Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE SEVILLA

E.T.S. de Ingeniería Informática
Avda. Reina Mercedes, S/N.
41012 Sevilla, SPAIN

Escuela Universitaria Politécnica
C/ Virgen de África, 7.
41011 Sevilla, SPAIN

Introducción a CoOs: Tareas, Timers, Banderas y Semáforos.

Objetivos de la práctica

- 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

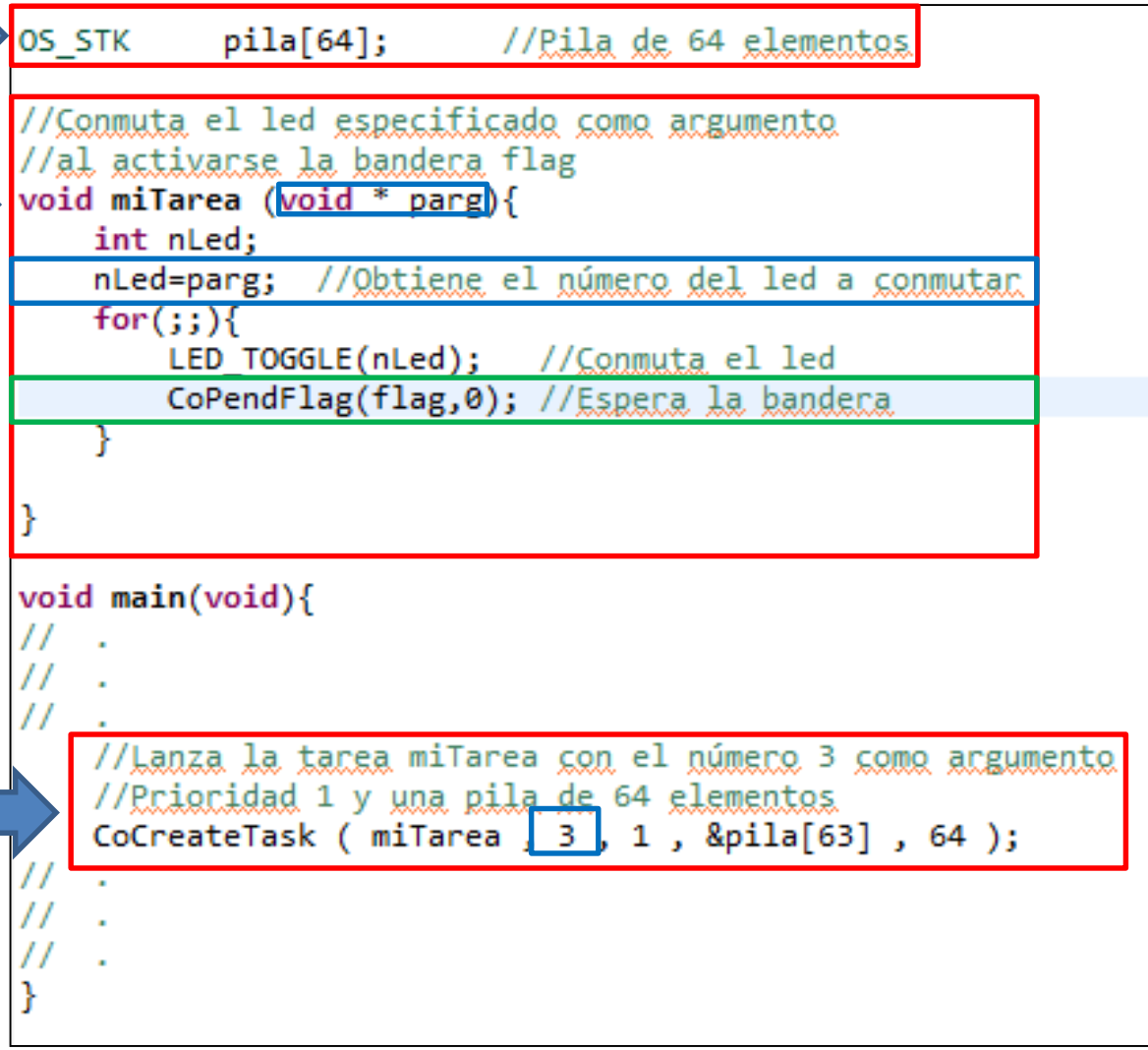
- **Para la implementación de una tarea se necesita:**
  - **La función** que implementa la tarea.
  - **Un vector de pila** para alojar las variables de la tarea.
  - **Un bloque control de tarea (TCB)**, interno al CoOs y que contiene toda la información relativa a la tarea.
- **Para crear una tarea 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

