SASE 2019

SIMPOSIO ARGENTINO DE SISTEMAS EMBEBIDOS

17 | 18 | 19 DE JULIO

RTOS WORKSHOP

MSC. ING. CARLOS CENTENO
G.IN.T.E.A - UTN FRC

TEMARIO

- Consideraciones del Workshop
- Topología Super Loop
- Topología RTOS
  - Generalidad. Requisitos de Hardware. Opciones Disponibles
  - Tareas
    - TCB
    - Estados Definidos
    - Prioridades

TEMARIO

- Sincronización con Eventos
  - Semaforos
  - Mailbox
  - Queues
  - Deadlock
- Cambio de Contexto
- Resumen de Aspectos Relevantes

CONSIDERACIONES ESPECIALES

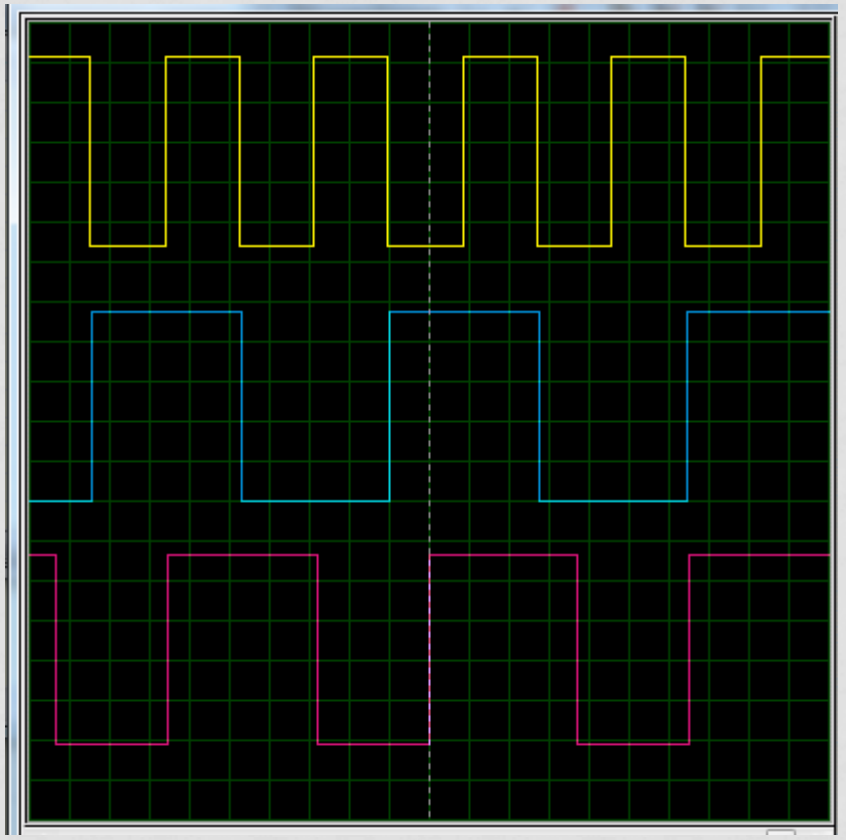
- Cuando usar RTOS
  - Es conveniente
  - Que recursos necesito
  - Cual es la definición concreta de RTOS
- Cuales son las opciones en el mercado
- En el workshop NO se explicará como usar un RTOS determinado.
- El objetivo es conocer los aspectos MAS Relevantes del uso de un RTOS.

SUPER BUCLE

- Resolvamos un pequeño problema.
  - Implementar un Sistema Embebido que controle tres secuencias temporales en salidas digitales.
  - Usar topología Super Loop.
  - El control de tiempo se realiza con espera pasiva.

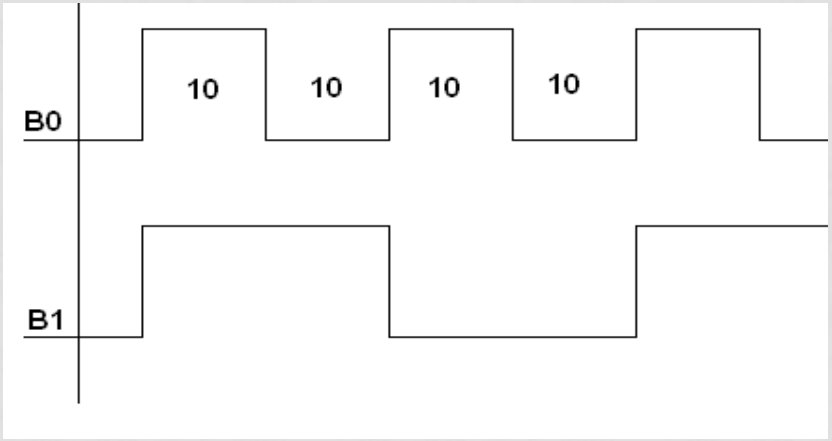
SECUENCIA

- Secuencia 1:
  - Alto : 1mS
  - Bajo : 1mS
- Secuencia 2:
  - Alto : 2mS
  - Bajo : 2mS
- Secuencia 3:
  - Alto : 3mS
  - BAjo: 4mS



SOLUCIÓN

```
B0=0;
B1=0;
while(1)
{
    B0 = B1 = 1;
    delay(10);
    B0=0;
    delay(10);
    B0=1;
    B1=0;
    delay(10);
    B0=0;
    delay(10);
}
```



Compilador 1

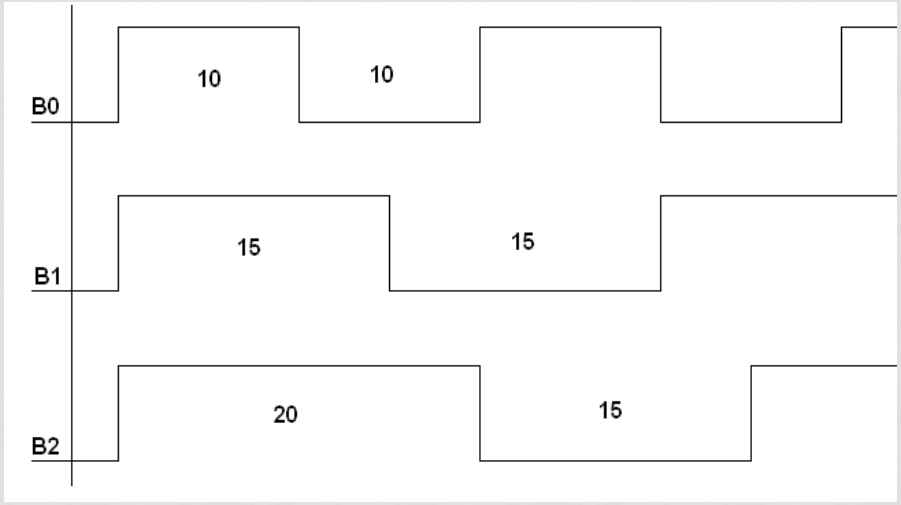
```
#define B0 PIN_B0
#define B1 PIN_B1
```

Compilador 2

```
#define PORTBIT(adr, bit)
    ((unsigned)(&adr)*8+(bit))
static bit          B0  @ PORTBIT(GPIO, 5);
```

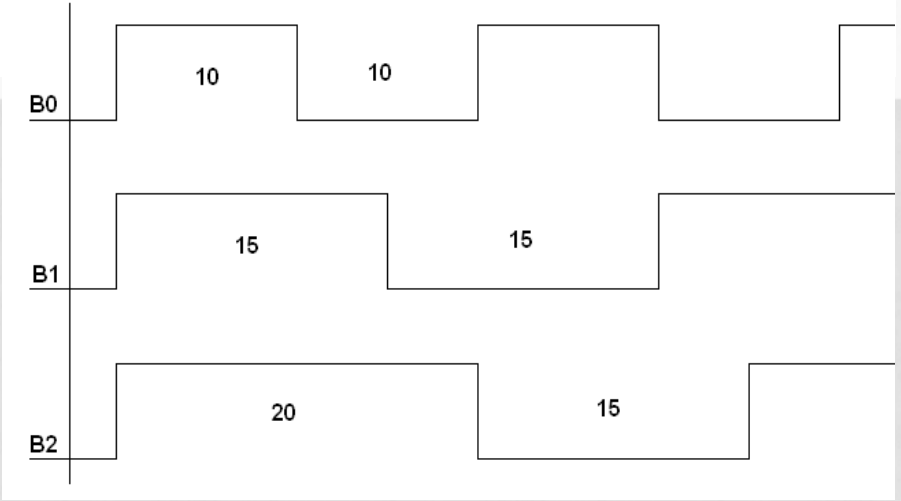
SOLUCIÓN

```
A = B = C = bandera = 0;
B0 = B1 = B2 = 1;
while(1){
    delay(5);
    A++;
    B++;
    C++;
    if(A >= 2){
        B0 ~= B0;
        A = 0;
    }
    if(B >= 4){
        B1 ~= B1;
        B = 0;
    }
```




```
#define masc1 0b00000111
#define masc2 0b11111011
#define masc3 0b11111101
#define masc4 0b11111110

INT bits_control
#define bandera bits_control.0
```

SOLUCIÓN

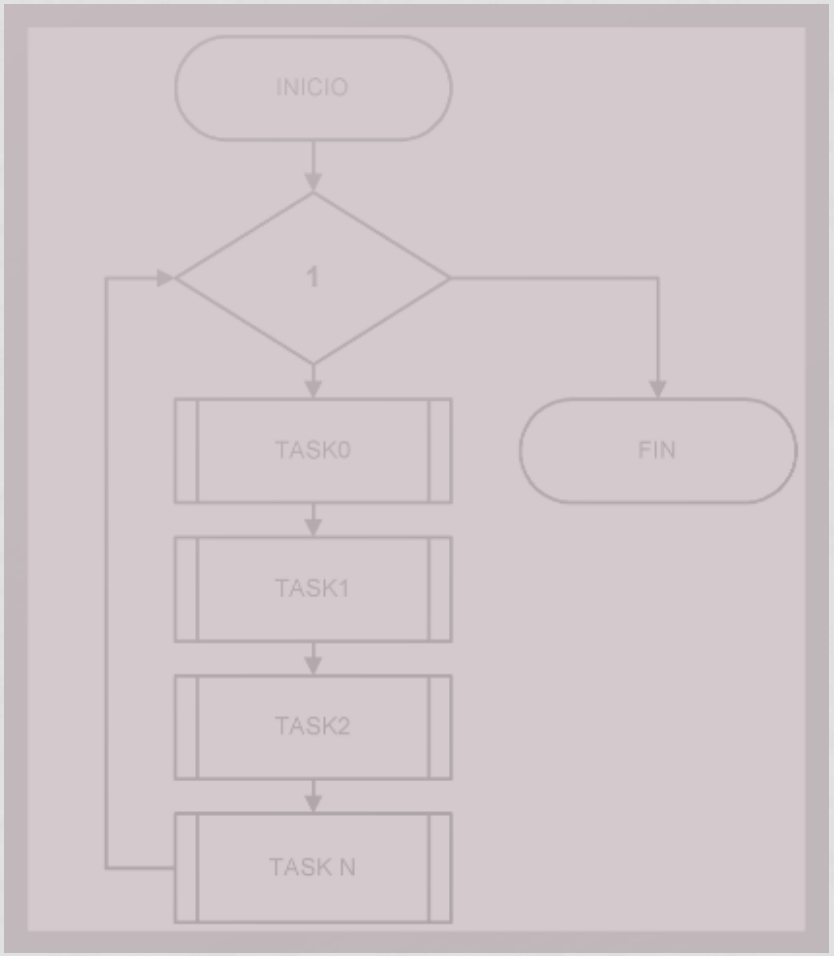



```
A = B = C = 0;
bandera = 0;
B0 = B1 = B2 = 1;
while(1){
    if(bits_tiempo){
        bits_tiempo = 0;
        if(A>=2)  {
            B0 ~= B0;
            A = 0;
        }
        if(B>=4){
            B1 ~= B1;
            B = 0;
        }
        if(C >=6 && bandera == 0){
            bandera = 1;
            B2 = 0;
        }else if(C >=8 && bandera == 1){
            B2 = 1;
            C = 0;
            bandera = 0;
        }
    }
}
```

```
static void interrupt isr(void) {
    if(TMR1IF && TMR1IE){
        TMR1IF=0;    // borro FLAG ISR
        TMR1H=0xCF;              // reinicio TIMER
        #asm
        nop
        nop
        #endasm
        TMR1L=0x32;              // 100mSeg
        bits_tiempo = 1;
        A++;  B++;  C++;
        }
    PEIE=1; GIE=1;
}
```

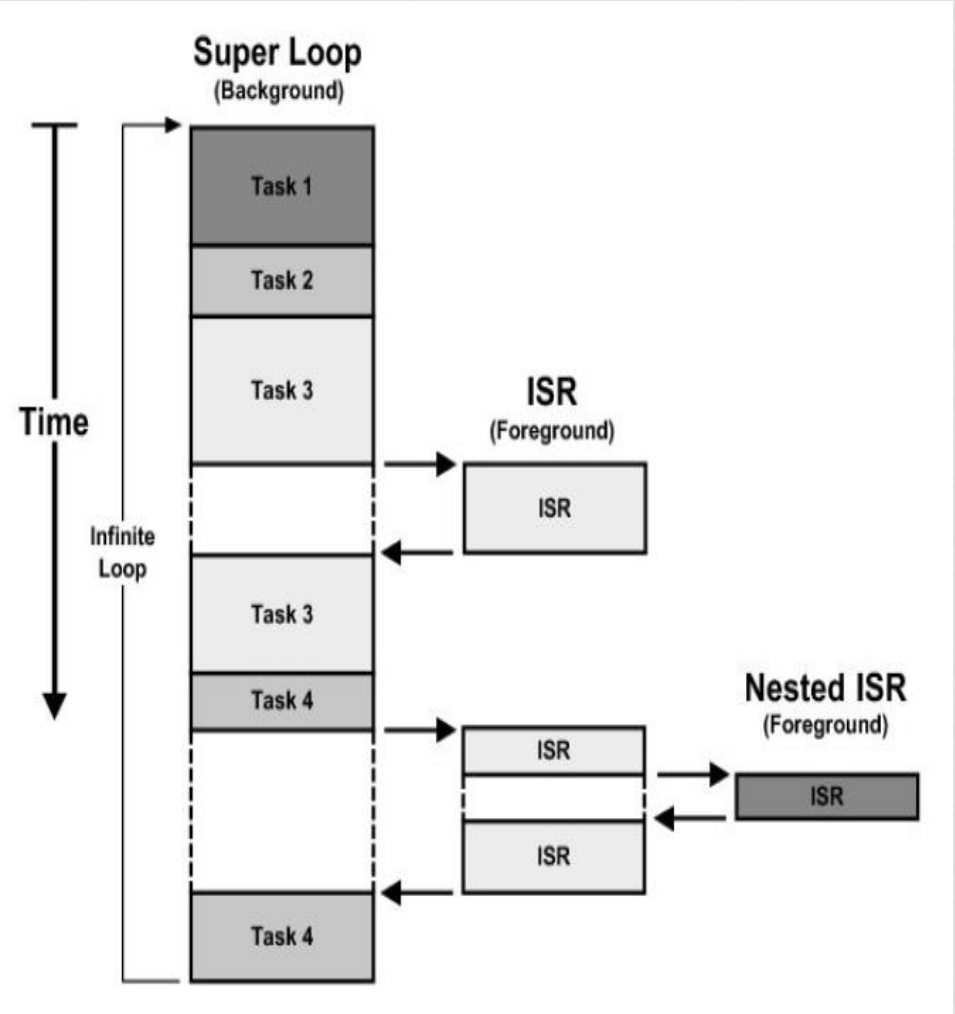
SUPER BUCLE

```
void main(void)
{
    for(;;)
    {
        Task1();
        Task2();
        Task3();
    }
}
```



SUPER BUCLE

- Se llama a MODULOS (funciones)
- Tenemos un servicio de interrupciones.
- El tiempo de ejecución depende de cuanto se demora en terminar el ciclo completo.
- Si modificamos el código este tiempo CAMBIA



SUPER BUCLE

- Cada ***"tarea"*** es una función en C.
- Se llaman por turno desde el bloque principal.
  - Se ejecutan rápidamente y regresan al bloque principal.
  - Pueden usar una variable de estado.
  - Se usan esperas pasivas – delay.
    - **NO** existen prioridades.
    - **NO** hay timers.
    - **NO** hay comunicación entre tareas

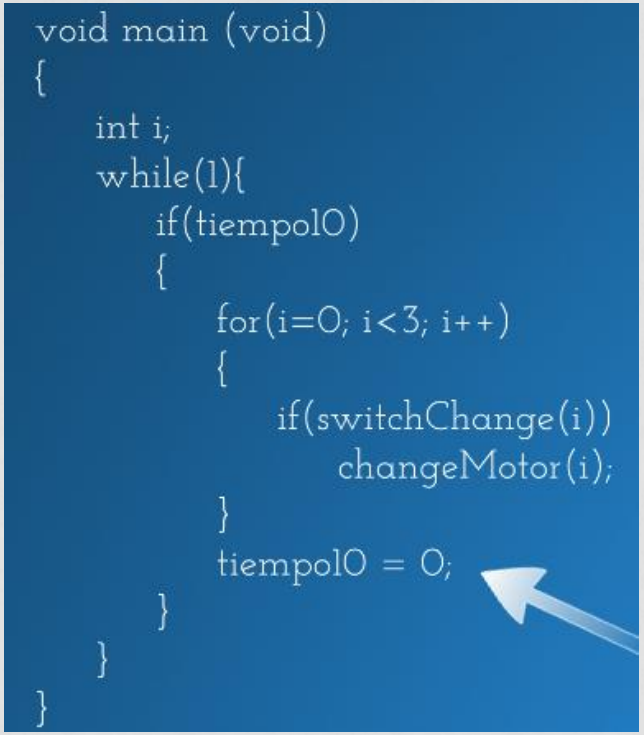
CONTROL DEL TIEMPO - TIMERS

- Para el control del Tiempo se puede utilizar la ISR de un timer para incrementar una variable Global.

```
unsigned int Tics;        // variable global
void timerIsr(void)
{
        Tics++;
}
```

CONTROL DEL TIEMPO

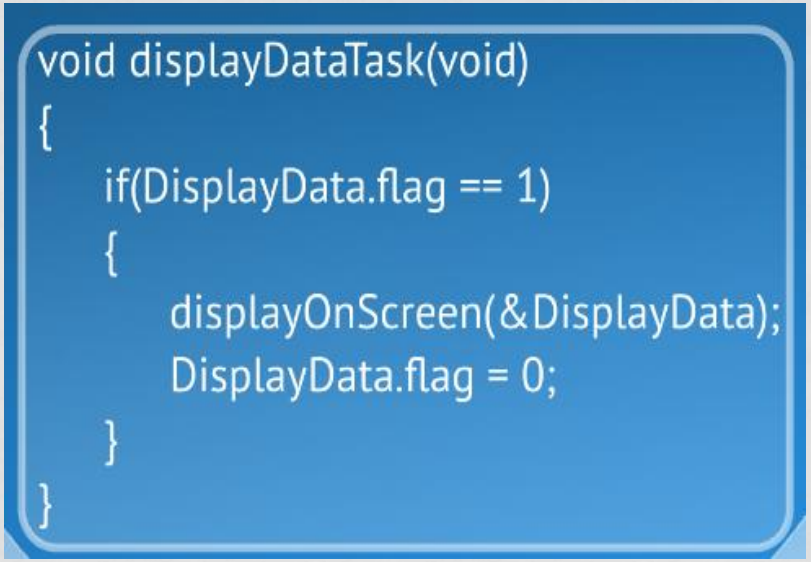
- La bandera tiempo10 se incrementa en la ISR de un timer.



```
void main (void)
{
    int i;
    while(1){
        if(tiempo10)
        {
            for(i=0; i<3; i++)
            {
                if(switchChange(i))
                    changeMotor(i);
            }
            tiempo10 = 0;
        }
    }
}
```

COMUNICACIÓN

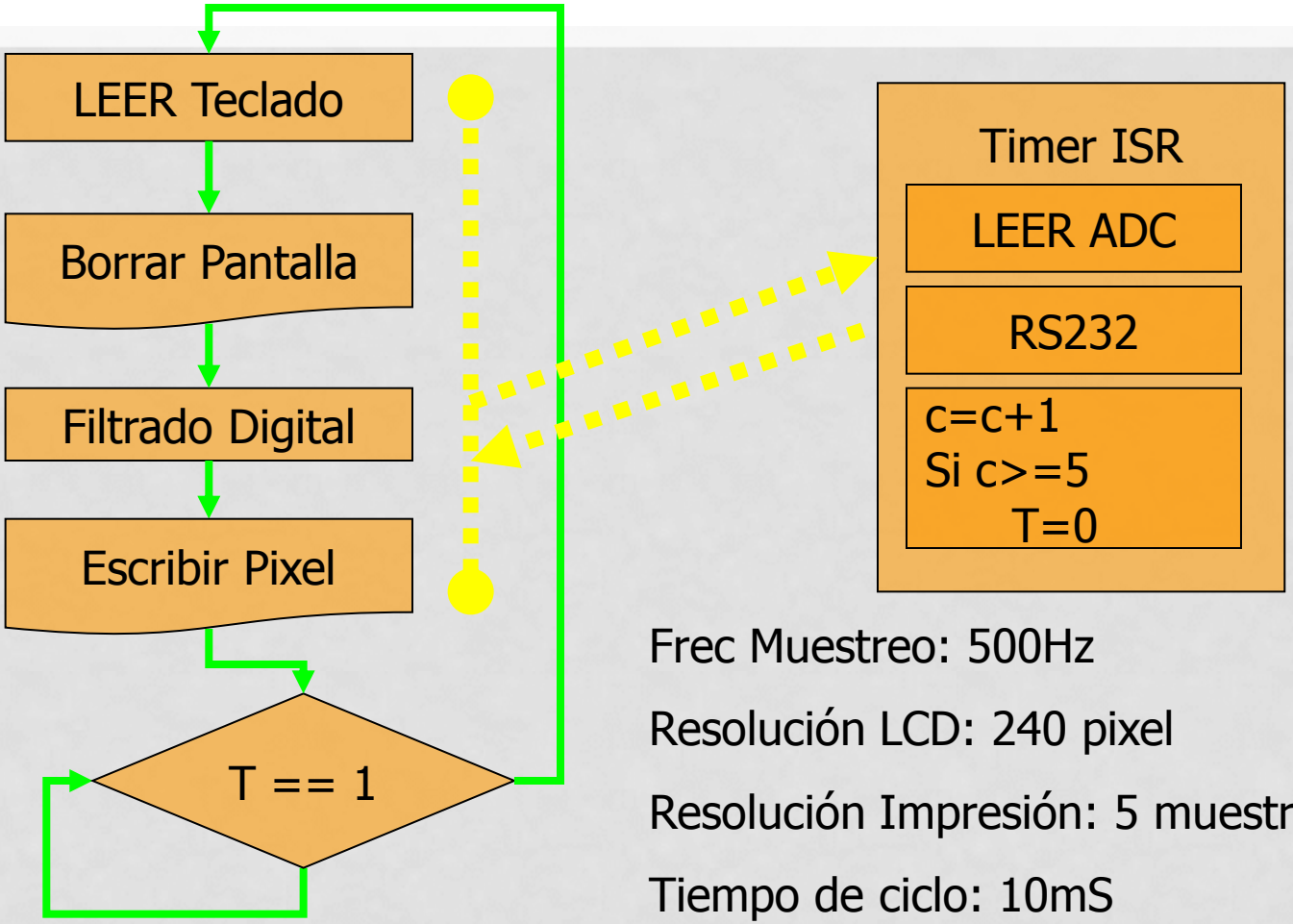
- Para la comunicación entre tareas se utiliza una variable global que contiene el flag de señalización y el dato propiamente dicho.



```
void displayDataTask(void)
{
    if(DisplayData.flag == 1)
    {
        displayOnScreen(&DisplayData);
        DisplayData.flag = 0;
    }
}
```

PRIORIDADES

- La prioridad de ejecución depende del orden en el bloque principal.
- Una proceso crítico debe ejecutarse dentro de un servicio de Interrupciones.



