Práctica 2: Introducción a CoOs: Tareas, Timers, Banderas y Semáforos

Alejandro Vega

Antonio Carlos Portillo

Objetivos.

Introducir un sistema operativo en tiempo real (CoOs).

Creación de un sistema multi tarea para el manejo de los leds.

Creación timers software para el muestro del joystick.

Sincronización de tareas mediante banderas.

Protección de acceso a recursos compartidos mediante semáforos.

Funciones para la gestión de tareas.

Para la creación y gestión de las tareas, el CoOs nos proporciona una serie de funciones:

– CoCreateTask: Crea una nueva tarea en un segmento de pila.
– CoExitTask: Finaliza una tarea.

– CoDelTask: Una tarea elimina a otra.

– CoSuspendTask: Una tarea suspende a otra.

– CoAwakeTask: Una tarea despierta a otra.

Implementación y creación de Tareas

• Para la implementación de una tarea se necesita:

– La función que implementa la tarea.

– Un vector de pila para alojar las variables de la tarea.

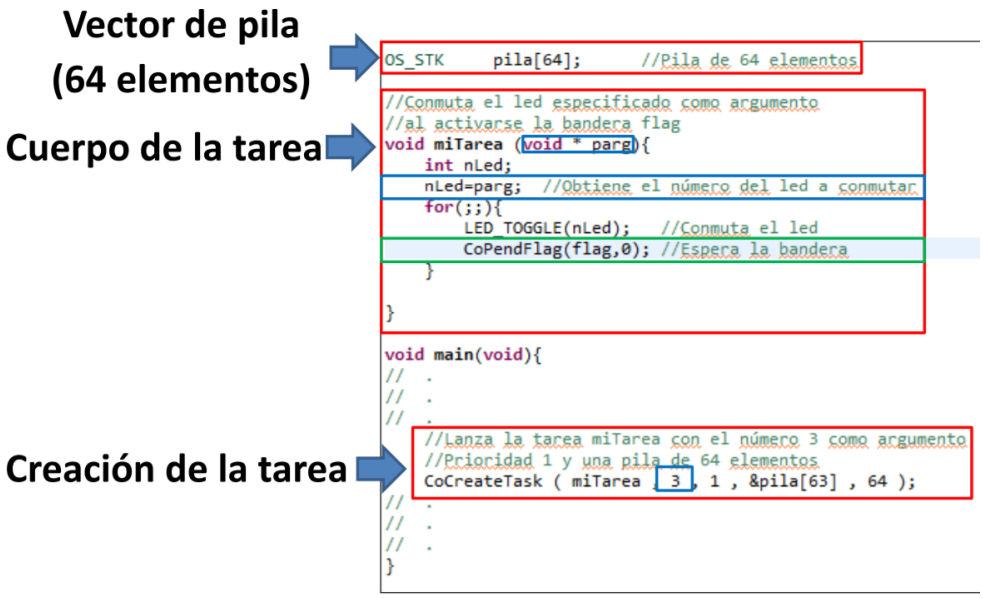
– Un bloque control de tarea (TCB), interno al CoOs y que contiene toda la información relativa a la tarea.

• Para crear una tarea llamamos a CoCreateTask, la cual recibe como parámetros:

1. Puntero a la función de la tarea.
2. Puntero a los argumentos de la tarea.
3. La prioridad de la tarea (cuando más alta, menos prioritaria).
4. Puntero a la primera posición libre de la pila.
5. Tamaño de la pila.

Creando una tarea

Al crear una tarea debemos tener en cuenta la creación de una pila, la creación de la tarea y el uso de CoCreateTask para la creación de la misma.



La tarea Idle

- CoOs lanza una tarea por defecto: CoIdleTask
- Tiene la mínima prioridad y sustituye al bucle principal: CFG_LOWEST_PRIO
- Cuando todas las tareas están bloqueadas, la ejecuta a la espera de tener tareas listas para su ejecución.
- Se encuentra en hook.c

Control del CoOs

- Funciones de inicialización y control del CoOs:
  - CoInitOS(void):
    - Inicializa las variables del sistema operativo.
  - CoStartOS(void):
    - Comienza la ejecución del planificador.
    - El CoOs toma el control de la ejecución.
    - Deben existir tareas creadas antes de lanzarlo.
  - CoSchedLock(void) y CoSchedUnLock(void):
    - Des/bloquean el planificador.
    - Usado internamente por el CoOs para el acceso a las regiones críticas.
    - Uso con cuidado por parte del usuario.

