Final Project

Objective : Implement, test and compare EDF against RM and DM.

EDF Design : Main steps are as below :-

Step 1 : Initialize EDF priorities based on shortest deadline.

Step 2 : Create and run all tasks.

Step 3 : In scheduler function, every time the sheduler wakes up, check for the earliest deadline of all tasks and update priorities of all tasks.

Note : Updation of priorities should only happen if there is a change in the deadline. However, in the current implementation, updation happens irrespective of deadline changes. This does not in any way affect the functionality but only adds unwanted load on the kernel. This can be optimized at a later stage.

Step 3 : Update absolute deadlines in the code as soon as a job completes execution so a call to update priorities will have the most updated deadlines of all the tasks.

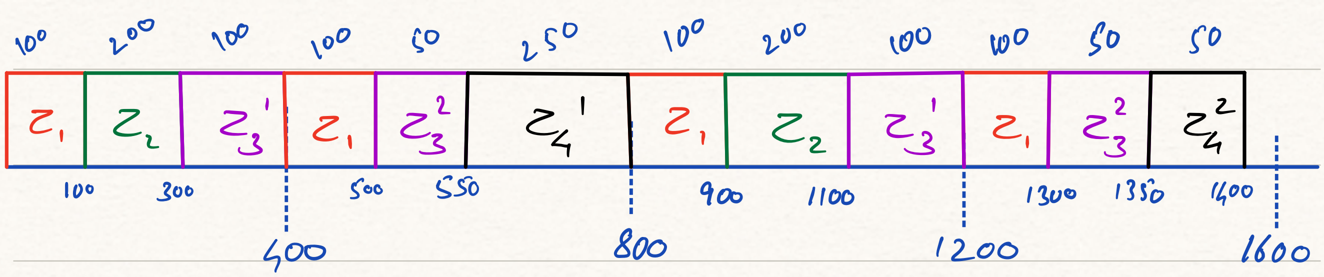
Only one extra function prvUpdateEDFPriorities function was added which assigns priorities in the increasing order of deadlines. It then makes a call to vTaskSetPriority to actually set the updated priorities.

Analysis of RM and DM :

Task Set 1 :-

Task	C	D	T
τ_1	100	400	400
τ_2	200	700	800
τ_3	150	1000	1000
τ_4	300	5000	5000

The timeline for RM for 1 cycle looks like :



Where $\pi(\tau_1) > \pi(\tau_2) > \pi(\tau_3) > \pi(\tau_4)$
as it depends on their periods.

The output timeline matches the timeline. Note that execution times of pre-empted tasks are not displayed in whole but only the first instance of the job so they are not to be considered in the analysis. However, the order of execution and context switches are correct.

```
15:05:17.906 -> t1 has priority 3
15:05:17.938 -> t2 has priority 2
15:05:17.971 -> t3 has priority 1
15:05:17.971 -> t4 has priority 0
15:05:18.012 -> t1 start
15:05:18.057 -> t1 end 80
15:05:18.057 -> t2 start
15:05:18.213 -> t2 end 176
15:05:18.213 -> t3 start
15:05:18.351 -> t1 start
15:05:18.430 -> t1 end 80
15:05:18.430 -> t3 end 144
15:05:18.474 -> t4 start
15:05:18.692 -> t4 end 256
15:05:18.725 -> t1 start
15:05:18.845 -> t1 end 80
15:05:18.845 -> t2 start
15:05:19.028 -> t2 end 176
15:05:19.028 -> t3 start
15:05:19.153 -> t1 start
15:05:19.232 -> t1 end 80
15:05:19.232 -> t3 end 128
15:05:19.553 -> t1 start
15:05:19.645 -> t1 end 80
15:05:19.645 -> t2 start
15:05:19.816 -> t2 end 176
15:05:19.956 -> t1 start
15:05:20.049 -> t1 end 80
15:05:20.049 -> t3 start
15:05:20.173 -> t3 end 128
15:05:20.359 -> t1 start
15:05:20.425 -> t1 end 80
15:05:20.469 -> t2 start
15:05:20.625 -> t2 end 176
15:05:20.750 -> t1 start
15:05:20.842 -> t1 end 80
15:05:21.035 -> t3 start
15:05:21.170 -> t1 start
15:05:21.263 -> t1 end 80
15:05:21.263 -> t2 start
15:05:21.432 -> t2 end 176
```