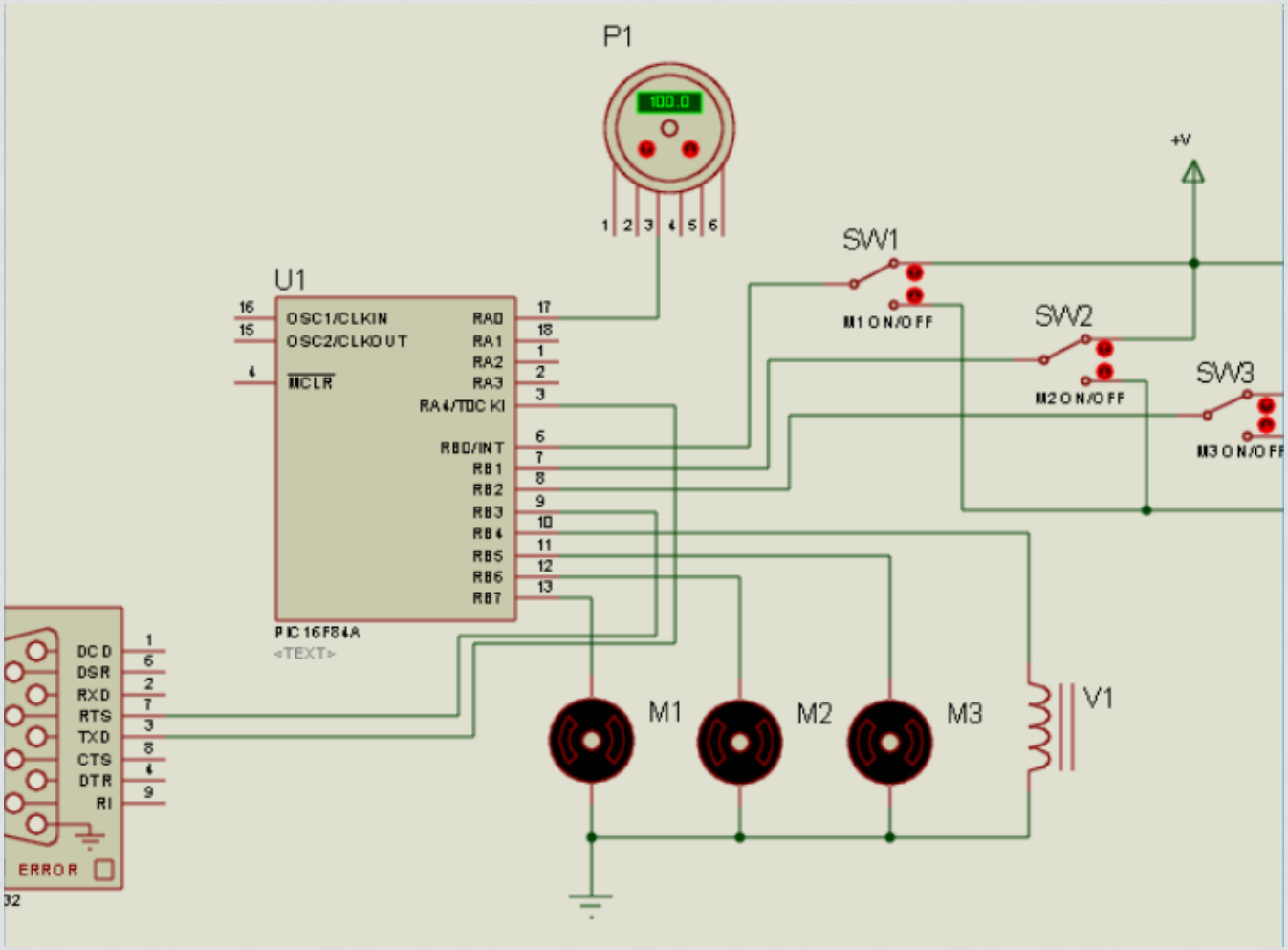
Frec Muestreo: 500Hz

Resolución LCD: 240 pixel

Resolución Impresión: 5 muestras por pixel

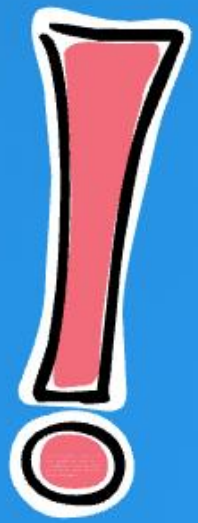
Tiempo de ciclo: 10mS

EJEMPLO DE SISTEMA



SOLUCIÓN PROPUESTA

```
void main (void)
{
    int i;
    for(;;)
    {
        checkMotorsSwitch();
        checkPresion();
        checkRS232();
    }

}
```



SOLUCIÓN PROPUESTA

```
void main (void)
{
    int i;
    for(;;)
    {
        checkMotorsSwitch();
        checkPresion();
        checkRS232();
    }

}
```

Si la velocidad de ejecución del lazo completo es menor que la velocidad de los datos que llegan por el puerto serial, es posible perder información

- Con control del Tiempo del ciclo para cada proceso

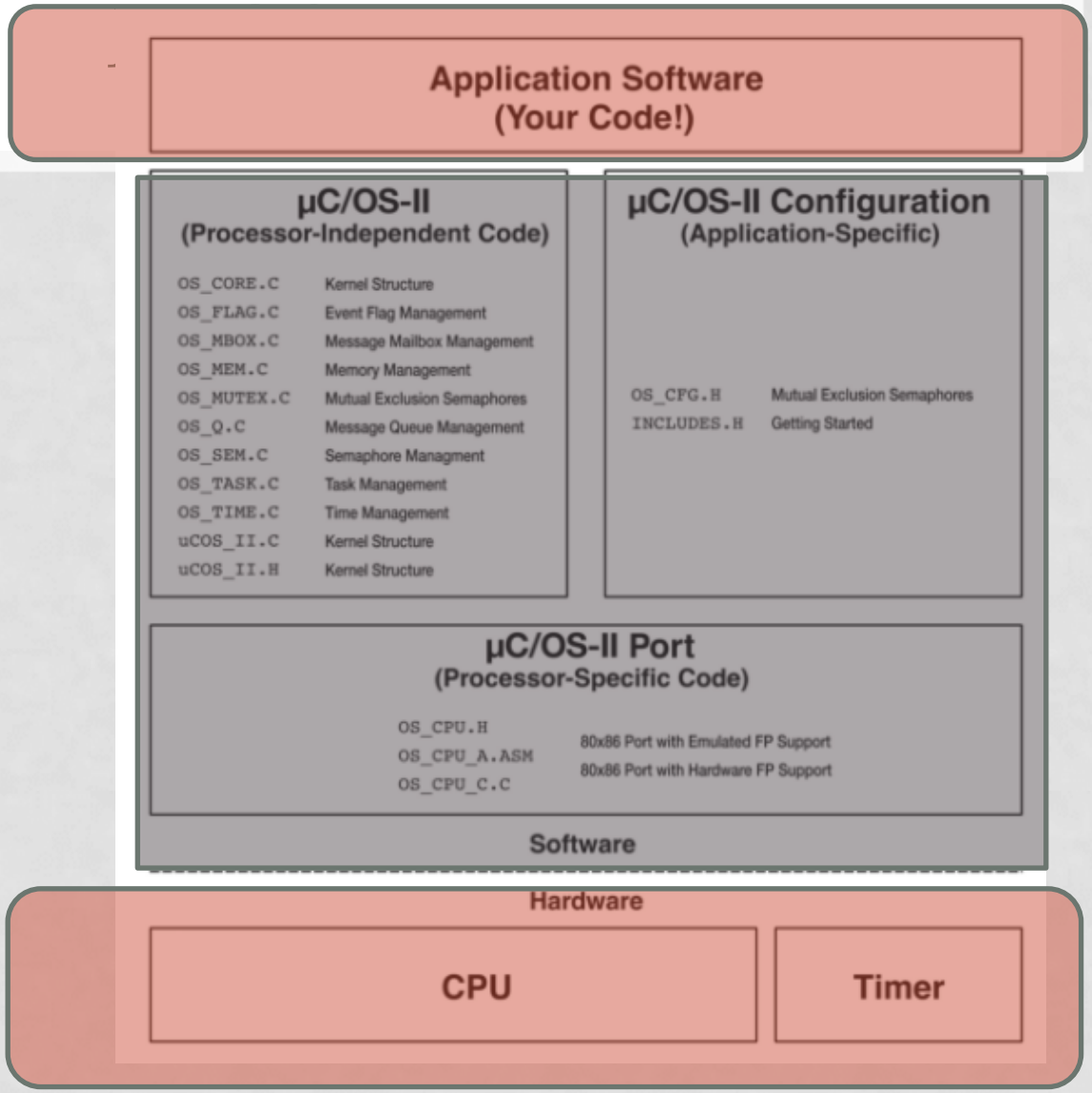
```
if(tiempo10)
{
    for(i=0; i<3; i++)
    {
        if(switchChange(i))
            changeMotor(i);
    }
    tiempo10 = 0;
}
```

```
if(tiempo50)
{
    switch(valveState)
    {
        case OPEN:
            if(presion < 90)
            {
                closeValve();
                valveState = CLOSE;
            }
            break;
        case CLOSE:
            if(presion > 100)
            {
                openValve();
                valveState = OPEN;
            }
            break;
    }
    tiempo50 = 0;
}
```

¿CONVIENE USAR RTOS?

- Se debe pensar en cambio de plataforma.
  - Uso de STACK
  - Micro del ejemplo no soporta RTOS
- La solución super loop es compacta y fácil de entender .
- No se pueden implementar prioridades.
- El tiempo del bucle depende de como se resolvió a nivel de código la solución del problema.

- Estructura final de software si usamos un RTOS como uCOS-II



PORQUE USAR UN RTOS

- Se evalúa el sistema y surge:
  - Hay tareas independientes
    - Botones para Configuración
      - TASK 1
    - Conversor Analógico Digital – ADC
      - TASK 2
    - Comunicación Serie
      - TASK 3
  - Hay un tiempo para respetar
  - Hay una comunicación serie que puede requerir una cola de mensajes

RTOS - GENERALIDAD

- Se introducen conceptos
  - ***Tareas***
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - Reentrancia
  - Eventos
    - ECB

- Las procesos del super loop se dividen en tareas.
  - Se debe asignar un Task Control Block (TCB).
  - Se debe asignar STACK.
  - Se debe asignar una PRIORIDAD.

RTOS - GENERALIDAD

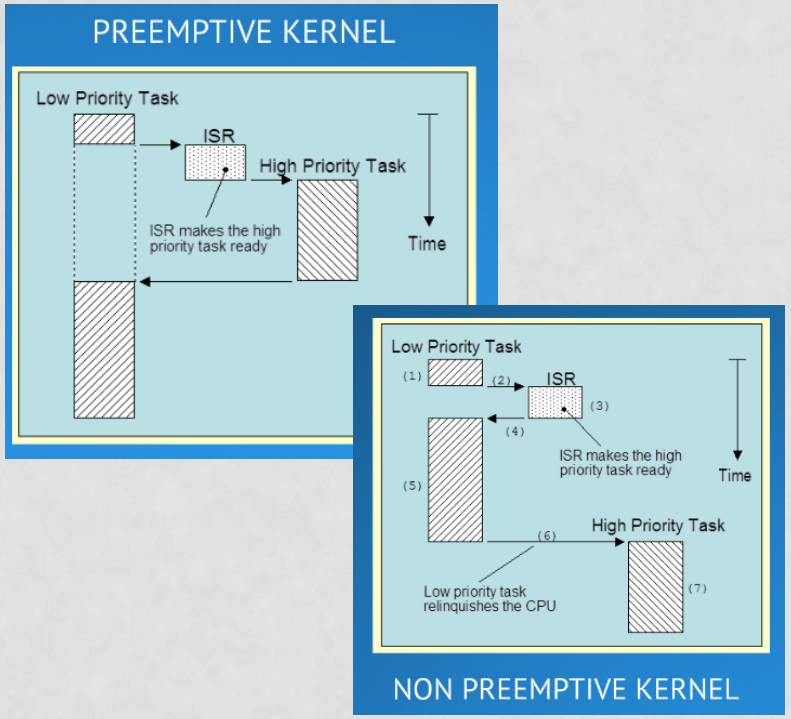
- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
  - ***Estados Definidos***
  - Preemptive
  - Reentrancia
  - Eventos
    - ECB

- Según el RTOS que se elija serán los estados posibles de cada tarea.

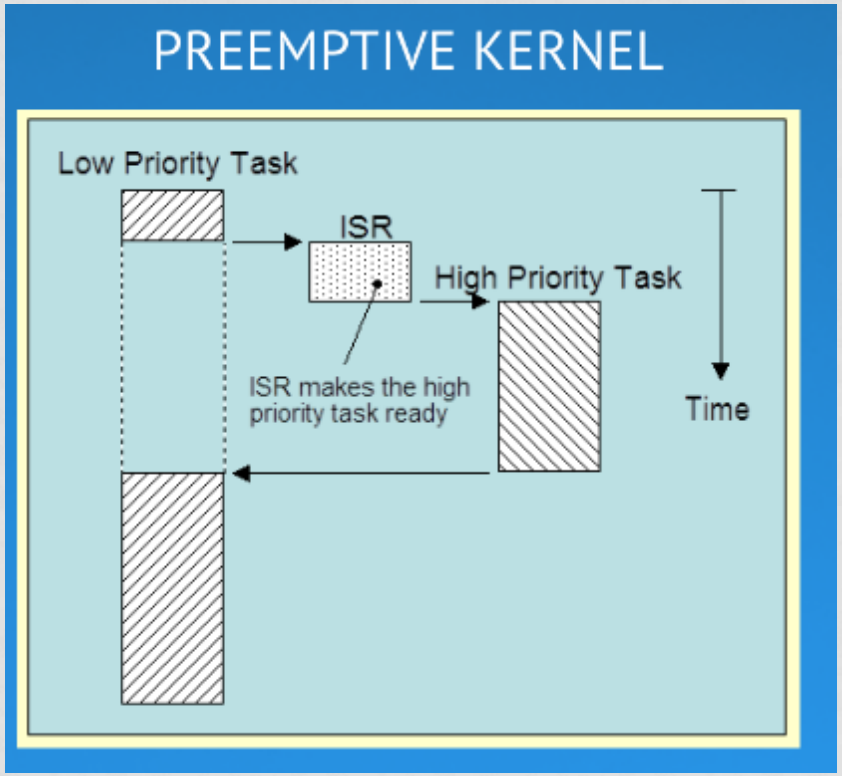
RTOS - GENERALIDAD

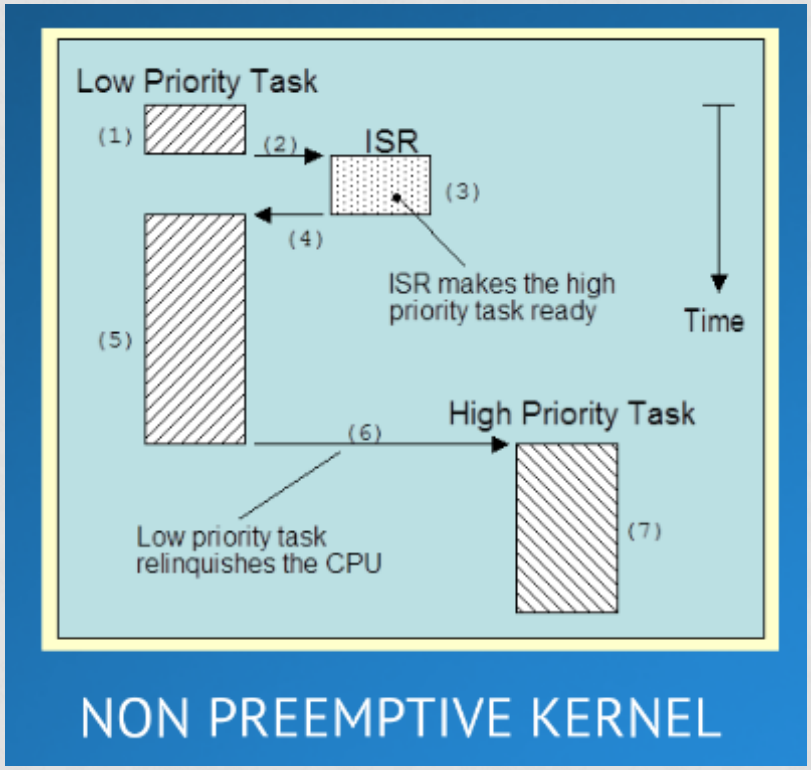
- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - ***Tipo RTOS***
  - Reentrancia
  - Eventos
    - ECB

- TIPOS



RTOS - GENERALIDAD

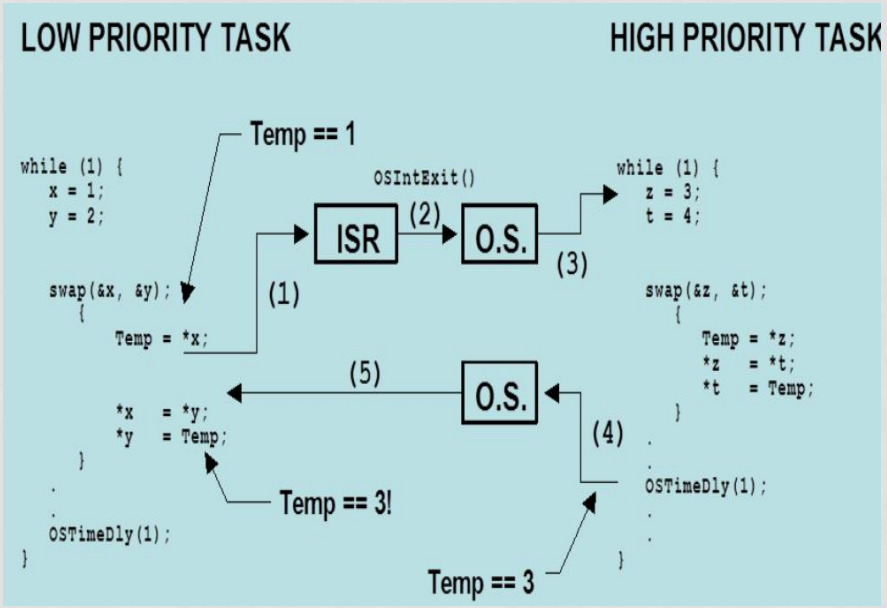




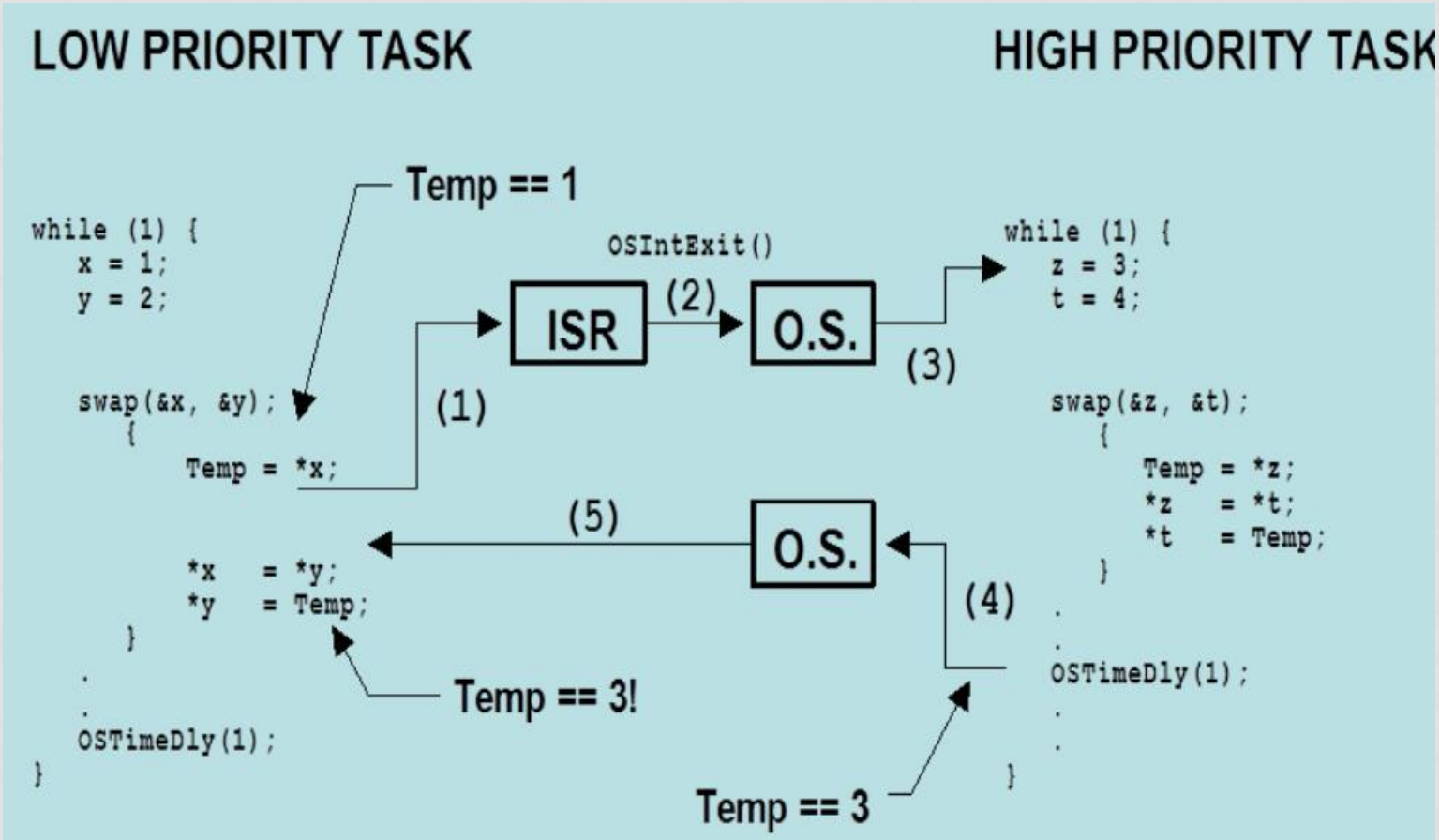
RTOS - GENERALIDAD

- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - ***Reentrancia***
  - Eventos
    - ECB

- Reentrancia



RTOS - GENERALIDAD



RTOS - GENERALIDAD

- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - Reentrancia
  - ***Eventos***
    - ECB

- Eventos
  - Semáforos
    - Binarios
    - MUTEX
    - Contadores
  - Mailbox
  - Queues

REQUISITOS DE HARDWARE

- Memoria RAM
  - Para almacenar las estructuras de control del RTOS.
    - TCB
    - ECB
    - Stack
- Memoria ROM
  - Para almacenar la Aplicación, el PORT, y las funcionalidades el RTOS.
- Timer
  - Para generar la base de tiempo del RTOS

OPCIONES RTOS

- uCOS III → https://www.micrium.com
  - Certificado FDA
  - Certificado Aeroespacial.
- freeRTOS → https://www.freertos.org/
  - Opción Certificada para industria, espacial, biomedicina →SafeRTOS.
  - freeRTOS+TCP
- AVIX RT → http://www.avix-rt.com/
- THREADX RTOS → https://rtos.com/

OPCIONES RTOS

- mbedOS
- NuttX
- ErikaOS
- QNX.
- Linux.
  - NO es un HARD - Real Time
  - Posee stacks de comunicaciones de primer nivel.
- Integriti
- ChibiOS

OPCIONES RTOS

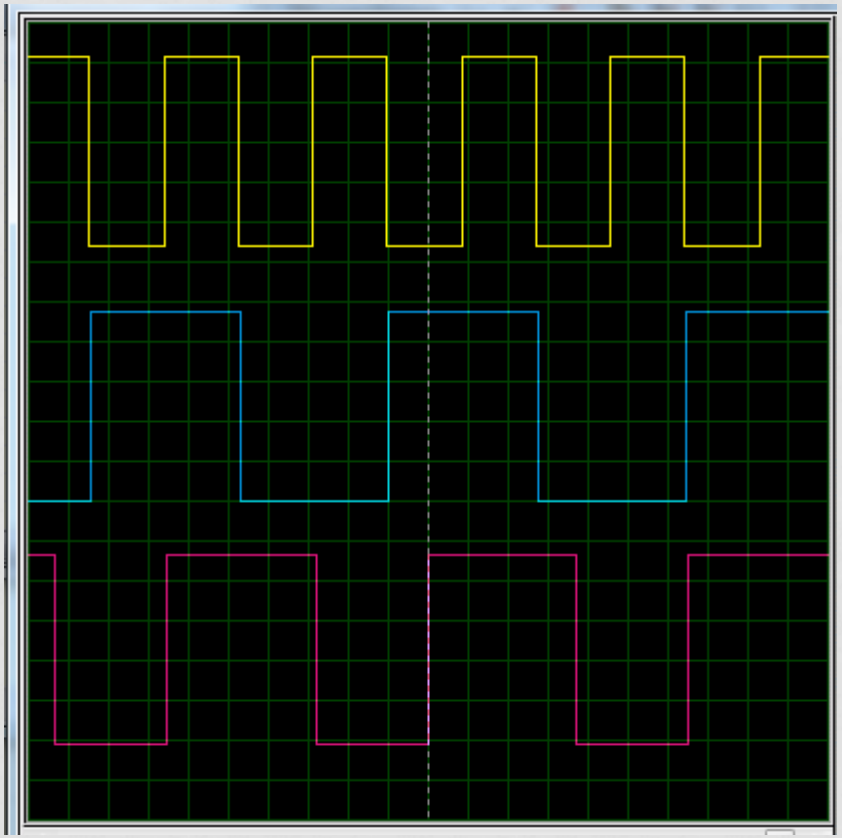
- **uCOS III**
  - https://www.micrium.com/rtos/kernels/
- **freeRTOS**
  - https://www.freertos.org/
- **AVIX RT**
  - http://www.avix-rt.com/
- **THREADX RTOS**
  - https://rtos.com/

EJEMPLO 1

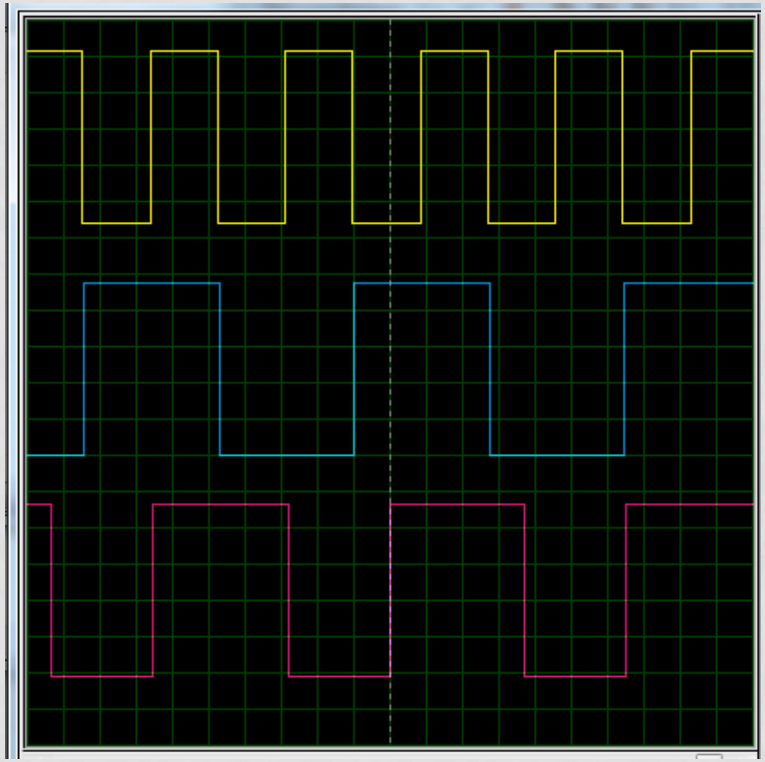
- Implementar un Sistema Embebido que controle tres secuencias temporales en salidas digitales.
- Usar topología Super Loop.
- El control de tiempo se realiza con espera pasiva.

SECUENCIA

- Secuencia 1:
  - Alto : 1mS
  - Bajo : 1mS
- Secuencia 2:
  - Alto : 2mS
  - Bajo : 2mS
- Secuencia 3:
  - Alto : 3mS
  - BAjo: 4mS



SECUENCIA - SOLUCIÓN



```
// SUPER loop
for(;;){
    switch(cont1){
        case 2: salidaLed1 = 1;
                break;
        case 4: salidaLed1 = 0;
                cont1 = 0;
                break;
    }
    switch(cont2){
        case 4: salidaLed2 = 1;
                break;
        case 8: salidaLed2 = 0;
                cont2 = 0;
                break;
    }
    switch(cont3){
        case 3: salidaLed3 = 1;
                break;
        case 7: salidaLed3 = 0;
                cont3 = 0;
                break;
    }
    Delay10TCYx(20L);
    cont1++;
    cont2++;
    cont3++;
}
```

Espera

SOLUCIÓN - RTOS

- Se deben conocer aspectos específicos de RTOS.
  - Secciones Críticas
  - Reentrancia
  - Tareas
  - Estado de las Tareas
  - Bloque de control de las tareas
  - Inicialización
  - Arranque del kernel

SECCIONES CRÍTICAS

SECCIONES CRÍTICAS

- Se pueden definir secciones que denominaremos “críticas” en donde se trabaja con memoria, registros, etc.
- El uso de estos identificadores logran evitar que se produzca durante esta operación alguna interrupción ya sea por software o por hardware.

SECCIONES CRÍTICAS

- OS_ENTER_CRITCAL();
- OS_EXIT_CRITICAL();

- A nivel de código se definen como

```
#define  OS_CRITICAL_METHOD   2

#if     OS_CRITICAL_METHOD == 1
#define  OS_ENTER_CRITICAL()  asm  CLI
        •    /* Disable interrupts                    */
#define  OS_EXIT_CRITICAL()   asm  STI
        •    /* Enable  interrupts                    */
#endif

#if     OS_CRITICAL_METHOD == 2
#define  OS_ENTER_CRITICAL()  asm {PUSHF; CLI}
        •    /* Disable interrupts                    */
#define  OS_EXIT_CRITICAL()   asm  POPF
        •    /* Enable  interrupts                    */
#endif
```

- // Estas estan DEFINIDAS EN OS_CPU.H

SECCIONES CRÍTICAS

- Ejemplo de uso de Secciones Críticas

```
OS_ENTER_CRITICAL();

PC_VectSet(0x08, OSTickISR);
/* Install uC/OS-II's clock tick ISR       */

PC_SetTickRate(OS_TICKS_PER_SEC);
/* Reprogram tick rate                     */

OS_EXIT_CRITICAL();
```

RTOS - KERNEL

- Conjunto de funciones que llevan a cabo todas las capacidades de un RTOS.