Vector de pila (64 elementos)

Cuerpo de la tarea

Creación de la tarea



```
OS_STK     pila[64];      //Pila de 64 elementos

//Conmuta el led especificado como argumento
//al activarse la bandera flag
void miTarea (void * parg){
    int nLed;
    nLed=parg;  //Obtiene el número del led a conmutar
    for(;;){
        LED_TOGGLE(nLed);   //Conmuta el led
        CoPendFlag(flag,0); //Espera la bandera
    }

}

void main(void){
//   .
//   .
//   .
   //Lanza la tarea miTarea con el número 3 como argumento
   //Prioridad 1 y una pila de 64 elementos
   CoCreateTask ( miTarea , 3 , 1 , &pila[63] , 64 );
//   .
//   .
//   .
}
```

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**

```
/**
 *********************************************************************************
 * @brief      IDLE task of OS
 * @param[in]  pdata    The parameter passed to IDLE task.
 * @param[out] None
 * @retval     None
 *
 * @par Description
 * @details    This function is system IDLE task code.
 *********************************************************************************
 */
void CoIdleTask(void* pdata)
{
    /* Add your codes here */
    for(; ;)
    {
        /* Add your codes here */
    }
}
```

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
    {
    }
}
```

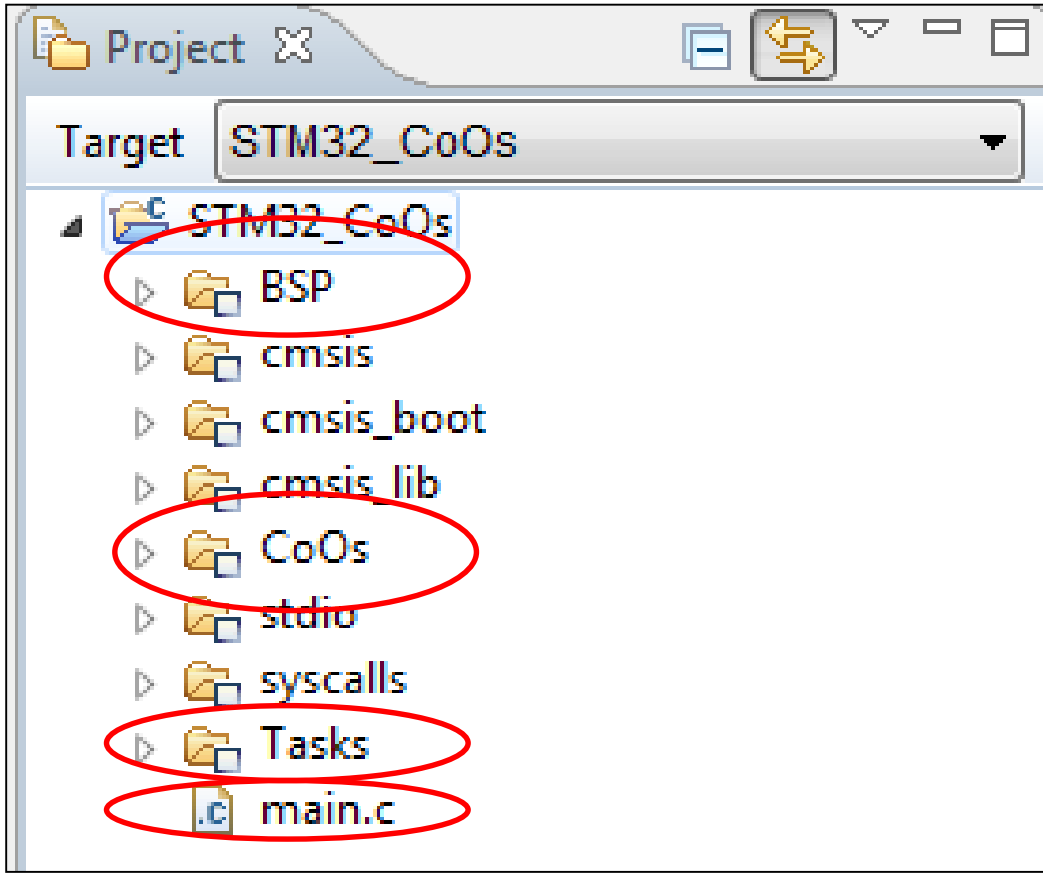
Lanzando tareas

- Las tareas se lanzan en dos pasos:
  1. Se crean los **elementos del CoOs compartidos 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**.

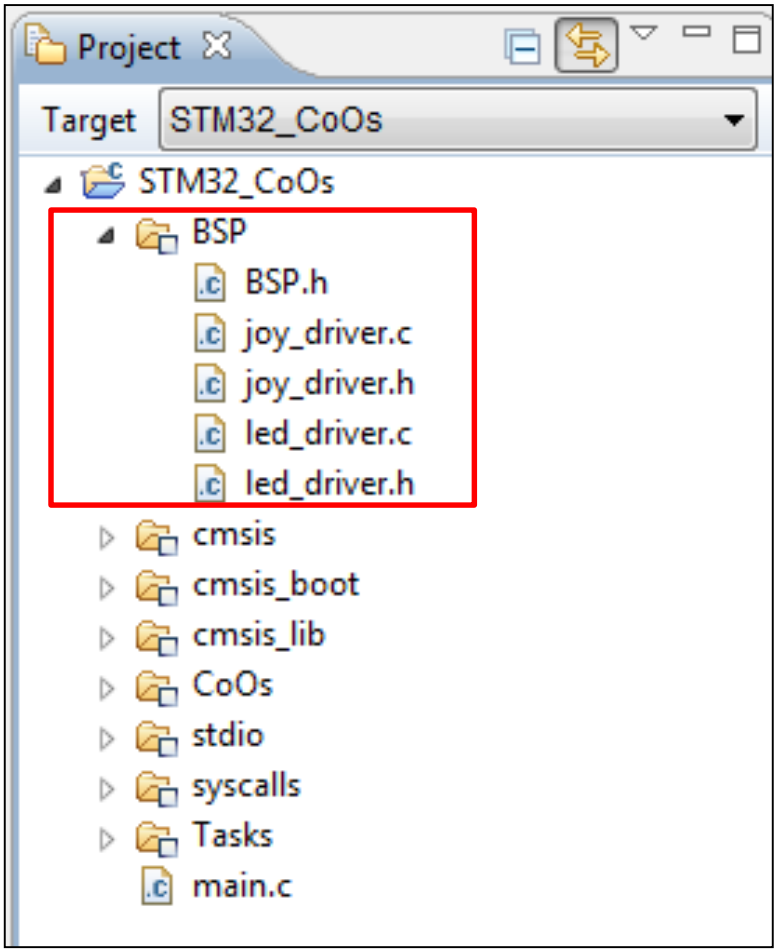
```
void CreateSystemObjetcs(void){
    //Inicialización de los elementos compartidos: flags, semaf., colas...
    CreateJoyFlags();       //Crear banderas del joystick
    CreateSerialQueue();    //Crear cola para el puerto serie
    CreateLCDSem();         //Crear Semaforo del LCD
}