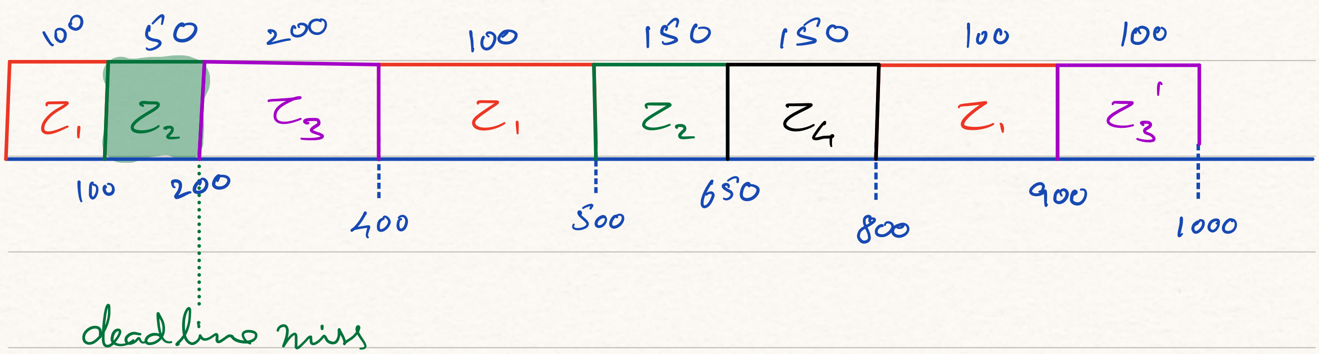
The timeline for DM for 1 exec cycle is the same as that of RM because the deadlines are proportional to their periods. Thus, the execution also matches the RM output.

```
14:59:48.821 -> t1 has priority 3
14:59:48.853 -> t2 has priority 2
14:59:48.853 -> t3 has priority 1
14:59:48.886 -> t4 has priority 0
14:59:48.919 -> t1 start
14:59:48.958 -> t1 end 80
14:59:48.958 -> t2 start
14:59:49.146 -> t2 end 176
14:59:49.146 -> t3 start
14:59:49.226 -> t1 start
14:59:49.345 -> t1 end 80
14:59:49.345 -> t3 end 144
14:59:49.345 -> t4 start
14:59:49.595 -> t4 end 256
14:59:49.628 -> t1 start
14:59:49.751 -> t1 end 80
14:59:49.751 -> t2 start
14:59:49.906 -> t2 end 176
14:59:49.952 -> t3 start
14:59:50.029 -> t1 start
14:59:50.151 -> t1 end 80
14:59:50.151 -> t3 end 128
14:59:50.464 -> t1 start
14:59:50.556 -> t1 end 80
14:59:50.556 -> t2 start
14:59:50.742 -> t2 end 176
14:59:50.865 -> t1 start
14:59:50.958 -> t1 end 80
14:59:50.958 -> t3 start
14:59:51.098 -> t3 end 128
14:59:51.271 -> t1 start
14:59:51.349 -> t1 end 80
14:59:51.349 -> t2 start
14:59:51.536 -> t2 end 176
14:59:51.660 -> t1 start
14:59:51.752 -> t1 end 80
14:59:51.970 -> t3 start
14:59:52.036 -> t1 start
14:59:52.157 -> t1 end 80
14:59:52.157 -> t2 start
14:59:52.315 -> t2 end 176
```

Consider task set 2 :-

Task	C	D	T
τ_1	100	400	400
τ_2	150	200	500
τ_3	200	700	800
τ_4	150	1000	1000

The timeline for RM for 1 cycle is as shown :