- Se encarga del cambio entre las tareas.
- Se encarga de la comunicación entre las tareas.
- Se encarga del manejo de semáforos, mailboxes, colas, demoras de tiempo.
- Necesita de ROM, RAM y tiempo de ejecución adicional para trabajar.

RTOS - SCHEDULER

- Es el encargado de determinar que tarea es la que se debe ejecutar.
- Necesita de la correcta distribución de prioridades para operar.
- Se ejecuta cada vez que se produce una interrupción del sistema.
  - ISRTick();
- Se ejecuta al final de la ejecución de los servicios del kernel.

RTOS – TIPOS DE KERNEL

- De acuerdo a la forma en que el scheduler administra el uso del CPU se dice que existen distintos tipos de Kernel.
- Predominan dos formas específicas
- Formas de trabajo del sistema:
  - PREEMPTIVE
  - Hard Real Time
  - NON PREEMPTIVE
  - Soft Real Time

RTOS – TIPOS DE KERNEL

- La diferencia entre Hard y Soft Real-time es la tolerancia a no cumplir con los tiempos establecidos, lo que pueda llevar a una falla catastrófica.
- Para sistemas Hard Real Time, no es opción no cumplir con los tiempos.
- En sistema Soft Real Time no es crítico el no cumplir con los plazos.

RTOS – TIPOS DE KERNEL

Hard Real Time

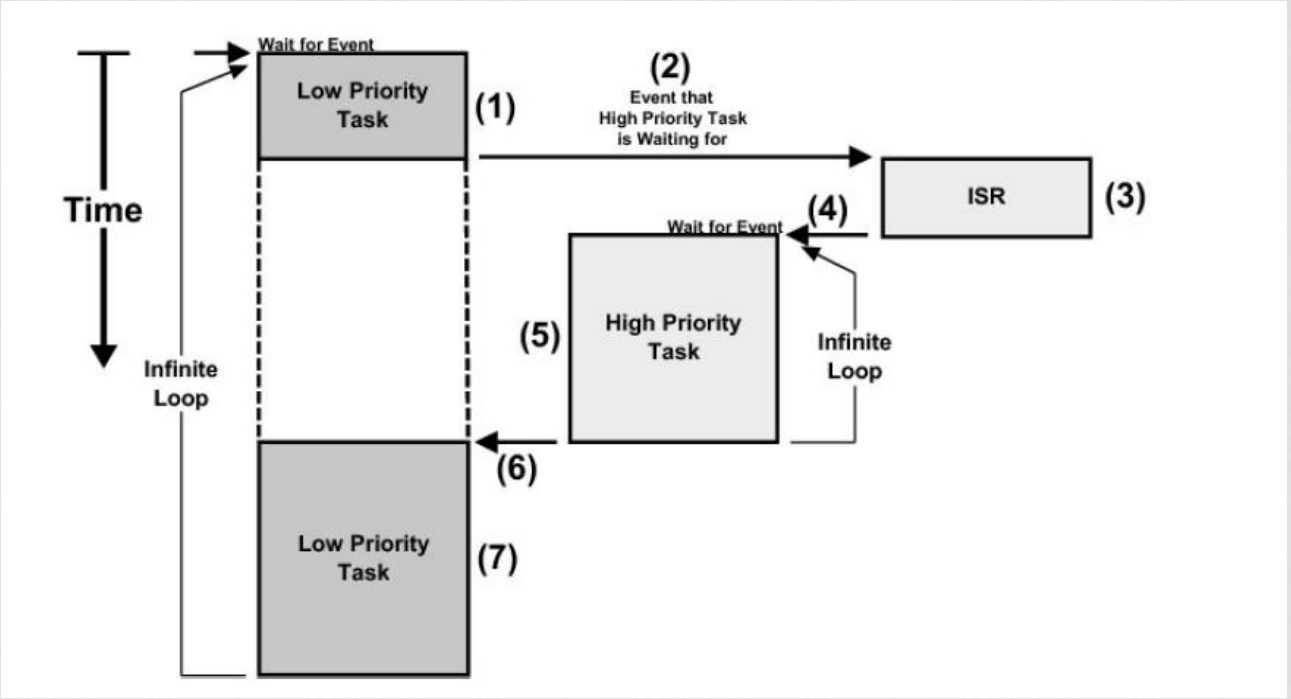
- Plazo de respuesta ESTRICTO.
  - Preemptive.
- Comportamiento temporal determinado por el diseño del sistema.

Soft Real Time

- Plazo de respuesta FLEXIBLE.
  - NON Preemtive
- Comportamiento temporal determinado por el computador.

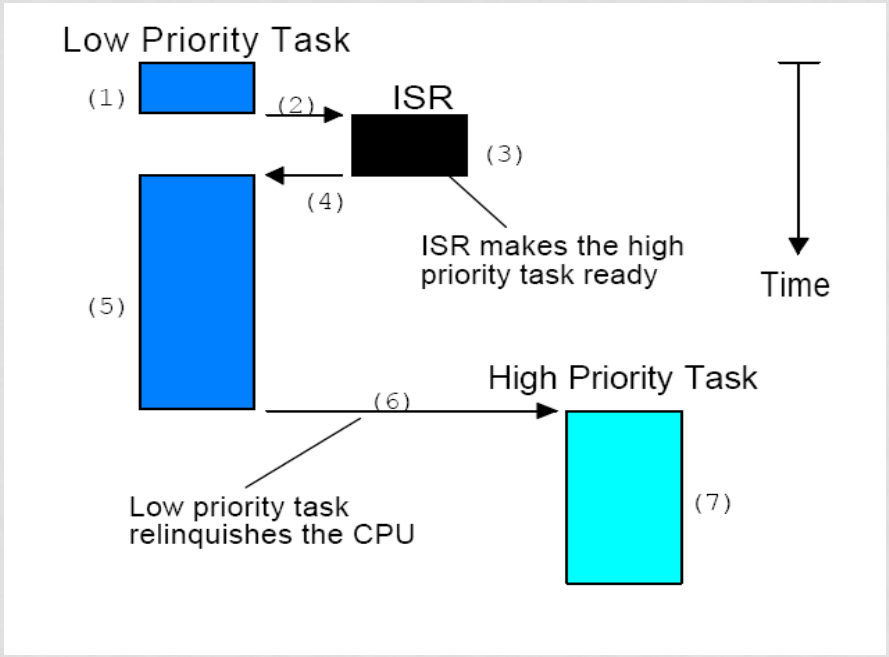
RTOS – Preemptive

- La de mas alta prioridad SIEMPRE se ejecuta cuando es necesario.
- Se deben usar funciones reentrantes.



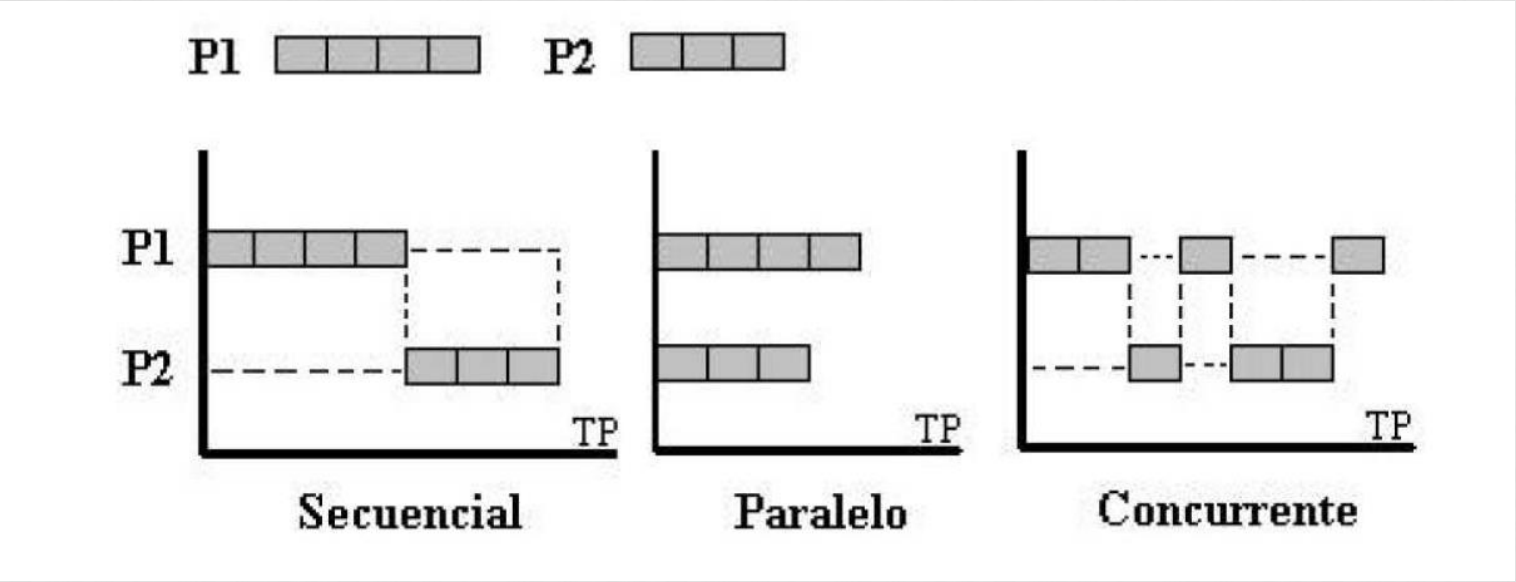
RTOS – NonPreemptive

- ISR → Tarea de Mayor prioridad
- Se vuelve a la TAREA interrumpida.
- La de mas alta prioridad "READY" solo se ejecuta cuando la "ACTUAL" termina su ejecución.
- Usar funciones reentrantes.



RTOS – MULTITAREA

- Se busca que el tiempo de procesamiento repartido entre las tareas (hilos) cree la ilusión de procesamiento concurrente.



SOLUCIÓN - RTOS

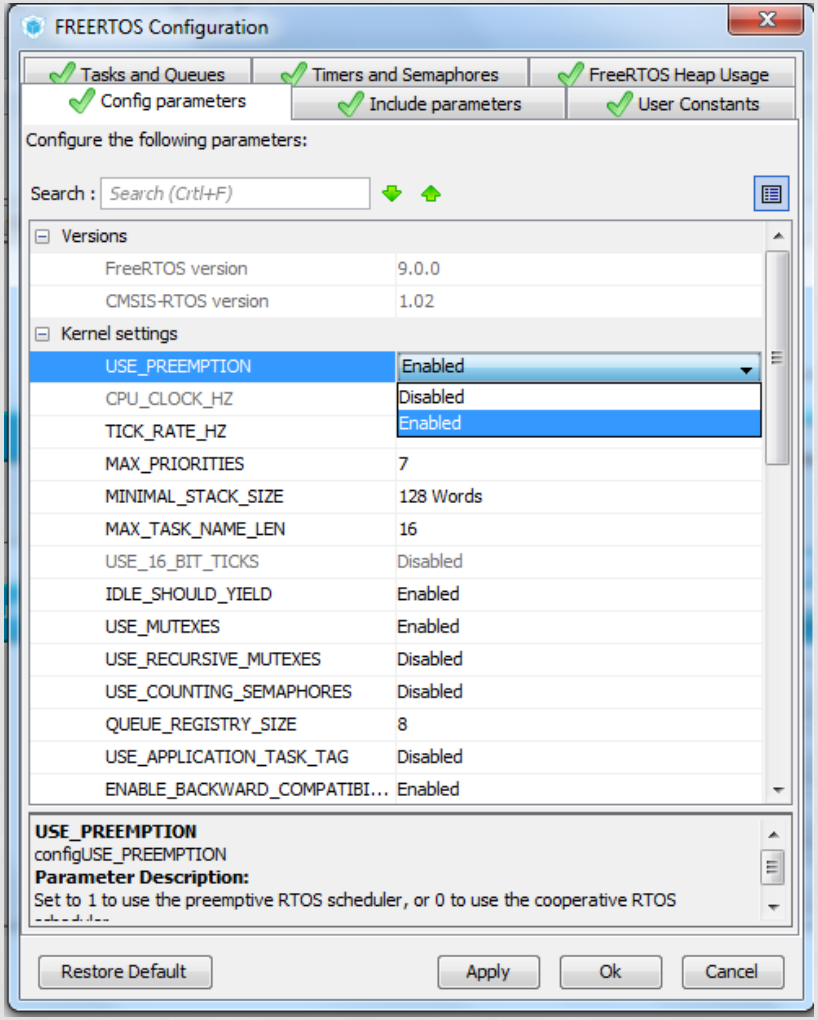
- Configuración Inicial del RTOS
- Debo Crear las Tareas
  - Requieren Task Control Block - TCB.
  - Requieren STACK.
  - Se deben definir Prioridades.
- Se deben conocer los estados posibles.
  - Se debe tener presente en el diseño el tiempo que necesita el RTOS para funcionar.
- Se deben usar SERVICIOS del KERNEL.

CONFIGURACIÓN INICIAL

- Se debe definir la estructura inicial del sistema RTOS.
  - Activar / Desactivar funcionalidades.
  - Establecer la cantidad de Tareas que se usarán.
  - Establecer la cantidad de Eventos que se utilizarán.
  - Definir el modo de trabajo.
    - Preemptive/Non Preemptive/Round Robin

CONFIGURACIÓN INICIAL FREERTOS

- Activar / Desactivar funcionalidades.
- Establecer la cantidad de Tareas que se usarán.
- Establecer la cantidad de Eventos que se utilizarán.
- Definir el modo de trabajo.
  - Preemptive/Non Preemptive/Round Robin

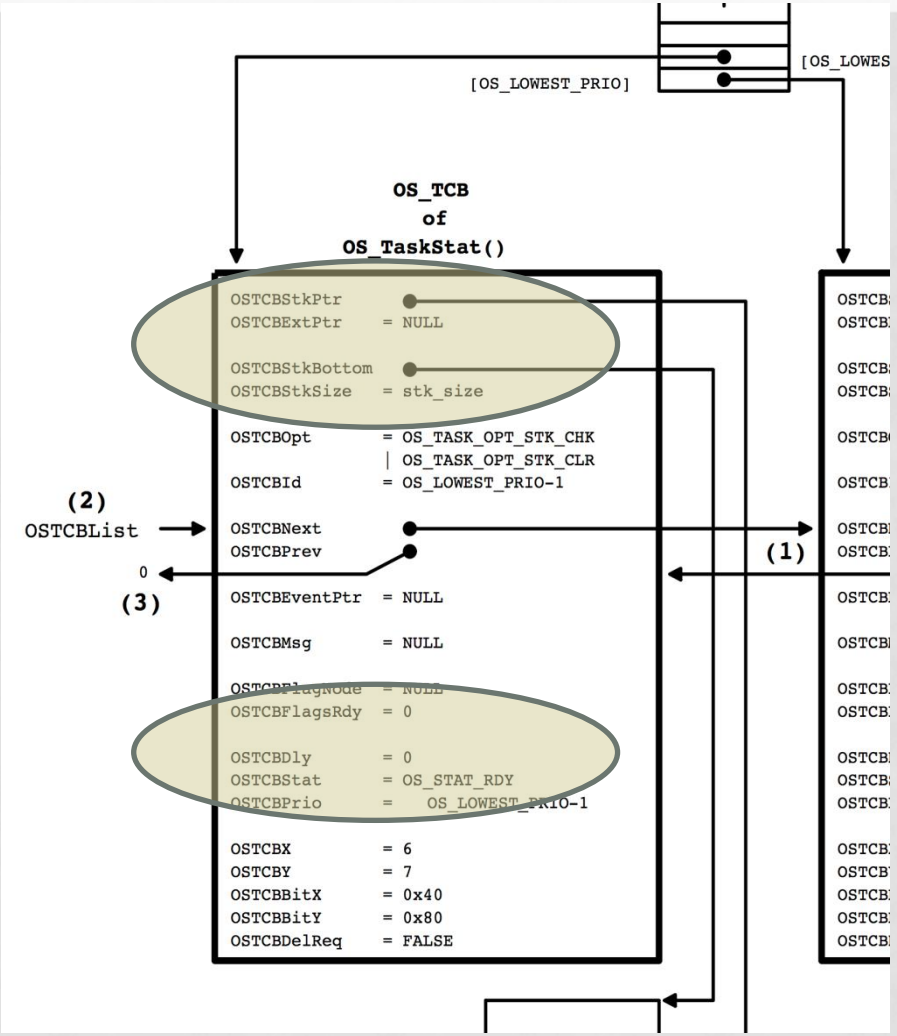


CONFIGURACIÓN INICIAL UCOS-II

```
#define OS_MAX_TASKS             8L    /* Max. number of tasks in your application ...                */
                                        /* ... MUST be >= 2                                            */

#define OS_LOWEST_PRIO          15L    /* Defines the lowest priority that can be assigned ...        */
                                        /* ... MUST NEVER be higher than 63!                           */

#define OS_TASK_IDLE_STK_SIZE  100L    /* Idle task stack size (# of OS_STK wide entries)             */

#define OS_TASK_STAT_EN          0     /* Enable (1) or Disable(0) the statistics task                */
#define OS_TASK_STAT_STK_SIZE  100L    /* Statistics task stack size (# of OS_STK wide entries)       */

#define OS_ARG_CHK_EN            1     /* Enable (1) or Disable (0) argument checking                 */
#define OS_CPU_HOOKS_EN          1     /* uC/OS-II hooks are found in the processor port files        */


                                        /* ---------------------- EVENT FLAGS ---------------------- */
#define OS_FLAG_EN               0     /* Enable (1) or Disable (0) code generation for EVENT FLAGS   */
#define OS_FLAG_WAIT_CLR_EN      1     /* Include code for Wait on Clear EVENT FLAGS                  */
#define OS_FLAG_ACCEPT_EN        1     /*     Include code for OSFlagAccept()                         */
#define OS_FLAG_DEL_EN           1     /*     Include code for OSFlagDel()                            */
#define OS_FLAG_QUERY_EN         1     /*     Include code for OSFlagQuery()                          */


                                        /* -------------------- MESSAGE MAILBOXES -------------------- */
#define OS_MBOX_EN               0           /* Enable (1) or Disable (0) code generation for MAILBOXES
#define OS_MBOX_ACCEPT_EN        1           /*     Include code for OSMboxAccept()
#define OS_MBOX_DEL_EN           1           /*     Include code for OSMboxDel()
#define OS_MBOX_POST_EN          1           /*     Include code for OSMboxPost()
#define OS_MBOX_POST_OPT_EN      1           /*     Include code for OSMboxPostOpt()
#define OS_MBOX_QUERY_EN         1           /*     Include code for OSMboxQuery()
```

TASK CONTROL BLOCK
TCB

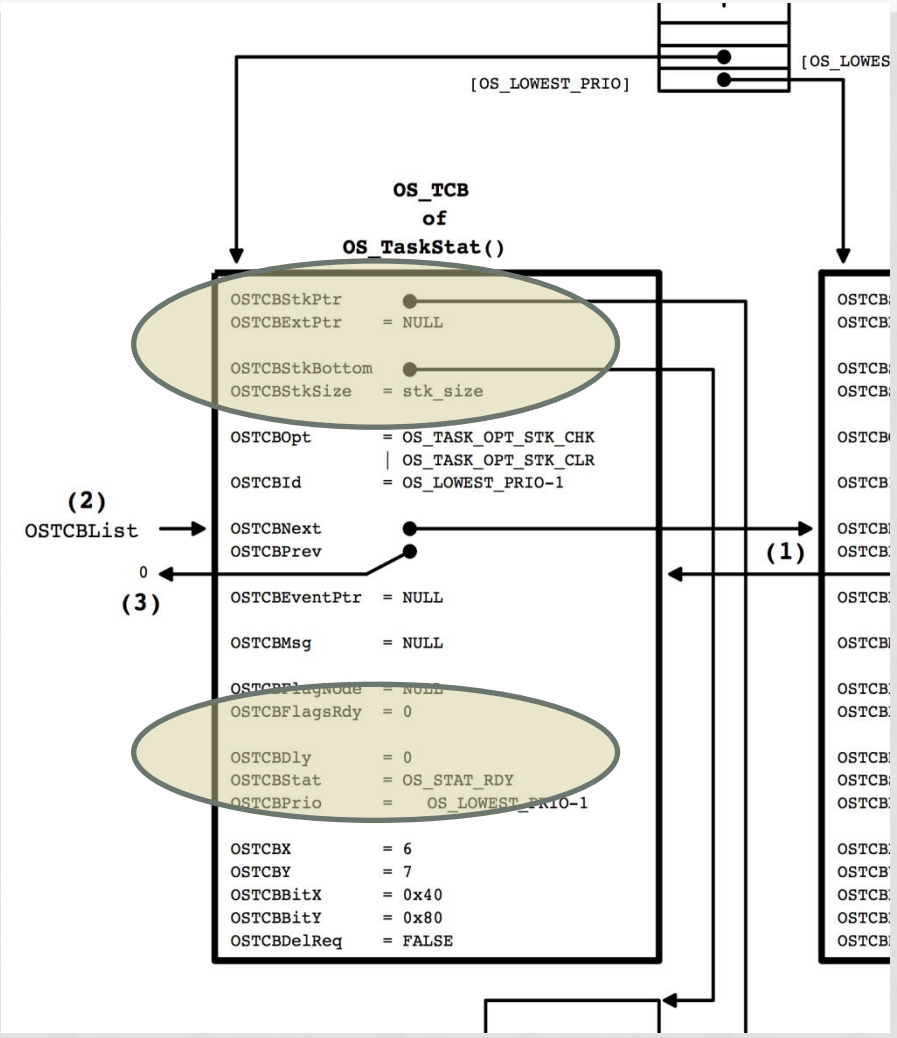
- Cada Tarea a ser usada requiere una estructura de control.
  - Estado
  - Timeout
  - Tamaño Stack
    - Punteros STACK
  - Prioridad



TCB – TASK CONTROL BLOCK

TASK CONTROL BLOCK TCB

- Cada Tarea a ser usada requiere una estructura de control.
  - Estado
  - Timeout
  - Tamaño Stack
    - Punteros STACK
  - Prioridad



BLOQUE DE CONTROL TCB

- A cada tarea creada se le asigna un TCB en el cual se almacenan diversos datos.
- **OSTCBStkPtr**: puntero al inicio del stack

MODO Extendido

- **OSTCBExtPtr**: puntero al inicio del stack en modo extendido
- **OSTCBStkBottom**: puntero al fin del stack
- Permite conocer el tamaño del stack usado en tiempo de ejecucion.
- **OSTCBStkSize**: define el tamaño del stack en bytes.
- **OSTCBOpt**: define si vamos a borrar, chequear, o usar punto flotante en el stack
- **OSTCBId**: para uso futuro

```
typedef struct os_tcb {
    OS_STK          *OSTCBStkPtr;

#if OS_TASK_CREATE_EXT_EN
    void            *OSTCBExtPtr;
    OS_STK          *OSTCBStkBottom;
    INT32U           OSTCBStkSize;
    INT16U           OSTCBOpt;
    INT16U           OSTCBId;
#endif

    struct os_tcb *OSTCBNext;
    struct os_tcb *OSTCBPrev;

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
    OS_EVENT        *OSTCBEventPtr;
#endif

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void            *OSTCBMsg;
#endif

    INT16U           OSTCBDly;
    INT8U            OSTCBStat;
    INT8U            OSTCBPrio;

    INT8U            OSTCBX;
    INT8U            OSTCBY;
    INT8U            OSTCBBitX;
    INT8U            OSTCBBitY;

#if OS_TASK_DEL_EN
    BOOLEAN          OSTCBDelReq;
#endif
} OS_TCB;
```

BLOQUE DE CONTROL TCB

- **OSTCBNext** y **OCTCBPrev**: Permiten generar una cadena de TCB, en la que se incluyen o se sacan tareas que necesitan administrar tiempo de alguna manera.
- **OSTCBEventPtr**: un puntero que se usara en el caso de comunicación entre tareas.
- **OSTCBMsg**: puntero a un mensaje entre tareas.

```
typedef struct os_tcb {
    OS_STK        *OSTCBStkPtr;

#if OS_TASK_CREATE_EXT_EN
    void          *OSTCBExtPtr;
    OS_STK        *OSTCBStkBottom;
    INT32U         OSTCBStkSize;
    INT16U         OSTCBOpt;
    INT16U         OSTCBId;
#endif

    struct os_tcb *OSTCBNext;
    struct os_tcb *OSTCBPrev;

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
    OS_EVENT      *OSTCBEventPtr;
#endif

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void          *OSTCBMsg;
#endif

    INT16U         OSTCBDly;
    INT8U          OSTCBStat;
    INT8U          OSTCBPrio;

    INT8U          OSTCBX;
    INT8U          OSTCBY;
    INT8U          OSTCBBitX;
    INT8U          OSTCBBitY;

#if OS_TASK_DEL_EN
    BOOLEAN        OSTCBDelReq;
#endif
} OS_TCB;
```

BLOQUE DE CONTROL TCB

- **OSTCBDly**: se usa para administrar tiempo. Si tiene un valor distinto de cero la tarea esta esperando por alguno de los eventos que la llevaron al estado WAITING.
- **OSTCBStat**: define el estado de la tarea. Si el valor es 0 , la tarea esta lista para ser ejecutada.
- Puede tomar otros valores:

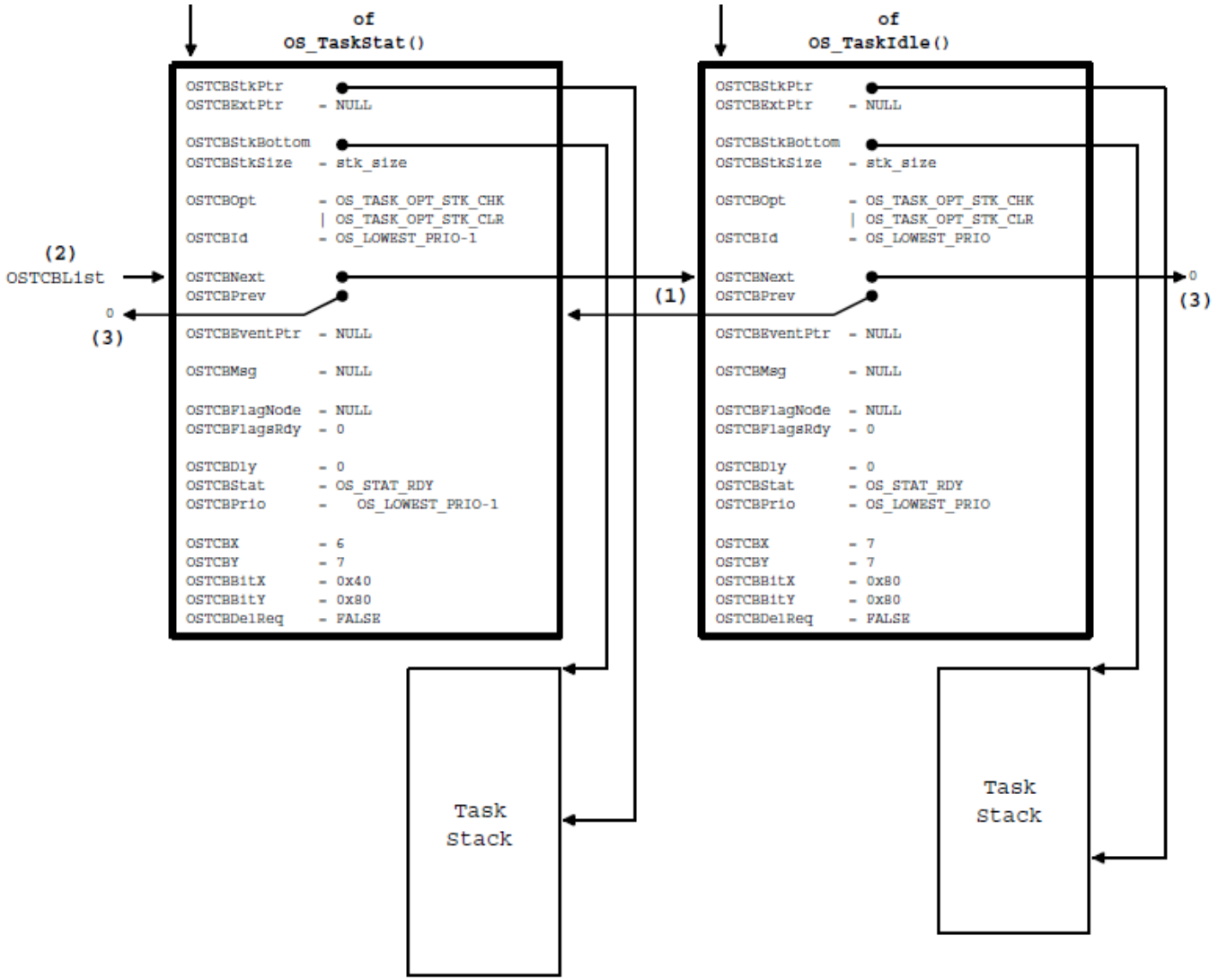
```
#define  OS_STAT_RDY          0x00
             /* Ready to run                    */
#define  OS_STAT_SEM          0x01
             /* Pending on semaphore            */
#define  OS_STAT_MBOX         0x02
             /* Pending on mailbox              */
#define  OS_STAT_Q            0x04
             /* Pending on queue                */
#define  OS_STAT_SUSPEND      0x08
             /* Task is suspended               */
```

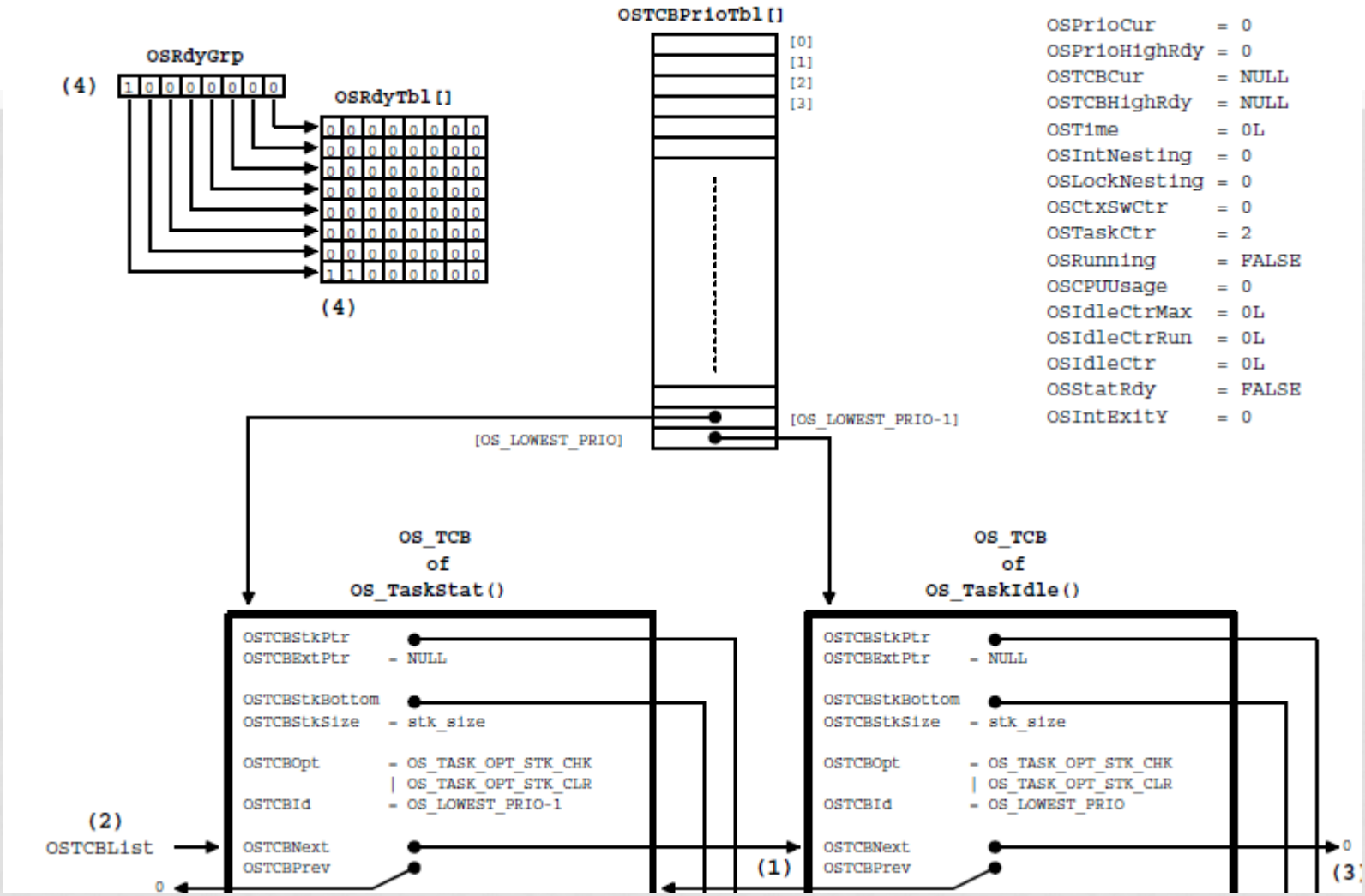
```
typedef struct os_tcb {
    OS_STK        *OSTCBStkPtr;

#if OS_TASK_CREATE_EXT_EN
    void          *OSTCBExtPtr;
    OS_STK        *OSTCBStkBottom;
    INT32U         OSTCBStkSize;
    INT16U         OSTCBOpt;
    INT16U         OSTCBId;
#endif

    struct os_tcb *OSTCBNext;
    struct os_tcb *OSTCBPrev;

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
    OS_EVENT      *OSTCBEventPtr;
#endif

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void          *OSTCBMsg;
#endif

    INT16U         OSTCBDly;
    INT8U          OSTCBStat;
    INT8U          OSTCBPrio;

    INT8U          OSTCBX;
    INT8U          OSTCBY;
    INT8U          OSTCBBitX;
    INT8U          OSTCBBitY;

#if OS_TASK_DEL_EN
    BOOLEAN        OSTCBDelReq;
#endif
} OS_TCB;
```

BLOQUE DE CONTROL TCB

- **OSTCBPrio**: define la prioridad de la tarea.
- **OSTCBX, OSTCBY, OSTCBBitX, OSTCBY**: se usan para determinar que tarea se encuentra en estado ready y es la de mayor prioridad.
- **OSTCBDelReq**: indica que la tarea debe o no ser borrada.

```
typedef struct os_tcb {
    OS_STK        *OSTCBStkPtr;

#if OS_TASK_CREATE_EXT_EN
    void          *OSTCBExtPtr;
    OS_STK        *OSTCBStkBottom;
    INT32U         OSTCBStkSize;
    INT16U         OSTCBOpt;
    INT16U         OSTCBId;
#endif

    struct os_tcb *OSTCBNext;
    struct os_tcb *OSTCBPrev;

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
    OS_EVENT      *OSTCBEventPtr;
#endif

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void          *OSTCBMsg;
#endif

    INT16U         OSTCBDly;
    INT8U          OSTCBStat;
    INT8U          OSTCBPrio;

    INT8U          OSTCBX;
    INT8U          OSTCBY;
    INT8U          OSTCBBitX;
    INT8U          OSTCBBitY;

#if OS_TASK_DEL_EN
    BOOLEAN        OSTCBDelReq;
#endif
} OS_TCB;
```


of
OS_TaskStat()
OSTCBStkPtr
OSTCBExtPtr - NULL
OSTCBStkBottom
OSTCBStkSize - stk_size
OSTCBOpt - OS_TASK_OPT_STK_CHK
| OS_TASK_OPT_STK_CLR
OSTCBId - OS_LOWEST_PRIO-1
(2)
OSTCBList
OSTCBNext
OSTCBPrev
0
(3)
OSTCBEventPtr - NULL
OSTCBMsg - NULL
OSTCBFlagNode - NULL
OSTCBFlagsRdy - 0
OSTCBDly - 0
OSTCBStat - OS_STAT_RDY
OSTCBPrio - OS_LOWEST_PRIO-1
OSTCBX - 6
OSTCBY - 7
OSTCBBitX - 0x40
OSTCBBitY - 0x80
OSTCBDelReq - FALSE
Task
Stack
of
OS_TaskIdle()
OSTCBStkPtr
OSTCBExtPtr - NULL
OSTCBStkBottom
OSTCBStkSize - stk_size
OSTCBOpt - OS_TASK_OPT_STK_CHK
| OS_TASK_OPT_STK_CLR
OSTCBId - OS_LOWEST_PRIO
(1)
OSTCBNext
OSTCBPrev
0
(3)
OSTCBEventPtr - NULL
OSTCBMsg - NULL
OSTCBFlagNode - NULL
OSTCBFlagsRdy - 0
OSTCBDly - 0
OSTCBStat - OS_STAT_RDY
OSTCBPrio - OS_LOWEST_PRIO
OSTCBX - 7
OSTCBY - 7
OSTCBBitX - 0x80
OSTCBBitY - 0x80
OSTCBDelReq - FALSE
Task
Stack


OSRdyGrp
(4)
OSRdyTbl[]
(4)
OSTCBPrioTbl[]
[0]
[1]
[2]
[3]
[OS_LOWEST_PRIO]
[OS_LOWEST_PRIO-1]
OSPrioCur = 0
OSPrioHighRdy = 0
OSTCBCur = NULL
OSTCBHighRdy = NULL
OSTime = 0L
OSIntNesting = 0
OSLockNesting = 0
OSCtxSwCtr = 0
OSTaskCtr = 2
OSRunning = FALSE
OSCPUUsage = 0
OSIdleCtrMax = 0L
OSIdleCtrRun = 0L
OSIdleCtr = 0L
OSStatRdy = FALSE
OSIntExitY = 0
OS_TCB of OS_TaskStat()
OSTCBStkPtr
OSTCBExtPtr = NULL
OSTCBStkBottom
OSTCBStkSize = stk_size
OSTCBOpt = OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR
OSTCBId = OS_LOWEST_PRIO-1
(2)
OSTCBList
OSTCBNext
OSTCBPrev
0
OS_TCB of OS_TaskIdle()
OSTCBStkPtr
OSTCBExtPtr = NULL
OSTCBStkBottom
OSTCBStkSize = stk_size
OSTCBOpt = OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR
OSTCBId = OS_LOWEST_PRIO
(1)
OSTCBNext
OSTCBPrev
0
(3)

BLOQUE DE CONTROL TCB

- Cuando se inicializa el sistema se crean la cantidad de TCB que se definen para el proyecto

```
#define OS_MAX_TASKS            11
/* Max. number of tasks in your application     */
/* ... MUST be >= 2                             */
```

- Un TCB se usa para la IDLE task.
- Un TCB se usa para la task de estadística
- Es importante definir correctamente la cantidad de tareas a utilizar para no sobrecargar la RAM.
- Se van asignando los TCB a medida que se crean las tareas.

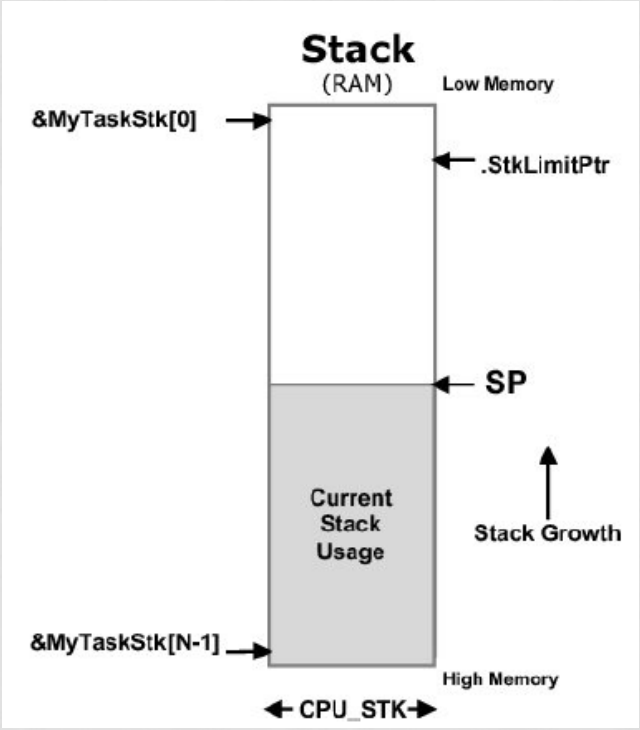
STACK - DEFINICIÓN

DEFINICIÓN TAMAÑO STACK

- Cada Tarea requiere un STACK que ocupará RAM
- Se almacenan los registros necesarios para realizar lo que se denomina Context Switch.
- Para evitar que exista overflow existen diversos mecanismos.

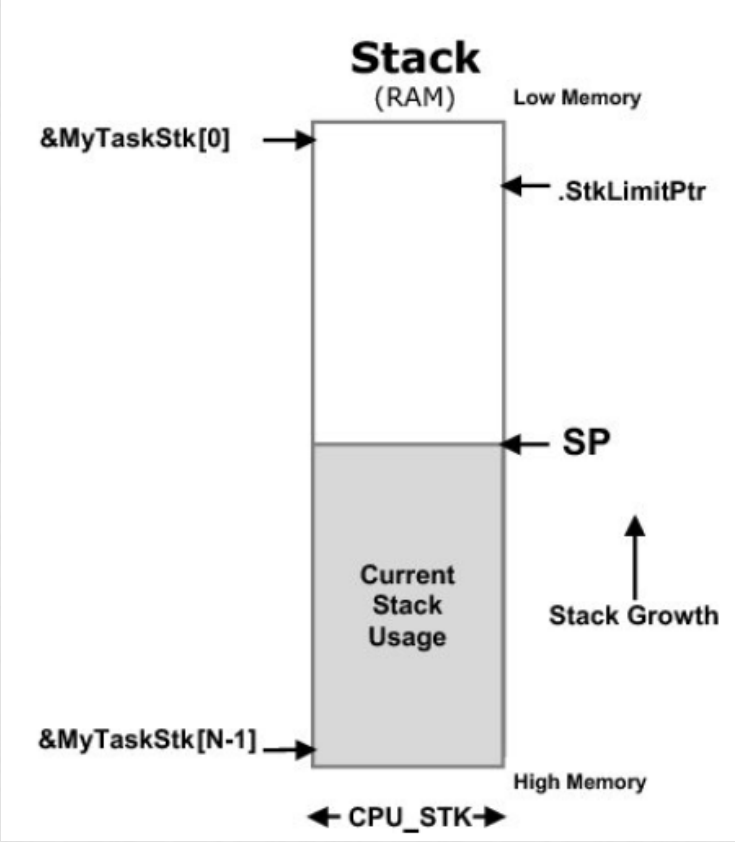
MECANISMOS PROTECCIÓN STACK

- Usar MMU o MPU. Sistemas de Hardware integrado para chequear el uso de stack.
- Usar detección por medio de registros específicos. En uCOS el valor .StkLimitPtr almacenado en el TCB permite detener el proceso que intenta escribir por fuera de ese rango. El valor asociado puede estar cerca de MyTaskStk[0].



MECANISMOS PROTECCIÓN STACK

- Usar funciones diseñadas a medida para simular la funcionalidad anterior. Se pueden usar las funciones hook(); Para este caso en particular la OSTaskSwHook().
- El valor de detección debe estar lo suficientemente alejado de MyTaskStk[0].



DEFINICIÓN TAMAÑO STACK

- Usar Zona Roja de detección.
- Se escribe en la zona roja caracteres específicos, el Kernel luego verifica si fueron sobrescritos.
- Se verifica que el puntero a stack no supere los limites establecidos para la tarea.

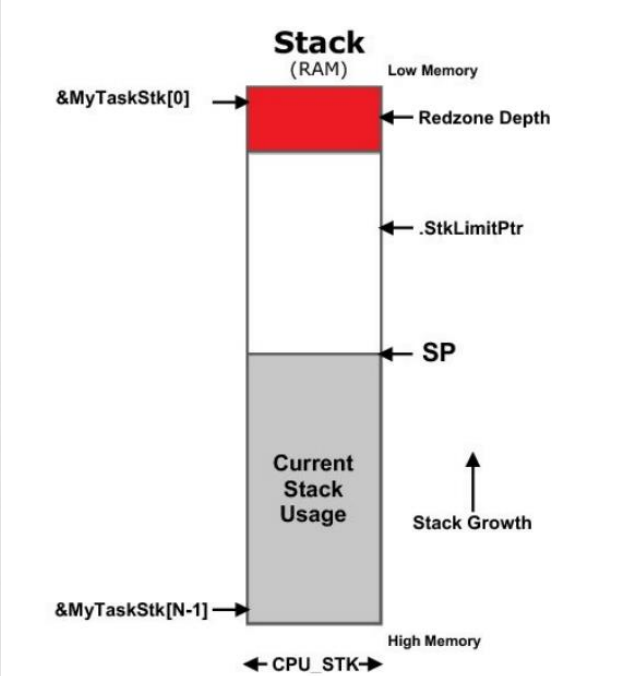
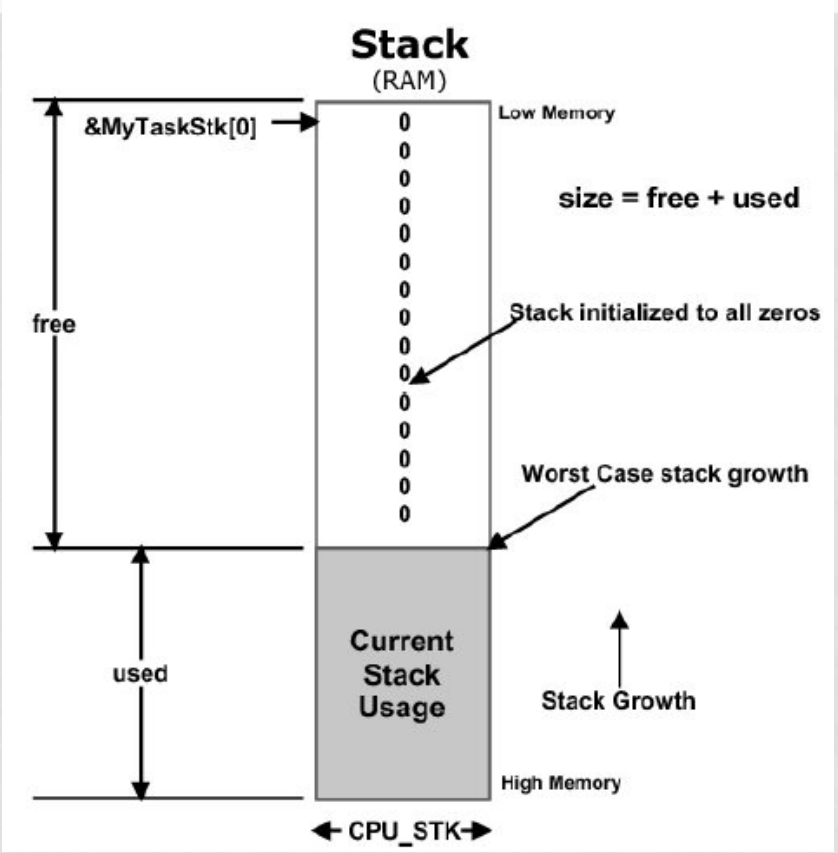


Figure - Redzone Stack Overflow detection

DEFINICIÓN TAMAÑO STACK

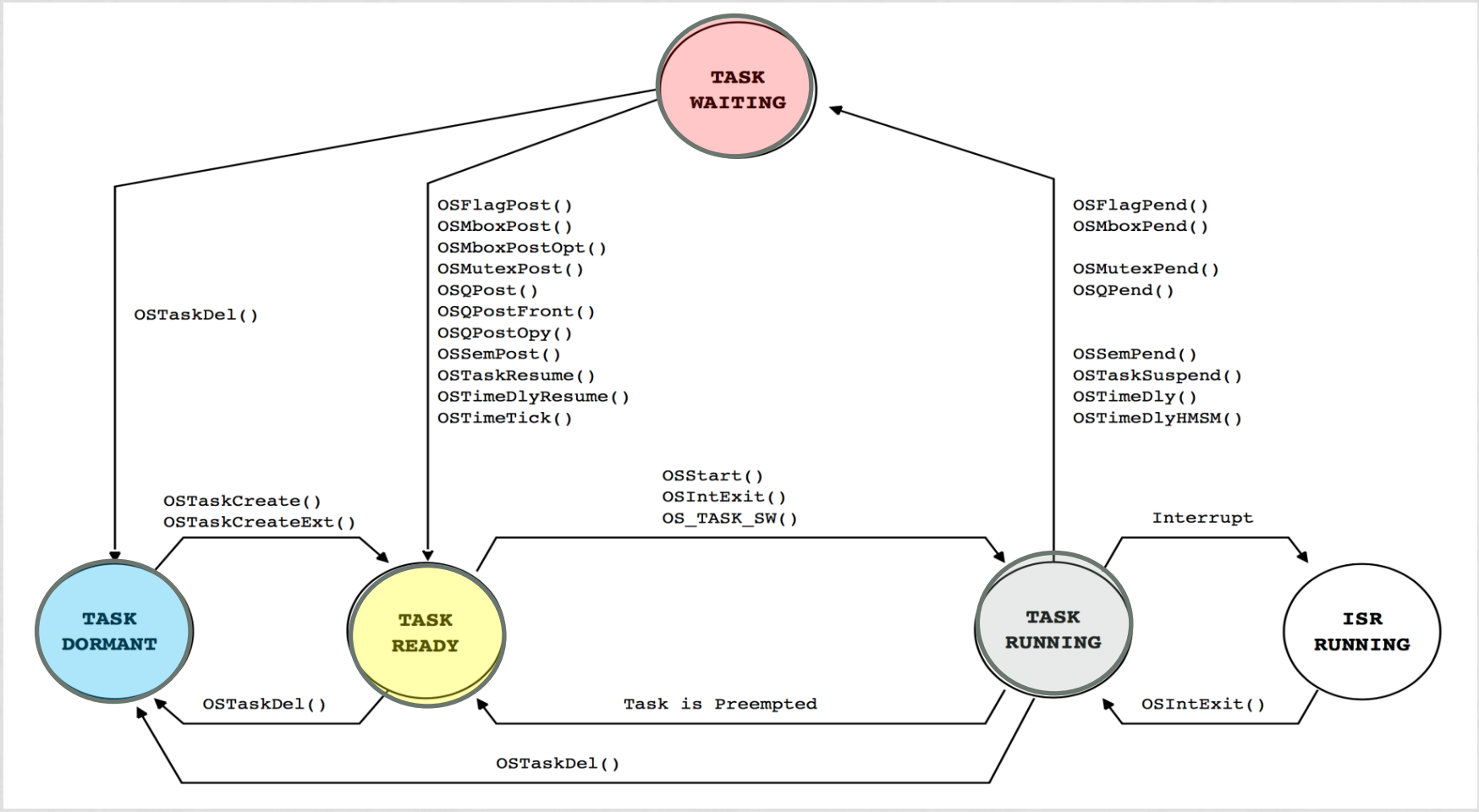
- Determinar el espacio libre de stack.
- Recorrer los stack contando los ceros iniciales.
- Definir con un valor mas elevado al necesario.
- Ajustar en diversos regrabaciones del codigo.



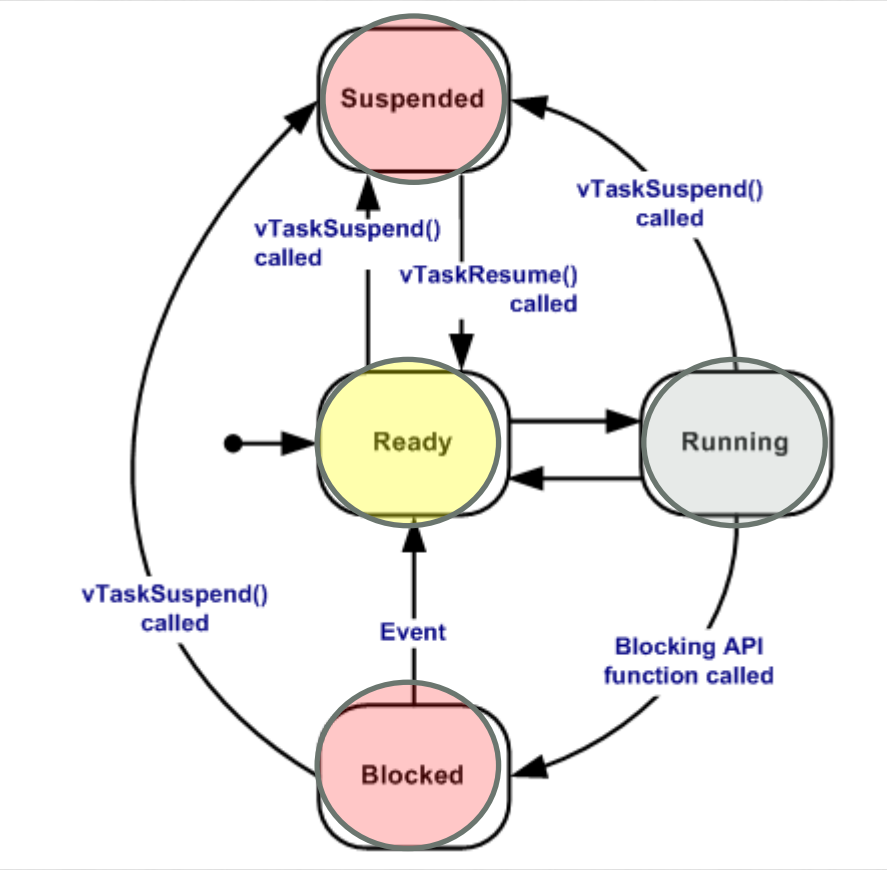
ESTADOS DEFINIDOS RTOS

- Es deseable conocer como RTOS administra el uso del CPU por parte de los distintas Tareas.
- El uso de los Servicios del RTOS determinará en que estado estará una Tarea.
  - Ready → Espera para usar el CPU
  - Run → Usa el CPU
  - Wait – Block - Suspend → Espera un timeout o la llegada de un evento.