Lanzando el CoOs

•Siempre se sigue el mismo proceso:

1.Se inicializa el reloj del sistema.

2.Se inicializan las estructuras internas del CoOs.

3.Se crean los elementos de sincronismo y comunicaciones (banderas, colas…)

4.Le crean las tareas de la aplicación

5.Se lanza el scheduler del CoOs.

```
int main(void)
{
    SystemInit();           //Inicialización del reloj

    CoInitOS ();            //Inicialización del CoOs

    CreateSystemObjetcs();  //Inicialización Sem, flags, queues...

    CreateUserTasks();      //Creación de Tareas

    CoStartOS ();           //Comienzo de ejecución del planificador

    while(1)                //La ejecución nunca debería llegar aquí
    {
    }
}
```

Lanzando tareas

•Las tareas se lanzan en dos pasos:

1.Se crean los **elementos del CoOscompartidos entre tareas**(banderas, colas, semáforos…) para evitar que **una tarea acceda a un elemento antes de ser creado**.

2.Se crean las tareas y quedan a la **espera de que se lance el planificador**.

```
void CreateSystemObjetcs(void){
    //Inicialización de los elementos compartidos: flags, semaf., colas...
    CreateJoyFlags();       //Crear banderas del joystick
    CreateSerialQueue();    //Crear cola para el puerto serie
    CreateLCDSem();         //Crear Semaforo del LCD
}

void CreateUserTasks(void){
    //Creación de las tareas de usuario
    CreateJoyTask();
    CreateLedTask();
    CreatSerialTask();
    CreateLCDTask();
}
```

El proyecto

Consta de una serie de carpetas, que vamos a ver ahora:

BoardSupportPackage

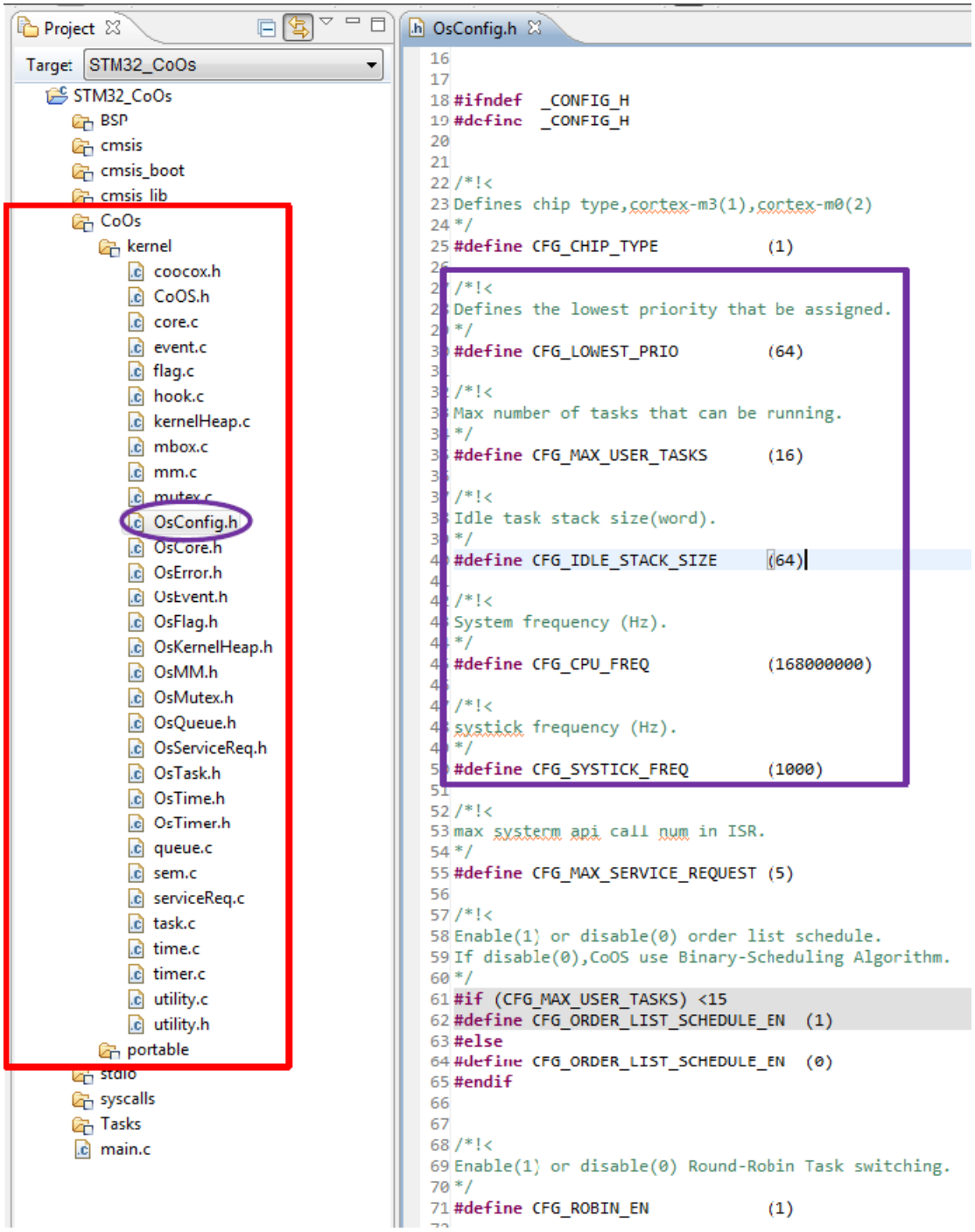
•Contiene el código de los drivers propios de cada micro/placa.

