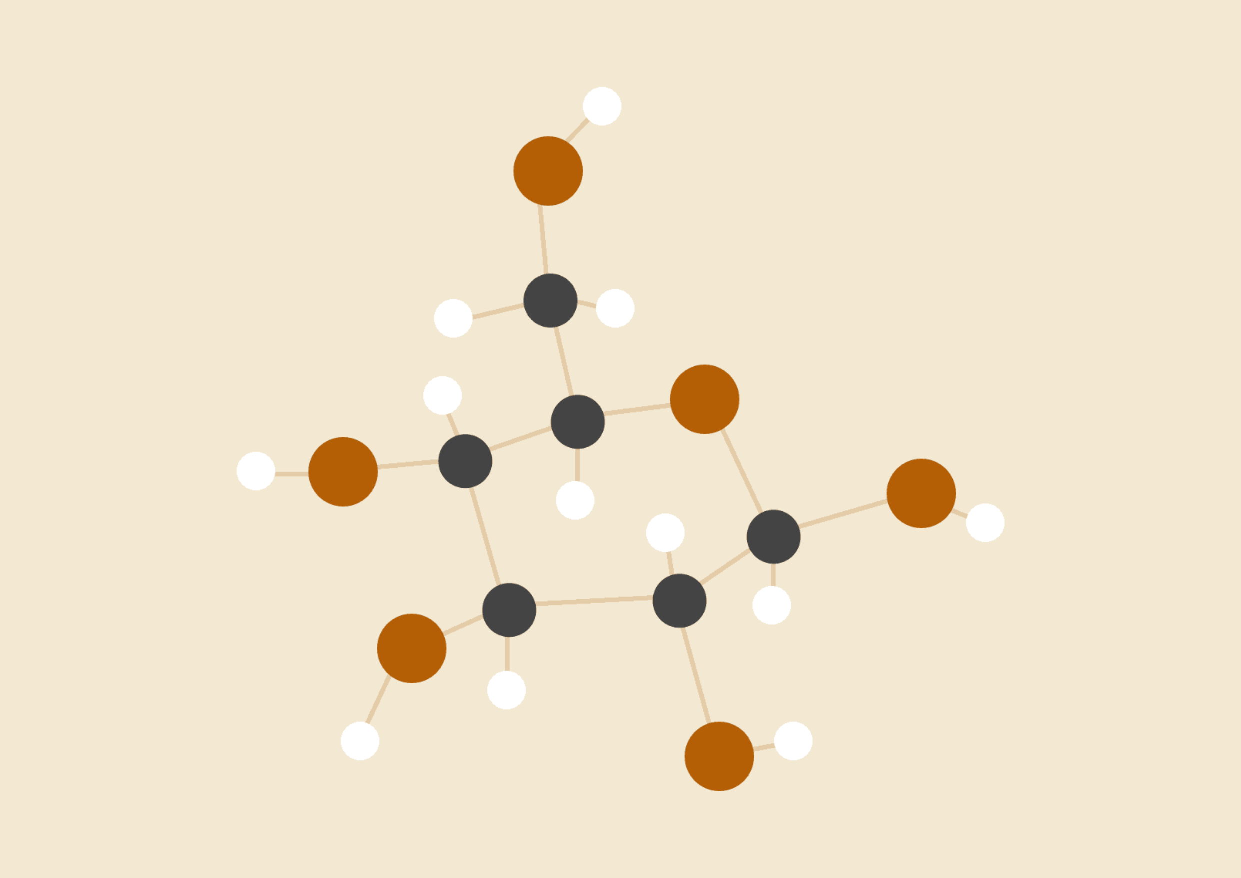
IMPLEMENTATION OF EDF SCHEDULER IN FREERTOS

RTOS MC GRADUATION PROJECT



Mahmoud Nageh Abdelkader

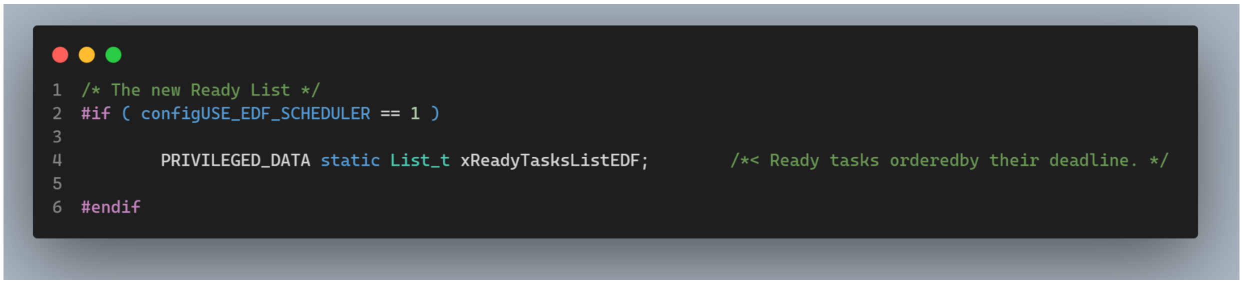
DESCRIPTION

In this project I’ll be implementing the EDF scheduler form “ **Implementation and Test of EDF and LLREF Schedulers in FreeRTOS** “ written by “ **EnricoCarraro** “ while making the necessary changes. Then the implementation will be tested with a chosen set of tasks, the tests will be manual analytics, offline simulator, and run-time analysis.