ESTADOS DEFINIDOS RTOS



ESTADOS DEFINIDOS RTOS



ESTADOS DEFINIDOS RTOS

- DORMANT: la tarea existe en ROM, RAM, pero no esta disponible para el Kernel.

- READY: cuando se invoca a las funciones OSTaskCreate(..) u OSTaskCreateExt(…), la tarea pasa a estar entre las disponibles para tomar el control del CPU. Las tareas pueden ser creadas antes de que se comience con la multitarea, o dinamicamente por una tarea que se encuentra corriendo.

ESTADOS DEFINIDOS RTOS

- RUNNING: solo una tarea puede estar en este estado. El kernel se encarga de que siempre la tarea de mayor prioridad se encuentre en este estado.

- WAITING: cuando una tarea debe esperar un tiempo determinado para efectuar una operación, esta puede ser enviada a este estado llamando a las funciones OSTimeDly() u OSTimeDlyHMSM(). Si la tarea debe sincronizase con otra, es posible utilizar OSSemPend(), u OSMboxPend(), OSQPend().

ESTADOS DEFINIDOS RTOS

- ISR: en este estado una interrupción externa puede detener la ejecución de la tarea que se encuentra en RUNNING. Luego que se retorna del estado ISR, el kernel determina que tarea debe ser ejecutada.

- Cuando todas las tareas están en estado WAITING, se ejecuta una tarea IDLE propia del sistema.

PRIORIDADES

- Se deben asignar prioridades a cada TAREA.
  - Estáticas
  - Dinámicas
  - Se pueden invertir
  - Mayor prioridad → Según Kernel (0 → uCOS)
  - Menor prioridad → 64
  - Existen prioridades reservadas para el sistema
  - Pueden Tener el mismo VALOR

PRIORIDADES

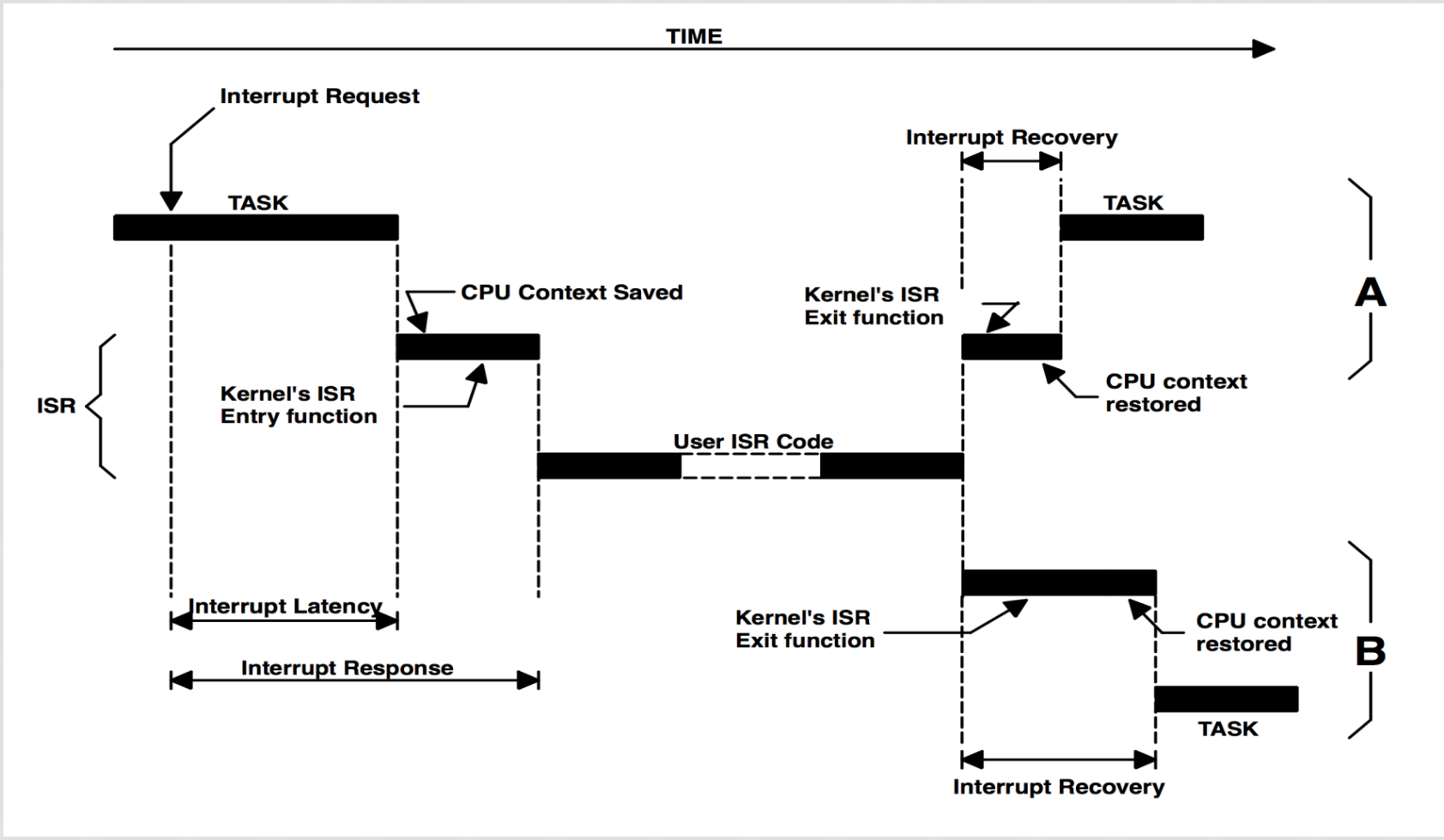
- El proceso de asignación de prioridades es el aquel en el cuál se asocia a cada tarea un número que definirá la celeridad en la atención a los estímulos que controle.

- Una tarea de ALTA prioridad tendrá una Latencia pequeña en contraste con una tarea de BAJA prioridad, la que tendrá mayor latencia.

CAMBIO DE CONTEXTO

- Es el proceso en el cual cambia la tarea que usará el CPU.
- El encargado de dicha administración es el Scheduler.
  - Depende del modo de trabajo.
  - Las prioridades de las tareas en estado Ready.
  - Resultan del Timer Tick.
  - Resultan del uso de un Servicio.

CAMBIO DE CONTEXTO



PREEMPTIVE CONTEXT SWITCH

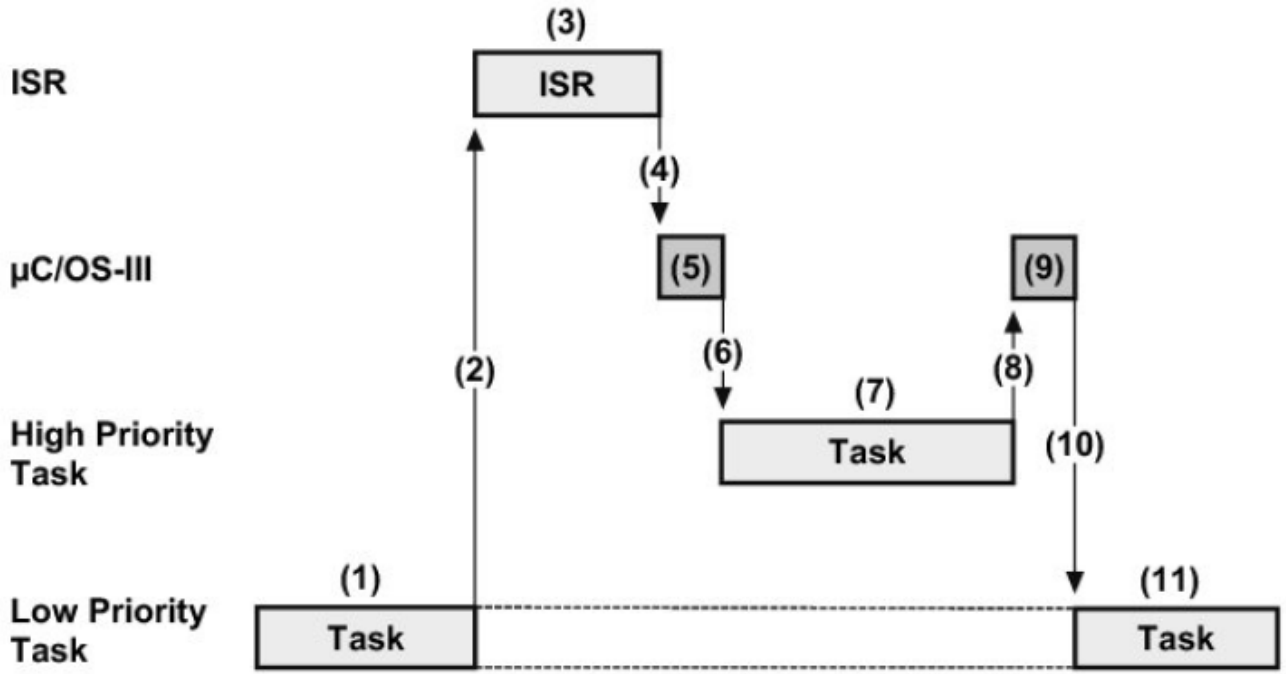


Figure - Preemptive scheduling

ROUND ROBIN CONTEXT SWITCH

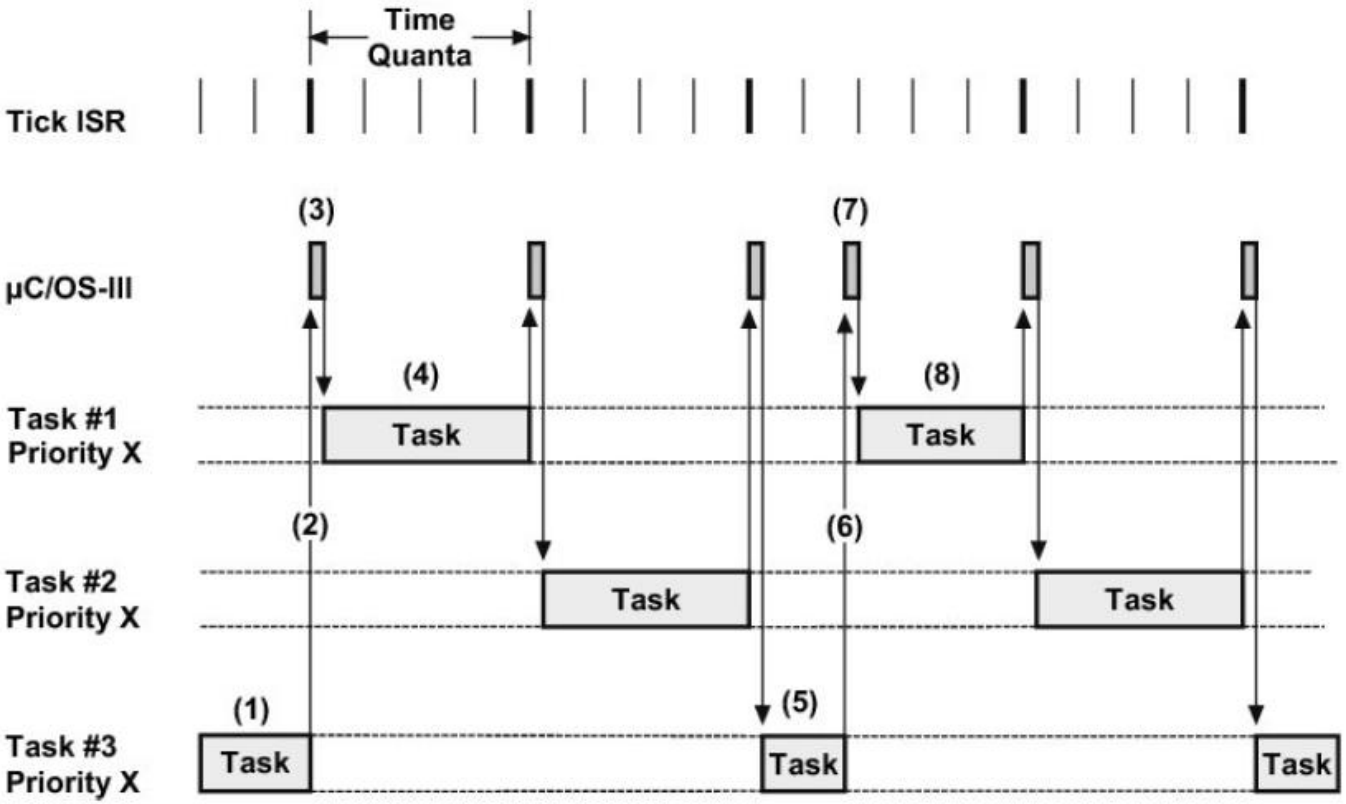
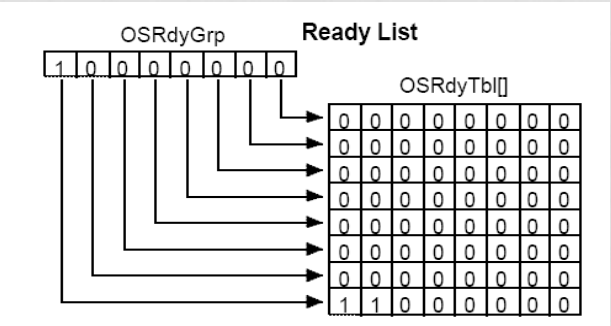
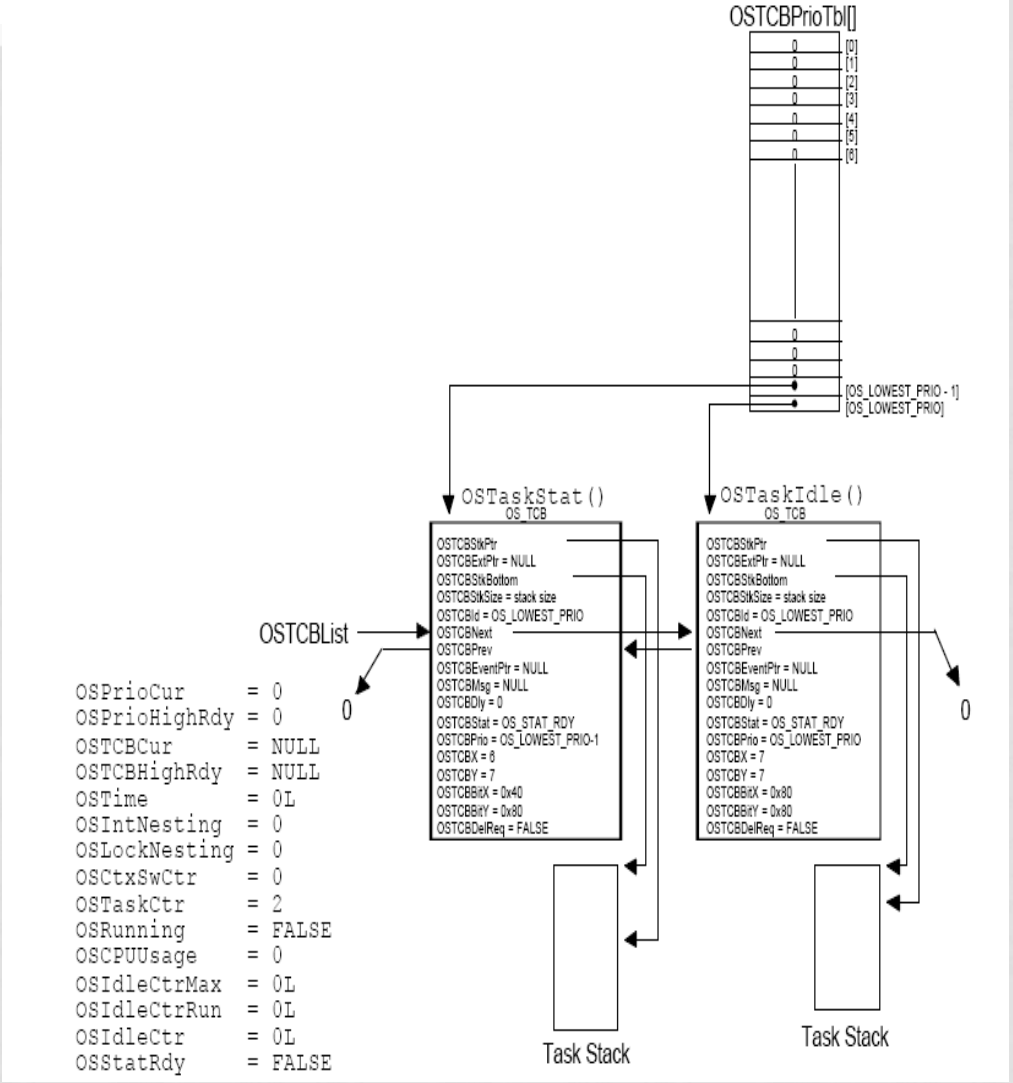


Figure - Round Robin Scheduling

INICIALIZACIÓN DEL SISTEMA

- Se debe inicializar el sistema con la función OSInit() antes de comenzar con la multitarea
- Esta función crea los TCB, y las tareas internas del kernel, (IDLE y la Estadística).
- Estas tareas creadas están listas para ser ejecutadas.
- Se enlazan los TCB





INICIALIZACIÓN DEL SISTEMA

- Tareas Internas según configuración del proyecto.
  - IDLE
  - Estadísticas
  - Ticks
  - Timers

$$CPU\_Utilization_{\%} = \left(100 - \frac{100 \times OSStatTaskCtr}{OSStatTaskCtrMax}\right)$$

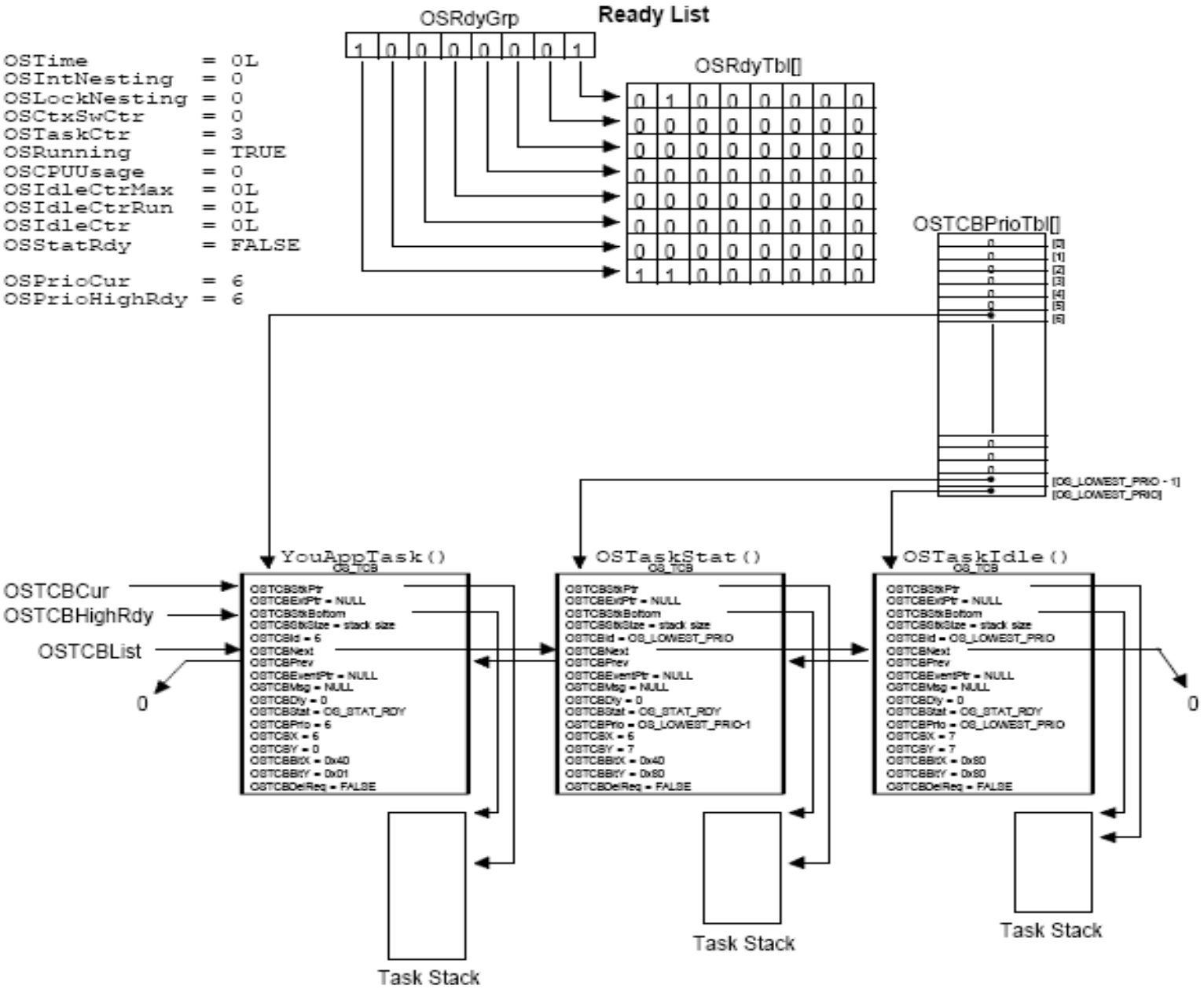
For example, if when redoing the test, `OSStatTaskCtr` reaches 7,500,000 the CPU is busy 25% of its time running application tasks:

$$25\% = \left(100 - \frac{100x7,500,000}{10,000,000}\right)$$

INICIALIZACIÓN DEL SISTEMA

- Antes de lanzar la multitarea se debe crear la primer tarea de nuestro sistema

```
void main (void)
{
   OSInit();                                  // Initialize uC/OS-II
   RandomSem = OSSemCreate(1);     // Crea semaphore

   OSTaskCreate(TaskStart, (void *)0, (void
   *)&TaskStartStk[TASK_STK_SIZE - 1], 0);

      OSStart();                                  // Start multitasking
}
```


OSTime = 0L
OSIntNesting = 0
OSLockNesting = 0
OSCtxSwCtr = 0
OSTaskCtr = 3
OSRunning = TRUE
OSCPUUsage = 0
OSIdleCtrMax = 0L
OSIdleCtrRun = 0L
OSIdleCtr = 0L
OSStatRdy = FALSE
OSPrioCur = 6
OSPrioHighRdy = 6
OSRdyGrp
Ready List
OSRdyTbl[]
OSTCBPrioTbl[]
[OS_LOWEST_PRIO - 1]
[OS_LOWEST_PRIO]
YouAppTask()
OSTaskStat()
OSTaskIdle()
OS_TCB
OSTCBCur
OSTCBHighRdy
OSTCBList
0
OSTCBStkPtr
OSTCBExtPtr = NULL
OSTCBStkBottom
OSTCBStkSize = stack size
OSTCBId = 6
OSTCBNext
OSTCBPrev
OSTCBEventPtr = NULL
OSTCBMsg = NULL
OSTCBDly = 0
OSTCBStat = OS_STAT_RDY
OSTCBPrio = 6
OSTCBX = 6
OSTCBY = 0
OSTCBBitX = 0x40
OSTCBBitY = 0x01
OSTCBDelReq = FALSE
OSTCBId = OS_LOWEST_PRIO
OSTCBPrio = OS_LOWEST_PRIO-1
OSTCBX = 7
OSTCBBitX = 0x80
OSTCBBitY = 0x80
Task Stack

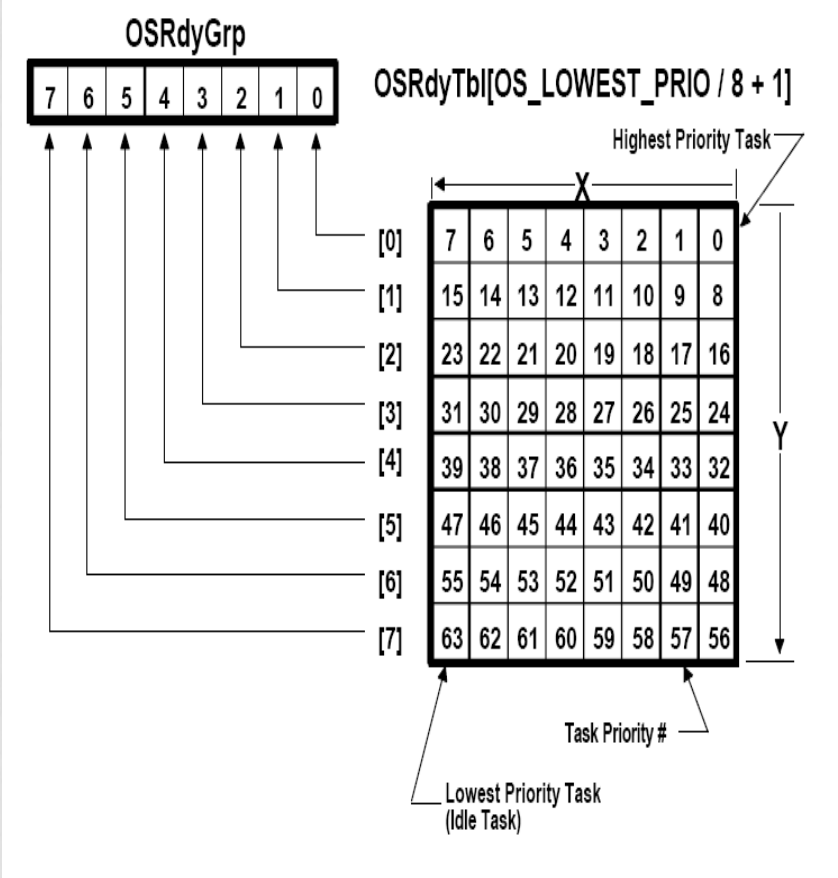
TAREA DE EJEMPLO

- Son bucles infinitos que no retornan ningún valor.
- Dentro de ellas se llaman a funciones que pueden o no retornar algún valor.

```
void Task (void *data)
{
    UBYTE x, y, err;

    for (;;)
    {
     OSSemPend(RandomSem, 0, &err);
     x = random(80);
     y = random(16);
     OSSemPost(RandomSem);
     OSTimeDly(1);
    }
}
```

TAREA PARA EJECUTAR

- Se define para cada tarea una prioridad diferente, la cual determina en que momento pasara al estado RUNNING.
- Las tareas READY se listan en dos variables: OSRdyGrp y OSRdyTbl[].
- OSRdyGrp agrupa las tareas cada 8.
- Cada tarea READY indica su estado con un bit dentro de la tabla OSrdyTbl[].