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

El proyecto

1. **Board Support Package**: drivers de periféricos.
2. Kernel del CoOs.
3. Tareas de usuario.
4. Main

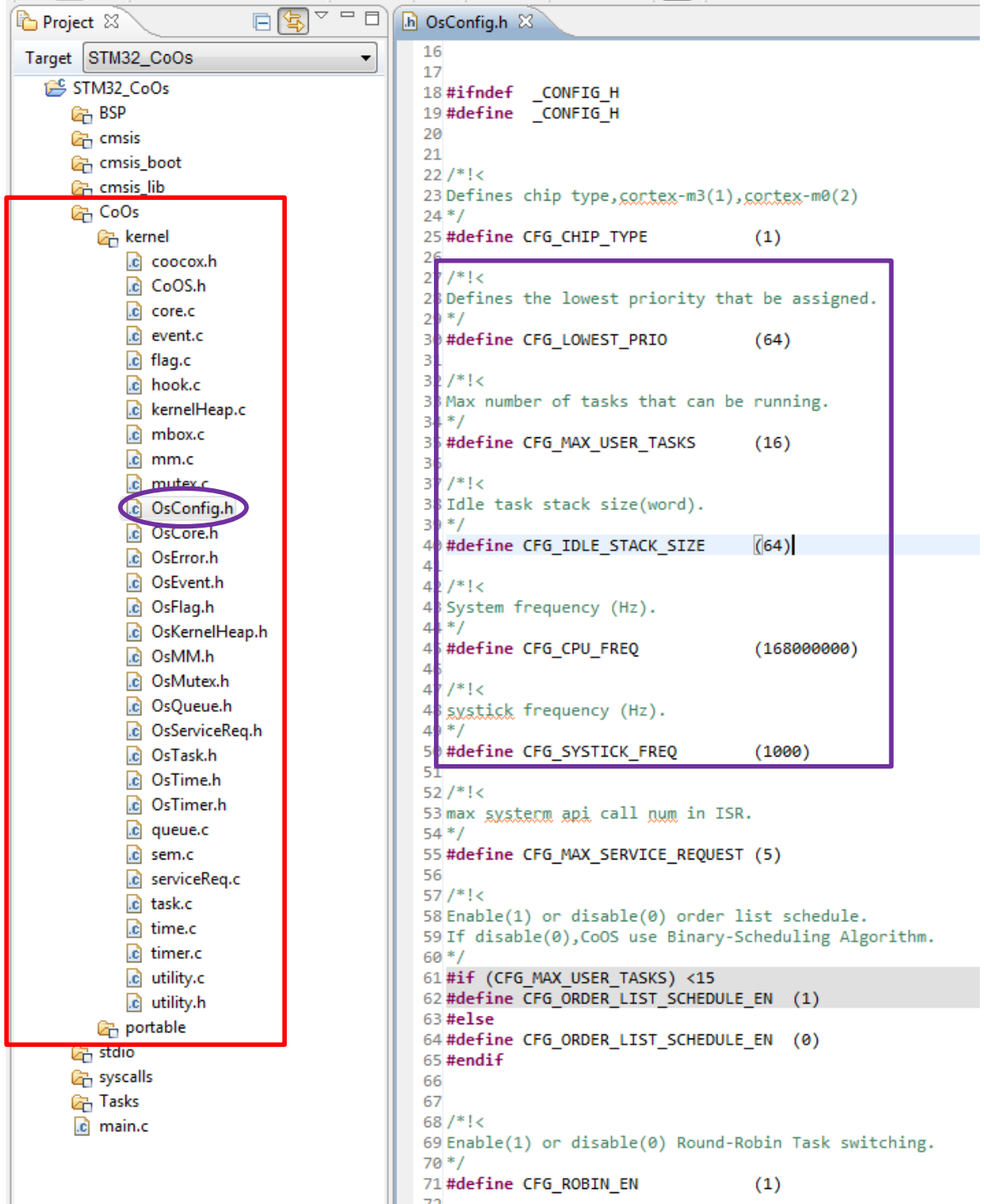


Board Support Package



- Contiene el código de los drivers propios de cada micro/placa.
- En nuestro caso sólo vamos a usar los leds y el joystick
- Por comodidad se agrupan los includes en **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
16
17
18 #ifndef _CONFIG_H
19 #define _CONFIG_H
20
21
22 /*!<
23 Defines chip type,cortex-m3(1),cortex-m0(2)
24 */
25 #define CFG_CHIP_TYPE (1)
26
27 /*!<
28 Defines the lowest priority that be assigned.
29 */
30 #define CFG_LOWEST_PRIO (64)
31
32 /*!<
33 Max number of tasks that can be running.
34 */
35 #define CFG_MAX_USER_TASKS (16)
36
37 /*!<
38 Idle task stack size(word).
39 */
40 #define CFG_IDLE_STACK_SIZE (64)
41
42 /*!<
43 System frequency (Hz).
44 */
45 #define CFG_CPU_FREQ (168000000)
46
47 /*!<
48 systick frequency (Hz).
49 */
50 #define CFG_SYSTICK_FREQ (1000)
51
52 /*!<
53 max systerm api call num in ISR.
54 */
55 #define CFG_MAX_SERVICE_REQUEST (5)
56
57 /*!<
58 Enable(1) or disable(0) order list schedule.
59 If disable(0),CoOS use Binary-Scheduling Algorithm.
60 */
61 #if (CFG_MAX_USER_TASKS) <15
62 #define CFG_ORDER_LIST_SCHEDULE_EN (1)
63 #else
64 #define CFG_ORDER_LIST_SCHEDULE_EN (0)
65 #endif
66
67
68 /*!<
69 Enable(1) or disable(0) Round-Robin Task switching.
70 */
71 #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

