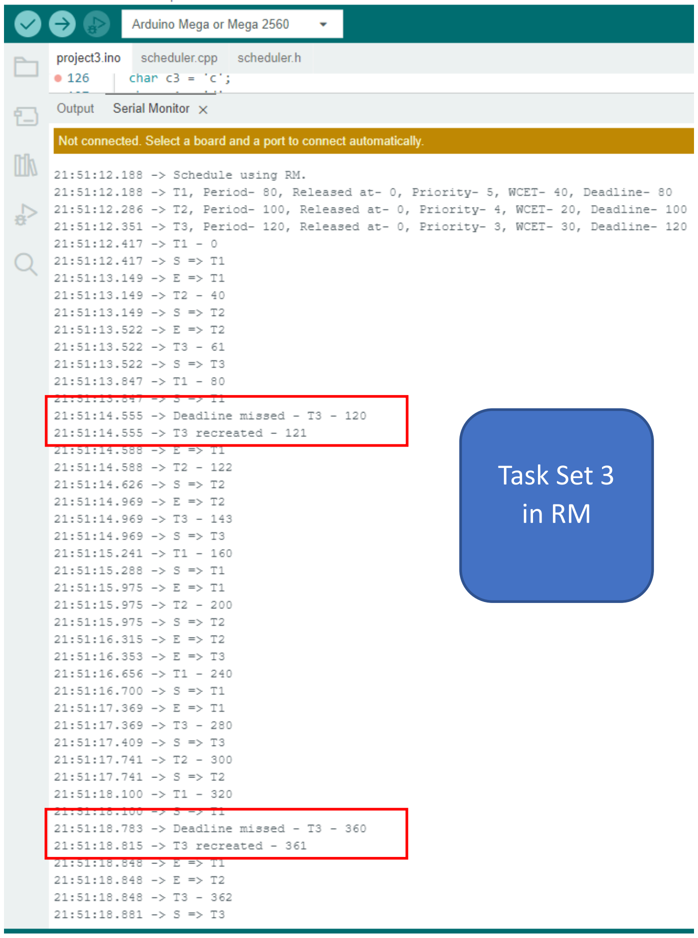
Notably for the task T1 and T2, as can be seen in Fig 4a and 4b, for EDF task T1 gets executed first and then the T2 as the deadline for T1 is much closer than the deadline for T2. However, in the case of RM/DM, T2 has the highest priority and gets completed first as it is fixed priority and the deadline or period of T2 is shorter than T1. This is the key difference of EDF from RM/DM scheduling policy and our implementation proves it.

For Task set 3, The tasks are not schedulable by the RM/DM scheduler which leads to deadline miss for the T3 tasks. But this can be resolved with the EDF scheduler. As can be seen in below logs. Task 3 misses its deadline in every cycle but such is not the case with the earliest deadline first scheduler as the task with the closes deadline gets the highest priority to be executed to meet the deadline. This again proves our implementation of EDF and showcase that EDF is better than RM and DM scheduler.


Arduino Mega or Mega 2560
project.ino
scheduler.cpp
scheduler.h
Output
Serial Monitor
Not connected. Select a board and a port to connect automatically.
Task Set 3
in EDF

```
#endif
```

```
21:49:48.257 -> T1, Period- 80, Released at- 0, *Priority- 4, WCET- 40, Deadline- 80
21:49:48.323 -> T2, Period- 100, Released at- 0, *Priority- 3, WCET- 20, Deadline- 100
21:49:48.389 -> T3, Period- 120, Released at- 0, *Priority- 2, WCET- 30, Deadline- 120
21:49:48.496 -> New T1
21:49:48.496 -> T1 - 0
21:49:48.496 -> S => T1
21:49:49.191 -> E => T1
21:49:49.191 -> New T2
21:49:49.191 -> T2 - 44
21:49:49.231 -> S => T2
21:49:49.555 -> E => T2
21:49:49.555 -> New T3
21:49:49.587 -> T3 - 65
21:49:49.587 -> S => T3
21:49:49.822 -> T1 - 80
21:49:49.822 -> S => T1
21:49:50.544 -> E => T1
21:49:50.544 -> T2 - 120
21:49:50.544 -> S => T2
21:49:50.885 -> E => T2
21:49:50.917 -> E => T3
21:49:50.917 -> T3 - 141
21:49:50.917 -> S => T3
21:49:51.255 -> T1 - 160
21:49:51.255 -> S => T1
21:49:51.968 -> E => T1
21:49:51.968 -> T2 - 200
21:49:51.968 -> S => T2
21:49:52.310 -> E => T2
21:49:52.349 -> E => T3
21:49:52.636 -> T1 - 240
21:49:52.677 -> S => T1
21:49:53.356 -> E => T1
21:49:53.356 -> T3 - 280
21:49:53.396 -> S => T3
21:49:53.696 -> T2 - 300
21:49:53.733 -> S => T2
21:49:54.061 -> T1 - 320
21:49:54.061 -> S => T1
21:49:54.790 -> E => T1
21:49:54.790 -> E => T2
21:49:54.790 -> E => T3
21:49:54.790 -> T3 - 360
21:49:54.836 -> S => T3
```