IMPLEMENTING EDF FROM THE THESIS

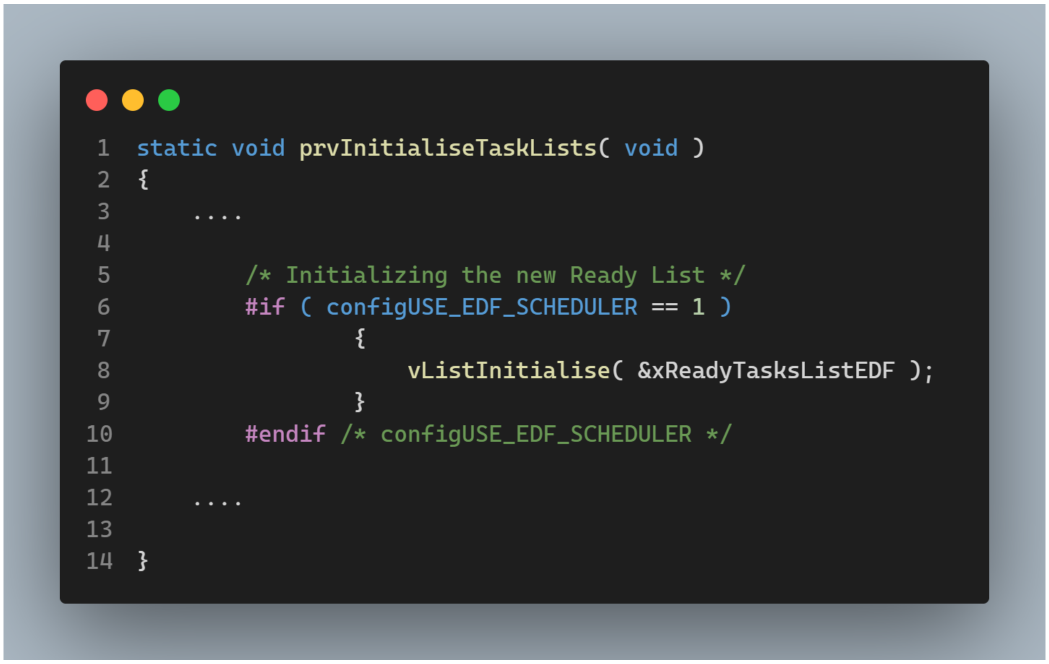
To start things off, all the changes will be done in task.c file and will be guarded with configUSEEDFSCHEDULER that will be set to 1 whenever the scheduler needs to be used.

- The new Ready List is declared **xReadyTasksListEDF** is a simple list structure.



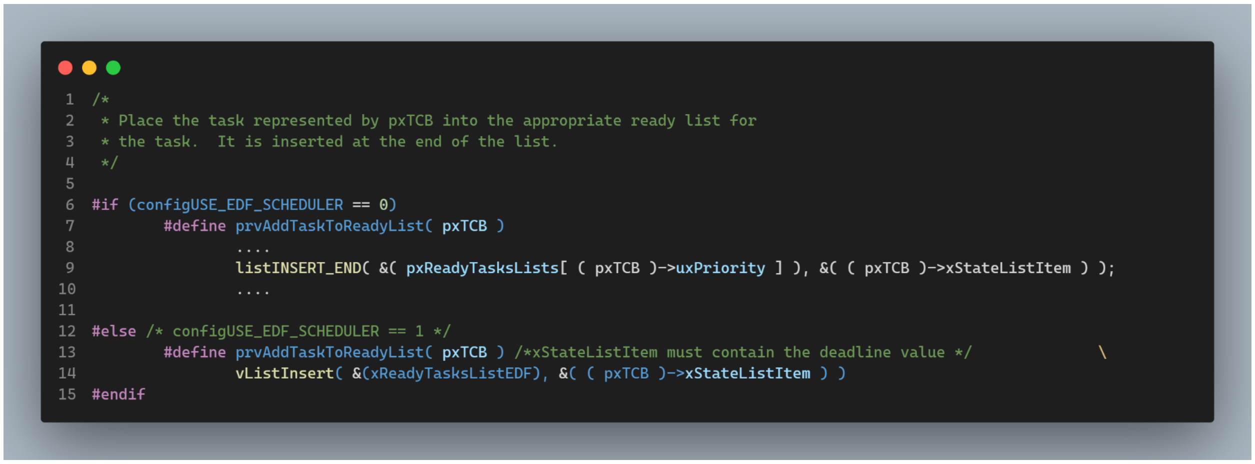
```c
/* The new Ready List */
#if ( configUSE_EDF_SCHEDULER == 1 )

        PRIVILEGED_DATA static List_t xReadyTasksListEDF;        /*< Ready tasks orderedby their deadline. */

#endif
```

- Adding the initialization of **xReadyTasksListEDF** in **prvInitialiseTaskLists().**



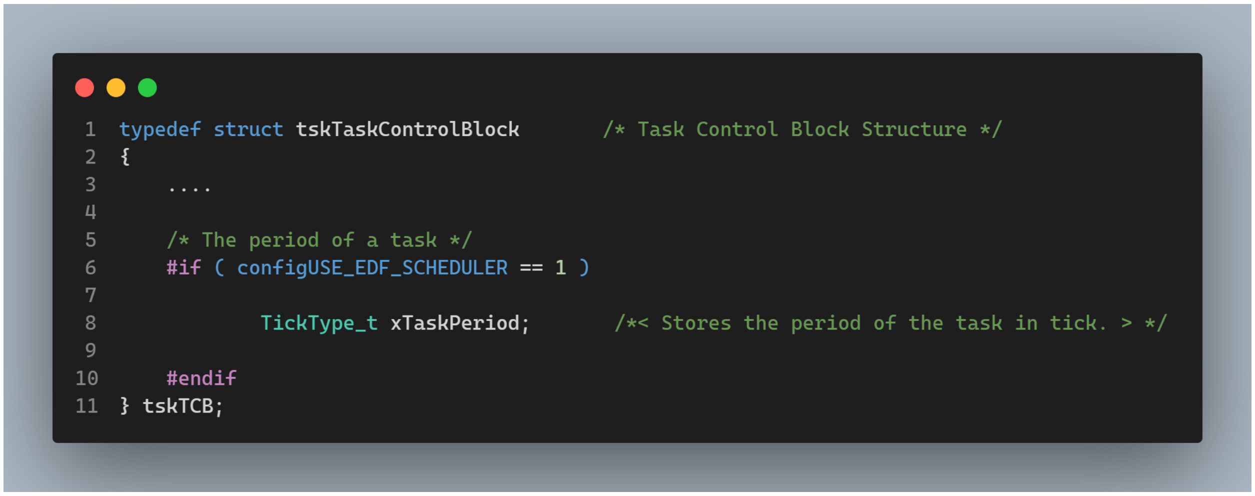
```c
static void prvInitialiseTaskLists( void )
{
    ....

        /* Initializing the new Ready List */
        #if ( configUSE_EDF_SCHEDULER == 1 )
                {
                    vListInitialise( &xReadyTasksListEDF );
                }
        #endif /* configUSE_EDF_SCHEDULER */

    ....

}
```

- Adding tasks to the Ready List according to their deadlines.



