Earliest Deadline First Scheduling Algorithm

Prepared by:Team6

Table of Contents

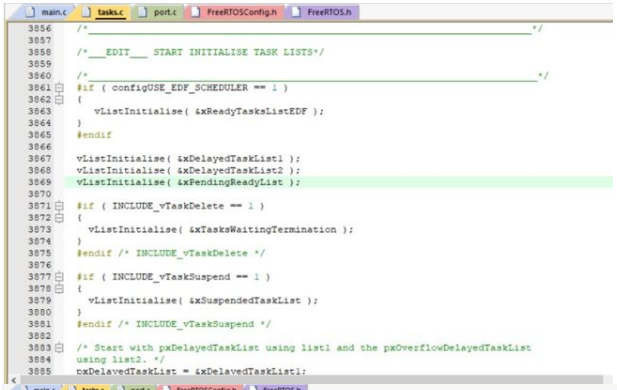
Project Description

The EDF “Earliest Deadline First” is a scheduling algorithm that adopts a dynamic priority-based preemptive scheduling policy, meaning that the priority of a task can change during its execution, and the processing of any task is interrupted by a request for any higher priority task.

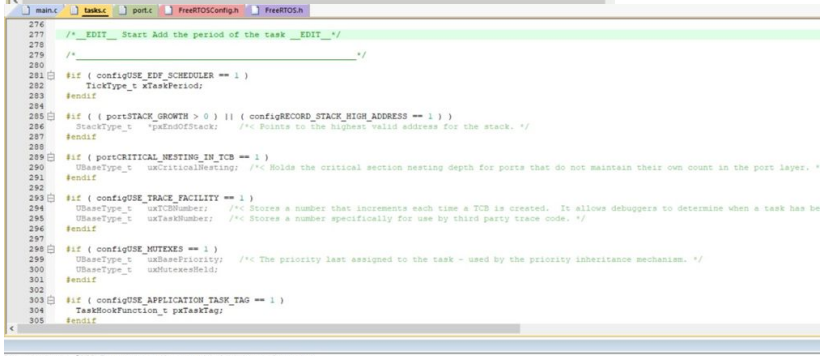
All the changes are implemented in the Tasks.c

The implementation adopts rising the priority of the tasks that have the earliest deadline which is calculated using the task period and the current tick.

Changes made in “Tasks.c”

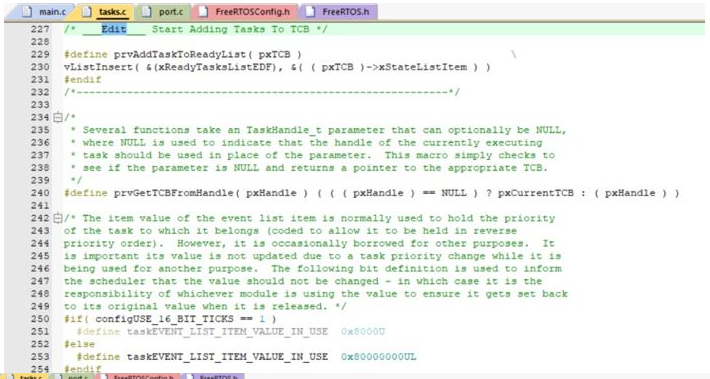


```
/*-----------------------------------------------------------*/

/*___EDIT___ START INITIALISE TASK LISTS*/

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

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

#if ( INCLUDE_vTaskDelete == 1 )
{
  vListInitialise( &xTasksWaitingTermination );
}
#endif /* INCLUDE_vTaskDelete */

#if ( INCLUDE_vTaskSuspend == 1 )
{
  vListInitialise( &xSuspendedTaskList );
}
#endif /* INCLUDE_vTaskSuspend */

/* Start with pxDelayedTaskList using list1 and the pxOverflowDelayedTaskList
using list2. */
pxDelayedTaskList = &xDelayedTaskList1;
```



```
/*__EDIT__ Start Add the period of the task __EDIT__*/

/*-----------------------------------------------------------*/
#if ( configUSE_EDF_SCHEDULER == 1 )
  TickType_t xTaskPeriod;
#endif

#if ( ( portSTACK_GROWTH > 0 ) || ( configRECORD_STACK_HIGH_ADDRESS == 1 ) )
  StackType_t  *pxEndOfStack;   /*< Points to the highest valid address for the stack. */
#endif

#if ( portCRITICAL_NESTING_IN_TCB == 1 )
  UBaseType_t  uxCriticalNesting;  /*< Holds the critical section nesting depth for ports that do not maintain their own count in the port layer. *
#endif

#if ( configUSE_TRACE_FACILITY == 1 )
  UBaseType_t  uxTCBNumber;    /*< Stores a number that increments each time a TCB is created.  It allows debuggers to determine when a task has be
  UBaseType_t  uxTaskNumber;   /*< Stores a number specifically for use by third party trace code. */
#endif

#if ( configUSE_MUTEXES == 1 )
  UBaseType_t  uxBasePriority;   /*< The priority last assigned to the task - used by the priority inheritance mechanism. */
  UBaseType_t  uxMutexesHeld;
#endif

#if ( configUSE_APPLICATION_TASK_TAG == 1 )
  TaskHookFunction_t pxTaskTag;
#endif
```

