Práctica 1

SISTEMAS TOLERANTES A FALLOS EN TIEMPO REAL

Sara Marcos Cornejo
José Francisco Romero Rodríguez

UNIVERSIDAD NEBRIJA

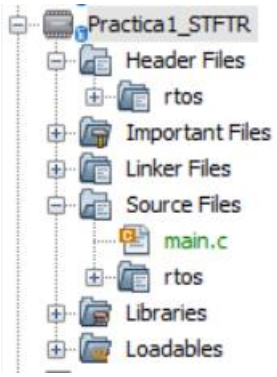
Índice:

Introducción:

En esta práctica se pide utlizar el entorno de MPLAB para crear 6 tareas/hilos diferentes, cada uno con prioridad distinta. Se utiliza el compilador XC de 32 bits, y varios archivos fuente (.c) y de cabecera (.h) ya proporcionados.

Procedimiento:

Para poder utilizar estos archivos correctamente, primero se debe acceder a la localización de la carpeta MPLABXProjects en nuestro PC, seleccionar el proyecto que acabamos de crear y dentro de nuestro proyecto crear una carpeta llamada **rtos**. Aquí se van a almacenar todos estos archivos .h y .c. Dentro de esta carpeta se crean dos nuevas carpetas, llamadas **include** y **source** (dentro de include se guardan todos los archivos .h, y en source todos los archivos .c). Una vez hecho esto, la configuración del explorador de archivos estaría lista, pero si se intenta abrir el proyecto desde MPLAB no aparecerán las modificaciones realizadas en cuanto a carpetas. Es por ello que en MPLAB se debe crear también la carpeta rtos correspondiente de la siguiente forma:

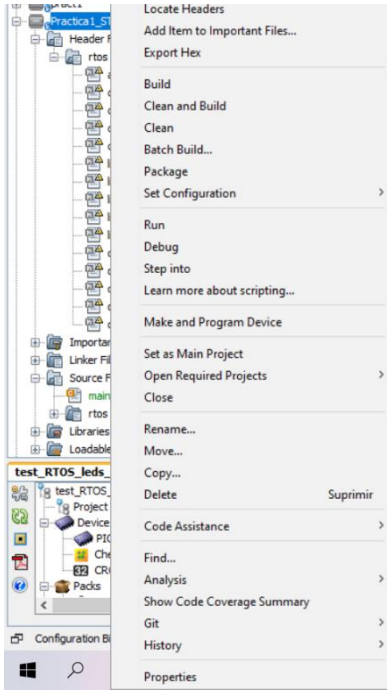


Dentro de los Header Files del proyecto, crear la carpeta rtos, guardar aquí todos los archivos .h proporcionados.

Seguir el mismo procedimiento con las Source Files, guardando esta vez en rtos los archivos .c

Crear dentro de Source Files un archivo main.c, donde se codificará la implementación de los 6 hilos

Para comprobar que no da errores, pulsando el botón derecho sobre nuestro proyecto, podemos acceder a sus propiedades:



Una vez aquí, se debe verificar que se han añadido correctamente los ficheros, y ejecutar el código. Si compila sin errores, se ha realizado correctamente el proceso.

Es importante tener en cuenta que se debe poner como proyecto principal (Set as Main Proyect), sino, será otro proyecto el que se esté ejecutando.

Código:

En el fichero main.c es donde se implementará la funcionalidad descrita.

En primer lugar, se deben crear los prototipos de las 6 funciones correspondientes a las tareas de la siguiente forma:

```
// Prototipos de funciones
void TASK_LED1 (void);
void TASK_LED2 (void);
void TASK_LED3 (void);
void TASK_LED4 (void);
void TASK_LED5 (void);
void TASK_LED6 (void);
```

A continuación, se crean los TCBs y Stack para el RTOS. Para administrar las tareas correctamente es necesario crear estas estructuras de datos.
El TCB es un puntero a la estructura de la tarea, que contiene la información necesaria para administrarla.
La stack sirve para guardar el contexto de la tarea.

```
// VARIABLES RELATIVAS AL RTOS. Task Control Block y Stacks
// TCB
OS_TCB task_led1_TCB;
OS_TCB task_led2_TCB;
OS_TCB task_led3_TCB;
OS_TCB task_led4_TCB;
OS_TCB task_led5_TCB;
OS_TCB task_led6_TCB;


// STK
CPU_STK task_led1_STK[1024]; // 1024*4 (32-bit system)
CPU_STK task_led2_STK[1024]; // 1024*4 (32-bit system)
CPU_STK task_led3_STK[1024]; // 1024*4 (32-bit system)
CPU_STK task_led4_STK[1024]; // 1024*4 (32-bit system)
CPU_STK task_led5_STK[1024]; // 1024*4 (32-bit system)
CPU_STK task_led6_STK[1024]; // 1024*4 (32-bit system)
```