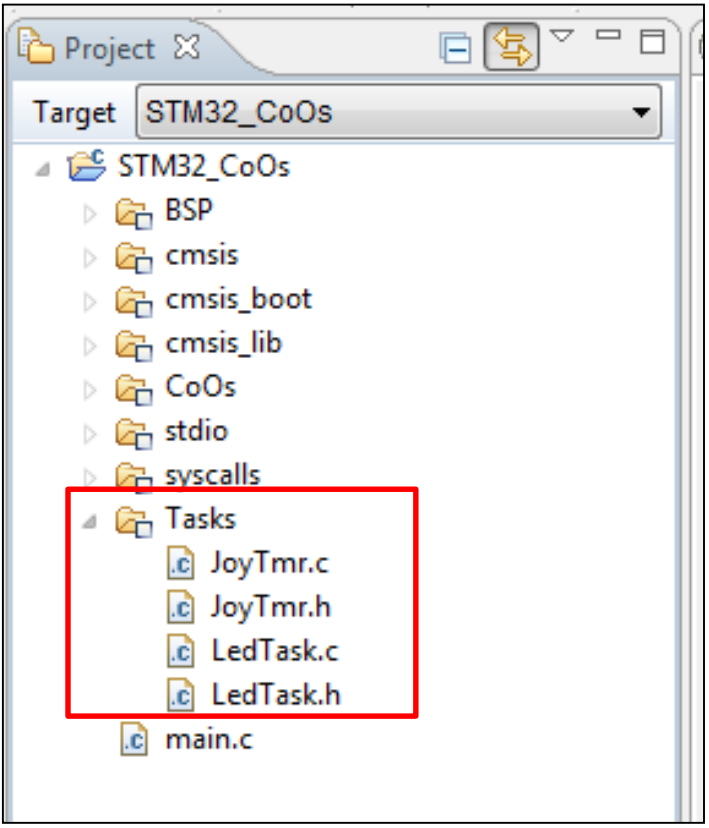
    CreateSystemObjetcs();  //Inicialización Sem, flags, queues...

    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

- **void CoTimeDelay (h, m, s ,ms)**
  - Duerme una tarea por un tiempo especificado.
- **void CoTickDelay (ticks)**
  - Duerme una tarea durante el número de ticks del SO especificado.
- **uint64_t CoGetOSTime(void)**
  - Nos devuelve el número absoluto de ticks que lleva ejecutados el SO.

Ejercicio 1

- 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?

Ejercicio 2

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

LED	Tiempo
1	200
2	300
3	400
4	500

- 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.

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.