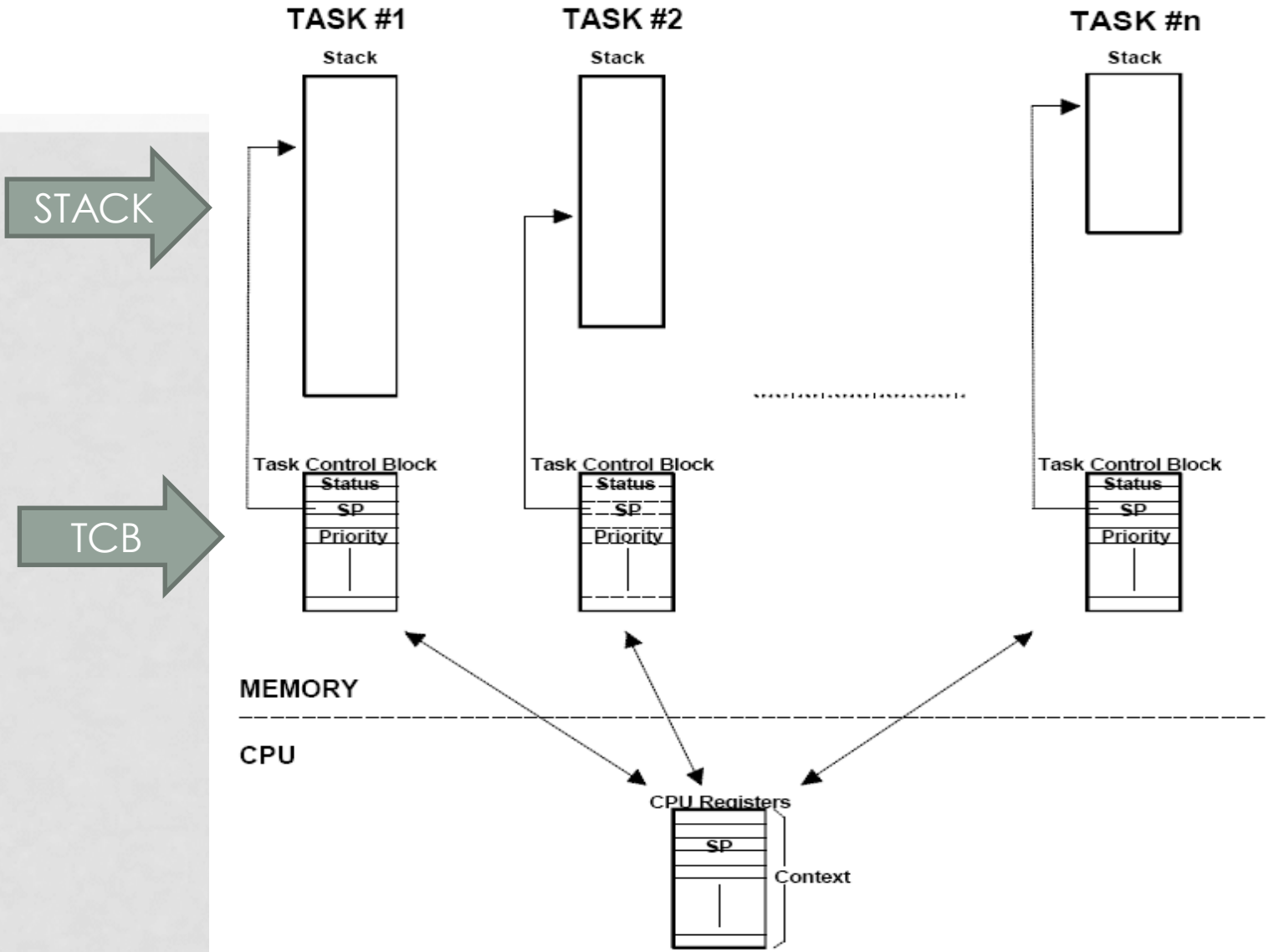


CREACIÓN DE TAREAS

- Debe estar definida en la configuración inicial la cantidad de Tareas
  - Reserva inicial de TCB.
- Se deben definir las prioridades.
- Se debe establecer el tamaño del STACK de cada Tarea.

RTOS - TAREAS

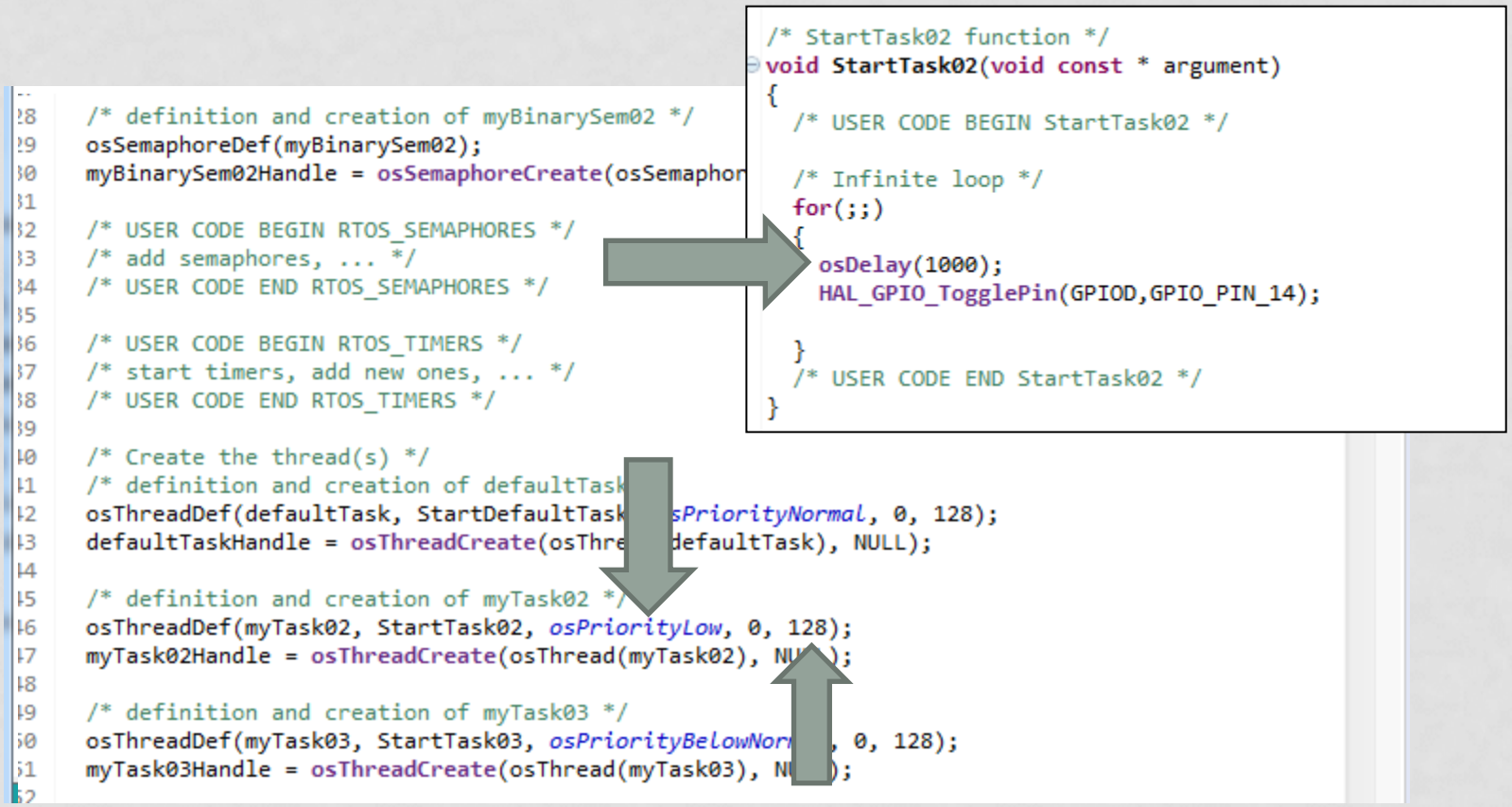
- Se llama HILO
- Se divide el trabajo en bloques, que resuelven una porción del problema.
- Cada Tarea tendrá asociado:
  - TCB(Task Control Block)
  - STACK
  - Prioridad
- Se deben conocer los estados posibles.
- Se deben conocer las funcionalidades asociadas.
  - Crear. Suspender. Reasumir. Borrar. Cambio de Prioridades


TASK #1
TASK #2
TASK #n
Stack
Stack
Stack
STACK
TCB
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
MEMORY
CPU
CPU Registers
SP
Context

CREACIÓN DE TAREAS FREERTOS

```
  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  (const portCHAR *)"Blink"   // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the
    ,  NULL );

  xTaskCreate(
    TaskAnalogRead
    ,  (const portCHAR *) "AnalogRead"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );
}
```

CREACIÓN DE TAREAS FREERTOS



```
/* definition and creation of myBinarySem02 */
osSemaphoreDef(myBinarySem02);
myBinarySem02Handle = osSemaphoreCreate(osSemaphor

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* Create the thread(s) */
/* definition and creation of defaultTask
osThreadDef(defaultTask, StartDefaultTask sPriorityNormal, 0, 128);
defaultTaskHandle = osThreadCreate(osThre defaultTask), NULL);

/* definition and creation of myTask02 */
osThreadDef(myTask02, StartTask02, osPriorityLow, 0, 128);
myTask02Handle = osThreadCreate(osThread(myTask02), NU );

/* definition and creation of myTask03 */
osThreadDef(myTask03, StartTask03, osPriorityBelowNor , 0, 128);
myTask03Handle = osThreadCreate(osThread(myTask03), N );
```

```
/* StartTask02 function */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */

  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_14);

  }
  /* USER CODE END StartTask02 */
}
```

CREACIÓN DE TAREAS

```
OSTaskCreate(Task1, (void *)0, &Task1Stk[0], task1PRIO);
OSTaskCreate(Task2, (void *)0, &Task2Stk[0], task2PRIO);
OSTaskCreate(Task3, (void *)0, &Task3Stk[0], task3PRIO);
```

```
//-----------------------------------------
void Task1(void *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

    for(;;)
    {
        salidaLed1 = 0;
        OSTimeDly(2);    // un tick --> 10mSeg
        salidaLed1 = 1;
        OSTimeDly(2);
    }
}
```

Ejemplo 1 RTOS

BORRADO DE TAREAS

- Si esta WAITING,
  - Se lleva el contador de tiempo a cero.
  - Se pone en estado READY.
  - Se llama a la función DUMMY();(acá se habilita las interrupciones y luego se vuelven a deshabilitar)
  - Se disminuye la cantidad de tareas
  - Se elimina la entrada en la tabla de prioridades
  - Se desenlaza el TCB correspondiente de la cadena de TCB's
  - Se llama al ***Scheduler***.
- Se llama a la función Dummy() para garantizar que al deshabilitar las interrupciones la ISR no tenga latente ningún proceso.
- Si la tarea a ser borrada posee algún evento compartido con otras tareas, primero debe liberar este para luego ser eliminada.

CAMBIO DE PRIORIDAD

- Se debe definir el nuevo valor de prioridad.
- No se puede cambiar la prioridad de la IDLE task.
- En el proceso de cambio de prioridad:
  - Parámetros: vieja y nueva prioridad
  - Verifico que ambos valores estén dentro del intervalo definido
  - Verifico que exista el nuevo valor de prioridad.
  - Se reserva el nuevo valor en la tabla de prioridades
  - Se modifican lo valores en el TCB relacionados con el nuevo valor de prioridad
  - Si la tarea estaba en WAITING, se remueve de la lista de tareas, y se reingresa ésta con la nueva prioridad.
- Al fin del proceso se llama al ***Scheduler***.

CAMBIO DE PRIORIDAD

```
#if OS_TASK_CHANGE_PRIO_EN
INT8U OSTaskChangePrio (INT8U oldprio, INT8U newprio)
{
    OS_TCB   *ptcb;
    OS_EVENT *pevent;
    INT8U     x;
    INT8U     y;
    INT8U     bitx;
    INT8U     bity;

    if ((oldprio >= OS_LOWEST_PRIO && oldprio != OS_PRIO_SELF)   ||
         newprio >= OS_LOWEST_PRIO) {
        return (OS_PRIO_INVALID);
    }
    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[newprio] != (OS_TCB *)0) {              /* New priority must not already exist */
        OS_EXIT_CRITICAL();
        return (OS_PRIO_EXIST);
    } else {
        OSTCBPrioTbl[newprio] = (OS_TCB *)1;                 /* Reserve the entry to prevent others */
        OS_EXIT_CRITICAL();
```

CAMBIO DE PRIORIDAD

```
OSTCBPrioTbl[newprio] = ptcb;                    /* Place pointer to TCB @ new priority */
        ptcb->OSTCBPrio      = newprio;                  /* Set new task priority               */
        ptcb->OSTCBY         = y;
        ptcb->OSTCBX         = x;
        ptcb->OSTCBBitY      = bity;
        ptcb->OSTCBBitX      = bitx;
        OS_EXIT_CRITICAL();

        OSSched();                                  /* Run highest priority task ready     */

        return (OS_NO_ERR);
    } else {
        OSTCBPrioTbl[newprio] = (OS_TCB *)0;             /* Release the reserved prio.          */
        OS_EXIT_CRITICAL();
        return (OS_PRIO_ERR);                       /* Task to change didn't exist         */
    }
  }
}
#endif
```

SUSPENSIÓN DE UNA TAREA

- Una tarea puede ser suspendida por otra o por si misma.
- No se puede suspender la IDLE task
- No se puede suspender una tarea con prioridad no definida.
- Necesita llamar al ***scheduler***
  - Si se suspende a si misma
  - Si es la de mas alta prioridad es ese momento
- Se pone en el status de la tarea en su TCB en estado SUSPEND

SUSPENSIÓN DE UNA TAREA

```
#if OS_TASK_SUSPEND_EN
INT8U OSTaskSuspend (INT8U prio)
{
    BOOLEAN   self;
    OS_TCB   *ptcb;

    if (prio == OS_IDLE_PRIO) {                          /* Not allowed to suspend idle task    */
        return (OS_TASK_SUSPEND_IDLE);
    }
    if (prio >= OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {       /* Task priority valid ?               */
        return (OS_PRIO_INVALID);
    }
    OS_ENTER_CRITICAL();
    if (prio == OS_PRIO_SELF) {                          /* See if suspend SELF                 */
        prio = OSTCBCur->OSTCBPrio;
        self = TRUE;
    } else if (prio == OSTCBCur->OSTCBPrio) {                   /* See if suspending self              */
        self = TRUE;
    } else {
        self = FALSE;                                    /* No suspending another task          */
    }
```

SUSPENSIÓN DE UNA TAREA

```
if ((ptcb = OSTCBPrioTbl[prio]) == (OS_TCB *)0) {              /* Task to suspend must exist     */
        OS_EXIT_CRITICAL();
        return (OS_TASK_SUSPEND_PRIO);
    } else {
        if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0) {     /* Make task not ready           */
            OSRdyGrp &= ~ptcb->OSTCBBitY;
        }
        ptcb->OSTCBStat |= OS_STAT_SUSPEND;                          /* Status of task is 'SUSPENDED'  */
        OS_EXIT_CRITICAL();
        if (self == TRUE) {                                  /* Context switch only if SELF    */

            OSSched();

        }
        return (OS_NO_ERR);
    }
}
#endif
```

REASUNCIÓN DE UNA TAREA

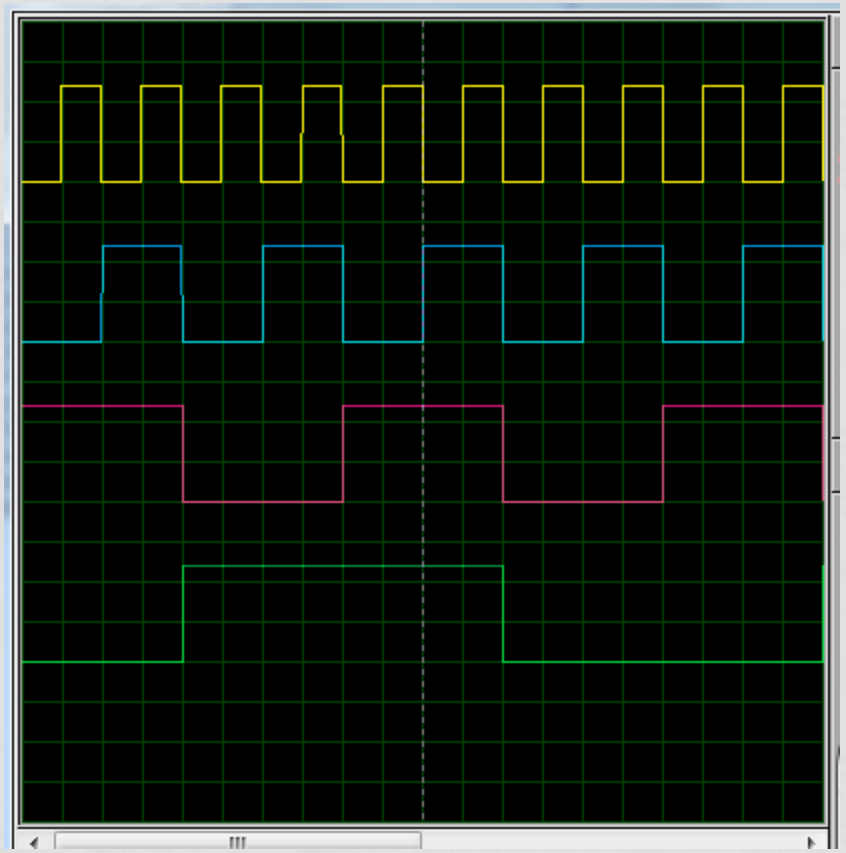
- Se vuelve del estado SUSPEND una tarea.
- Se verifica que exista la prioridad
- Se verifica que esté dentro del intervalo definido
- Se modifica el estado de la tarea en el TCB
- Se ingresa la tarea en la tabla de tareas READY si no tiene un tiempo de WAITING pendiente.
  - Se llama al ***Scheduler***.

EJEMPLO 2

- Implementar un programa que genere una secuencia en salidas digitales, garantizando la evolución temporal de cada una de ellas.
  - Las salidas deben estar ***Sincronizadas*** con la salida 1.
- Usar topología RTOS.

SECUENCIA TEMPORAL

- Posible Solución
  - Sincronización con Semáforos Binarios.

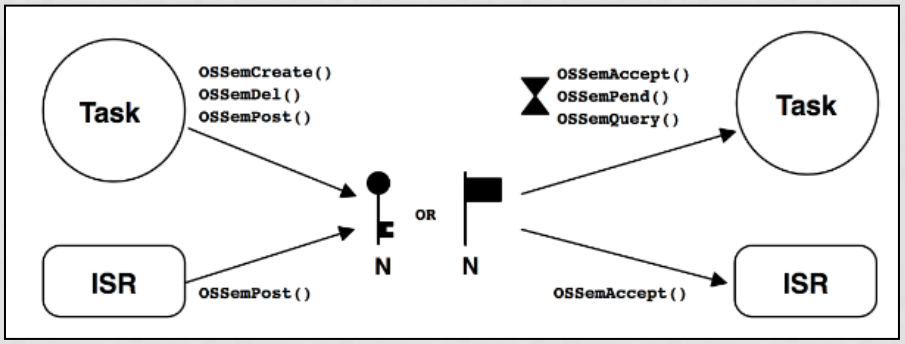


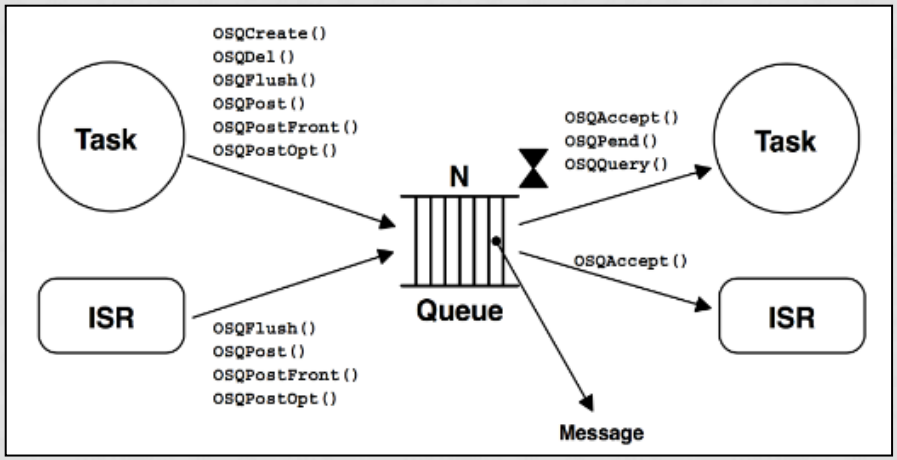
SINCRONIZACIÓN

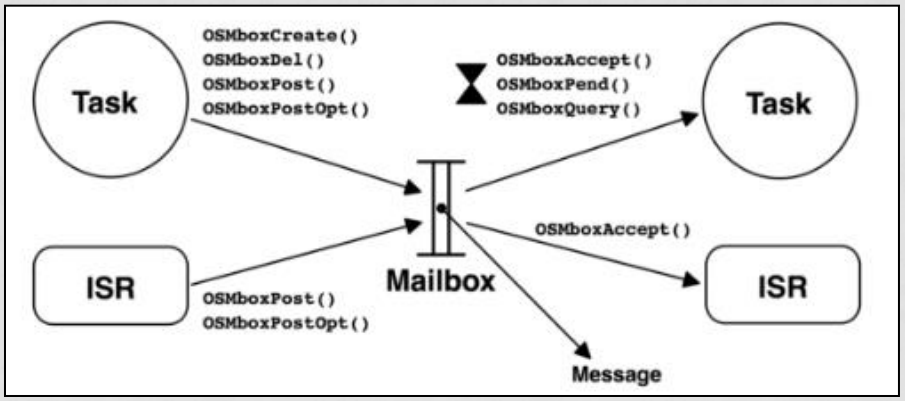
- Se utilizan entidades denominadas EVENTOS.
- Requieren estructuras de control denominadas ECB.
- Precauciones.
  - Se debe definir correctamente como será la relación entre Tareas.
    - Evitar el “deadlock”.
    - Usar timeout para detectar esta instancia.

EVENTOS

- Semáforos
- Mailbox
- Queues

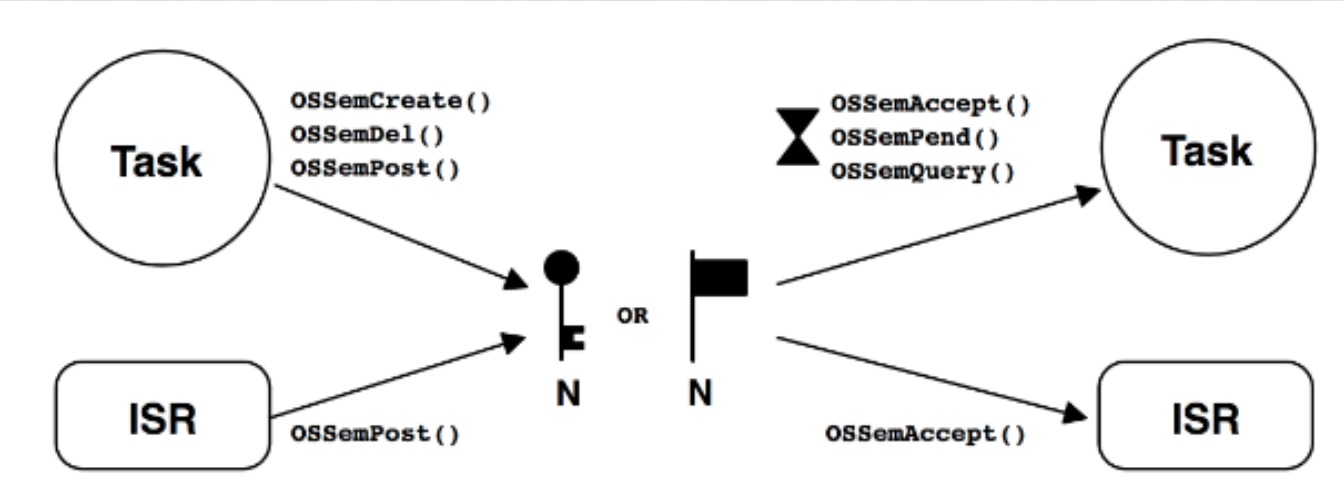






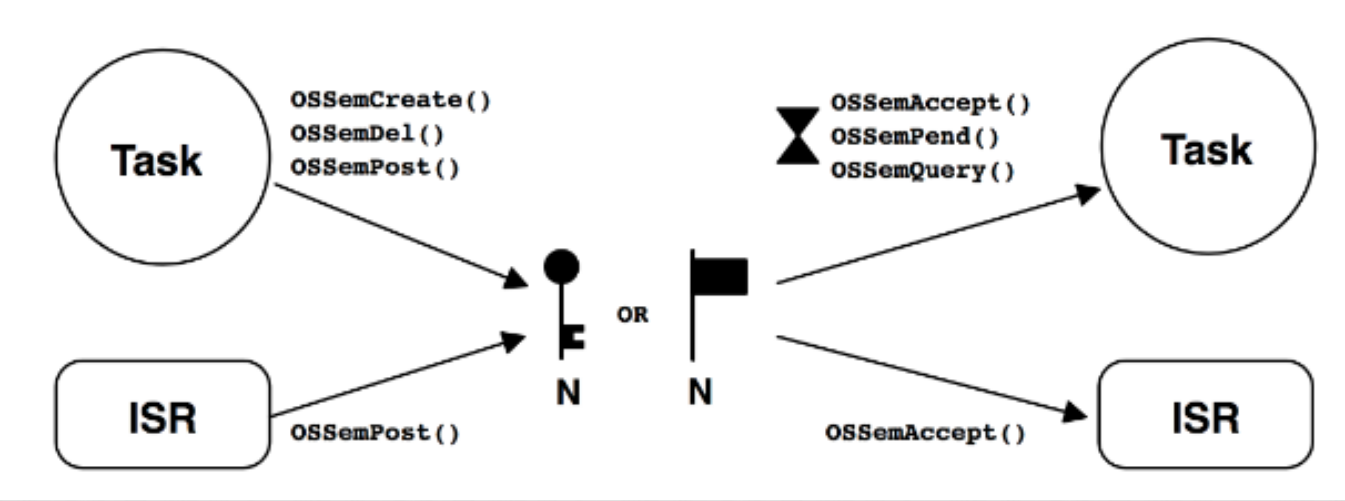
EVENTOS - SEMÁFOROS

- Semáforos Binarios
- Semáforos Contadores
- Semáforos Mutex



EVENTOS - SEMÁFOROS

- Semáforos Binarios
- Semáforos Contadores
- Semáforos Mutex
- Broadcasting



EVENTOS - SEMÁFOROS

- Es posible señalizar varias tareas con un único evento (semáforo) usando la opción OS_OPT_POST_ALL cuando se hace post.