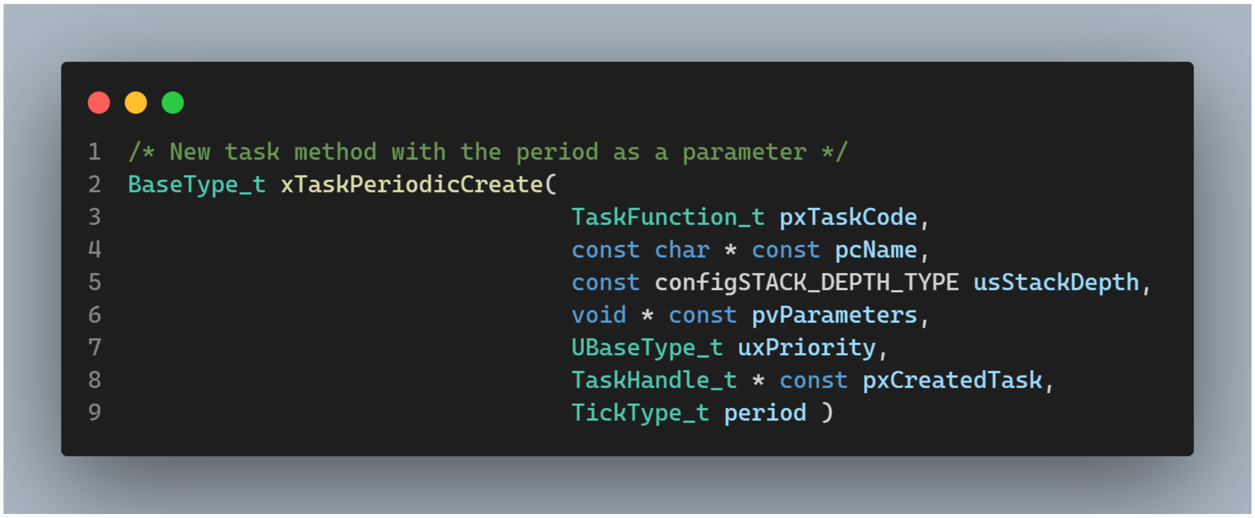
```
/*
 * Place the task represented by pxTCB into the appropriate ready list for
 * the task.  It is inserted at the end of the list.
 */

#if (configUSE_EDF_SCHEDULER == 0)
        #define prvAddTaskToReadyList( pxTCB )
                ....
                listINSERT_END( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB )->xStateListItem ) );
                ....

#else /* configUSE_EDF_SCHEDULER == 1 */
        #define prvAddTaskToReadyList( pxTCB ) /*xStateListItem must contain the deadline value */              \
                vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xStateListItem ) )
#endif
```

- A new period variable is added in the **tskTaskControlBlock.**



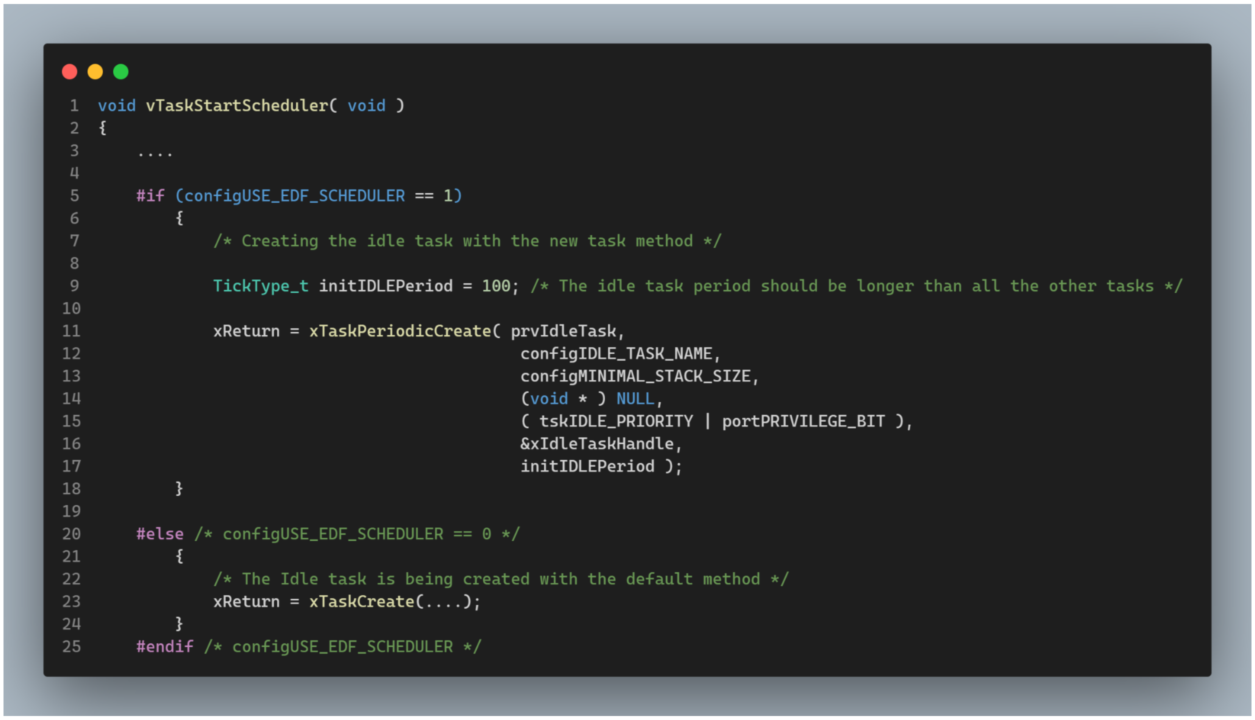
```
typedef struct tskTaskControlBlock       /* Task Control Block Structure */
{
    ....

    /* The period of a task */
    #if ( configUSE_EDF_SCHEDULER == 1 )

            TickType_t xTaskPeriod;       /*< Stores the period of the task in tick. > */

    #endif
} tskTCB;
```