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?**

Parte 2

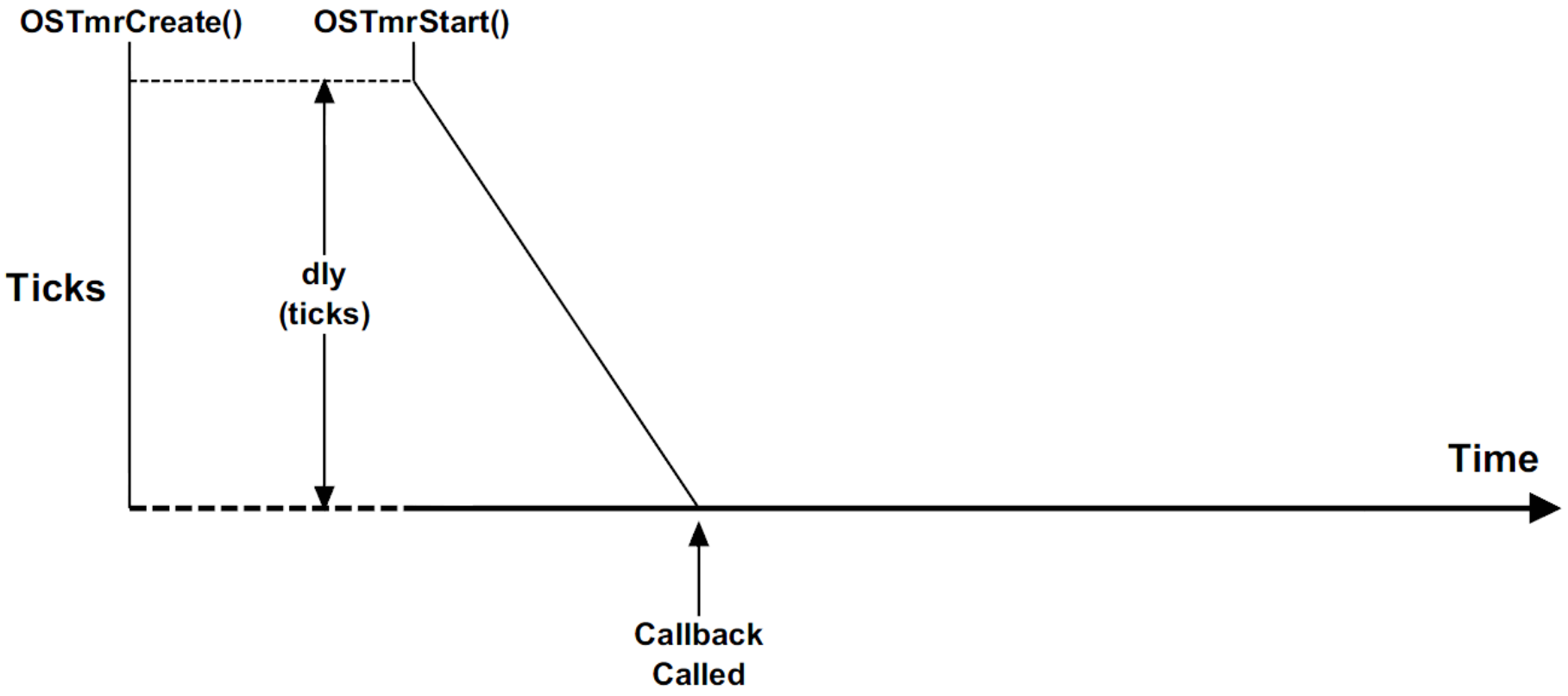
Objetivos

- 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