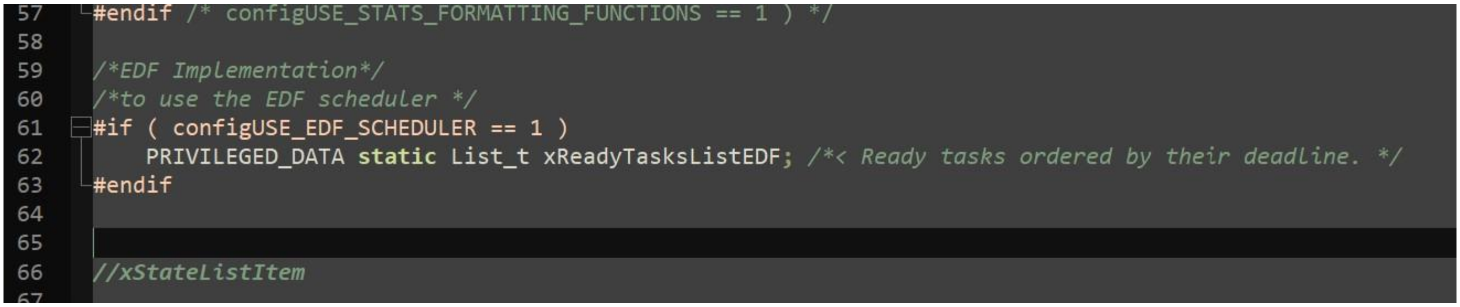
# Implementation of EDF from the Thesis:

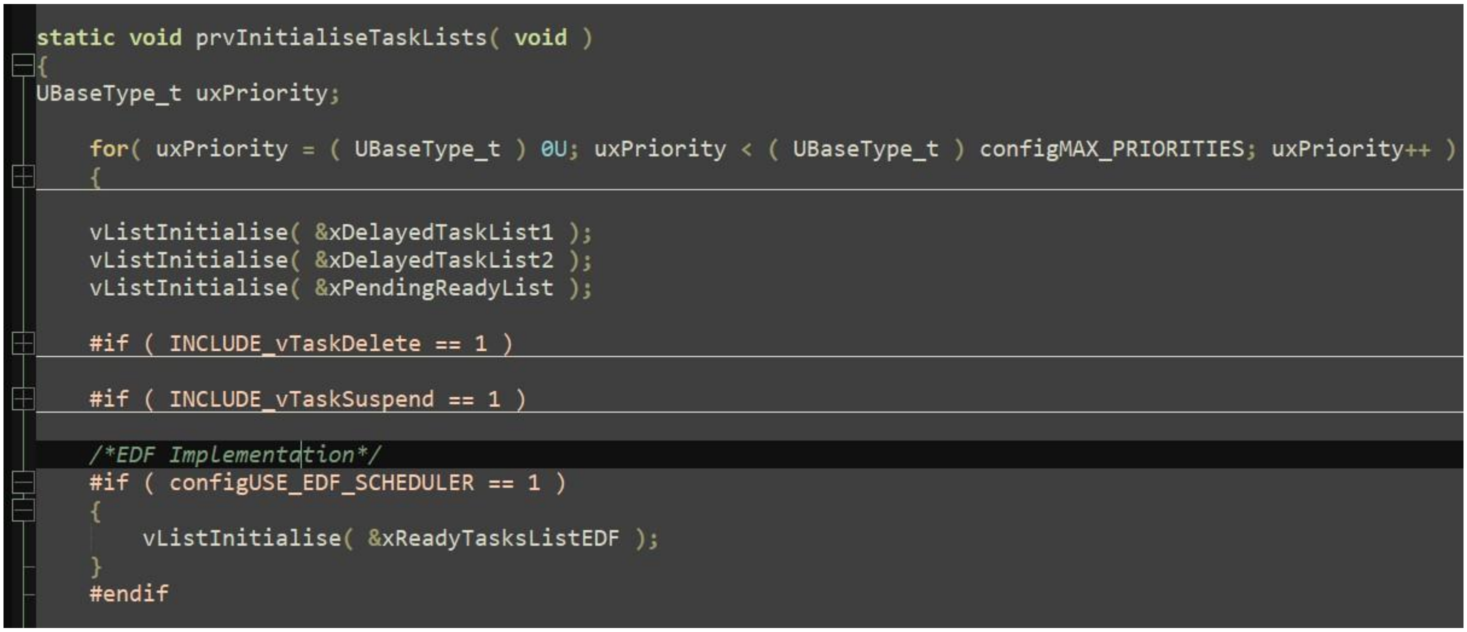
All changes made in Tasks.c & executed when the macro (configUSE_EDF_SCHEDULER) is enabled