•En nuestro caso sólo vamos a usar los ledsy el joystick

•Por comodidad se agrupan los includesen **BSP.h**

•En las tareas sólo hay que incluir **BSP.h**

CoOs Kernel


Project
Target: STM32_CoOs
STM32_CoOs
BSP
cmsis
cmsis_boot
cmsis_lib
CoOs
kernel
coocox.h
CoOS.h
core.c
event.c
flag.c
hook.c
kernelHeap.c
mbox.c
mm.c
mutex.c
OsConfig.h
OsCore.h
OsError.h
OsEvent.h
OsFlag.h
OsKernelHeap.h
OsMM.h
OsMutex.h
OsQueue.h
OsServiceReq.h
OsTask.h
OsTime.h
OsTimer.h
queue.c
sem.c
serviceReq.c
task.c
time.c
timer.c
utility.c
utility.h
portable
stdio
syscalls
Tasks
main.c
OsConfig.h
#ifndef _CONFIG_H
#define _CONFIG_H
/*!<
Defines chip type,cortex-m3(1),cortex-m0(2)
*/
#define CFG_CHIP_TYPE (1)
/*!<
Defines the lowest priority that be assigned.
*/
#define CFG_LOWEST_PRIO (64)
/*!<
Max number of tasks that can be running.
*/
#define CFG_MAX_USER_TASKS (16)
/*!<
Idle task stack size(word).
*/
#define CFG_IDLE_STACK_SIZE (64)
/*!<
System frequency (Hz).
*/
#define CFG_CPU_FREQ (168000000)
/*!<
systick frequency (Hz).
*/
#define CFG_SYSTICK_FREQ (1000)
/*!<
max systerm api call num in ISR.
*/
#define CFG_MAX_SERVICE_REQUEST (5)
/*!<
Enable(1) or disable(0) order list schedule.
If disable(0),CoOS use Binary-Scheduling Algorithm.
*/
#if (CFG_MAX_USER_TASKS) <15
#define CFG_ORDER_LIST_SCHEDULE_EN (1)
#else
#define CFG_ORDER_LIST_SCHEDULE_EN (0)
#endif
/*!<
Enable(1) or disable(0) Round-Robin Task switching.
*/
#define CFG_ROBIN_EN (1)