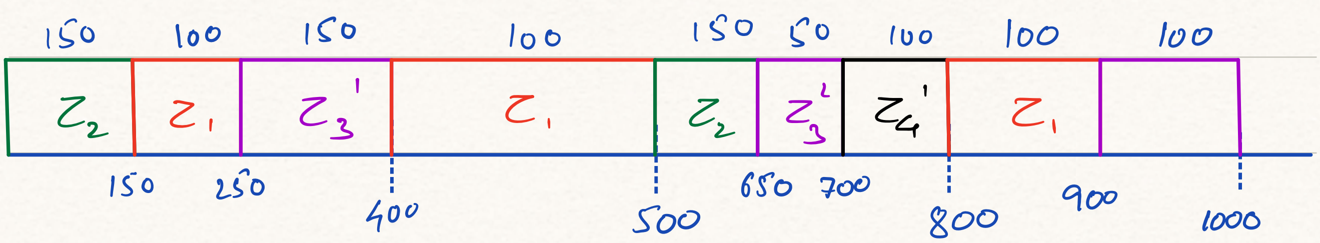


But since task τ_2 has to be first deleted and recreated, there is going to be a overhead involved after τ_2 has missed its deadline thereby pushing all task execution by a certain time.

The practical output is as shown:

```
15:26:34.194 -> t1 has priority 3
15:26:34.194 -> t2 has priority 2
15:26:34.227 -> t3 has priority 1
15:26:34.227 -> t4 has priority 0
15:26:34.260 -> t1 start
15:26:34.292 -> t1 end 96
15:26:34.338 -> t2 start
15:26:34.452 -> t2 task deadline missed
15:26:34.494 -> t2 task recreated
15:26:34.494 -> t3 start
15:26:34.618 -> t1 start
15:26:34.698 -> t1 end 80
15:26:34.698 -> t2 start
15:26:34.853 -> t2 end 144
15:26:34.886 -> t3 end 192
15:26:34.886 -> t4 start
15:26:35.022 -> t1 start
15:26:35.087 -> t1 end 80
15:26:35.131 -> t3 start
15:26:35.247 -> t2 start
15:26:35.247 -> t4 task deadline missed
15:26:35.286 -> t4 task recreated
15:26:35.410 -> t2 end 144
15:26:35.410 -> t1 start
15:26:35.488 -> t1 end 80
15:26:35.521 -> t3 exec time exceeded
15:26:35.563 -> t4 start
```

The timeline for DM for 1 cycle is as shown:



The practical output is as shown below:

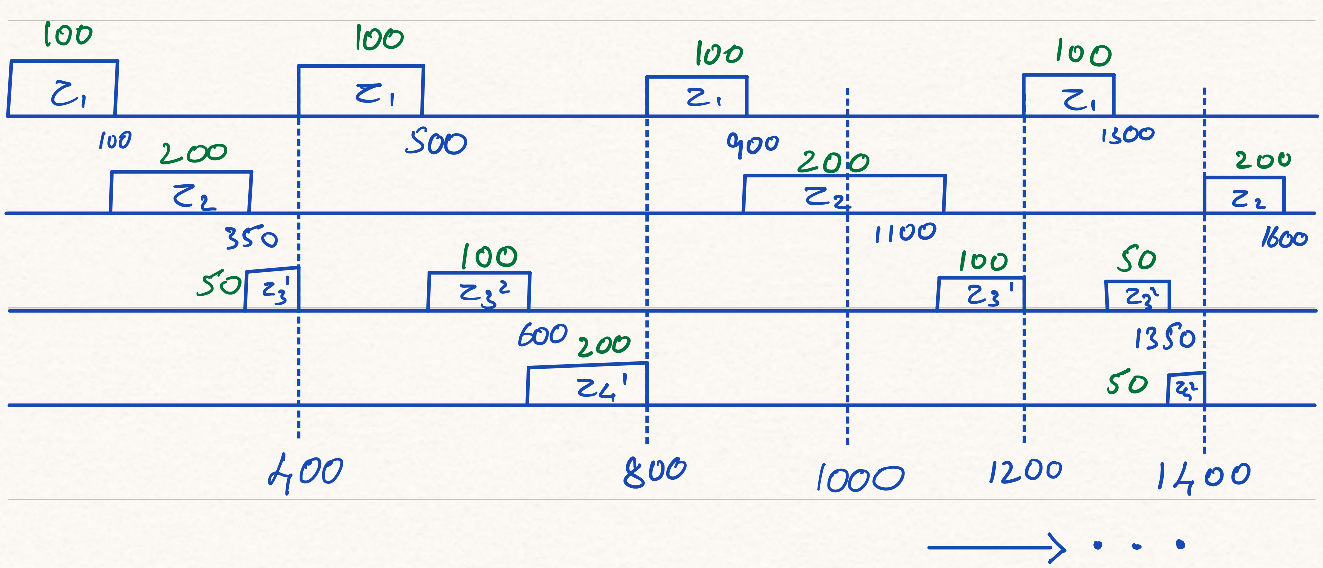
```
15:32:49.196 -> t1 has priority 2
15:32:49.229 -> t2 has priority 3
15:32:49.229 -> t3 has priority 1
15:32:49.262 -> t4 has priority 0
15:32:49.262 -> t2 start
15:32:49.379 -> t2 end 144
15:32:49.379 -> t1 start
15:32:49.471 -> t1 end 80
15:32:49.471 -> t3 start
15:32:49.596 -> t1 start
15:32:49.712 -> t1 end 80
15:32:49.712 -> t3 end 176
15:32:49.751 -> t4 start
15:32:49.751 -> t2 start
15:32:49.889 -> t2 end 128
15:32:50.003 -> t4 end 144
15:32:50.046 -> t1 start
15:32:50.123 -> t1 end 80
15:32:50.123 -> t3 start
15:32:50.236 -> t2 start
15:32:50.404 -> t2 end 128
15:32:50.404 -> t1 start
15:32:50.519 -> t1 end 80
15:32:50.519 -> t3 end 192
15:32:50.519 -> t4 start
15:32:50.681 -> t4 end 128
15:32:50.774 -> t2 start
15:32:50.900 -> t2 end 128
15:32:50.946 -> t1 start
15:32:51.023 -> t1 end 80
15:32:51.023 -> t3 start
15:32:51.192 -> t3 end 176
15:32:51.239 -> t1 start
15:32:51.270 -> t2 start
15:32:51.433 -> t2 end 128
15:32:51.473 -> t1 end 80
15:32:51.473 -> t4 start
15:32:51.600 -> t4 end 144
15:32:51.644 -> t1 start
15:32:51.737 -> t1 end 80
15:32:51.737 -> t3 start
```