- A new initialization task method is created **xTaskPeriodicCreate().**



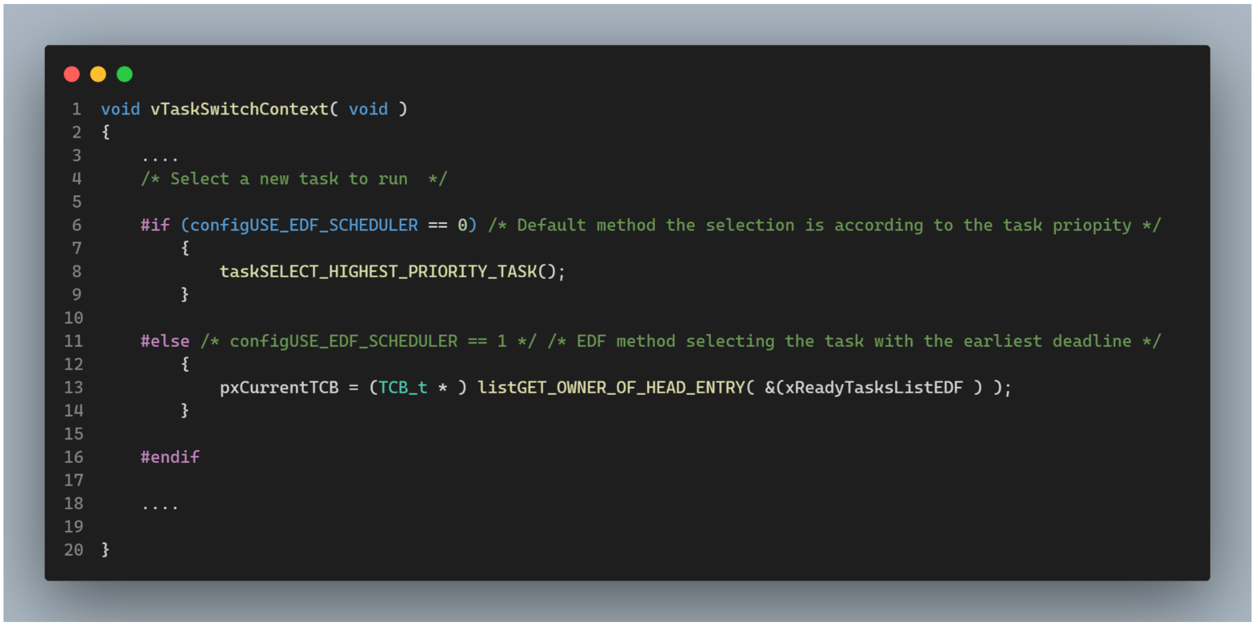
```
/* New task method with the period as a parameter */
BaseType_t xTaskPeriodicCreate(
                                TaskFunction_t pxTaskCode,
                                const char * const pcName,
                                const configSTACK_DEPTH_TYPE usStackDepth,
                                void * const pvParameters,
                                UBaseType_t uxPriority,
                                TaskHandle_t * const pxCreatedTask,
                                TickType_t period )
```

- Creating the idle task with the new task method in **vTaskStartScheduler().**



```c
void vTaskStartScheduler( void )
{
    ....

    #if (configUSE_EDF_SCHEDULER == 1)
        {
            /* Creating the idle task with the new task method */

            TickType_t initIDLEPeriod = 100; /* The idle task period should be longer than all the other tasks */

            xReturn = xTaskPeriodicCreate( prvIdleTask,
                                            configIDLE_TASK_NAME,
                                            configMINIMAL_STACK_SIZE,
                                            (void * ) NULL,
                                            ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ),
                                            &xIdleTaskHandle,
                                            initIDLEPeriod );
        }

    #else /* configUSE_EDF_SCHEDULER == 0 */
        {
            /* The Idle task is being created with the default method */
            xReturn = xTaskCreate(....);
        }
    #endif /* configUSE_EDF_SCHEDULER */
```

- **taskSELECTHIGHESTPRIORITYTASK()** method in **vTaskSwitchContext** is replaced in order to assign to **pxCurrentTCB** the task at the first place of the new Ready List.