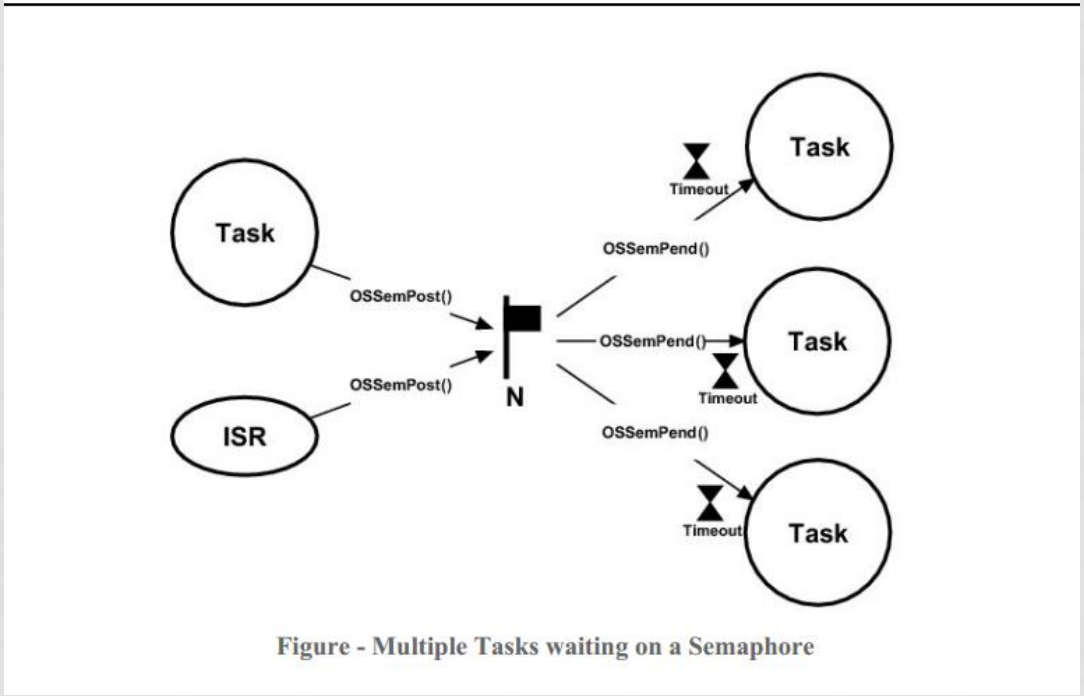
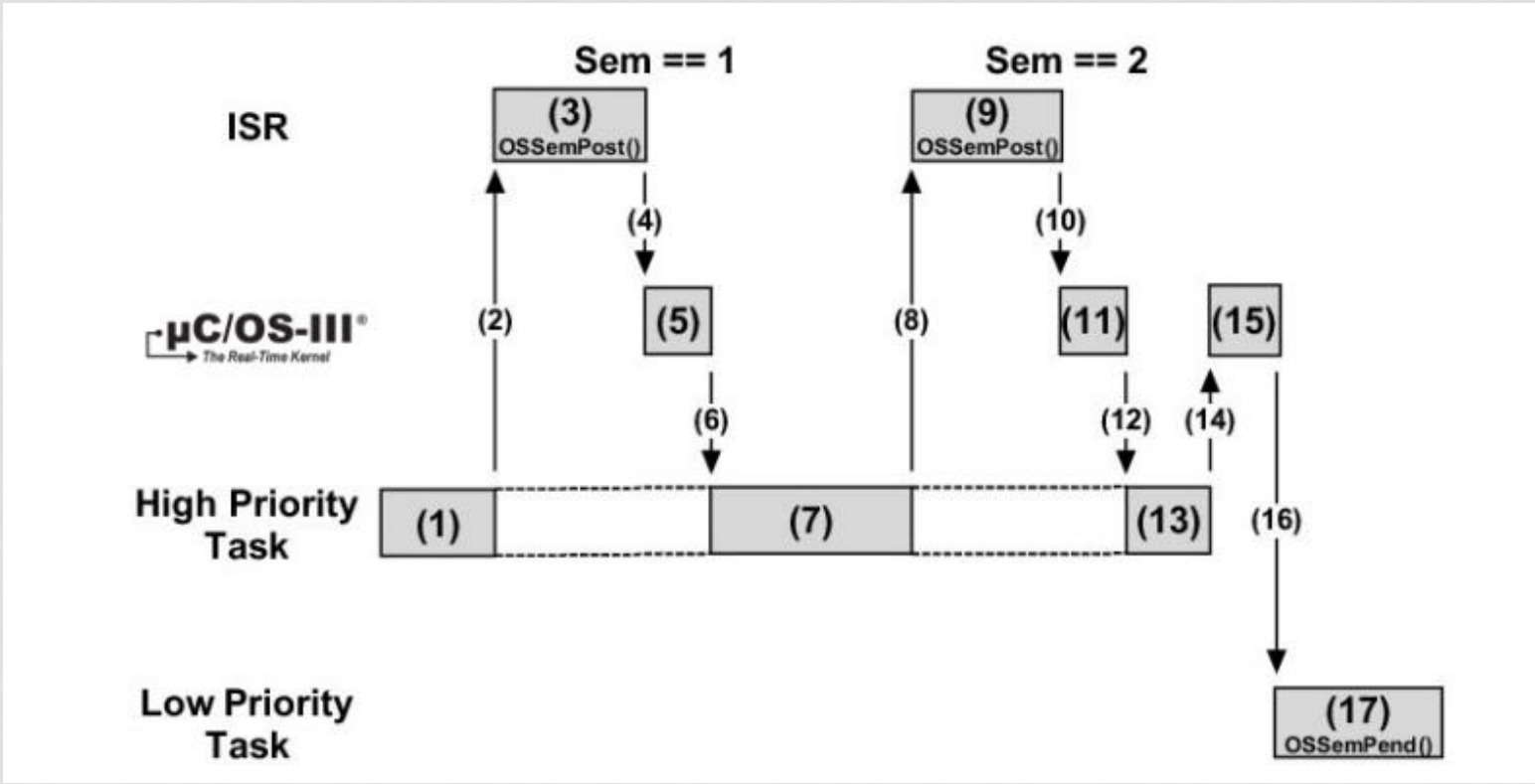


Figure - Multiple Tasks waiting on a Semaphore

EVENTOS – SEMÁFORO CONTADOR

- Semáforos Contadores



Prioridad 1

```
//-----------------------------------
void Task1(void *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

    for(;;)
    {
        salidaLed_1 = ~salidaLed_1;
        if(salidaLed_1==0)
            OSSemPost(Semaforo1);
        OSTimeDly(10);
    }
}
```

Prioridad 2

```
for(;;)
{
    OSSemPend(Semaforo1,0,&err);
    salidaLed_2 = ~salidaLed_2;
    if(salidaLed_2==0)
        OSSemPost(Semaforo2);
    OSTimeDly(10);
}
```

Prioridad 3

```
for(;;)
{
    OSSemPend(Semaforo2,0,&err);
    salidaLed_3 = ~salidaLed_3;
    if(salidaLed_3==0)
        OSSemPost(Semaforo3);
    OSTimeDly(10);
}
```

Prioridad 4

```
INT8U *err;
INT16U timeOut=0;

for(;;)

    OSSemPend(Semaforo3,timeOut,err);
    salidaLed_4 = ~salidaLed_4;
    OSTimeDly(10);
}
```

Ejemplo 2 SEM

RTOS - DEADLOCK

- SOLUCION
  - La manera más sencilla de evitar deadlock es utilizar timeout.
  - Es necesario cuando se libera el semáforo binario o el semáforo mutex, verificar el resultado de la operación.
  - OSMutexPend() devuelve el resultado.
    - OS_ERR_NONE
    - OS_ERR_MUTEX_NESTING
    - ETC

OCURRENCIA DE DEADLOCK

```
INT16U timeOut = 0;
for(;;)
{
    OSSemPend(Semaforo2,timeOut,error);
    switch(*error){
        case OS_NO_ERR:
            Nop();
            break;
        case OS_TIMEOUT:
            Nop();
            break;
    }

    salidaLed_1 = 1;
    OSSemPost(Semaforo1);
    OSTimeDly(20);
}
```

```
for(;;)
{
    OSSemPend(Semaforo1,0,err);
    salidaLed_2 = 1;
    OSSemPost(Semaforo2);
    OSTimeDly(20);
}
```

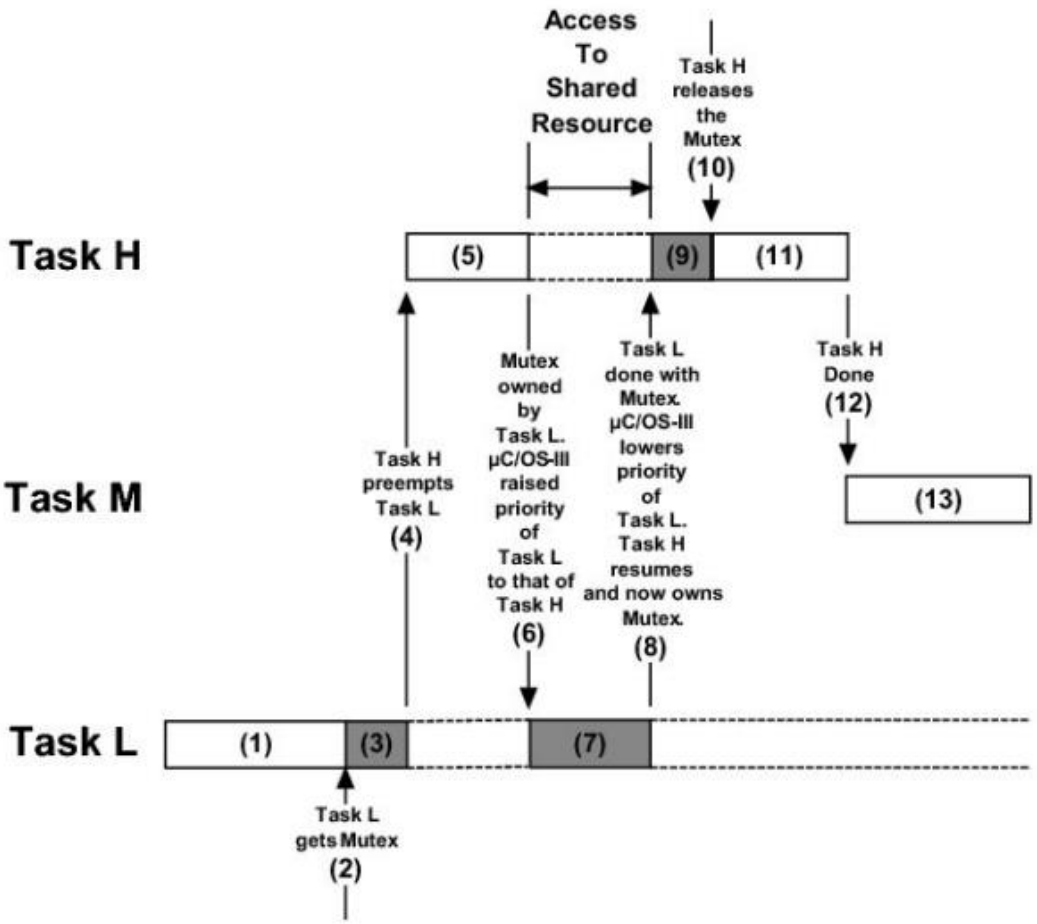
Ejemplo 3 DEAD

OCURRENCIA DE DEADLOCK

EVENTOS - MUTEX

- Cuando se requiere usar un recurso/periférico por más de una tarea.
  - Ejemplo SPI
    - ADC
    - LCD
    - MEMORIA

EVENTOS - MUTEX

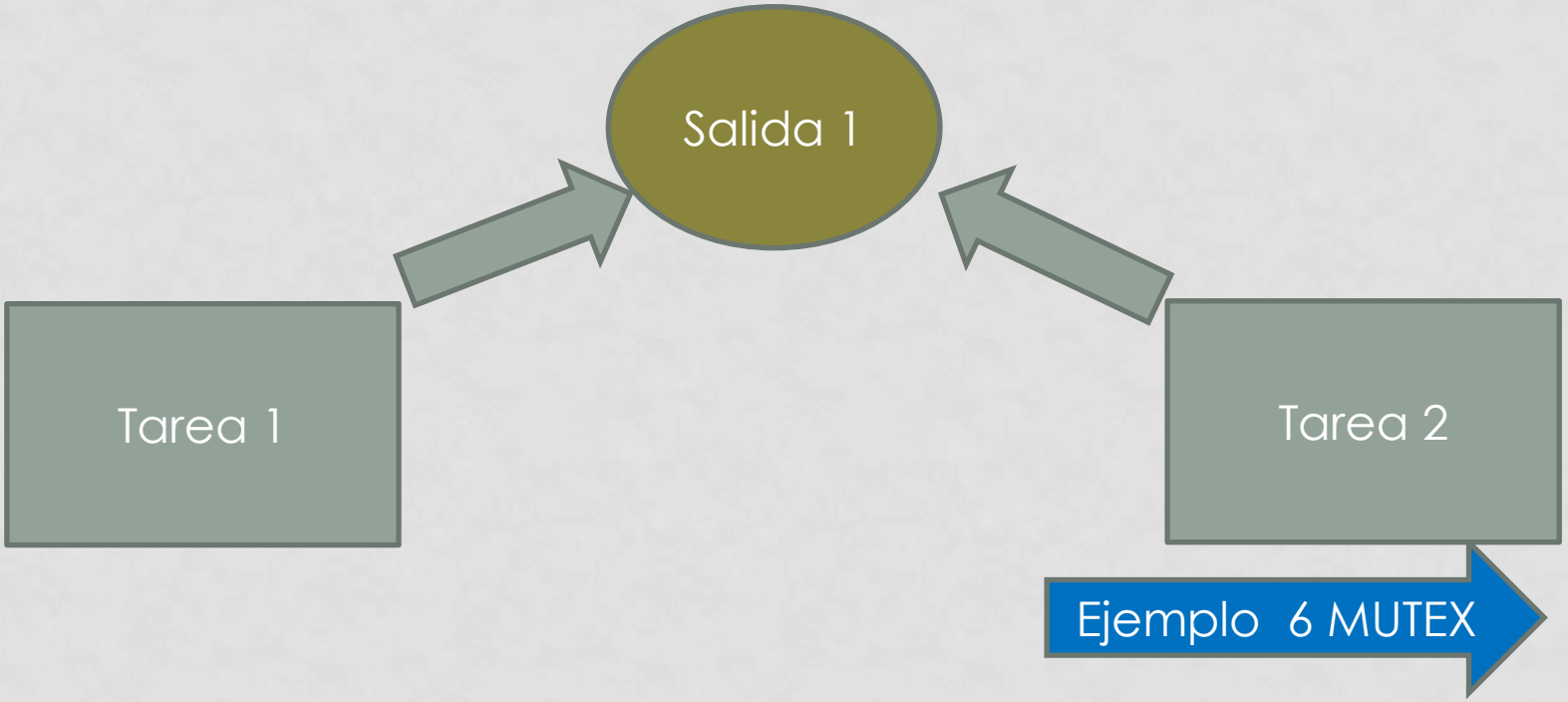


EVENTOS - MUTEX

- PRECAUCIONES de USO
  - Es la alternativa de uso más sencilla.
  - Se modifican las reglas de ejecución.
    - La tarea de mas alta prioridad es la que se ejecuta.
  - Se debe tener presente :
    - La inversión de prioridades
    - El deadlock.

EJEMPLO

- Dos tareas desean acceder a un puerto I/O.



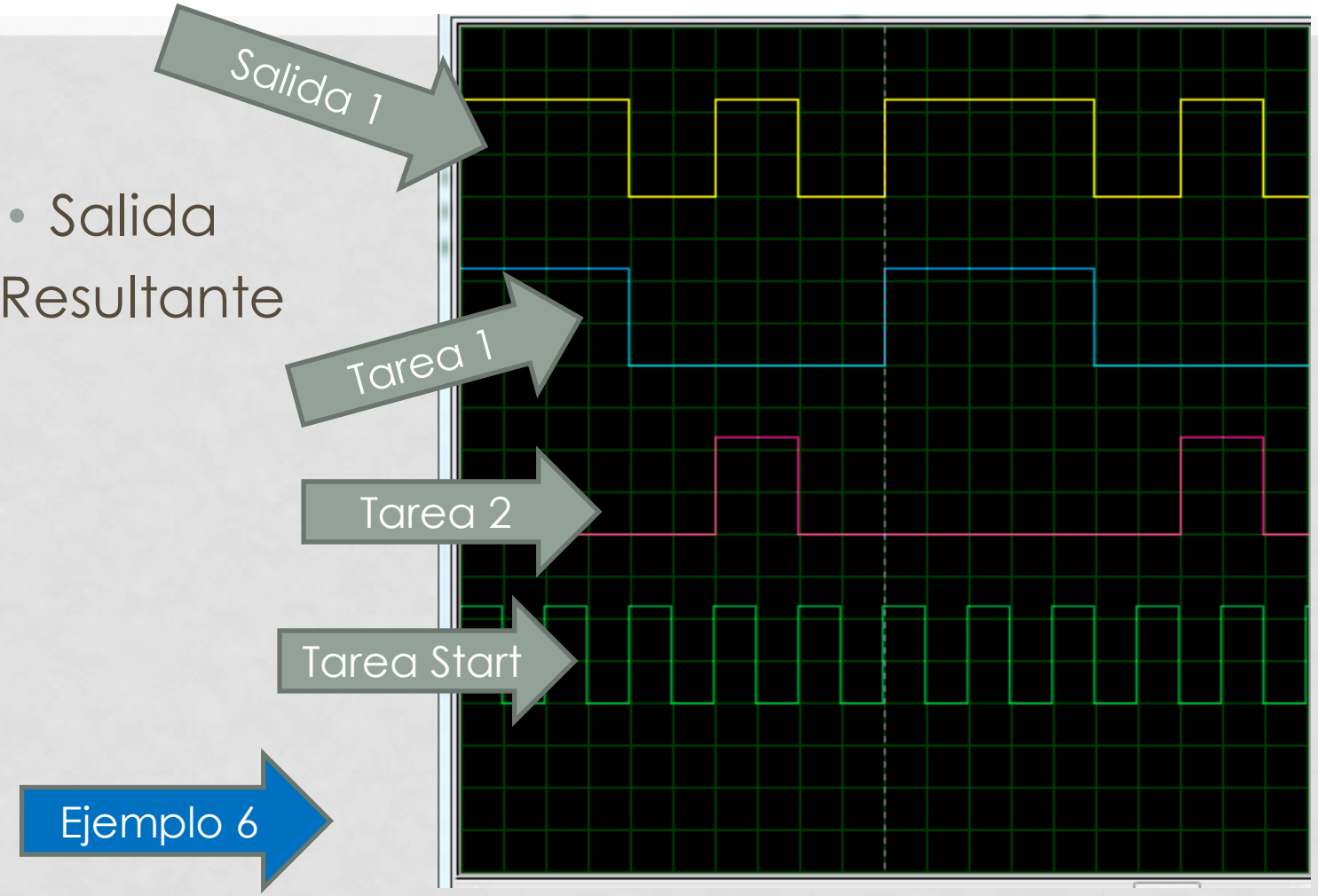
EJEMPLO

- Dos tareas desean acceder a un puerto I/O.

```
for(;;) {
    OSMutexPend(ADC_Mutex, 0, &err);

    salidaLED_1 = 1;
    salidaLED_2 = 1;
    OSTimeDly(5);

    salidaLED_1 = 0;
    salidaLED_2 = 0;
    OSTimeDly(2);

    OSMutexPost(ADC_Mutex);
}
```

```
for(;;){
    OSMutexPend(ADC_Mutex, 0, &err);

    salidaLED_1 = 1;
    salidaLED_3 = 1;
    OSTimeDly(2);

    salidaLED_1 = 0;
    salidaLED_3 = 0;
    OSTimeDly(2);

    OSMutexPost(ADC_Mutex);
}
```

- Salida Resultante



Ejemplo 6

EJEMPLO

- Dos tareas desean acceder a un puerto I/O.
- Usar Activación/ Desactivación de Interrupciones.
- Usar Activación/Desactivación del Scheduler.

SOLUCIÓN

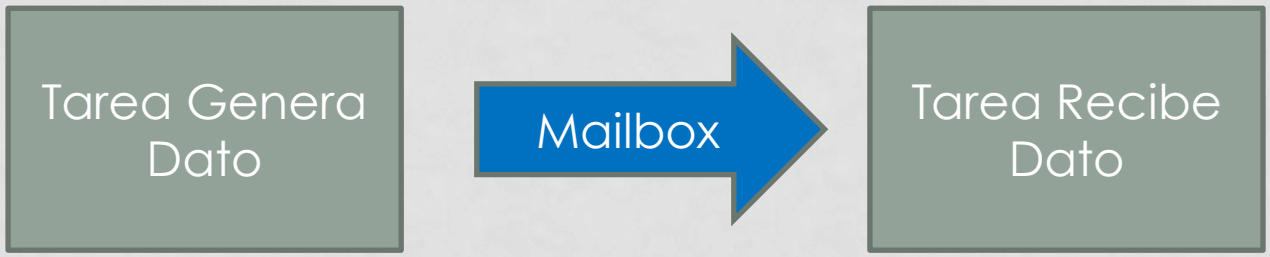
```
for(;;){
    OS_ENTER_CRITICAL();

    salidaLED_1 = 1;
    salidaLED_2 = 1;

    OS_EXIT_CRITICAL();

    OSTimeDly(2);

    salidaLED_1 = 0;
    salidaLED_2 = 0;
    OSTimeDly(2);
}
```

```
for(;;){
    OSSchedLock();

    salidaLED_1 = 1;
    salidaLED_3 = 1;
    OSTimeDly(5);

    salidaLED_1 = 0;
    salidaLED_3 = 0;
    OSTimeDly(5);

    OSSchedUnlock();
```

Ejemplo 10 CRITICAL

EVENTOS - MAILBOX

- Entidad que permite transferir información entre Tareas.
- Se puede usar para Sincronizar dos Tareas.
- Se asemeja a una variable de tipo global.



MAILBOX - EJEMPLO

- Control de una salida Digital en base a un valor que resulta de un canal ADC.

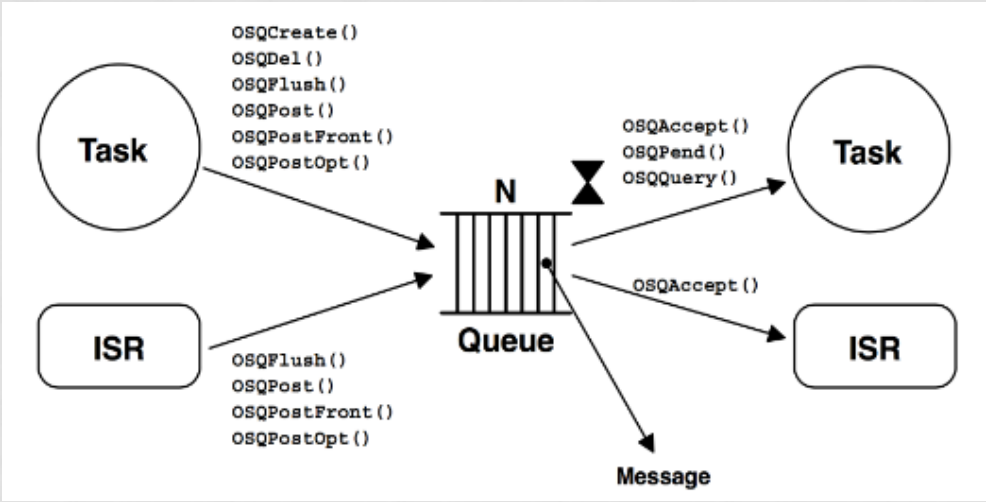
```
for(;;)
{
    OSTimeDly(5);
    uiValorADC = convertirADC();
    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

Ejemplo 5 MBOX

```
{
    rxmsg = OSMboxPend(ADC_Mbox, 0, err);
    inc_P = *rxmsg;

    if(*rxmsg >= 250)
        salidaRELE = 1;
    else
        salidaRELE = 0;

    OSTimeDly(1);
}
```

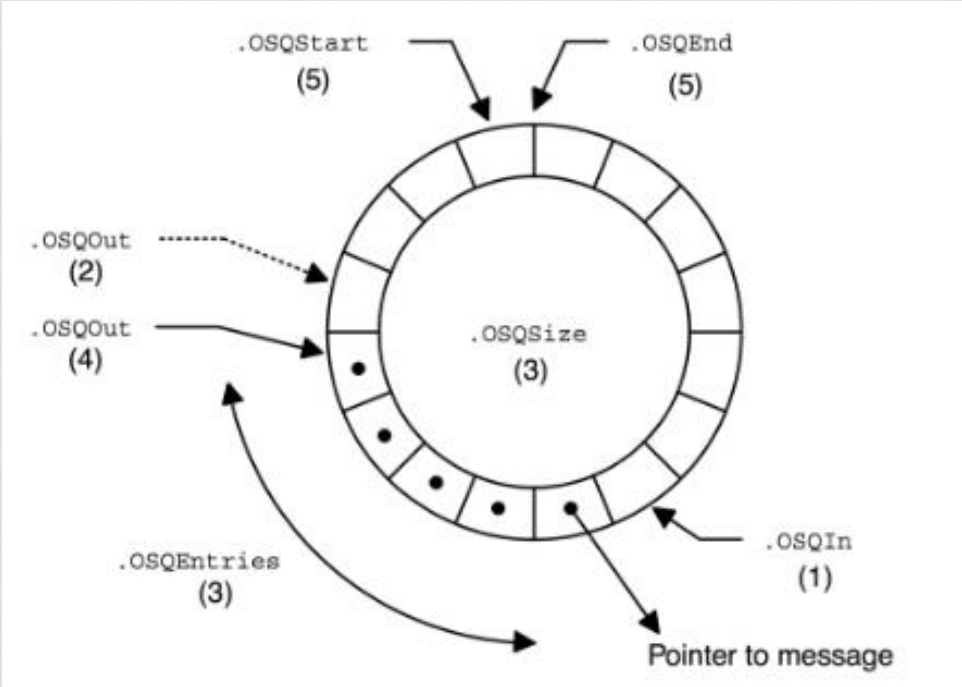
EVENTOS - QUEUES

- Entidad que permite transferir una cola de mensajes.
- Es posible almacenar mensajes siguiendo una lógica FIFO o LIFO.



EVENTOS - QUEUES

- Se almacenan los mensajes en una estructura de buffer circular.



EVENTOS - QUEUES

- Ejemplo de USO

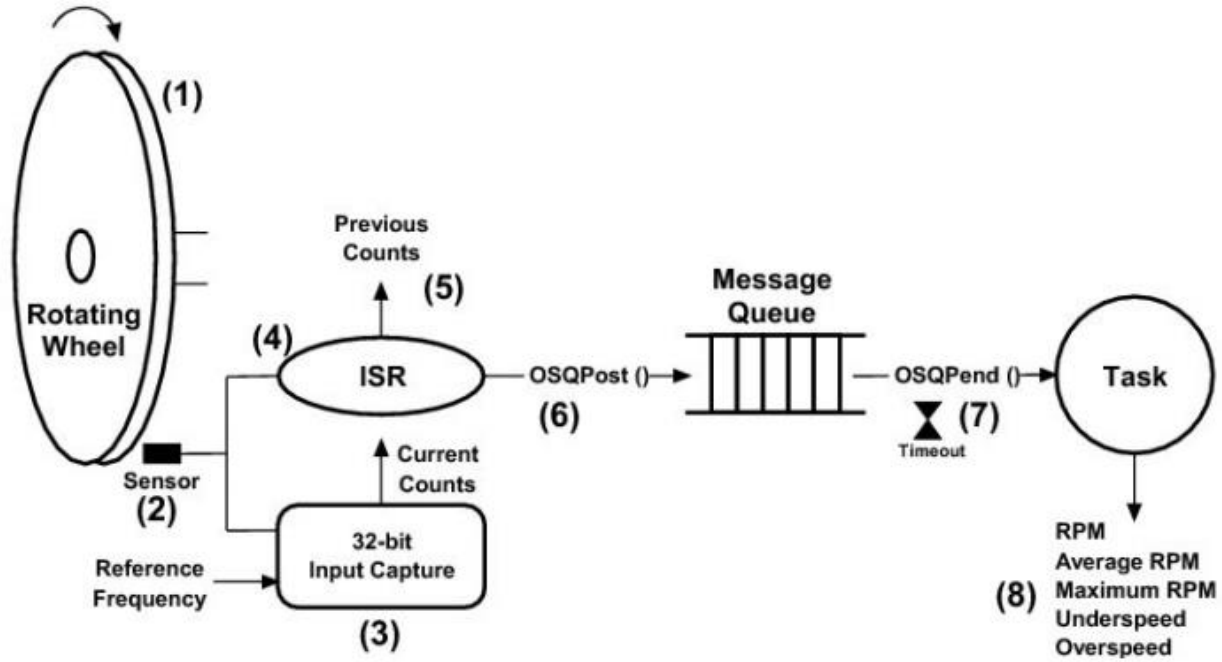
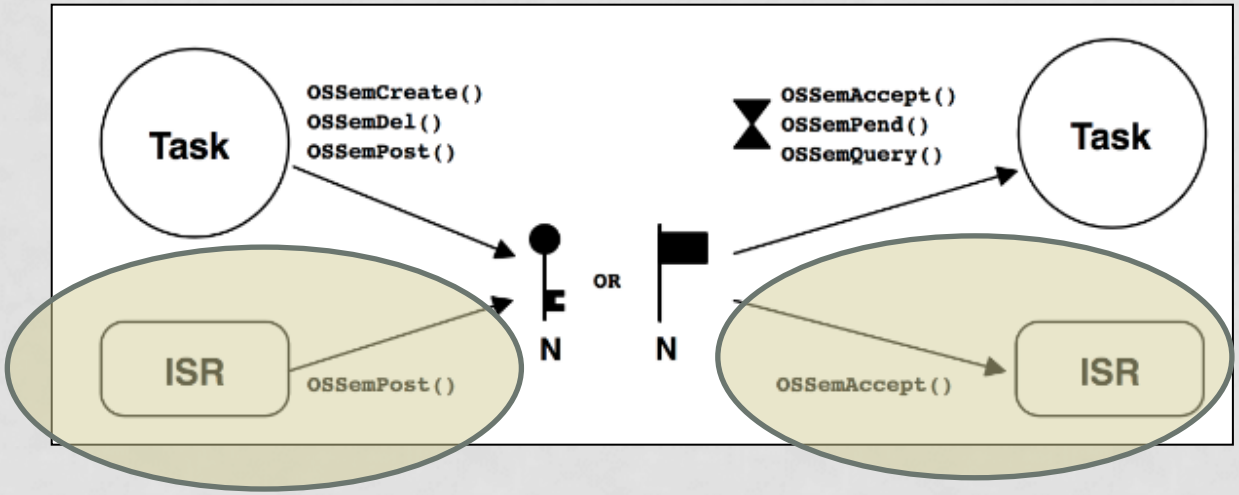


Figure - Measuring RPM

SINCRONIZACIÓN DESDE ISR

- Es posible usando EVENTOS sincronizar una Tarea con la ocurrencia de una Interrupción.



REENTRANCIA

- El concepto es crucial al momento de la definiciones de la solución del problema.
- Recordar que existen cambios de contexto durante toda la vida operacional del software desarrollado.

REENTRANCIA

Baja Prioridad