Arduino Mega or Mega 2560
project3.ino
scheduler.cpp
scheduler.h
Output
Serial Monitor
Not connected. Select a board and a port to connect automatically.
Task Set 3
in RM

```
char c3 = 'c';
```

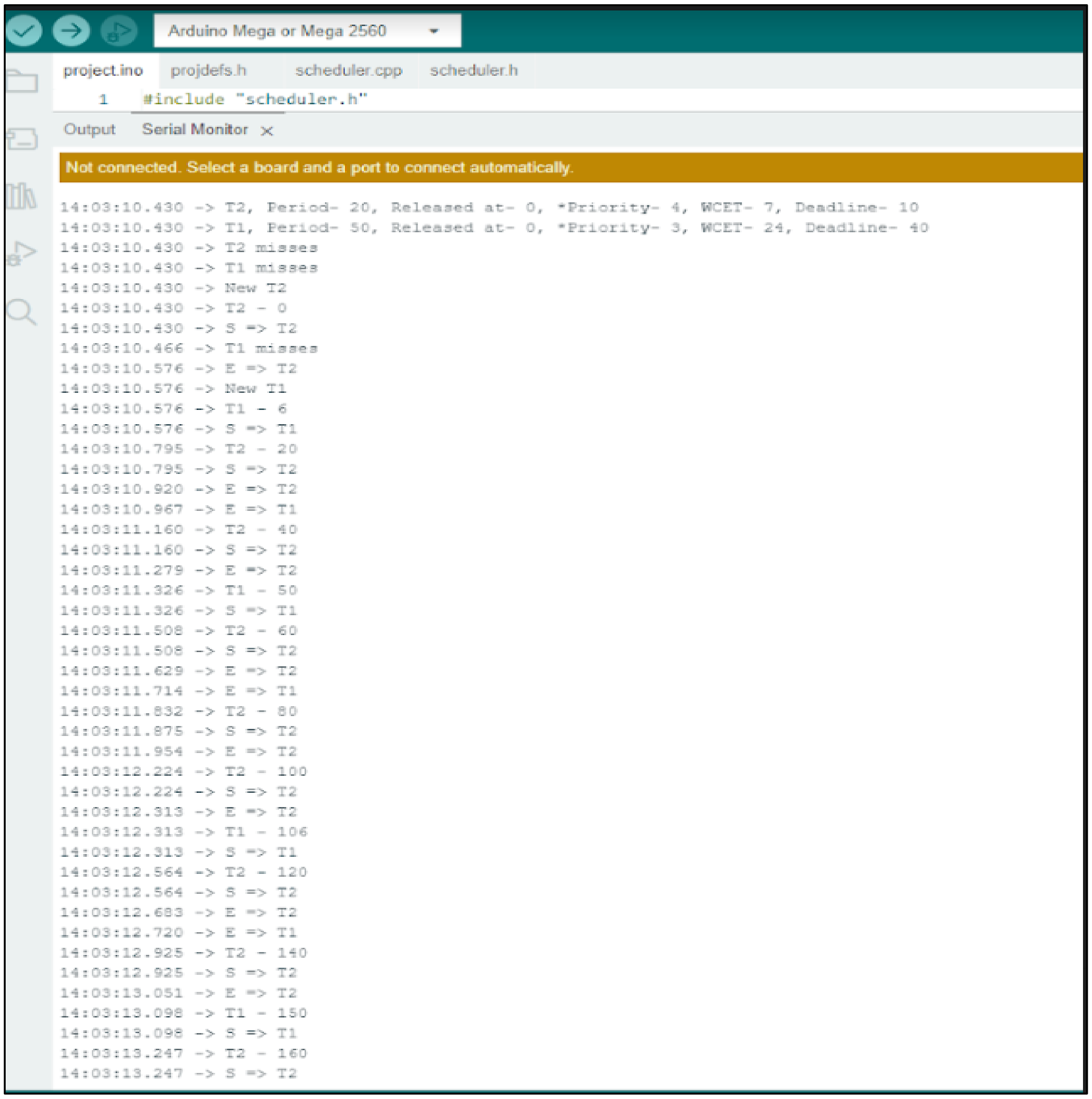
```
21:51:12.188 -> Schedule using RM.
21:51:12.188 -> T1, Period- 80, Released at- 0, Priority- 5, WCET- 40, Deadline- 80
21:51:12.286 -> T2, Period- 100, Released at- 0, Priority- 4, WCET- 20, Deadline- 100
21:51:12.351 -> T3, Period- 120, Released at- 0, Priority- 3, WCET- 30, Deadline- 120
21:51:12.417 -> T1 - 0
21:51:12.417 -> S => T1
21:51:13.149 -> E => T1
21:51:13.149 -> T2 - 40
21:51:13.149 -> S => T2
21:51:13.522 -> E => T2
21:51:13.522 -> T3 - 61
21:51:13.522 -> S => T3
21:51:13.847 -> T1 - 80
21:51:13.847 -> S => T1
21:51:14.555 -> Deadline missed - T3 - 120
21:51:14.555 -> T3 recreated - 121
21:51:14.588 -> E => T1
21:51:14.588 -> T2 - 122
21:51:14.626 -> S => T2
21:51:14.969 -> E => T2
21:51:14.969 -> T3 - 143
21:51:14.969 -> S => T3
21:51:15.241 -> T1 - 160
21:51:15.288 -> S => T1
21:51:15.975 -> E => T1
21:51:15.975 -> T2 - 200
21:51:15.975 -> S => T2
21:51:16.315 -> E => T2
21:51:16.353 -> E => T3
21:51:16.656 -> T1 - 240
21:51:16.700 -> S => T1
21:51:17.369 -> E => T1
21:51:17.369 -> T3 - 280
21:51:17.409 -> S => T3
21:51:17.741 -> T2 - 300
21:51:17.741 -> S => T2
21:51:18.100 -> T1 - 320
21:51:18.100 -> S => T1
21:51:18.783 -> Deadline missed - T3 - 360
21:51:18.815 -> T3 recreated - 361
21:51:18.848 -> E => T1
21:51:18.848 -> E => T2
21:51:18.848 -> T3 - 362
21:51:18.881 -> S => T3
```