First of all, the new Ready List is declared: *xReadyTasksListEDF* is a simple list structure.



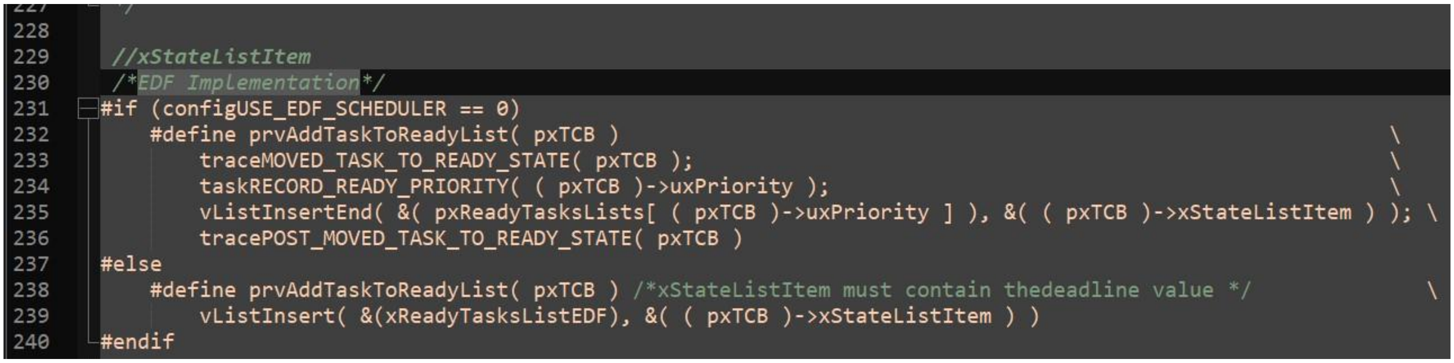
```
#endif /* configUSE_STATS_FORMATTING_FUNCTIONS == 1 ) */

/*EDF Implementation*/
/*to use the EDF scheduler */
#if ( configUSE_EDF_SCHEDULER == 1 )
    PRIVILEGED_DATA static List_t xReadyTasksListEDF; /*< Ready tasks ordered by their deadline. */
#endif


//xStateListItem
```

Then, the *prvInitialiseTaskLists*() method is modified adding the initialization of *xReadyTasksListEDF* :