.**

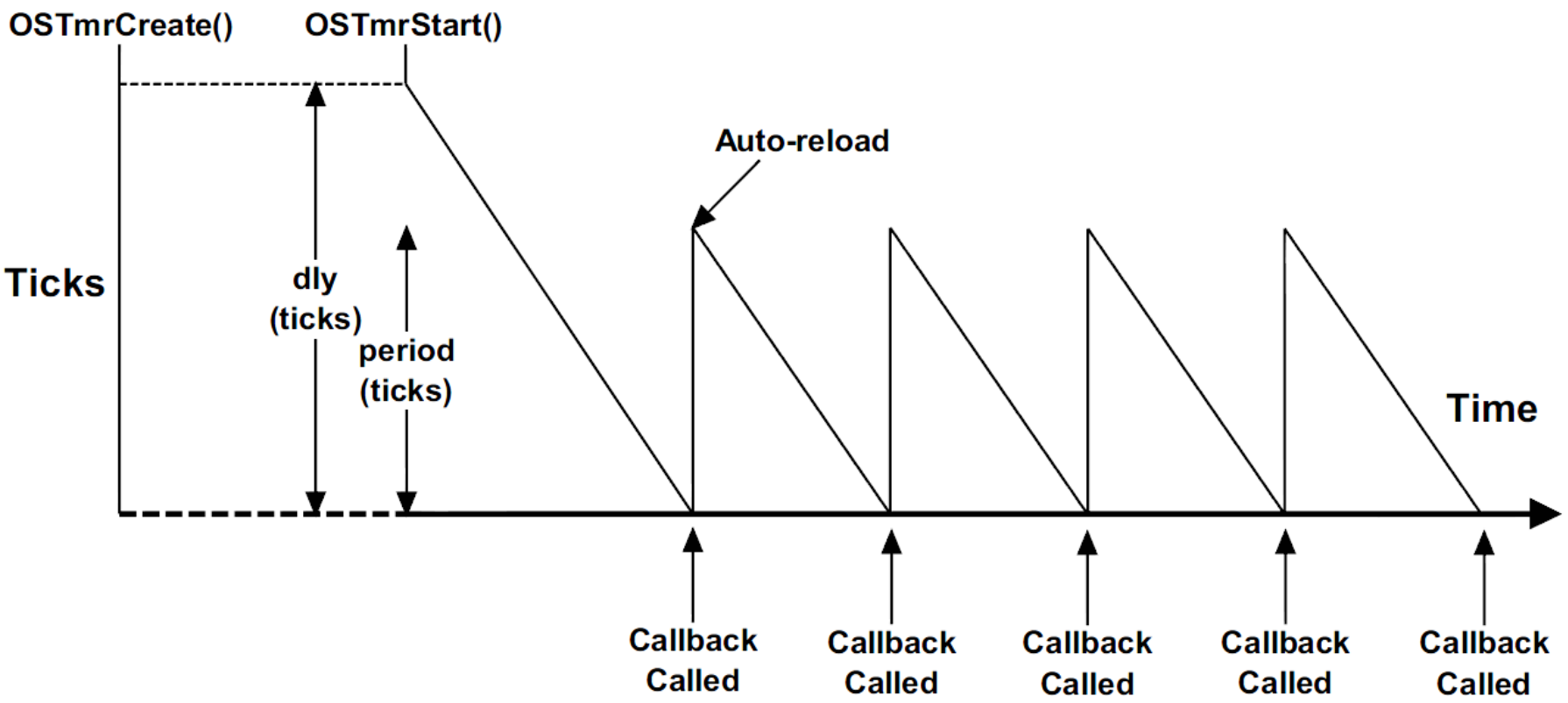
Timers Software

- 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**).