main.c

main.c

```c
#include <CoOS.h>
#include "stm32f4xx_conf.h"
#include "LedTask.h"
#include "JoyTmr.h"

void SystemInit(void);

void CreateSystemObjetcs(void){
    //Inicialización de los elementos compartidos: flags, semaf., colas...
    CreateJoyFlags();       //Crear banderas del joystick
}

void CreateUserTasks(void){
    //Creación de las tareas de usuario
    CreateLedTask();    //Crear tareas para los leds
    CreateJoyTimer();   //Crear timer de muestreo del joystick
}


int main(void)
{
    SystemInit();           //Inicialización del reloj

    CoInitOS ();            //Inicialización del CoOs

    CreateSystemObjetcs();  //Inicialización Sem, flags, queues...

    CreateUserTasks();      //Creación de Tareas

    CoStartOS ();           //Comienzo de ejecución del planificador

    while(1)                //La ejecución nunca debería llegar aquí
    {
    }
}
```

Tareas de usuario

- En la carpeta Taks están las plantillas de las tareas de usuario.
- En este caso tenemos:
  - **LedTask.c:** Tareas con animaciones de leds.
  - **JoyTmr.c**: Timer de muestreo del joystick.

LedTask.c

```
//Pila de la tarea
OS_STK     led_stk[64];
```

```
void CreateLedTask(void){
    uint16_t i;

    Init_Leds();    //Inicialización de los leds

    //Creación de tareas
    CoCreateTask (LedToogleTask, 0 , 2 ,&led_stk[63],64);

}
```

```
void LedToogleTask(void * parg){
    //Inicialización de la tarea
    LED_Off(0);

    //Cuerpo de la tarea
    for (;;) {
        LED_Toggle(0);
        CoTimeDelay(0,0,0,100);
    }

}
```

1. Pila de la tarea
2. Creación de la tarea.
3. Cuerpo de la tarea.

Gestión del tiempo en CoOs

- **voidCoTimeDelay(h, m, s ,ms)**
  - –Duerme una tarea por un tiempo especificado.
- **voidCoTickDelay(ticks)**
  - –Duerme una tarea durante el número de ticksdel SO especificado.
- **uint64_t CoGetOSTime(void)**
  - –Nos devuelve el número absoluto de ticksque lleva ejecutados el SO.

Ejercicio 1

Pasamos ahora a desarrollar los ejercicios de la primera sesión de ésta práctica.

•Hacer una tarea genérica para todos los leds, crearla 4 veces con distintos parámetros.

•La idea es que cada tarea conmute un led.

•Depurar para comprobar que se han creado 4 instancias de la misma tarea.

•Añadir un BreakPoint en la tarea **CoIdleTask** en hook.c. ¿Cuándo se ejecuta esta tarea?

La tarea se ejecuta cuando las otras tareas se bloquean.

```
17 void CreateLedTask(void){
18     uint16_t i;
19 
20     Init_Leds();    //Inicialización de los leds
21 
22     //Creación de tareas
23     CoCreateTask (LedToogleTask, 1 , 2 ,&led_stk[0][63],64);
24     CoCreateTask (LedToogleTask, 2 , 2 ,&led_stk[1][63],64);
25     CoCreateTask (LedToogleTask, 3 , 2 ,&led_stk[2][63],64);
26     CoCreateTask (LedToogleTask, 4 , 2 ,&led_stk[3][63],64);
27 
28 }
29 
30 void LedToogleTask(void * parg){
31 
32     int i;
33     i = (int) parg;
34 
35     //Inicialización de la tarea
36     LED_Off(i);
37 
38     //Cuerpo de la tarea
39     for (;;) {
40         LED_Toggle(i);
41         CoTimeDelay(0,0,0,100);
42     }
43 
```

Ejercicio 2

•Modificar la tarea **LedToogleTask** para que cada led parpadee a un ritmo distinto.

•Crearla 4 veces con parámetros y pilas distintas.

•Comprobar que al crearlas con distintas prioridades siempre se ejecuta primero la tarea más prioritaria.