Task Set 2 in EDF and RM scheduler.

Performance metrics:

The performance of the EDF scheduler can be costly as it involves calling the scheduler every time before and after task execution for each task and going through the iterative list for finding the right priority. Hence the scheduler overhead is larger incase of EDF scheduler. Tasks with short and critical deadlines can get affect by this overhead but not much. But in case of RM/DM scheduler, the scheduler overhead is less as its only job is to check for deadline misses or timing errors.

Runtime Overhead of EDF > Overhead of RM/DM


Arduino Mega or Mega 2560
project.ino
projdefs.h
scheduler.cpp
scheduler.h
Output
Serial Monitor
Not connected. Select a board and a port to connect automatically.

```
#include "scheduler.h"
```

```
14:03:10.430 -> T2, Period- 20, Released at- 0, *Priority- 4, WCET- 7, Deadline- 10
14:03:10.430 -> T1, Period- 50, Released at- 0, *Priority- 3, WCET- 24, Deadline- 40
14:03:10.430 -> T2 misses
14:03:10.430 -> T1 misses
14:03:10.430 -> New T2
14:03:10.430 -> T2 - 0
14:03:10.430 -> S => T2
14:03:10.466 -> T1 misses
14:03:10.576 -> E => T2
14:03:10.576 -> New T1
14:03:10.576 -> T1 - 6
14:03:10.576 -> S => T1
14:03:10.795 -> T2 - 20
14:03:10.795 -> S => T2
14:03:10.920 -> E => T2
14:03:10.967 -> E => T1
14:03:11.160 -> T2 - 40
14:03:11.160 -> S => T2
14:03:11.279 -> E => T2
14:03:11.326 -> T1 - 50
14:03:11.326 -> S => T1
14:03:11.508 -> T2 - 60
14:03:11.508 -> S => T2
14:03:11.629 -> E => T2
14:03:11.714 -> E => T1
14:03:11.832 -> T2 - 80
14:03:11.875 -> S => T2
14:03:11.954 -> E => T2
14:03:12.224 -> T2 - 100
14:03:12.224 -> S => T2
14:03:12.313 -> E => T2
14:03:12.313 -> T1 - 106
14:03:12.313 -> S => T1
14:03:12.564 -> T2 - 120
14:03:12.564 -> S => T2
14:03:12.683 -> E => T2
14:03:12.720 -> E => T1
14:03:12.925 -> T2 - 140
14:03:12.925 -> S => T2
14:03:13.051 -> E => T2
14:03:13.098 -> T1 - 150
14:03:13.098 -> S => T1
14:03:13.247 -> T2 - 160
14:03:13.247 -> S => T2
```

EDF Scheduling policy – **Task set 2**

Conclusion :

We have implemented the EDF scheduling policy with the expected performance and effectiveness and have contrasted its performance with that of RM and DM scheduling policy. For smaller set of non-varying tasks RM/DM scheduler works fine but for more complex systems which needs much stronger scheduling policy based on the nearest deadline, EDF scheduler has proven to be an effective solution.