The observation is that since tasks are running almost to their WCET but not completely in the first release, there are windows to sneak in job instances before periodic release of other higher priority jobs. Thus we may see many pre-emptions and context switches.

The general idea is that while RM missed job deadline, DM was the optimal choice for task set 2. It did not matter for task set 1 because both were optimal.

Analysis of EDF :-

Task Set 1 :- The expected pattern is :-

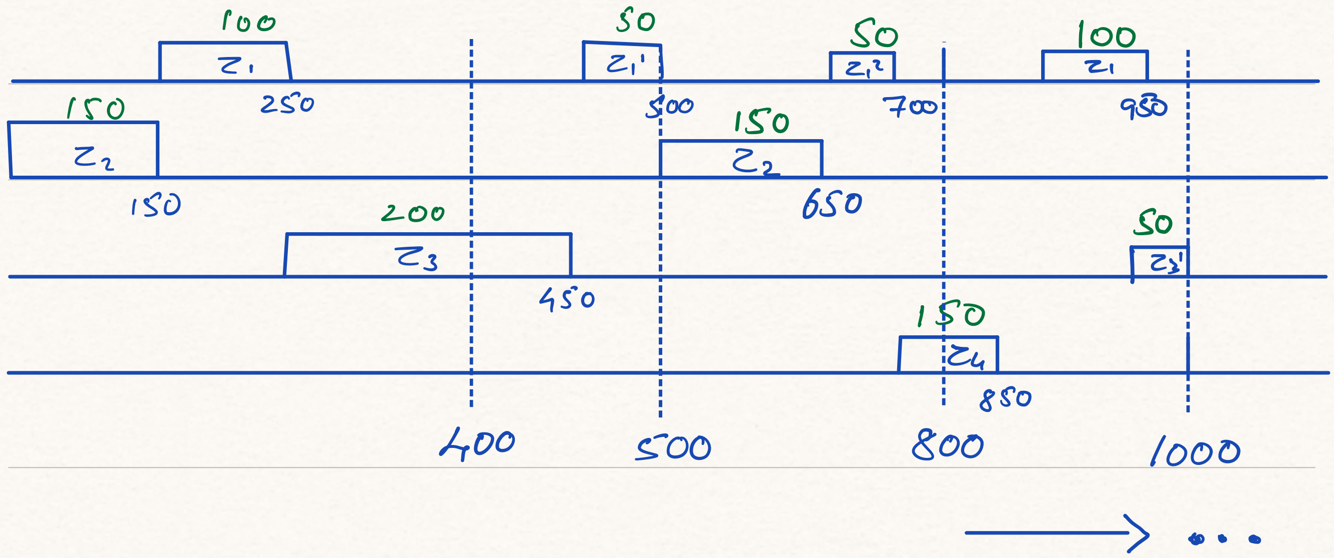


The observed practical output is :

```
17:18:45.824 -> t1 has priority 4
17:18:45.824 -> t2 has priority 3
17:18:45.857 -> t3 has priority 2
17:18:45.857 -> t4 has priority 1
17:18:45.902 -> t1 start
17:18:45.934 -> t1 stop
17:18:45.934 -> t2 start
17:18:46.119 -> t2 stop
17:18:46.119 -> t3 start
17:18:46.243 -> t1 start
17:18:46.321 -> t1 stop
17:18:46.321 -> t3 stop
17:18:46.366 -> t4 start
17:18:46.645 -> t4 stop
17:18:46.645 -> t1 start
17:18:46.737 -> t1 stop
17:18:46.737 -> t2 start
17:18:46.923 -> t2 stop
17:18:46.923 -> t3 start
17:18:47.045 -> t1 start
17:18:47.110 -> t1 stop
17:18:47.152 -> t3 stop
17:18:47.417 -> t1 start
17:18:47.540 -> t1 stop
17:18:47.540 -> t2 start
17:18:47.724 -> t2 stop
17:18:47.848 -> t1 start
17:18:47.940 -> t1 stop
17:18:47.940 -> t3 start
17:18:48.078 -> t3 stop
17:18:48.248 -> t1 start
17:18:48.340 -> t1 stop
17:18:48.340 -> t2 start
17:18:48.526 -> t2 stop
17:18:48.649 -> t1 start
17:18:48.742 -> t1 stop
17:18:48.959 -> t3 start
17:18:49.052 -> t1 start
```

Although it is different from RM and DM, we can see that there are no deadline misses and that all 3 policies are optimal for this set.

Consider task set 2 :-



The obtained results are as below :-

```
17:21:03.175 -> t2 has priority 4
17:21:03.207 -> t1 has priority 3
17:21:03.240 -> t3 has priority 2
17:21:03.240 -> t4 has priority 1
17:21:03.281 -> t2 start
17:21:03.375 -> t2 stop
17:21:03.375 -> t1 start
17:21:03.467 -> t1 stop
17:21:03.467 -> t3 start
17:21:03.671 -> t3 stop
17:21:03.671 -> t1 start
17:21:03.748 -> t2 start
17:21:03.872 -> t2 stop
17:21:03.872 -> t1 stop
17:21:03.872 -> t4 start
17:21:04.042 -> t4 stop
17:21:04.042 -> t1 start
17:21:04.121 -> t1 stop
17:21:04.121 -> t3 start
17:21:04.228 -> t2 start
17:21:04.413 -> t2 stop
17:21:04.444 -> t3 stop
17:21:04.491 -> t1 start
17:21:04.567 -> t1 stop
17:21:04.567 -> t4 start
17:21:04.707 -> t4 stop
17:21:04.785 -> t2 start
17:21:04.922 -> t2 stop
17:21:04.922 -> t1 start
17:21:04.999 -> t1 stop
17:21:04.999 -> t3 start
17:21:05.215 -> t3 stop
17:21:05.215 -> t1 start
17:21:05.307 -> t2 start
17:21:05.432 -> t2 stop
17:21:05.476 -> t1 stop
17:21:05.476 -> t4 start
17:21:05.596 -> t4 stop
```

In this case, RM runs into deadline misses but DM does not. Similarly, EDF also is an optimal scheduling policy for this task.

<u>Conclusion</u> :- EDF performs well when the task sets are not overloaded. It shows similar characteristics to Deadline Monotonic scheduling policy and proves to be a good scheduling algorithm when catering to tasks that are very critical and data dependent. However, it performs poorly and runs into multiple deadline misses due to multiple context switches when trying to schedule overloaded task sets.

Unlike RM and DM, there is slightly higher scheduler overhead when using EDF policy due to dynamic priority checking.