•Usando la función **CoGetOSTime** comprobar que cada led parpadea con el ritmo adecuado.

```
15 OS_STK     led_stk[4][64];
16
17 void CreateLedTask(void){
18     uint16_t i;
19
20     Init_Leds();    //Inicialización de los leds
21
22     //Creación de tareas
23     CoCreateTask (LedToogleTask, 1 , 4 ,&led_stk[0][63],64);
24     CoCreateTask (LedToogleTask, 2 , 5 ,&led_stk[1][63],64);
25     CoCreateTask (LedToogleTask, 3 , 3 ,&led_stk[2][63],64);
26     CoCreateTask (LedToogleTask, 4 , 2 ,&led_stk[3][63],64);
27
28 }
29
30 void LedToogleTask(void * parg){
31
32     int i;
33     i = (int) parg;
34
35     uint16_t tiempo;
36     U64 total;
37     U64 principio;
38
39     principio = CoGetOSTime();
40
41
```

```
39
40      if(i==1){
41          tiempo = 200;
42      }else if(i==2){
43          tiempo = 300;
44      }else if(i==3){
45          tiempo = 400;
46      }else if(i==4){
47          tiempo = 500;
48      }else{
49
50      }
51
52      //Inicialización de la tarea
53      LED_Off(i);
54
55      //Cuerpo de la tarea
56      for (;;) {
57          LED_Toggle(i);
58          CoTimeDelay(0,0,0,tiempo);
59          total = CoGetOSTime() - principio;
60          principio = CoGetOSTime();
61
62      }
63
64  }
```

Ejercicio 3

1.Implementar las animaciones de la práctica anterior como **funciones independientes** usando el retraso temporal del CoOs.

2.Crear **una única tarea** usando la plantilla **LedAnimationTask**, que llame a cada animación dependiendo del botón pulsado. Llamar a **Init_Joy()** al inicializar la tarea. Usar **Read_Joy()** y una estructura **switch/case**.

3.Comprobar el funcionamiento de la tarea.

```
void LedAnimationTask(void * parg){
    //Inicialización de la tarea
    Init_Joy();
    uint8_t joy;
    //Cuerpo de la tarea
    for (;;) {
        joy = Read_Joy();
        switch(joy){
        case 1:
            animation1();
            break;
        case 2:
            animation2();
            break;
        case 3:
            animation3();
            break;
        case 4:
            animation4();
            break;
        default:
            break;
        }
    }

}
```

Ejercicio 4

•De cara a la próxima sesión:

1.Implementar cada animación como **una tarea independiente**.

2.Usar las plantillas en **LedTask.c**: LedTask0, LedTask1, LedTask2, y LedTask3.

3.Crearlas una a una para comprobar su correcto funcionamiento.

4.Crear las cuatro tareas simultáneamente. ¿Qué está ocurriendo?

5.¿Cómo influyen las prioridades?

Al crear las tareas simultáneamente, pero con prioridades distintas les estamos indicando el orden con el que deben ejecutarse.

```
void LedTask0(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea
    for (;;) {
        animation1();
    }

}



void LedTask1(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea
    for (;;) {
        animation2();
    }
}
void LedTask2(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea
    for (;;) {
        animation3();
    }

}


void LedTask3(void * parg){
    //Inicialización de la tarea
```

Parte 2 de la práctica

Objetivos

•Implementar **un timer software** que **muestree periódicamente el joystick**.

•**Sincronizar** el **timer software con tareas** de animaciones de leds **mediante banderas**.

•**Controlar** el **acceso simultáneo** a los leds, **recursos compartidos**, por parte de las tareas de las animaciones **usando semáforos.**

Timers Software

•Los RTOS además de las tareas, permiten la creación de "**timer software**".

•**Un timer software es una función que se ejecuta con una periodicidad predefinida.**

•Como particularidad, la función que ejecuta **el timer no puede llamar a funciones del RTOS que la bloqueen.**

•**Pueden postear:** banderas, semáforos, colas…

•**Existen dos tipos:**

–Timers periódicos (**Periodic Timer**).

–Timers que se ejecutan una sola vez tras un tiempo determinado (**One-Shot Timer**).