0): warning: #188-D: enumerated type mixed with another type



```
/* __Edit__ Start Adding Tasks To TCB */

#define prvAddTaskToReadyList( pxTCB )                                  \
vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xStateListItem ) )
#endif
/*-----------------------------------------------------------*/

/*
 * Several functions take an TaskHandle_t parameter that can optionally be NULL,
 * where NULL is used to indicate that the handle of the currently executing
 * task should be used in place of the parameter.  This macro simply checks to
 * see if the parameter is NULL and returns a pointer to the appropriate TCB.
 */
#define prvGetTCBFromHandle( pxHandle ) ( ( ( pxHandle ) == NULL ) ? pxCurrentTCB : ( pxHandle ) )

/* The item value of the event list item is normally used to hold the priority
of the task to which it belongs (coded to allow it to be held in reverse
priority order).  However, it is occasionally borrowed for other purposes.  It
is important its value is not updated due to a task priority change while it is
being used for another purpose.  The following bit definition is used to inform
the scheduler that the value should not be changed - in which case it is the
responsibility of whichever module is using the value to ensure it gets set back
to its original value when it is released. */
#if( configUSE_16_BIT_TICKS == 1 )
  #define taskEVENT_LIST_ITEM_VALUE_IN_USE  0x8000U
#else
  #define taskEVENT_LIST_ITEM_VALUE_IN_USE  0x80000000UL
#endif
```

```
/*__EDIT__ DECLARE LIST OF TASKS LIST EDF */

/*-----------------------------------------------------------*/
#if ( configUSE_EDF_SCHEDULER == 1 )
PRIVILEGED_DATA static List_t xReadyTasksListEDF;
#endif
PRIVILEGED_DATA static List_t xDelayedTaskList1;          /*< Delayed tasks. */
PRIVILEGED_DATA static List_t xDelayedTaskList2;          /*< Delayed tasks (two lists are used - one for delays that have overflowed the current tick c
PRIVILEGED_DATA static List_t * volatile pxDelayedTaskList;   /*< Points to the delayed task list currently being used. */
PRIVILEGED_DATA static List_t * volatile pxOverflowDelayedTaskList;   /*< Points to the delayed task list currently being used to hold tasks that have ove
PRIVILEGED_DATA static List_t xPendingReadyList;          /*< Tasks that have been readied while the scheduler was suspended.  They will be moved to the

#if( INCLUDE_vTaskDelete == 1 )

  PRIVILEGED_DATA static List_t xTasksWaitingTermination;      /*< Tasks that have been deleted - but their memory not yet freed. */
  PRIVILEGED_DATA static volatile UBaseType_t uxDeletedTasksWaitingCleanUp = ( UBaseType_t ) 0U;

#endif

#if ( INCLUDE_vTaskSuspend == 1 )

  PRIVILEGED_DATA static List_t xSuspendedTaskList;        /*< Tasks that are currently suspended. */

#endif

/* Global POSIX errno. Its value is changed upon context switching to match
 the errno of the currently running task. */
#if ( configUSE_POSIX_ERRNO == 1 )
  int FreeRTOS_errno = 0;
#endif
```

Changes made in “Tasks.c”

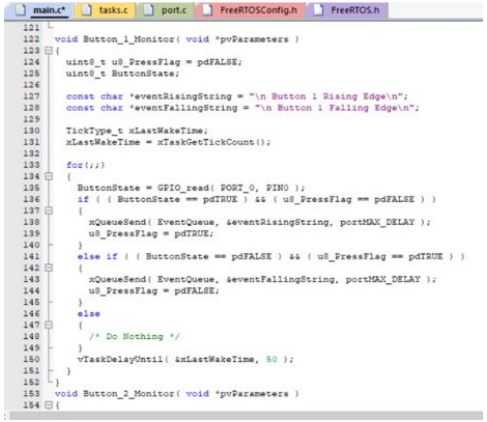
```
928     /*__EDIT___ Initialising The period of Task*/
929
930     /*_____________________________________________________________________*/
931
932     pxNewTCB->xTaskPeriod = period;
933
934     prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, pxNewTCB, NULL );
935
936     /* __insert the period value in the generic list iteam before to add the task __*/
937     currentTick = xTaskGetTickCount();
938
939     listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), ( pxNewTCB )->xTaskPeriod + currentTick );
940
941     prvAddNewTaskToReadyList( pxNewTCB );
942
943     xReturn = pdPASS;
944   }
945   else
946   {
947     xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
948   }
949
950   return xReturn;
951 }
952
953 #endif /* configSUPPORT_DYNAMIC_ALLOCATION */
954 /*-----------------------------------------------------------*/
```