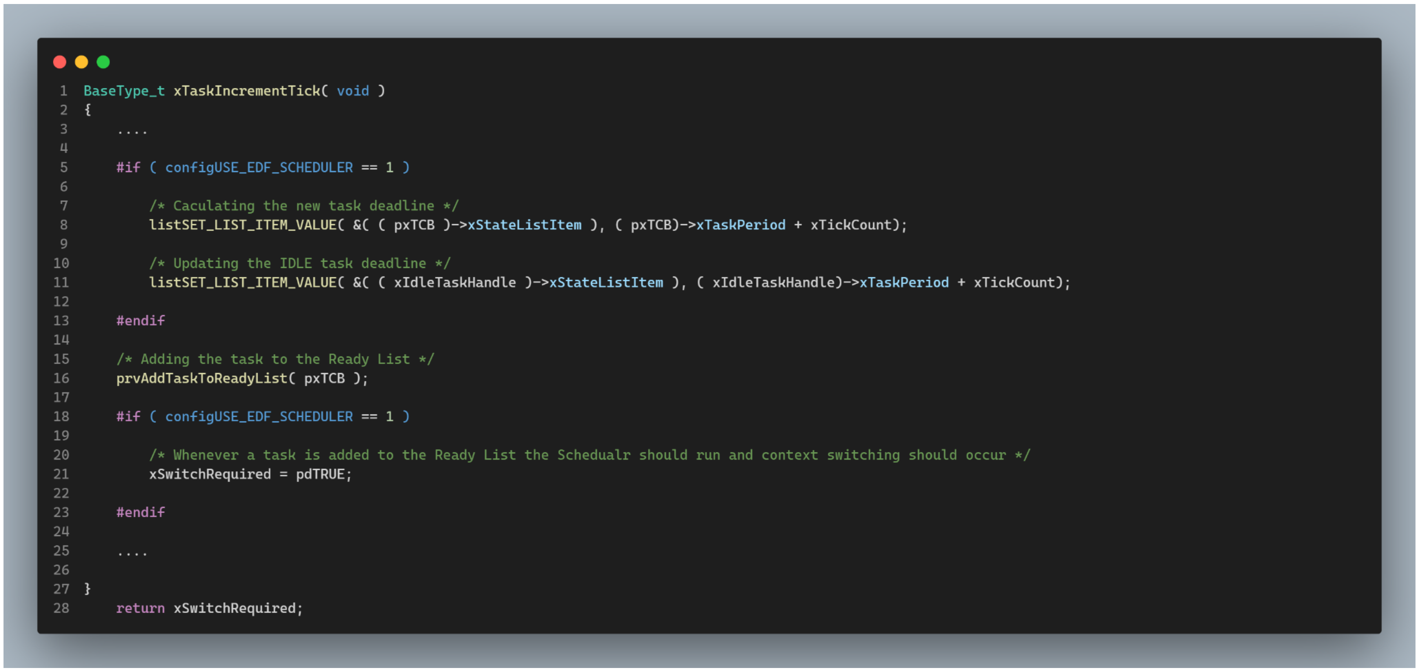
```c
void vTaskSwitchContext( void )
{
    ....
    /* Select a new task to run  */

    #if (configUSE_EDF_SCHEDULER == 0) /* Default method the selection is according to the task priopity */
        {
            taskSELECT_HIGHEST_PRIORITY_TASK();
        }

    #else /* configUSE_EDF_SCHEDULER == 1 */ /* EDF method selecting the task with the earliest deadline */
        {
            pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );
        }

    #endif

    ....

}
```

UPDATING THE IMPLEMENTATION

There are three main changes that have to be made to the implementation in order to have a fully functioning EDF Scheduler.

- The deadline of the IDLE task should be always updating, to prevent the idle task from running while there are ready tasks .
- The new task deadline should be calculated before adding the task to the Ready List.
- Whenever a task is added to the ready list a context switching is required (The scheduler should be called)

All the previous updates were done in **xTaskIncrementTick()** because it looks like the optimal place to do all these changes.
With each tick it looks for a ready task, if there is any then it will calculate it's new deadline and will be added to the ready list while also updating the IDLE task deadline. Then a switch context will occur.



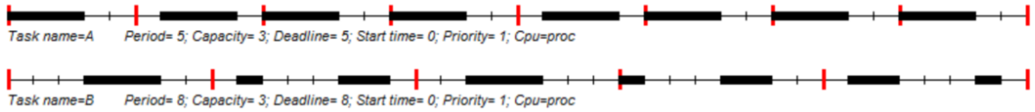
```c
BaseType_t xTaskIncrementTick( void )
{
    ....

    #if ( configUSE_EDF_SCHEDULER == 1 )

        /* Caculating the new task deadline */
        listSET_LIST_ITEM_VALUE( &( ( pxTCB )->xStateListItem ), ( pxTCB)->xTaskPeriod + xTickCount);

        /* Updating the IDLE task deadline */
        listSET_LIST_ITEM_VALUE( &( ( xIdleTaskHandle )->xStateListItem ), ( xIdleTaskHandle)->xTaskPeriod + xTickCount);

    #endif

    /* Adding the task to the Ready List */
    prvAddTaskToReadyList( pxTCB );

    #if ( configUSE_EDF_SCHEDULER == 1 )

        /* Whenever a task is added to the Ready List the Schedualr should run and context switching should occur */
        xSwitchRequired = pdTRUE;

    #endif

    ....

}
    return xSwitchRequired;
```

TASKS

TASKS	PERIODICITY	DEADLINE	EXECUTION TIME	PRIORITY
TASK 1	5 ms	5 ms	3 ms	1
TASK 2	8 ms	8 ms	3 ms	1

Testing the EDF Scheduler will be through these two tasks which were taken from the thesis.

EDF scheduling of task 1 (T=5, D=5, C=3) and task 2 (T=8, D=8 C=3)



The detection of a perfectly working EDF will be at tick = 10 when Task 2 is running but Task 1 becomes ready and the deadline of TASK 1 is 15 while TASK 2 is 16, so the scheduler should pick TASK 1 because it has the earliest deadline.