El siguiente paso es la creación de las tareas. Se utiliza la función OSTaskCreate(), pasándole como parámetros:

- la dirección de la TCB
- el nombre de la tarea
- la dirección de la función de la tarea
- el parámetro de la función de la tarea
- la prioridad de la tarea
- la dirección de la stack de la tarea
- el tamaño de la stack de la tarea
- el tamaño de los mensajes que se van a enviar a la tarea
- el tiempo de espera de la tarea
- el parámetro de la función de la tarea
- las opciones de la tarea
- el puntero al error

Tendría la siguiente forma:

Tarea 1:

```
OSTaskCreate(
             (OS_TCB     *)&task_led1_TCB,
             (CPU_CHAR    *)"Tarea 1. Control led y mensaje uart",
             (OS_TASK_PTR  )TASK_LED1,
             (void        *)0,
             (OS_PRIO      )1,
             (CPU_STK     *)&task_led1_STK[0],
             (CPU_STK_SIZE )0u,
             (CPU_STK_SIZE )1024u,
             (OS_MSG_QTY   )0u,
             (OS_TICK      )10u, //10*System Tick period
             (void        *)0,
             (OS_OPT       )(OS_OPT_TASK_STK_CHK|OS_OPT_TASK_STK_CLR),
             (OS_ERR      *)&os_err);
```

Tarea 6:

```
OSTaskCreate(
             (OS_TCB     *)&task_led6_TCB,
             (CPU_CHAR    *)"Tarea 6. Control led y mensaje uart",
             (OS_TASK_PTR  )TASK_LED6,
             (void        *)0,
             (OS_PRIO      )6,
             (CPU_STK     *)&task_led6_STK[0],
             (CPU_STK_SIZE )0u,
             (CPU_STK_SIZE )1024u,
             (OS_MSG_QTY   )0u,
             (OS_TICK      )10u, //10*System Tick period
             (void        *)0,
             (OS_OPT       )(OS_OPT_TASK_STK_CHK|OS_OPT_TASK_STK_CLR),
             (OS_ERR      *)&os_err);
```

La prioridad más alta es 1. Cuanto más alto es el número, más baja es la prioridad. De tal forma que la tarea 1 tiene la prioridad más alta y la tarea 6 la más baja, por lo que la tarea 6 se ejecuta cuando no hay nada más que hacer. En caso de que dos tareas tengan la misma prioridad, se ejecuta primero la que se haya creado antes.

Justo debajo de la creación de cada tarea, se debe incluir una sentencia if que compruebe si hay algún error en la creación de cada tarea. Si es el caso, se envía un mensaje por la uart y se queda en un bucle infinito, si no, se inicia el rtos.

```
    if(os_err != OS_ERR_NONE)
{
    sprintf(txdata," ERROR en la creación de las Tareas \r\n");
    EnviarString(txdata, Uart3);
    while(1);
}
   else
   {
       OSStart(&os_err);
       while(1)
       {
           sprintf(txdata," ERROR en la inicialización del RTOS\r\n");
           EnviarString(txdata, Uart3);
       }
   }
```

Por último, se crean 6 funciones que se ejecutan en paralelo con el resto de tareas. Para el primer hilo, se cambia el estado del led rojo (toggle) cada 100ms (10Hz) y se envía un mensaje por la UART cada vez que se ejecuta.
Cada función tendrá un color de led distinto (LED_RED, LED_ORANGE, LED_GREEN, LED_BLUE1, LED_BLUE2, LED_BLUE3)

```
//HILOS DEL SISTEMA
void TASK_LED1 (void)
{
    OS_ERR os_err;

    while(1)
    {
        sprintf(txdata,"TASK1\r\n");
        EnviarString(txdata, Uart3);
        LED_RED = !LED_RED_Read;
        OSTimeDly(100,OS_OPT_TIME_DLY, &os_err);  //100 ticks del RTOS -> 100ms

    }
}
```

Conclusiones:

En esta práctica se ha inicializado el espacio de trabajo, se han creado las tareas de forma que se realice un toggle de los leds cada 100ms y se envía un mensaje por la uart cada vez que se ejecuta una tarea. Por último, se ha iniciado el RTOS.