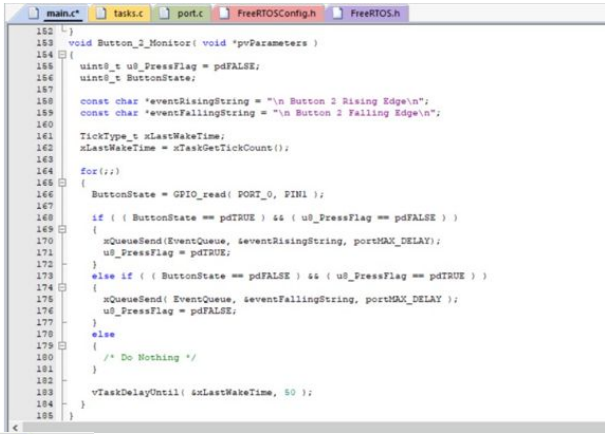
```
2145     /* __EDIT___ INITIALIZING IDLE TASK */
2146
2147     TickType_t initIDLEPeriod = ( TickType_t ) configIDLE_TSK_PERIOD;
2148     xReturn = xTaskPeriodicCreate( prvIdleTask,
2149                                    configIDLE_TASK_NAME,
2150                                    configMINIMAL_STACK_SIZE,
2151                                    ( void * ) NULL,
2152                                    ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ),
2153                                    &xIdleTaskHandle,
2154                                    initIDLEPeriod );
2155   }
2156   #else
2157   {
2158     xReturn = xTaskCreate(  prvIdleTask,
2159                 configIDLE_TASK_NAME,
2160                 configMINIMAL_STACK_SIZE,
2161                 ( void * ) NULL,
2162                 portPRIVILEGE_BIT, /* In effect ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), but tskIDLE_PRIORITY is zero. */
2163                 &xIdleTaskHandle ); /*lint !e961 MISRA exception, justified as it is not a redundant explicit cast to all sup
2164
2165   }
2166   #endif
2167 }
2168 #endif /* configSUPPORT_STATIC_ALLOCATION */
2169
2170 #if ( configUSE_TIMERS == 1 )
2171 {
2172   if( xReturn == pdPASS )
2173   {
2174     xReturn = xTimerCreateTimerTask();
```

```
3227     /*__ EDIT__ Check For Tasks that has the Earliest Dead Line */
3228
3229     /*_____________________________________________________________________*/
3230     #if (configUSE_EDF_SCHEDULER == 0)
3231     {
3232       taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-r
3233     }
3234     #else
3235     {
3236
3237       pxCurrentTCB = ( TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &( xReadyTasksListEDF ) );
3238     }
3239     #endif
3240     traceTASK_SWITCHED_IN();
3241
3242     /* After the new task is switched in, update the global errno. */
3243     #if( configUSE_POSIX_ERRNO == 1 )
3244     {
3245       FreeRTOS_errno = pxCurrentTCB->iTaskErrno;
3246     }
3247     #endif
3248
3249     #if ( configUSE_NEWLIB_REENTRANT == 1 )
3250     {
3251       /* Switch Newlib's _impure_ptr variable to point to the _reent
3252       structure specific to this task.
3253       See the third party link http://www.nadler.com/embedded/newlibAndFreeRTOS.html
3254       for additional information. */
3255       _impure_ptr = &( pxCurrentTCB->xNewLib_reent );
3256     }
```

```
3858     /*__EDIT___ START INITIALISE TASK LISTS*/
3859
3860     /*_____________________________________________________________________*/
3861     #if ( configUSE_EDF_SCHEDULER == 1 )
3862     {
3863       vListInitialise( &xReadyTasksListEDF );
3864     }
3865     #endif
3866
3867     vListInitialise( &xDelayedTaskList1 );
3868     vListInitialise( &xDelayedTaskList2 );
3869     vListInitialise( &xPendingReadyList );
3870
3871     #if ( INCLUDE_vTaskDelete == 1 )
3872     {
3873       vListInitialise( &xTasksWaitingTermination );
3874     }
3875     #endif /* INCLUDE_vTaskDelete */
3876
3877     #if ( INCLUDE_vTaskSuspend == 1 )
3878     {
3879       vListInitialise( &xSuspendedTaskList );
3880     }
3881     #endif /* INCLUDE_vTaskSuspend */
3882
3883     /* Start with pxDelayedTaskList using list1 and the pxOverflowDelayedTaskList
3884     using list2. */
3885     pxDelayedTaskList = &xDelayedTaskList1;
3886     pxOverflowDelayedTaskList = &xDelayedTaskList2;
3887 }
```