HYPERPERIOD

Hyperperiod = LCM(Pi)   which is 40 ms

URM CALCULATIONS

$$U = \sum_{i-1}^{n} \frac{Ci}{Pi} \leq n(2^{\frac{1}{n}} - 1)$$

U = (3 / 5) + (3 / 8) = 0.6 + 0.375
U = 0.975

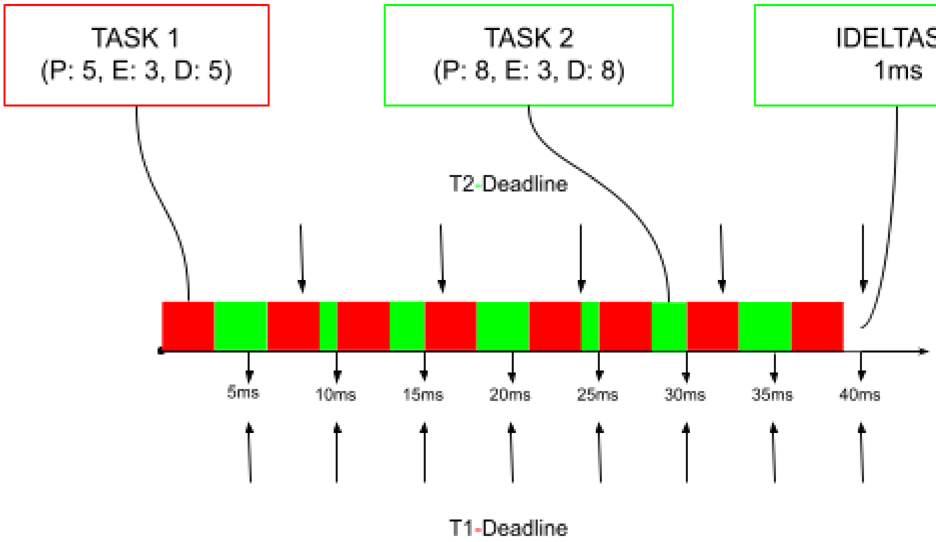
Urm = $2(2^{\frac{1}{2}} - 1)$ = 0.83

U > Urm

System guaranteed **NOT** schedulable in a **Rate-Monotonic Scheduler**.

CPU LOAD / SYSTEM SCHEDULABILITY MANUALLY

Applying the previous calculations we can plot out the tasks on this 40 ms hyperperiod timeline (Taking in consideration it’s a EDF Scheduler):



CPU manual calculations:

R (Busy Time)   = (8*3) + (5*3)

= 39 ms

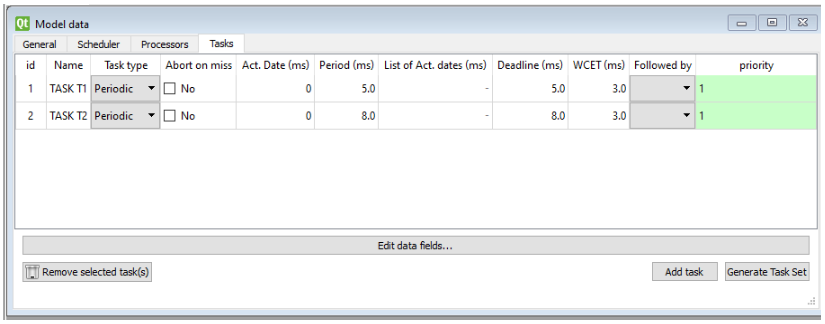
C (Total Time) = 40 ms

CPU Load = R/C = 39 / 40 = 97.5%

According to the timeline and cpu load calculations the system is loaded but barely schedulable. Every task comes at the right time and no task misses the deadline.

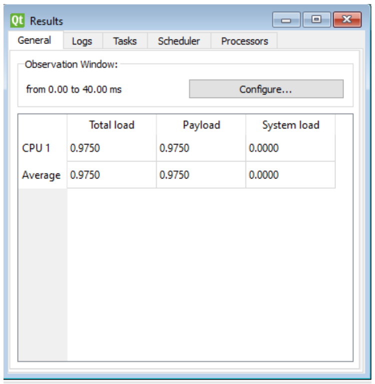
CPU LOAD / SYSTEM SCHEDULABILITY ON SIMULATION

Set the Tasks with the periodicity, deadline, execution time, and priority.



id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)	Followed by	priority
1	TASK T1	Periodic	□ No	0	5.0	-	5.0	3.0		1
2	TASK T2	Periodic	□ No	0	8.0	-	8.0	3.0		1

The CPU load is as it was manually calculated and that confirms the calculations.

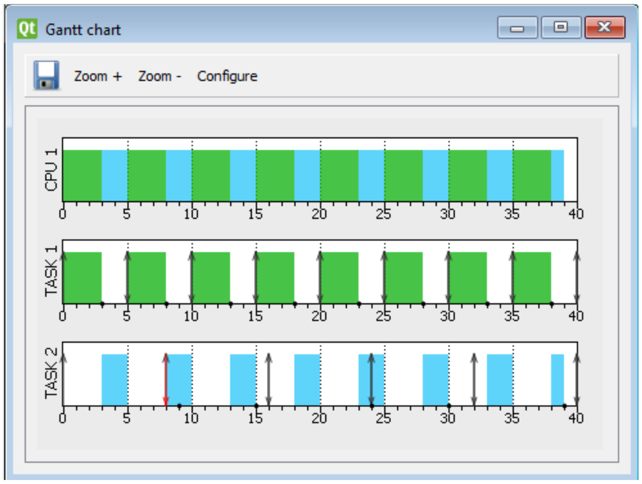


Observation Window:

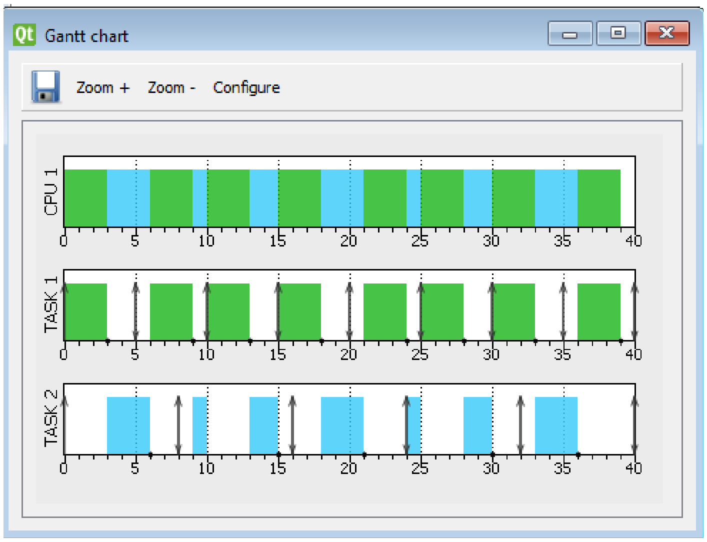
from 0.00 to 40.00 ms

	Total load	Payload	System load
CPU 1	0.9750	0.9750	0.0000
Average	0.9750	0.9750	0.0000

Using the **Rate-Monotonic Scheduler** we can clearly see that it is **NOT** schedulable as we calculated with URM. TASK 2 missed the first deadline at tick = 8.