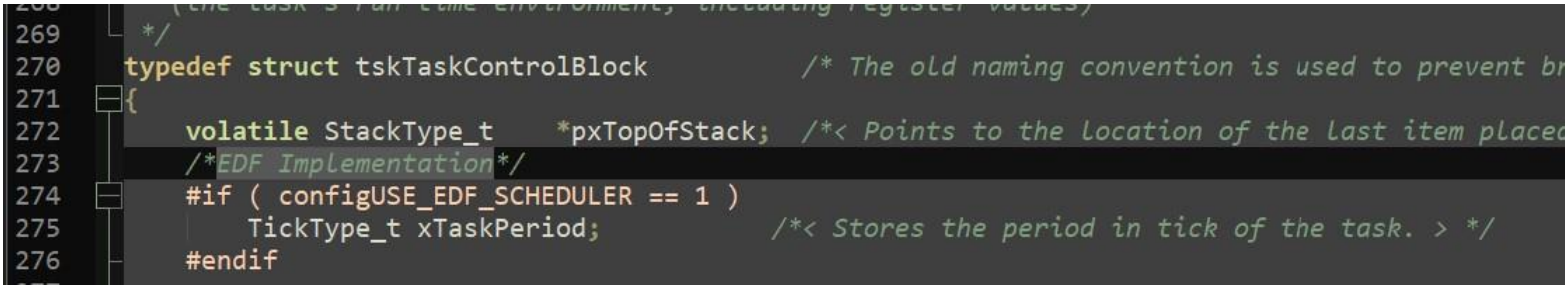
```
static void prvInitialiseTaskLists( void )
{
UBaseType_t uxPriority;

    for( uxPriority = ( UBaseType_t ) 0U; uxPriority < ( UBaseType_t ) configMAX_PRIORITIES; uxPriority++ )
    {

    vListInitialise( &xDelayedTaskList1 );
    vListInitialise( &xDelayedTaskList2 );
    vListInitialise( &xPendingReadyList );

    #if ( INCLUDE_vTaskDelete == 1 )

    #if ( INCLUDE_vTaskSuspend == 1 )

    /*EDF Implementation*/
    #if ( configUSE_EDF_SCHEDULER == 1 )
    {
        vListInitialise( &xReadyTasksListEDF );
    }
    #endif
```

prvAddTaskToReadyList*() method that adds a task to the Ready List is then modified*



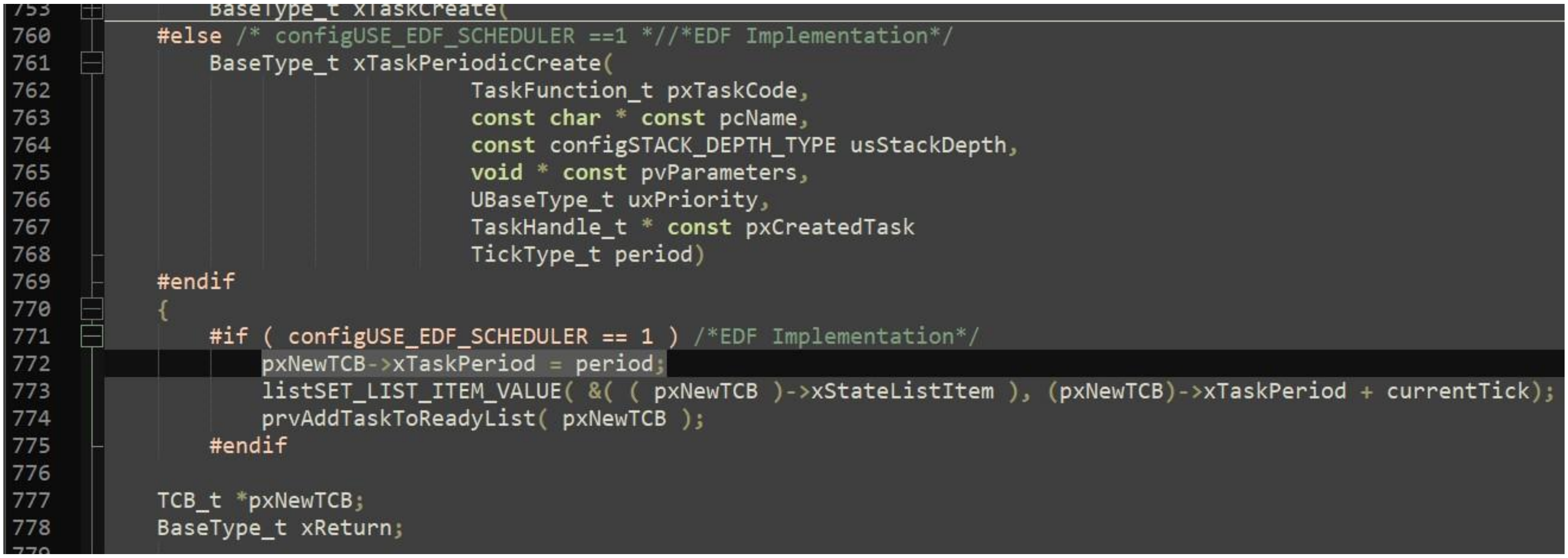
```
*/

 //xStateListItem
 /*EDF Implementation*/
#if (configUSE_EDF_SCHEDULER == 0)
    #define prvAddTaskToReadyList( pxTCB )                                                          \
        traceMOVED_TASK_TO_READY_STATE( pxTCB );                                                    \
        taskRECORD_READY_PRIORITY( ( pxTCB )->uxPriority );                                         \
        vListInsertEnd( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB )->xStateListItem ) ); \
        tracePOST_MOVED_TASK_TO_READY_STATE( pxTCB )
#else
    #define prvAddTaskToReadyList( pxTCB ) /*xStateListItem must contain thedeadline value */          \
        vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xStateListItem ) )
#endif
```