Funciones para la creación de timers

- •Para crear un timer software hay que definir:
  - –Una variable del tipo **OS_TCID**: identificador del timer.
- •OS_TCID miTimerID;
  - –Puntero a la función a ejecutar:
- •void miTimer(void){

  //…

  }
- •El CoOs proporciona las siguientes funciones:
  - •OS_TCID **CoCreateTmr**(tmrType, tmrCnt, tmrReload, func);
  - •StatusType**CoStartTmr**(tmrID);
  - •StatusType**CoDelTmr**(tmrID);

Ejemplo de creación de un timer software

```
void TimerCreate(void){

    OS_TCID timerId;     //Identificiador del timer
//  ...
    //Creación
    timerId=CoCreateTimer(TMR_TYPE_PERIODIC, ticksDelay, ticksPeriod, miTimer );
    //Inicio
    CoStartTmr(timerId);

//  ...

}

void miTimer (void){
    uint8_t key;
    key = readJoy();
    LED_TOOGLE(key);
}
```

Vamos ahora con los ejercicios propuestos para esta sesión:

Ejercicio 1

- Modificar **LedTask.c**:
  - Comentar temporalmente, en la función **CreateLedTask**, las llamadas a **CoCreateTask** de las tareas de las animaciones.
- Modificar **JoyTmr.c**:
  - Crear un timer software que ejecute la función **JoyTimer** cada 100 miliSeg.
  - Completar las funciones proporcionadas.

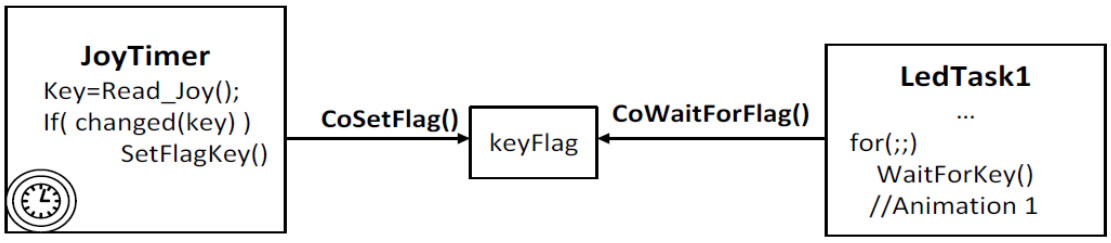
```
//Creación del timer del joy
void CreateJoyTimer(void){
    //ID del timer
    OS_TCID joyId;

    //Inicialización del joystick
    Init_Joy();

    //Creación e inciacion del timer
    joyId=CoCreateTmr(TMR_TYPE_PERIODIC, 100, 100, JoyTimer);
    CoStartTmr(joyId);


}


//Funcion del timer software
void JoyTimer(void){
    uint8_t key;

    key = Read_Joy();

    LED_Toggle(key);
}
```

Banderas en CoOs

- •Una bandera es una variable del tipo:
  - •**OS_FlagIDmiBandera;**
- •Creando una bandera:
  - •**miBandera= CoCreateFlag(autoReset, initialState)**
    - –autoReset= 0 => La bandera se resetea manualmente.
    - –autoReset= 1 => La bandera se “consume” al despertar a una tarea.
    - –initialState: Estado inicial de la bandera (1: Ready, 0: Non-ready)
- •Modificando su estado:
  - •Activar: **CoSetFlag(miBandera)**
  - •Desactivar: **CoClearFlag(miBandera)**
- •Esperando su activación:
  - •**CoWaitForSingleFlag (miBandera, TimeOut)**
    - –**TimeOut:** Tiempo de espera en ticksdel sistema (1 mSeg). Si vale 0, la espera es indefinida.