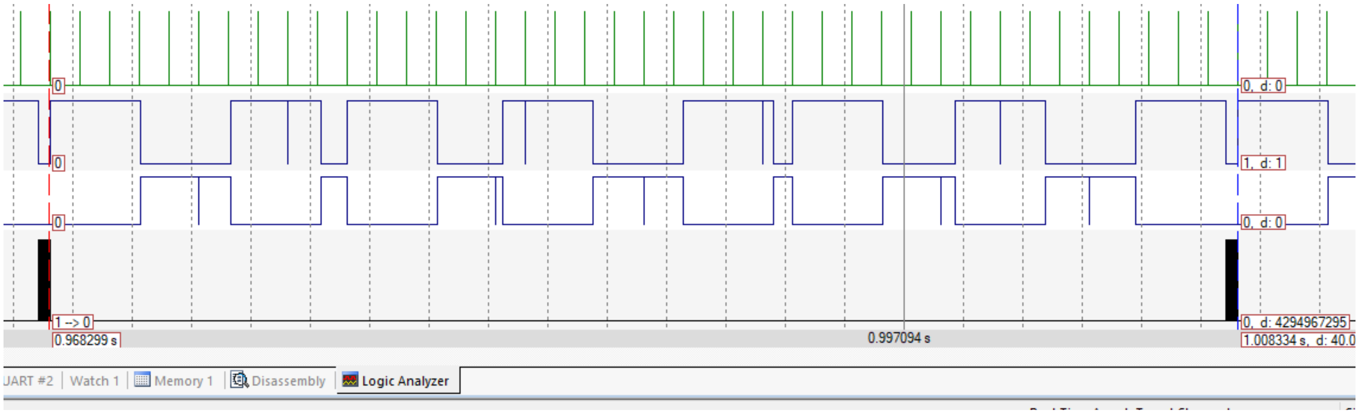
On the other hand using the **EDF Scheduler**, the simulation timeline is identical to the manually drawn timeline, and confirms that the system **IS** schedulable and no task missed the deadline.

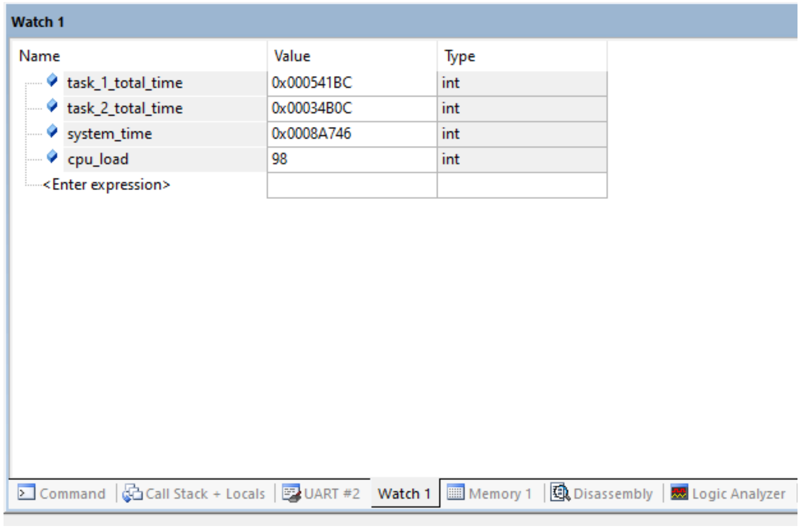


RUN-TIME ANALYSIS

To confirm that the implemented EDF Scheduler is working as it is intended, the run time analysis should be identical to the previous EDF simulation output.

With the help of trace hooks, we can clearly see in this 40 ms hyperperiod that the implemented EDF scheduler is working exactly as it is supposed to. Also the cpu load is identical to the manually calculated and simulated results.





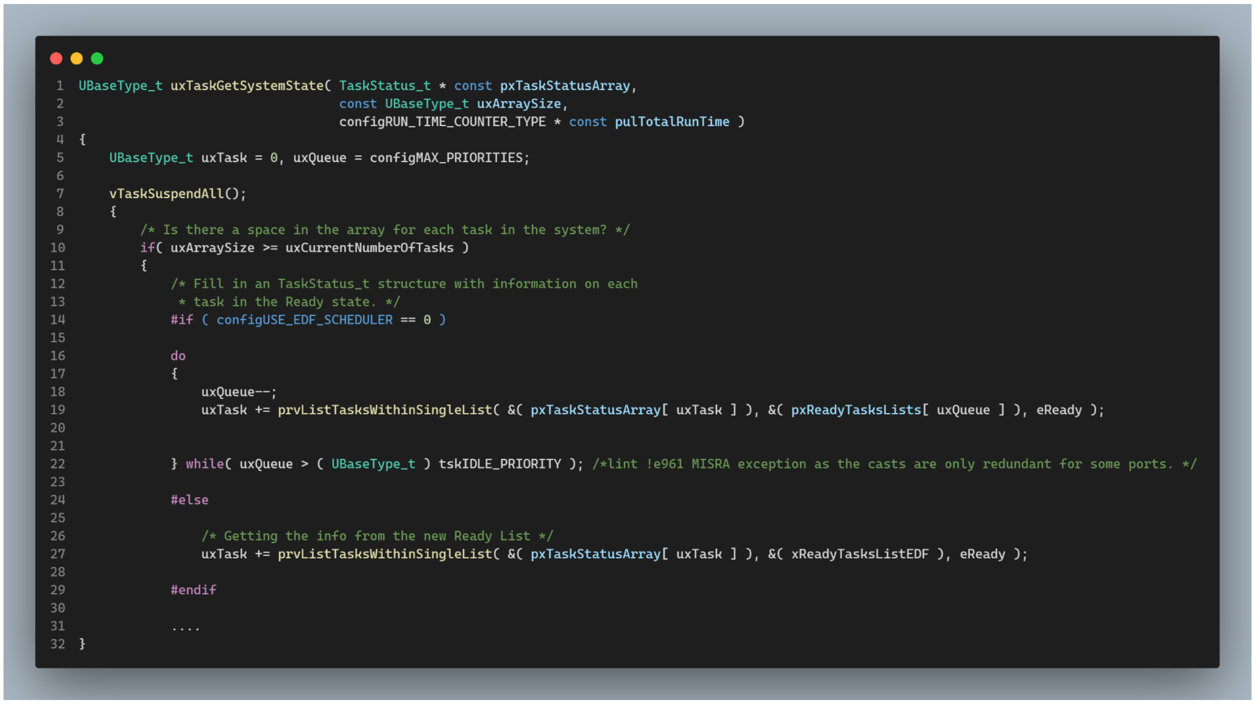
Name	Value	Type
task_1_total_time	0x000541BC	int
task_2_total_time	0x00034B0C	int
system_time	0x0008A746	int
cpu_load	98	int
<Enter expression>		