A new variable is added in the tskTaskControlBlock structure (TCB):



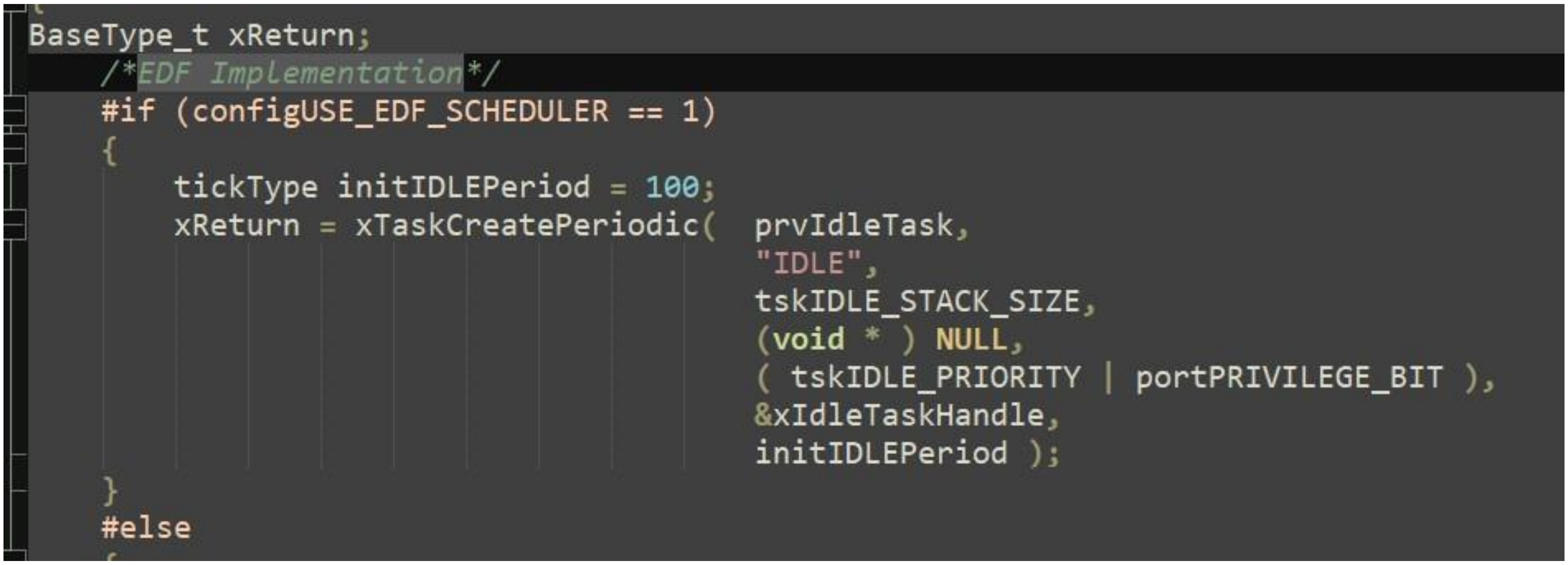
```
    (the task's run time environment, including register values)
 */
typedef struct tskTaskControlBlock          /* The old naming convention is used to prevent br
{
    volatile StackType_t    *pxTopOfStack;  /*< Points to the location of the last item placed
    /*EDF Implementation*/
    #if ( configUSE_EDF_SCHEDULER == 1 )
        TickType_t xTaskPeriod;             /*< Stores the period in tick of the task. > */
    #endif
```