Ejercicio 2: Comenzando con una bandera

- •**Objetivo:** ejecutar una animación una sola vez cuando se pulse cualquier botón.
- •La gestión de las banderas la vamos a implementar en **JoyTimer.c**
- •El primer paso es declarar una bandera como una variable global:
  - –**OS_FlagIDkeyFlag;**
- •Para **no tener que compartir la variable global** entre distintos ficheros fuente, como regla de estilo, se implementan “**envoltorios**” para su uso.
- •Completar dos funciones usando las plantillas proporcionadas:
  - **1.voidSetFlagKey(uint8_t key)**
  - •Activa la bandera usando **CoSetFlag.**
  - **2.voidwaitForKey (uint8_t key, uint32_t timeOut)**
  - •Espera la bandera usando **CoWaitForSingleFlag.**



1.En JoyTimer.c:

1.Crear la bandera en la función **CreateJoyFlags**

```
//Inicializacion de las banderas
void  CreateJoyFlags(void){


        //Creacion de la bandera


        keyFlag=CoCreateFlag(1, 0);


}
```

2.Llamar, en el cuerpo del timer, a **SetFlagKey(0)**cada vez que el valor devuelto por **Read_Joy** cambie.

```
//Funcion del timer software
void JoyTimer(void){
        uint8_t key;
        key = Read_Joy();
        if(key != oldkey && oldkey == 0){
                SetFlagKey(0);
        }
        oldkey = key;

}
```

2.En **LedTask.c**, modificar la tarea de alguna animación para que espere la bandera antes de ejecutarse llamando a **WaitForKey(0,0)**.

```
void LedTask0(void * parg){

        //Inicialización de la tarea

        //Cuerpo de la tarea

        for (;;) {

                waitForKey(0,0);

                animation1();

        }

}
```

3.Crear la tarea de la animación del led que se ha modificado.

```
void CreateLedTask(void){
        uint16_t i;
        Init_Leds();//Inicialización de los leds
        //Creación de tareas
        CoCreateTask (LedTask0, 1 , 2 ,&led_stk[0][63],64);
        //CoCreateTask (LedTask1, 1 , 2 ,&led_stk[1][63],64);
        //CoCreateTask (LedTask2, 1 , 2 ,&led_stk[2][63],64);
        //CoCreateTask (LedTask3, 1 , 2 ,&led_stk[3][63],64);
}
```

Ejercicio 3: Sistema multi-bandera

•La idea es tener **una bandera asociada a cada tecla**, activándola cada vez que se pulse.

•Por otro lado, **lanzamos las tareas con las animaciones** de los leds, a la **espera de la bandera de una tecla**.

•Cuando se active la bandera, la tarea con la animación asociada a la tecla se ejecutará.

•En JoyTimer.c:

1.Modificar la declaración de la bandera para que ahora sea **un vector de 4 banderas**

```
8
9 //Declaración de las banderas
10 OS_FlagID keyFlag[4];
11 uint8_t oldkey = 0;
12
```

2.Modificar la función **CreateJoyFlags**para que ahora **inicialice las 4 banderas**. Usar un simple bucle for.

```
2 //Inicializacion de las banderas
3 void  CreateJoyFlags(void){
4     int i;
5     //Creacion de la bandera
6     for(i = 0; i<4; i++){
7         keyFlag[i]=CoCreateFlag(1, 0);
8     }
9 }
```

3.Modificar la función **SetFlagKey**para que sólo active la bandera asociada a la tecla key que recibe como parámetro.

```
//Funcion que activa la bandera correspondiente a una tecla
void SetFlagKey(uint8_t key){

    CoSetFlag(keyFlag[key-1]);


}
```

4.Modificar la función **WaitForKey**para que se quede a la espera de la bandera asociada a la tecla key que recibe como parámetro.

```
//Funcion de espera a que una tecla se pulse
uint8_t waitForKey(uint8_t key, uint32_t timeout){

    //Esperar la bandera

    CoWaitForSingleFlag(keyFlag[key-1], timeout);

    return key;
}
```