BOUNS:

To modify FreeRTOS **vTaskGetRunTimeStats** function to read the new **ReadyListEDF** and gather the required information about the tasks.

Following the **vTaskGetRunTimeStats** we can see that it calls **uxTaskGetSystemState** to get info about the tasks. The old Ready List was a bunch of lists in a list (each one for a certain priority)

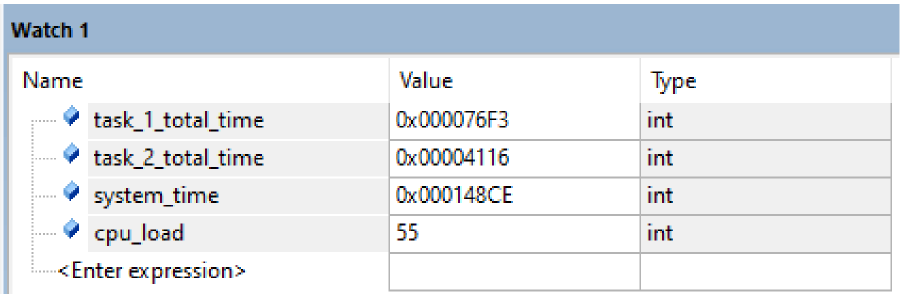
So this is the part where it needs to be modified because the new Ready List is just one list with no more lists inside it.



```
UBaseType_t uxTaskGetSystemState( TaskStatus_t * const pxTaskStatusArray,
                                  const UBaseType_t uxArraySize,
                                  configRUN_TIME_COUNTER_TYPE * const pulTotalRunTime )
{
    UBaseType_t uxTask = 0, uxQueue = configMAX_PRIORITIES;

    vTaskSuspendAll();
    {
        /* Is there a space in the array for each task in the system? */
        if( uxArraySize >= uxCurrentNumberOfTasks )
        {
            /* Fill in an TaskStatus_t structure with information on each
             * task in the Ready state. */
            #if ( configUSE_EDF_SCHEDULER == 0 )

            do
            {
                uxQueue--;
                uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &( pxReadyTasksLists[ uxQueue ] ), eReady );


            } while( uxQueue > ( UBaseType_t ) tskIDLE_PRIORITY ); /*lint !e961 MISRA exception as the casts are only redundant for some ports. */

            #else

                /* Getting the info from the new Ready List */
                uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &( xReadyTasksListEDF ), eReady );

            #endif

            ....
}
```

I had to change the execution time of both tasks because the system was loaded so adding **vTaskGetRunTimeStats** made the system not schedule.

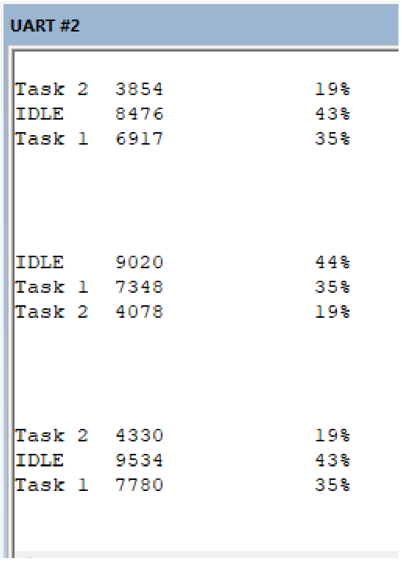
New CPU load: 55%



Watch 1

Name	Value	Type
task_1_total_time	0x000076F3	int
task_2_total_time	0x00004116	int
system_time	0x000148CE	int
cpu_load	55	int
<Enter expression>		

vTaskGetRunTimeStats OUTPUT:
The numbers do add up to confirm the previous cpu load calculated.



```
Task 2   3854            19%
IDLE     8476            43%
Task 1   6917            35%



IDLE     9020            44%
Task 1   7348            35%
Task 2   4078            19%



Task 2   4330            19%
IDLE     9534            43%
Task 1   7780            35%
```

CONCLUSION

The EDF Scheduler was successfully implemented and tested with a task set that utilizes the EDF scheduling.

The selected tasks system was verified for its schedulability with three different methods: manual analytical calculations, offline simulator, and in Run-Time analysis.

The implemented EDF Scheduler lived up to the expectations, and passed all the tests.

Further dive and updates to the source code were made to change the run time stats so that even the run time analysis could be made by the schedler itself and printed out the cpu and tasks stats.