Tasks created





Verifying the system implementation using analytical methods

1. Calculating the Hyper-Period:

   Hyper period (H) = LCM(Pi) =100ms

2. Calculating the CPU load:

   U = R/C = (0.01 * 2+0.01 * 2+0.01* 1+0.01*5+5*10+12)/100= 0.621=62.1%

3. Check system schedulability using URM:

   U = 0.01/50 + 0.01/50 + 0.01/100 + 0.01/20 +5/10+ 12/100= 0.621

   URM = 6*(2^(⅙)-1) = 0.735

   U < URM So the system is schedulable

Verifying the system implementation using analytical methods

4. Check system schedulability using Time demand analysis:

Tasks Priority:

Task_5 > Task_4 > Task_1, Task_2 > Task_3, Task_6

Task_1 schedulability:

$W(50) = 0.01 + 50/20 * 0.01 + 50/10 * 5 = 25.04 < 50$ (Task_1 is schedulable)

Task_2 schedulability:

$W(50) = 25.04 < 50$ (Task_2 is schedulable)

Verifying the system implementation using analytical methods

Task_3 schedulability:

W(100) = 0.01 +(100/50) *0.01 + (100/50)*0.01 + (100/20)*0.01 + (100/10)*5 = 50.1<100 (Task_3 is schedulable)

Task_4 schedulability:

W(20) = 0.01+(20/10) * 5 = 10.01<20 (Task_4 is schedulable)

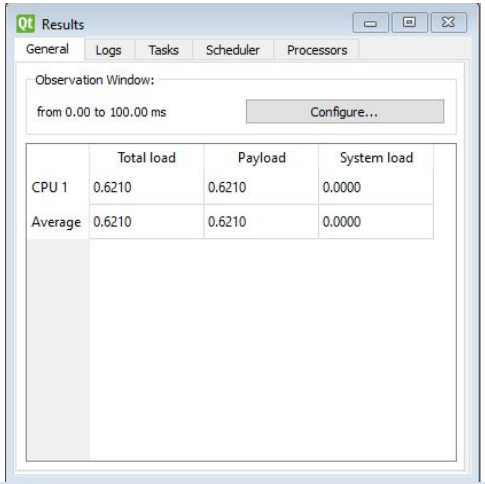
Task_5 schedulability:

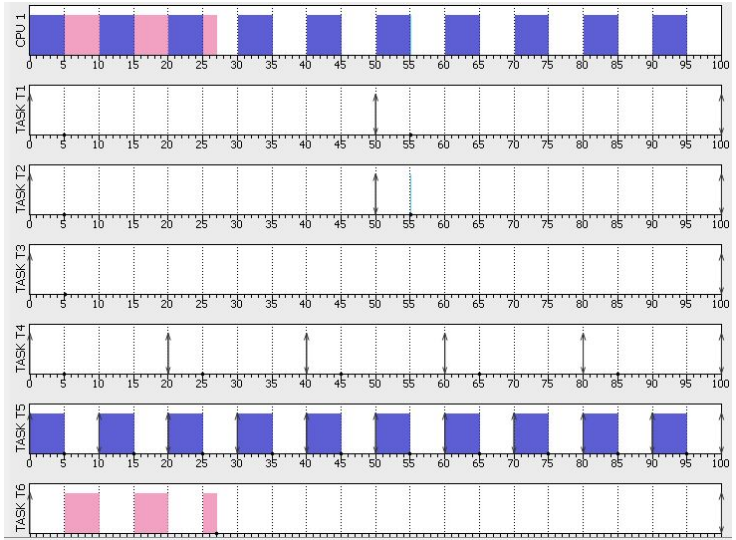
W(10) = 5 <10 (Task_5 is schedulable)

Task_6 schedulability:

W(100) = 12 + (100/5) *0.01 +(100/5) *0.01 +(100/20) *0.01 + (100/10) *5 =62.09<100 (Task_6 is schedulable)

Simulating the tasks using SimSo offline simulator





General | Scheduler | Processors | Tasks

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	☑ Yes	0.0	50.0	-	50.0	0.01		2
2	TASK T2	Periodic	☑ Yes	0	50	-	50	0.01		2
3	TASK T3	Periodic	☑ Yes	0	100	-	100	0.01		1
4	TASK T4	Periodic	☑ Yes	0	20	-	20	0.01		3
5	TASK T5	Periodic	☑ Yes	0	10	-	10	5		4
6	TASK T6	Periodic	☑ Yes	0	100		100	12		1

Simulating using the Keil simulator