One-Shot Timer



Periodic 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);
}
```

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 100miliSeg.
  - Completar las funciones proporcionadas.

Banderas en CoOs

- Una bandera es una variable del tipo:
  - **OS_FlagID miBandera;**
- 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 ticks del sistema (1 mSeg). Si vale 0, la espera es indefinida.

Esperando una bandera

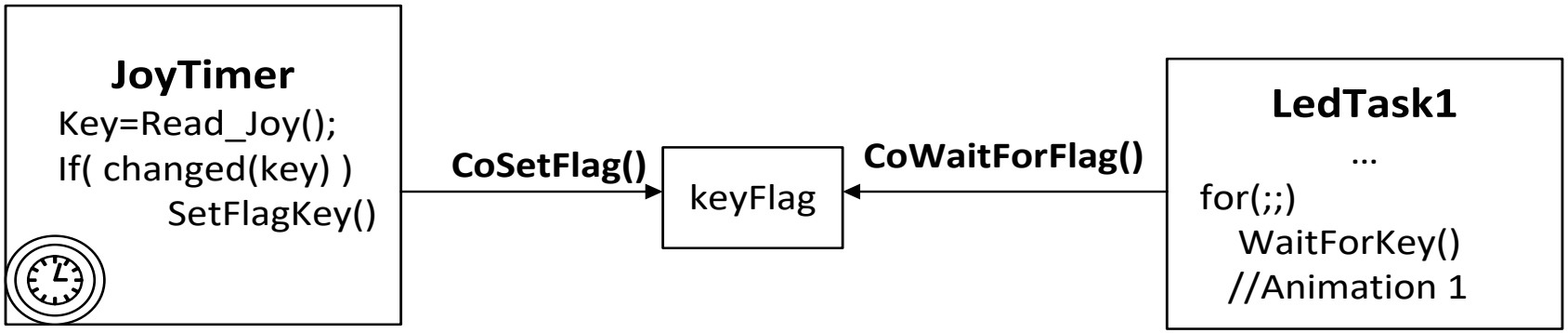
```
void myTaskA(void* pdata)
{
    ..........
    flagID = CoCreateFlag(0,0);       // Reset manually, the original state is not-ready
    CoWaitForSingleFlag(flagID,0);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoSetFlag(flagID);
    ......
}
```

Esperando múltiples banderas