xTaskPeriodicCreate*() is a modified version of the standard method* xTaskCreate*()*



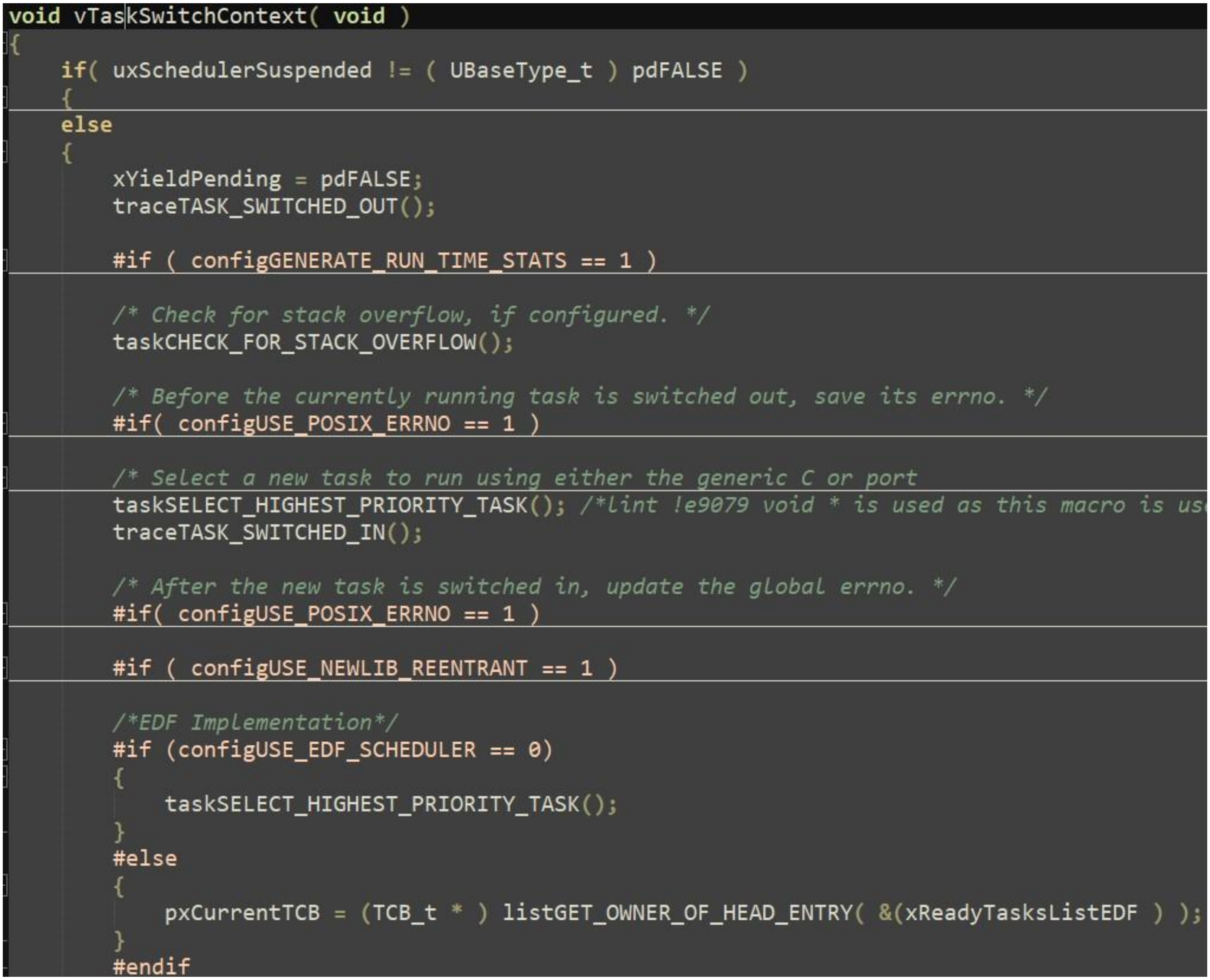
```
        BaseType_t xTaskCreate(
    #else /* configUSE_EDF_SCHEDULER ==1 *//*EDF Implementation*/
        BaseType_t xTaskPeriodicCreate(
                            TaskFunction_t pxTaskCode,
                            const char * const pcName,
                            const configSTACK_DEPTH_TYPE usStackDepth,
                            void * const pvParameters,
                            UBaseType_t uxPriority,
                            TaskHandle_t * const pxCreatedTask
                            TickType_t period)
    #endif
    {
        #if ( configUSE_EDF_SCHEDULER == 1 ) /*EDF Implementation*/
            pxNewTCB->xTaskPeriod = period;
            listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), (pxNewTCB)->xTaskPeriod + currentTick);
            prvAddTaskToReadyList( pxNewTCB );
        #endif

    TCB_t *pxNewTCB;
    BaseType_t xReturn;
```