En LedTask.c:
1.Modificar **las tareas de las 4 animaciones** para que se queden a **la espera de una tecla** usando **WaitForKey**. La tecla asignada a cada animación se debe de indicar en el parámetro de la tarea.

```
void LedTask0(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea

    for (;;) {
        waitForKey(1,0);
        CoPendSem(ledSem, 0);
        animation1();
        CoPostSem(ledSem);
    }

}


void LedTask1(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea
    for (;;) {
        waitForKey(2,0);
        CoPendSem(ledSem, 0);;
        animation2();
        CoPostSem(ledSem);
    }
}
```

```
void LedTask2(void * parg){
    //Inicialización de la tarea


    //Cuerpo de la tarea
    for (;;) {
        waitForKey(3,0);
        CoPendSem(ledSem, 0);
        animation3();
        CoPostSem(ledSem);
    }

}


void LedTask3(void * parg){
    //Inicialización de la tarea

    //Cuerpo de la tarea
    for (;;) {
        waitForKey(4,0);
        CoPendSem(ledSem, 0);
        animation4();
        CoPostSem(ledSem);
    }
}
```

2.Modificar la función **CreateLedTask**para que cree las 4 tareas con las animaciones, y le asigne una tecla a cada una como parámetro.

```
void CreateLedTask(void){
    uint16_t i;

    Init_Leds();    //Inicialización de los leds
    ledSem = CoCreateSem(1, 1, EVENT_SORT_TYPE_PRIO);
    //Creación de tareas
    CoCreateTask (LedTask0, 1 , 2 ,&led_stk[0][63],64);
    CoCreateTask (LedTask1, 1 , 2 ,&led_stk[1][63],64);
    CoCreateTask (LedTask2, 1 , 2 ,&led_stk[2][63],64);
    CoCreateTask (LedTask3, 1 , 2 ,&led_stk[3][63],64);

}
```

Semáforos en CoOs

- •Un semáforo es una variable del tipo:
  - •**OS_EventIDmiSem;**
- •Creando un semáforo:
  - •**miSem= CoCreateSem(initCount, maxCnt, sortType)**
    - –**initCount:** Contador inicial del semáforo: **1**.
    - –**maxCnt:** Valor máximo del contador: **1**.
    - –**sortType:** Ordenación de tareas a la espera:
      - »EVENT_SORT_TYPE_FIFO: se encolan en una fifo.
      - »**EVENT_SORT_TYPE_PRIO**: se ordenan en base a la prioridad.
- •Esperar a que un semáforo esté libre:
  - •**CoPendSem(miSem, timeout)**
- •Liberando un semáforo:
  - •**CoPostSem(miSem)**

Ejercicio 4: Acceso a un recurso compartido

•En el ejercicio anterior las animaciones se mezclaban al compartir todas las tareas los leds.

•Añadir un semáforo que sólo permita acceder a una tarea a los leds.

•En LedTask.c:

1.Declarar un semáforo como una variable global.

```
7
8 //Declaración del semáforo
9 OS_EventID ledSem;
0
1
```

2.Modificar la función **CreateLedTask**para que cree un semáforo binario.

```
6
7 void CreateLedTask(void){
8     uint16_t i;
9
0     Init_Leds();    //Inicialización de los leds
1     ledSem = CoCreateSem(1, 1, EVENT_SORT_TYPE_PRIO);
2     //Creación de tareas
3     CoCreateTask (LedTask0, 1 , 2 ,&led_stk[0][63],64);
4     CoCreateTask (LedTask1, 1 , 2 ,&led_stk[1][63],64);
5     CoCreateTask (LedTask2, 1 , 2 ,&led_stk[2][63],64);
6     CoCreateTask (LedTask3, 1 , 2 ,&led_stk[3][63],64);
7
8 }
```

3.Añadir a cada tarea la espera y liberación del semáforo:

1.Acceder al semáforo (pend) después de la función **WaitForKey.**

2.Liberar el semáforo al final de la animación (post).

```
4
5 void LedTask0(void * parg){
6     //Inicialización de la tarea
7
8
9     //Cuerpo de la tarea
0
1     for (;;) {
2         waitForKey(1,0);
3         CoPendSem(ledSem, 0);
4         animation1();
5         CoPostSem(ledSem);
6     }
7
8 }
9
```

Conclusiones

Se ha podido realizar las dos sesiones de prácticas sin problemas. Habiendo servido como introducción al uso de Tareas, Banderas y Semáforos. El profesor ha indicado convenientemente la teoría tanto en clase como durante las sesiones para poder realizarlas sin problemas.