```
void myTaskA(void* pdata)
{
    U32 flag;
    StatusType err;
    ..........
    flagID1 = CoCreateFlag(0,0);     // Reset manually, the original state is not-ready
    flagID2 = CoCreateFlag(0,0);     // Reset manually, the original state is not-ready
    flagID3 = CoCreateFlag(0,0);     // Reset manually, the original state is not-ready
    flag = flagID1 | flagID2 | flagID3;
    CoWaitForMultipleFlags(flag,OPT_WAIT_ANY,0,&err);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoSetFlag(flagID1);
    ......
}
void myISR(void)
{
    CoEnterISR();
    ......
    isr_SetFlag(flagID2);
    CoExitISR();
}
```

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_FlagID keyFlag;**
- 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. **void SetFlagKey(uint8_t key)**
     - Activa la bandera usando **CoSetFlag.**
  2. **void waitForKey (uint8_t key, uint32_t timeOut)**
     - Espera la bandera usando **CoWaitForSingleFlag.**

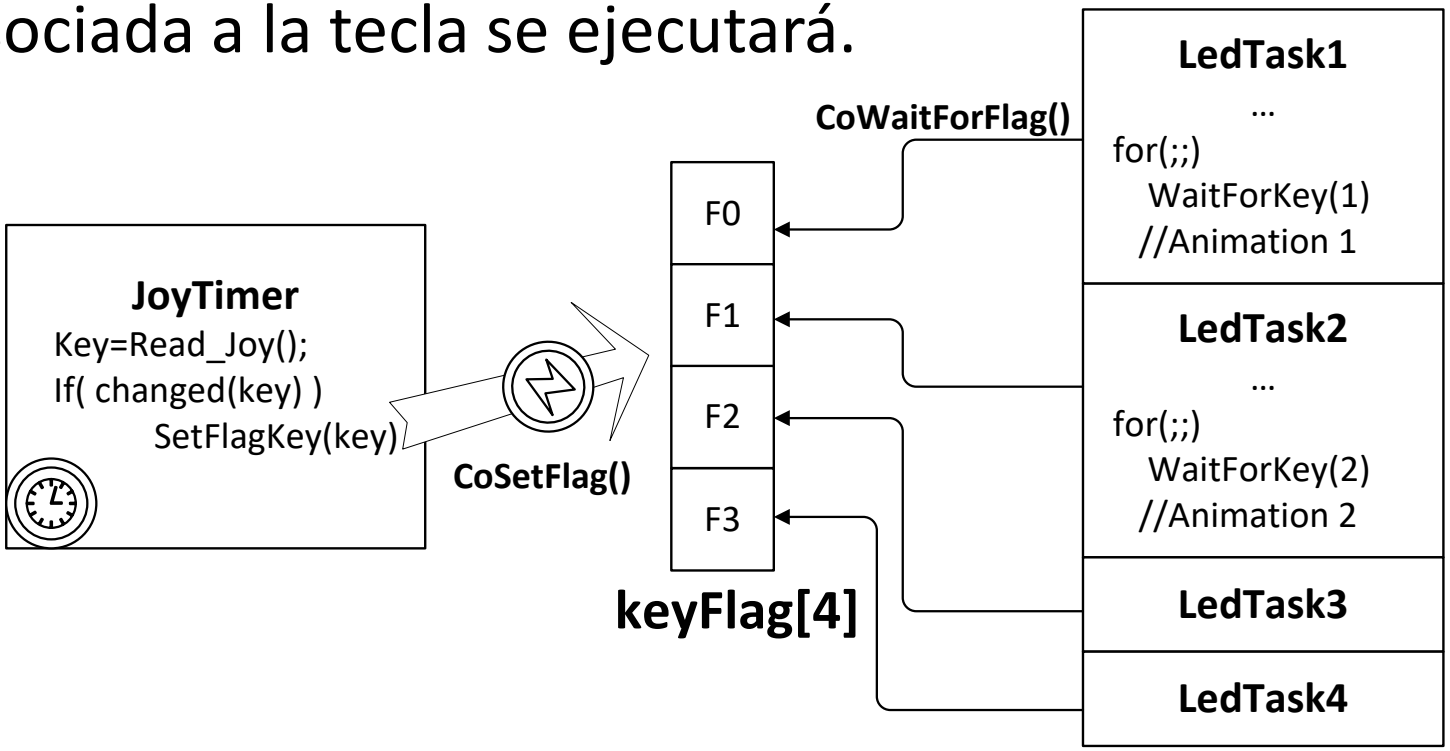


Ejercicio 2: Creando, activando y esperando banderas

1. En **JoyTimer.c**:
    1. Crear la bandera en la función **CreateJoyFlags**
    2. Llamar, en el cuerpo del timer, a **SetFlagKey(0)** cada vez que el valor devuelto por **Read_Joy** cambie.
2. En **LedTask.c**, modificar la tarea de alguna animación para que espere la bandera antes de ejecutarse llamando a **WaitForKey(0,0)**.
3. **Crear la tarea de la animación del led que se ha modificado**.

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.



Ejercicio 3: Sistema multi-bandera

- **En JoyTimer.c**:
  1. Modificar la declaración de la bandera para que ahora sea **un vector de 4 banderas**.
  2. Modificar la función **CreateJoyFlags** para que ahora **inicialice las 4 banderas**. Usar un simple bucle for.
  3. Modificar la función **SetFlagKey** para que sólo active la bandera asociada a la tecla key que recibe como parámetro.
  4. Modificar la función **WaitForKey** para que se quede a la espera de la bandera asociada a la tecla key que recibe como parámetro.

Key	keyFlag
1	keyFlag[0]
2	keyFlag[1]
3	keyFlag[2]
4	keyFlag[3]

Ejercicio 3: Sistema multi-bandera

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

Semáforos en CoOs

- Una semáforo es una variable del tipo:
  - **OS_EventID miSem;**
- 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)**

Ejemplo semáforo binario (inicialmente cerrado)

```
void myTaskA(void* pdata)
{
    ..........
    semID = CoCreateSem(0,1,EVENT_SORT_TYPE_FIFO);
    CoPendSem(semID,0);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoPostSem(semID);
    ......
}
```

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

Ejercicio 4: Acceso a un recurso compartido

```
//Declaración del semáforo
OS_EventID ledSem;

void createLedTask(void){
        Init_Leds();
        ledSem = CoCreateSem(1, 1, EVENT_SORT_TYPE_PRIO);
        //Creación de las tareas de los leds
        //…
}

void LedTaskN(void * parg){
        //Inicializacion de la tarea
        //…
        for (; ;){
                WaitForKey(nKey,0);
                CoPendSem(ledSem, 0);
                //Animación
                //…
                CoPostSem(ledSem);
        }

}
```