The IDLE task management modification



```
BaseType_t xReturn;
    /*EDF Implementation*/
    #if (configUSE_EDF_SCHEDULER == 1)
    {
        tickType initIDLEPeriod = 100;
        xReturn = xTaskCreatePeriodic(  prvIdleTask,
                                        "IDLE",
                                        tskIDLE_STACK_SIZE,
                                        (void * ) NULL,
                                        ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ),
                                        &xIdleTaskHandle,
                                        initIDLEPeriod );
    }
    #else
```

Last change needed involves the switch context mechanism.

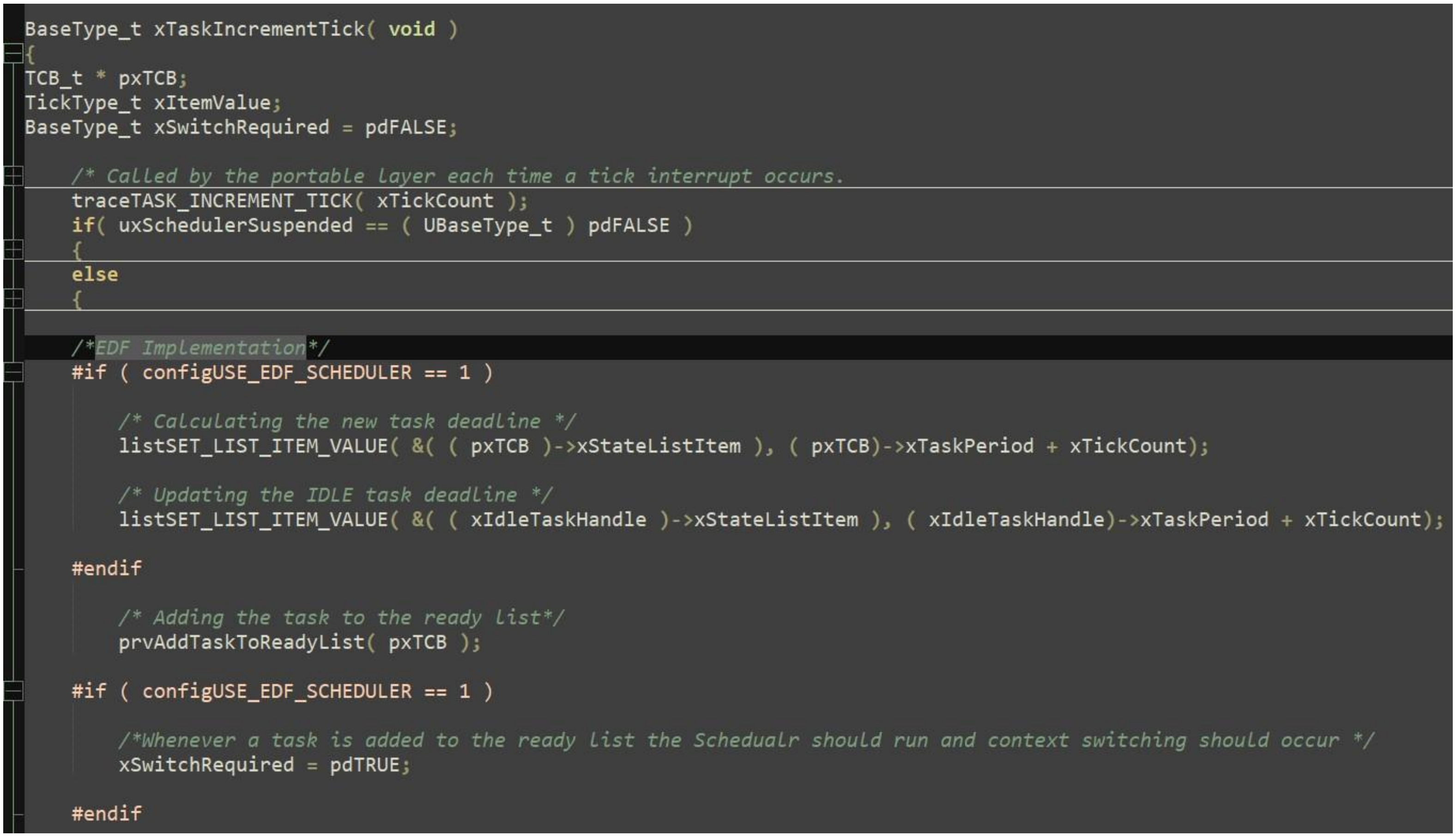


```
void vTaskSwitchContext( void )
{
    if( uxSchedulerSuspended != ( UBaseType_t ) pdFALSE )
    {
    else
    {
        xYieldPending = pdFALSE;
        traceTASK_SWITCHED_OUT();

        #if ( configGENERATE_RUN_TIME_STATS == 1 )

        /* Check for stack overflow, if configured. */
        taskCHECK_FOR_STACK_OVERFLOW();

        /* Before the currently running task is switched out, save its errno. */
        #if( configUSE_POSIX_ERRNO == 1 )

        /* Select a new task to run using either the generic C or port
        taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is us
        traceTASK_SWITCHED_IN();

        /* After the new task is switched in, update the global errno. */
        #if( configUSE_POSIX_ERRNO == 1 )

        #if ( configUSE_NEWLIB_REENTRANT == 1 )

        /*EDF Implementation*/
        #if (configUSE_EDF_SCHEDULER == 0)
        {
            taskSELECT_HIGHEST_PRIORITY_TASK();
        }
        #else
        {
            pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );
        }
        #endif
```

The 5% remaining to operate

1. Updating The deadline of the IDLE task
2. Calculating the new task deadline
3. context switching is required when a new task is added

All these modification is made in xTaskIncrementTick()



```
BaseType_t xTaskIncrementTick( void )
{
TCB_t * pxTCB;
TickType_t xItemValue;
BaseType_t xSwitchRequired = pdFALSE;

    /* Called by the portable layer each time a tick interrupt occurs.
    traceTASK_INCREMENT_TICK( xTickCount );
    if( uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )
    {
    else
    {

    /*EDF Implementation*/
    #if ( configUSE_EDF_SCHEDULER == 1 )

        /* Calculating 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
```

