Thesis Implementation

1. Define Macro

- In FreeRTOSConfig.h

```
75
76 /* EDF Scheduler */
77 #define configUSE_EDF_SCHEDULER 1
78
79
```

2. Define The New EDF List

- In tasks.c

```
403 #if (configUSE_EDF_SCHEDULER == 1)
404   PRIVILEGED_DATA static List_t xReadyTasksListEDF;
405 #endif
406
```

3. Initialize The New EDF List

- In tasks.c in `prvInitialiseTaskLists()`

```
3645
3646 #if (configUSE_EDF_SCHEDULER == 1)
3647   vListInitialise(&xReadyTasksListEDF);
3648 #endif
```

4. Modify The Method That Adds A Task To The Ready List

- In tasks.c

```
229 #if configUSE_EDF_SCHEDULER == 1
230
231 #define prvAddTaskToReadyList(pxTCB) \
232   traceMOVED_TASK_TO_READY_STATE(pxTCB); \
233   vListInsert( &(xReadyTasksListEDF), & ((pxTCB)->xStateListItem)); \
234   tracePOST_MOVED_TASK_TO_READY_STATE(pxTCB)
235
236 #else
237
238 #define prvAddTaskToReadyList(pxTCB)
239   traceMOVED_TASK_TO_READY_STATE(pxTCB);
240   taskRECORD_READY_PRIORITY((pxTCB)->uxPriority);
241   listINSERT_END(&(pxReadyTasksLists[(pxTCB)->uxPriority]),
242                  &((pxTCB)->xStateListItem));
243   tracePOST_MOVED_TASK_TO_READY_STATE(pxTCB)
244
245 #endif /* configUSE_EDF_SCHEDULER */
246
```

- Note: When adding a new task using `vListInsert()` function it inserts this new `xStateListItem` node in the `xReadyTasksListEDF` list at a position according to the value inside the member variable `xStateListItem.xItemValue` in such a way so that the nodes inside the list are sorted in ascending order according to this value. So We should make `xItemValue` of each task node hold the task deadline

---

5. Modify TCB Struct

- In tasks.c

```
379
380 #if (configUSE_EDF_SCHEDULER == 1)
381   TickType_t xTaskPeriod;
382 #endif
383
384 } tskTCB;
385
386 /* The old tskTCB name is maintained above th
387  * below to enable the use of older kernel aw
388 typedef tskTCB TCB_t;
389
```

---

6. Create A New Task Initialization Method

- In tasks.c

```
911
912     return xReturn;
913 }
914 #else
915 BaseType_t xTaskPeriodicCreate(
916       TaskFunction_t pxTaskCode,
917       const char *const pcName, /*lint !e971 Unqualified char types are allowed
918                                    for strings and single characters only. */
919       const configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters,
920       UBaseType_t uxPriority, TaskHandle_t *const pxCreatedTask, TickType_t period) {
921     TCB_t *pxNewTCB;
922     BaseType_t xReturn;
923
924
```

```
999                             NULL);
1000
1001      /* INITIALIZE THE PERIOD */
1002      pxNewTCB->xTaskPeriod = period;
1003
1004      /* STORE THE DEADLINE INSIDE OF TASK NODE BEFORE ADDING IT TO THE READY LIST */
1005      listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ),
1006      ( pxNewTCB->xTaskPeriod + xTickCount));
1007
1008      /* USING THE MODIFIED METHOD ADD THE TASK TO THE READY LIST */
1009      prvAddNewTaskToReadyList(pxNewTCB);
1010
1011      xReturn = pdPASS;
```

---

7. Modify Initialization of IDLE Task

- In tasks.c in `vTaskStartScheduler()`

```
2103      /* The idle task is being created using dynamically allocated RAM. */
2104
2105      #if(configUSE_EDF_SCHEDULER == 1)
2106        TickType_t IDLEPeriod = 100;
2107        xReturn = xTaskPeriodicCreate(
2108          prvIdleTask, configIDLE_TASK_NAME, configMINIMAL_STACK_SIZE,
2109          (void *)NULL,
2110          portPRIVILEGE_BIT,
2111          &xIdleTaskHandle,
2112          IDLEPeriod);
2113      #else
2114
2115      xReturn = xTaskCreate(
2116          prvIdleTask, configIDLE_TASK_NAME, configMINIMAL_STACK_SIZE,
2117          (void *)NULL,
2118          portPRIVILEGE_BIT, /* In effect ( tskIDLE_PRIORITY | portPRIVILEGE_BIT
2119                                ), but tskIDLE_PRIORITY is zero. */
2120          &xIdleTaskHandle); /*lint !e961 MISRA exception, justified as it is not
2121                                a redundant explicit cast to all supported
2122                                compilers. */
2123      #endif /* configUSE_EDF_SCHEDULER */
2124    }
2125
```

- Note: We will have to make sure that the IDLE tasks stays at the end of the EDF list. This is just initialization if we didn't do anything else when the system starts running for a while the IDLE task will eventually preempt other application tasks.

> *Every time IDLE task executes (i.e. no other tasks are in the Ready List), it calls a method that increments its deadline in order to guarantee that IDLE task will remain in the last position of the Ready List.*

---

8. Choose The Task At The Head Of The EDF List When Context Switching

- In tasks.c in `vTaskSwitchContext()`

```
3082          * optimised asm code. */
3083
3084         #if(configUSE_EDF_SCHEDULER == 0)
3085          taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this
3086                                              macro is used with timers and
3087                                              co-routines too.  Alignment is known
3088                                              to be fine as the type of the pointer
3089                                              stored and retrieved is the same. */
3090         #else
3091
3092          pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );
3093
3094         #endif /* configUSE_EDF_SCHEDULER */
3095
3096         traceTASK SWITCHED IN();
```

- Note: All That this function `vTaskSwitchContext()` does is it selects the task that will run and assigns it to `pxCrruntTCB`