EDF IMPLEMENTION

```
832  BaseType_t xTaskPeriodicCreate( TaskFunction_t pxTaskCode,
833              const char * const pcName,    /*lint !e971 Unqualified char types are allowed for strings and single char
834              const configSTACK_DEPTH_TYPE usStackDepth,
835              void * const pvParameters,
836              UBaseType_t uxPriority,
837              TaskHandle_t * const pxCreatedTask,
838              TickType_t Period)
839  {
840  TCB_t *pxNewTCB;
841  BaseType_t xReturn;
842    ////////////////////////////////////////////////////////////////////////////
843  TickType_t currentTick = xTaskGetTickCount();   // get the current tick count to update the deadline
844    ////////////////////////////////////////////////////////////////////////////
845
846    /* If the stack grows down then allocate the stack then the TCB so the stack
```

We get the current tick count

```
912 ////////////////////////////////////////////////////////////////////////////////////////////////////
913       prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, p
914
915     #if(configUSE_EDF_SCHEDULER ==1)
916     { pxNewTCB->xTaskPeriod = Period;
917
918       /*E.C. : insert the period value in the generic list iteam before to add the task in RL: */
919       listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), ( pxNewTCB)->xTaskPeriod + currentTick);
920
921       ////////////////////////////////////////////////////////////////////////////
922       prvAddNewTaskToReadyList(pxNewTCB);   ////////////////////////////////////////////////////////
923     }
```

Update the xStateListItem to hold the update deadline to insert the task according to its deadline …. Then call prvAddNewTaskToReadyList(pxNewTCB);

```
1271 static void prvAddNewTaskToReadyList( TCB_t * pxNewTCB )
1272 {
1273     /* Ensure interrupts don't access the task lists while the lists are being
1274      * updated. */
1275     taskENTER_CRITICAL();
1276     {
1277         uxCurrentNumberOfTasks++;
1278
1279         if( pxCurrentTCB == NULL )
1280         {
1281             /* There are no other tasks, or all the other tasks are in
1282              * the suspended state - make this the current task. */
1283             pxCurrentTCB = pxNewTCB;
1284
1285             if( uxCurrentNumberOfTasks == ( UBaseType_t ) 1 )
1286             {
```

this will start the last created task which is a bug as the Idle task is the last created and lower priority and it will be the 1st excuted

'

So we will set the current task after creating Idle task to be the nearest deadline

```
2194  #endif /* ( ( INCLUDE_xTaskResumeFromISR == 1 ) && ( INCLUDE_vTaskSuspend == 1 )
2195  /*-----------------------------------------------------------*/
2196
2197  void vTaskStartScheduler( void )
2198  {
2199      BaseType_t xReturn;
2200
```

Inside void vTaskStartScheduler(void) function

```
2228    #else /* if ( configSUPPORT_STATIC_ALLOCATION == 1 ) */
2229  {
2230      ////////////////////////////////////////////////////////////////////////////////
2231    #if (configUSE_EDF_SCHEDULER == 1)
2232    {
2233      TickType_t initIDLEPeriod = 100;
2234      xReturn = xTaskPeriodicCreate( prvIdleTask, "IDLE", 100,
2235      (void * ) NULL, 0, NULL,
2236      IDLE_PERIOD );
2237
2238      pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );  /////// i  added this
2239    }
2240    ////////////////////////////////////////////////////////////////////////////////
```

After creating the Idle task we will get the nearest deadline to be first executed

```
2962    /*-----------------------------------------------------------*/
2963    uint32_t IdlePeriod =0;
2964    uint32_t IdleDeadLine =0;
2965    BaseType_t xTaskIncrementTick( void )
2966    {
2967        TCB_t * pxTCB;
2968        TickType_t xItemValue;
2969        BaseType_t xSwitchRequired = pdFALSE;
2970
2971        /* Called by the portable layer each time a tick interrupt occurs.
2972         * Increments the tick then checks to see if the new tick value will cause any
```

Inside xTaskIncrementTick(void) function

```
3055  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3056          #if (  configUSE_EDF_SCHEDULER == 1 )
3057          {
3058          TCB_t * IdlepxTCB;
3059          IdlepxTCB = (TCB_t*)((&((&(xReadyTasksListEDF))->xListEnd))->pxPrevious->pvOwner); //get TCB pointer of Idle task
3060          IdlePeriod = IdlepxTCB->xTaskPeriod;                                              //monotoring idle task period
3061          IdleDeadLine = (&((&(xReadyTasksListEDF))->xListEnd))->pxPrevious->xItemValue;    //monitoring idle task xItemValue
3062
3063          //changing Idle Task xItemValue to hold the current deadline of idle task as idle task is the last element
3064          (&((&(xReadyTasksListEDF))->xListEnd))->pxPrevious->xItemValue = IdlepxTCB->xTaskPeriod + xTickCount;
3065
3066
3067
3068          // changing the deadline of the comming back task to be inserted correctly inside ready list
3069          listSET_LIST_ITEM_VALUE( &( ( pxTCB )->xStateListItem ), ( pxTCB)->xTaskPeriod + xTickCount);
3070
3071          prvAddTaskToReadyList( pxTCB );
3072          xSwitchRequired = pdTRUE;     //  it is the time to request switching
3073          }
3074          #else
3075          {
```

We will

1- Get the pointer to idle task as it is the last item in linked list and the ListEnd small list item is the pointer to the end of list so pxPrevious is pointing to the Idle task
2- Get the period from idel task
3- Update the item value of the idle task to be the farest deadline after adding its period to the tickcounter
4- Changing the xStateListItem of the new task removed from the delayed list to be inserted according to its new deadline inside EDF ready list
5- THE REQUEST FOR CONTEXT SWITCHING

AS WE ARE IN PEEMPTION WE DIDN'T HAVE TO UPDATE IDLE TASK DEADLINE CONTINOUSLY INSIDE THE IDLE PERIOD FUNCTION

```
2748
2749    UBaseType_t uxTaskGetSystemState( TaskStatus_t * const pxTaskStatusArray,
2750                                      const UBaseType_t uxArraySize,
2751                                      configRUN_TIME_COUNTER_TYPE * const pulTotalRunTime )
2752    {
2753        UBaseType_t uxTask = 0, uxQueue = configMAX_PRIORITIES;
2754
2755        vTaskSuspendAll();
```

Inside the uxTaskGetSystemState function

```
2763
2764        #if(configUSE_EDF_SCHEDULER == 1)
2765          {
2766          uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &(xReadyTasksListEDF), eReady );
2767
2768        }
2769        #else
2770        {
2771          do
2772          {
2773          uxQueue--;
2774          uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), (List_t*)&( pxReadyTasksLists[ uxQueu
2775          } while( uxQueue > ( UBaseType_t ) tskIDLE_PRIORITY ); /*lint !e961 MISRA exception as the casts are only redun
2776
2777        }
2778        #endif
```

We change the parameter (List_t*)&(pxReadyTasksLists[uxQueue]) to &(xReadyTasksListEDF) to get the freeRTOS 0statistics