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

T1 (P: 5, E: 3, D: 5)

T2 (P: 8, E: 3, D: 8)

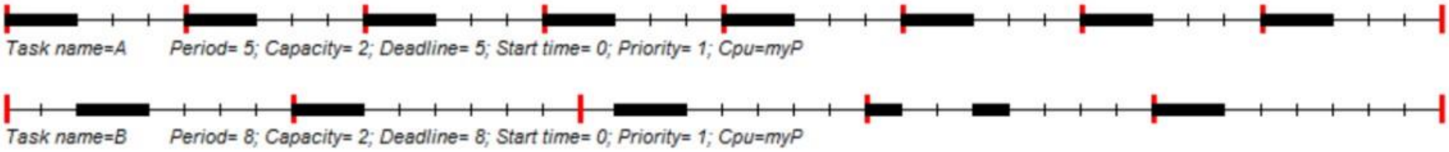


*Figure 1Representation from the thesis*

## Analytical methods:

Hyperperiod =8*5=40 ms

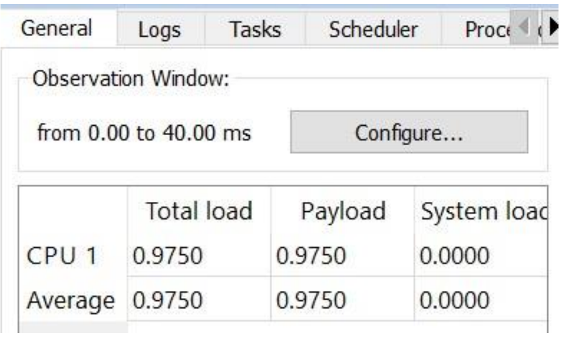
U =(3/5) + (3/8) = 0.975

Urm = 2*(2^(1/2) -1) = 0.828

U > Urm then the system is guaranteed to be not schedulable And will be a task that misses its deadline

Offline simulation:

The CPU load is as calculated

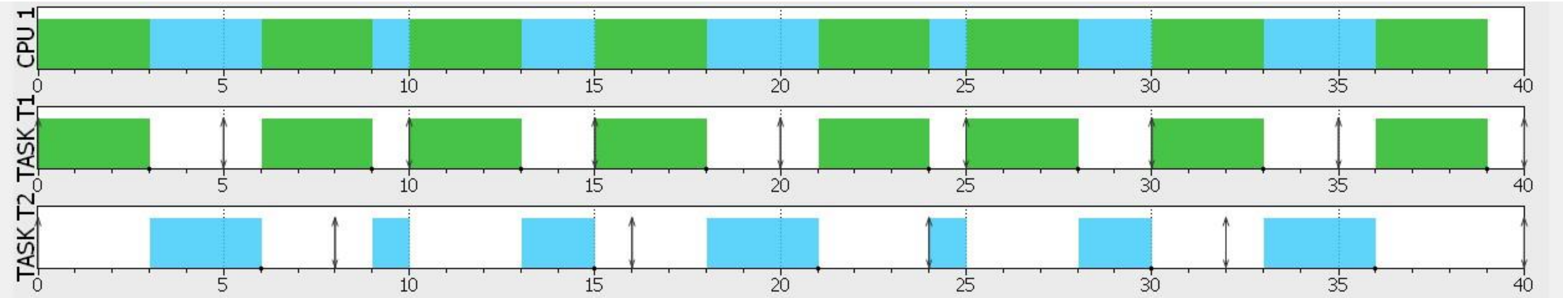


Observation Window:

from 0.00 to 40.00 ms

| | Total load | Payload | System loac |
|---|---|---|---|
| CPU 1 | 0.9750 | 0.9750 | 0.0000 |
| Average | 0.9750 | 0.9750 | 0.0000 |

The time line with EDF scheduler:



The system is guaranteed to be not schedulable with rate monotonic schedulers but schedulable with the EDF scheduler