```
for(;;)
{
    OSTimeDly(5);
    uiValorADC = convertirADC();

    uiValorADC = 222;
    demoraLarga(5);

    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

Alta Prioridad

```
for(;;)
{
    OSTimeDly(2);
    uiValorADC = convertirADC();
    uiValorADC = 333;
    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

RTOS - REENTRANCIA

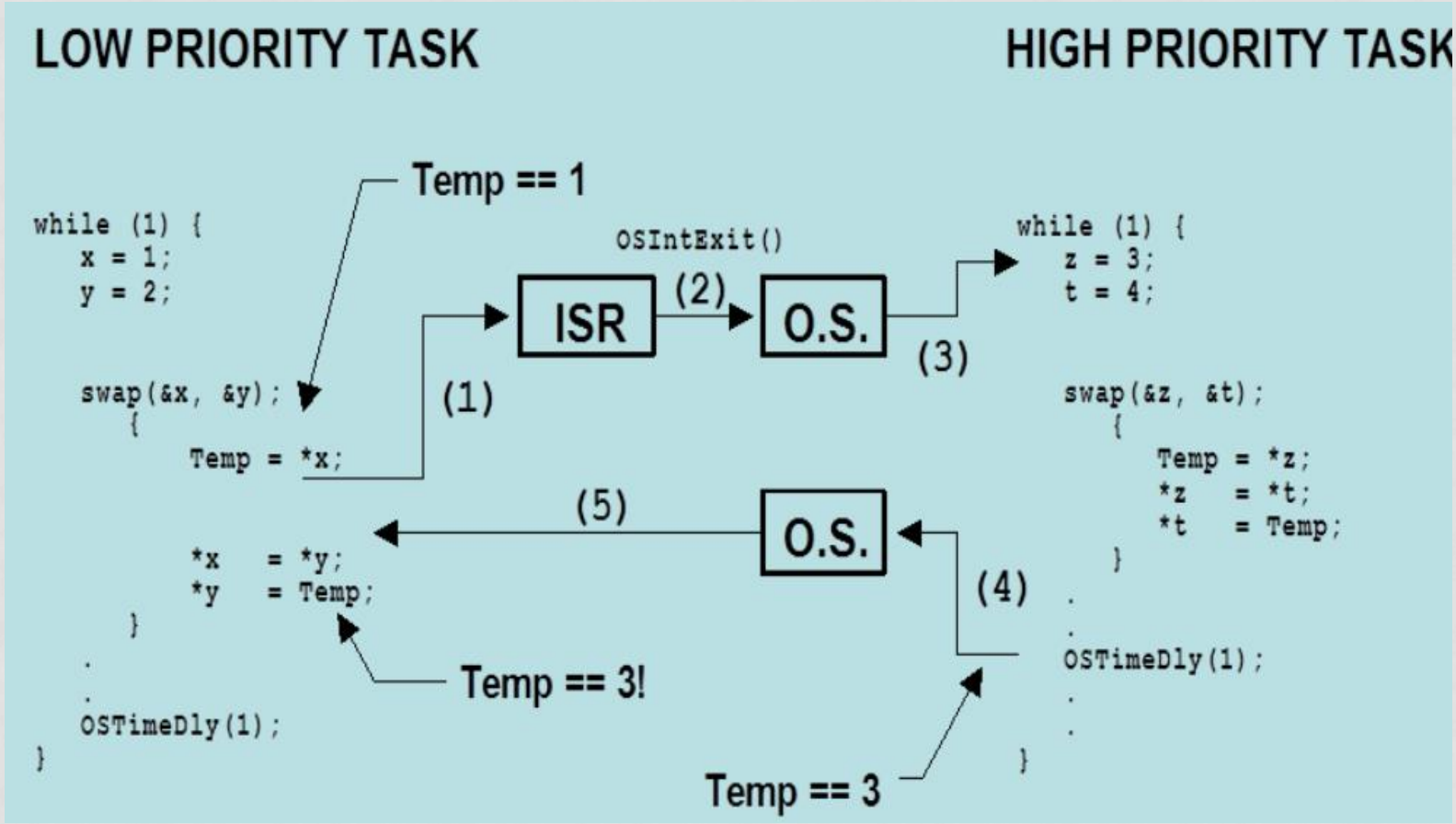
- Para trabajar en forma segura en un entorno multitarea las funciones deben ser reentrantes.

```
void strcpy(char *dest, char *src)
{
     while (*dest++ = *src++) {
          ;
     }
     *dest = NUL;
}
```

- En el ejemplo se verifica la no reentrancia de la función swap().

```
int Temp;

void swap(int *x, int *y)
{
     Temp = *x;
     *x   = *y;
     *y   = Temp;
}
```

RTOS - REENTRANCIA



CAMBIOS DE CONTEXTO

- Se producen cambios de contexto cuando expira el tiempo establecido para funcionar.
- Se producen cuando una Tarea solicita un Servicio del RTOS.
- Los cambios de contexto pueden provocar instancias de Deadlock si no son tenidos en cuenta.

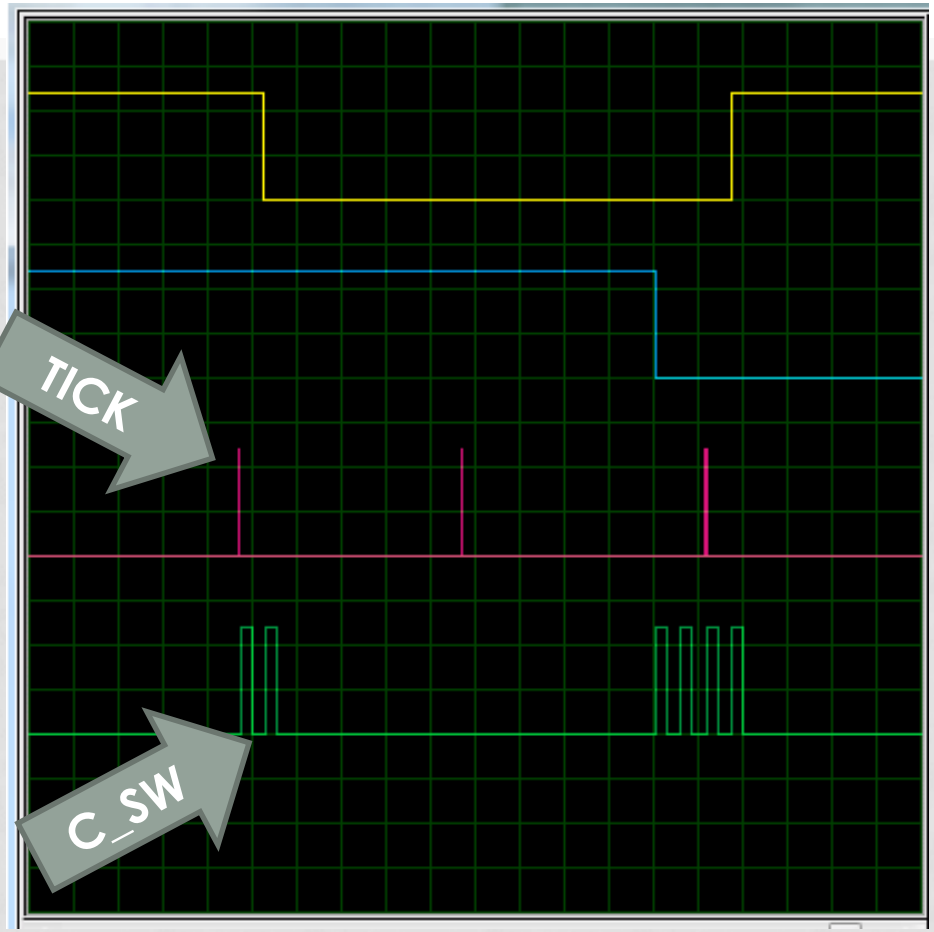
Ejemplo 9 PREEMPTIVE

CAMBIOS DE CONTEXTO

- Se puso para el ejemplo un led que se activa y desactiva cuando se produce el Context Switch,
- Se puso un led que se activa cuando se ejecuta la ISR del timer.

```
void CPUInterruptHook(void)
{
    if(INTCONbits.TMR0IF) {

        salidaLed3 = 1;

        INTCONbits.TMR0IF = 0;

        TMR0H = 60;
        TMR0L = 251;

        OSTimeTick();
        salidaLed3 = 0;
    }
}
```

```
void OSTaskSwHook (void)
{
    salidaLed4 = 1;
    Delay10TCYx(10L);
    salidaLed4 = 0;
    Delay10TCYx(10L);
}
```



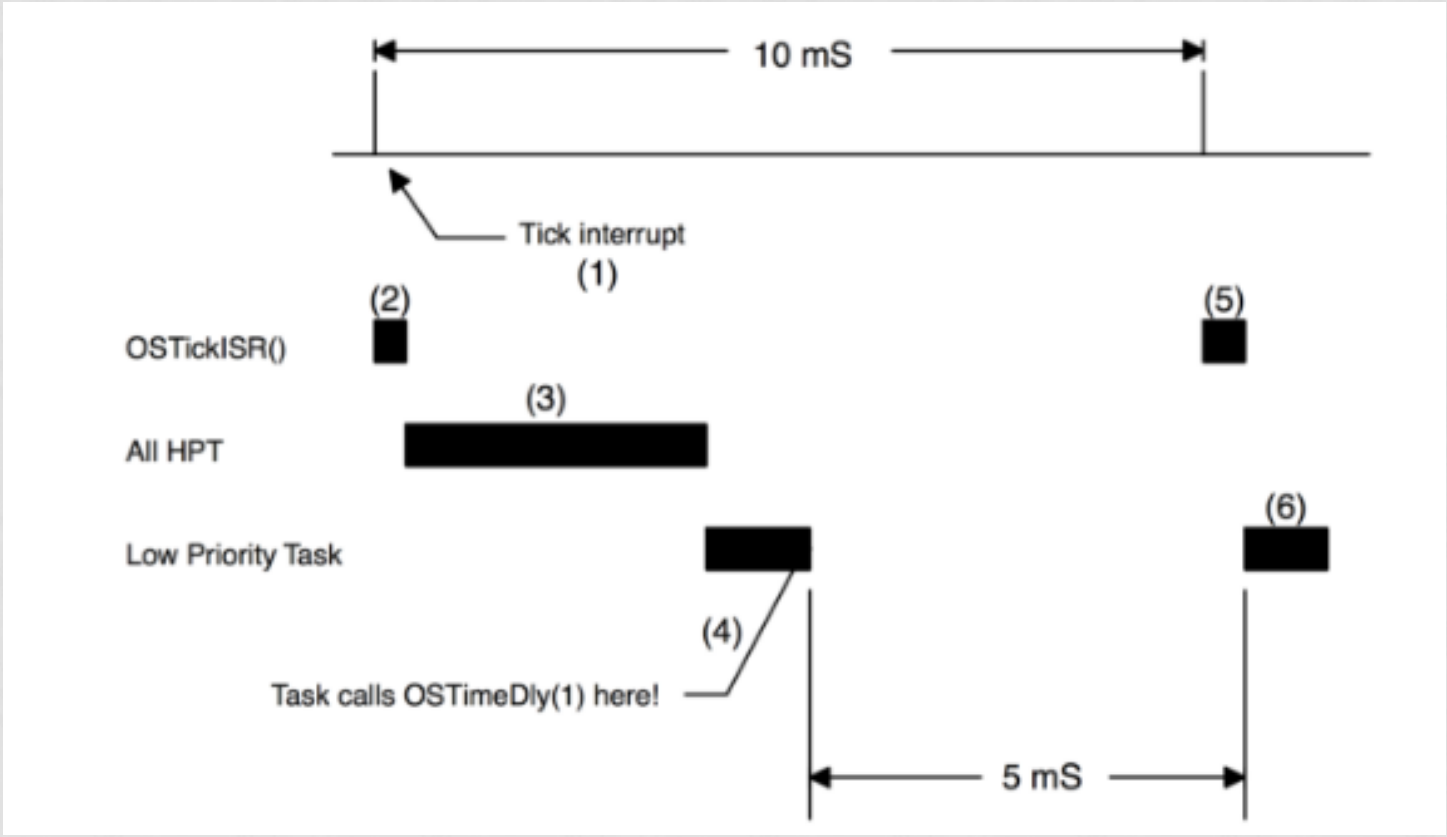
Ejemplo 9

RESOLUCIÓN DE TIEMPOS

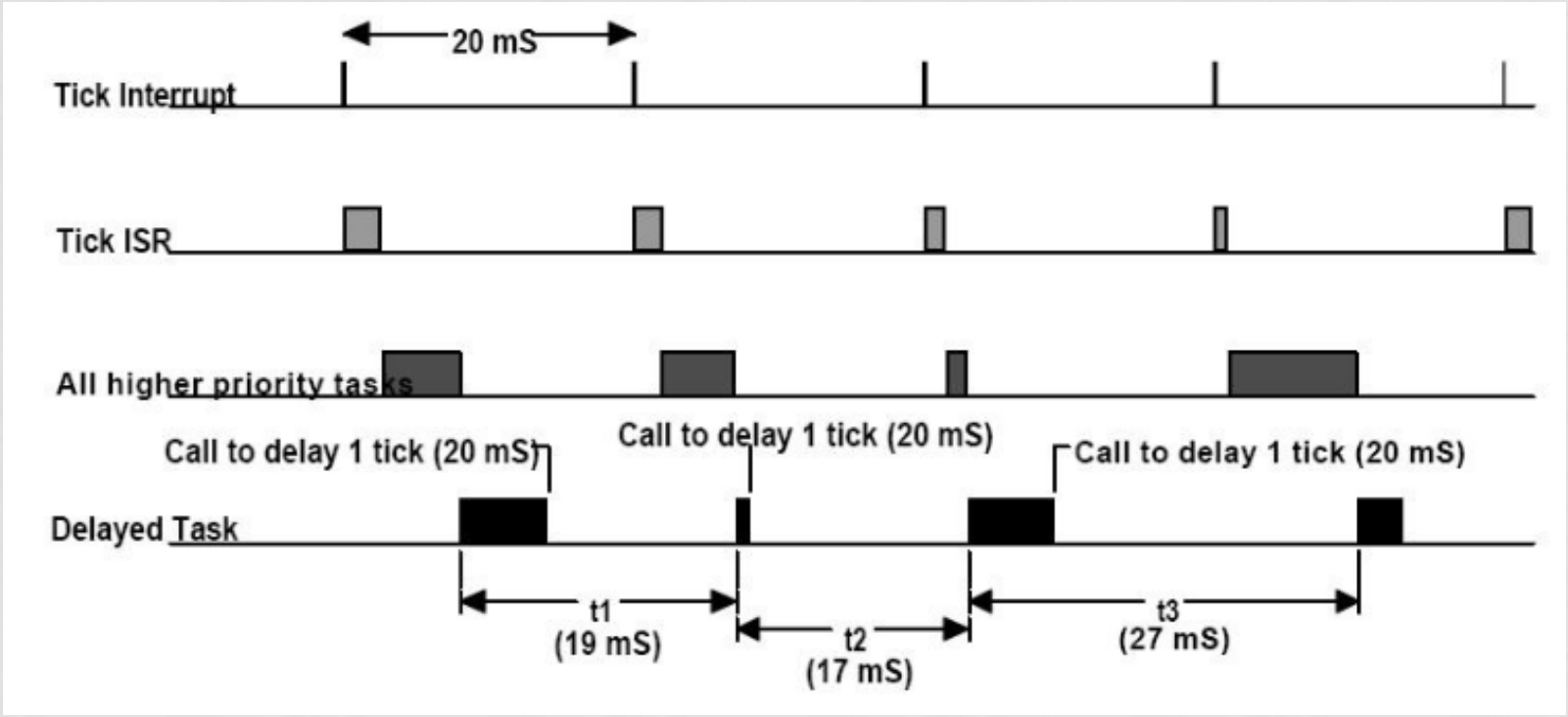
- La tarea de MAS Alta prioridad tendrá la posibilidad de ejecutarse dentro del tiempo establecido.
- Las tareas de menor prioridad ocuparán al CPU pudiendo no ejecutarse dentro del tiempo del sistema.
  - Ejecución completa antes de que se produzca un tick del sistema.

RESOLUCIÓN DE TIEMPO

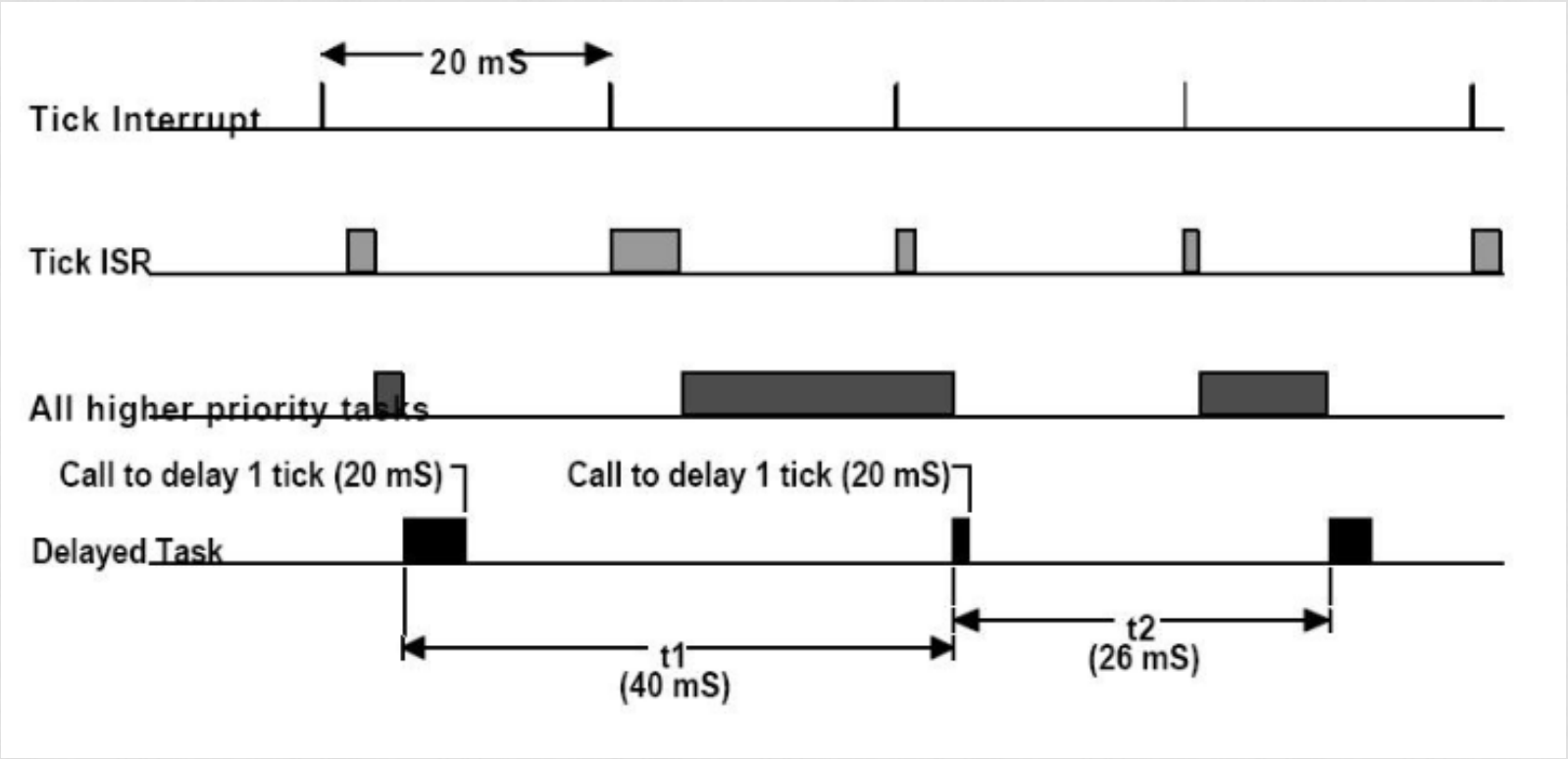
- La tarea de MAS Alta prioridad tendrá la posibilidad de ejecutarse dentro del tiempo establecido.



RESOLUCIÓN DE TIEMPO

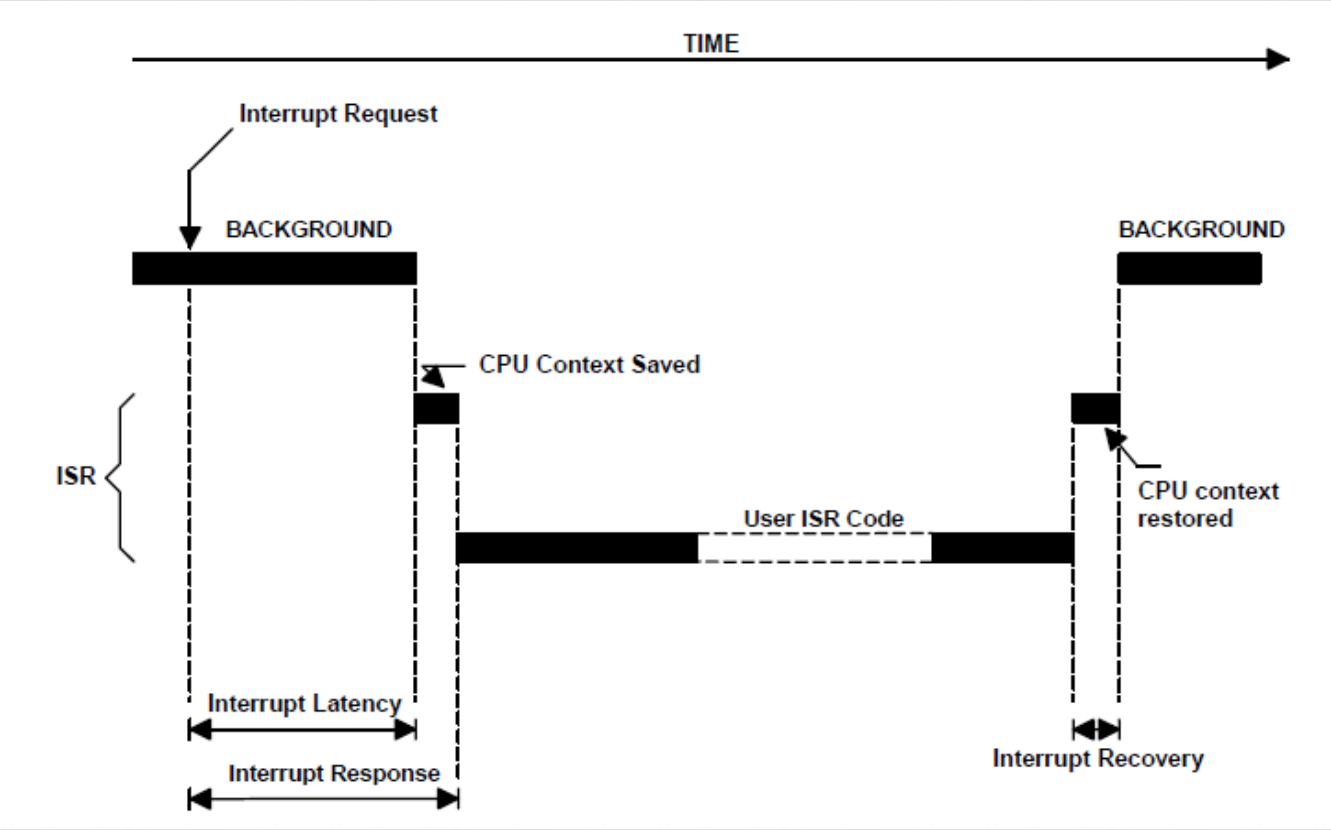


RESOLUCIÓN DE TIEMPO



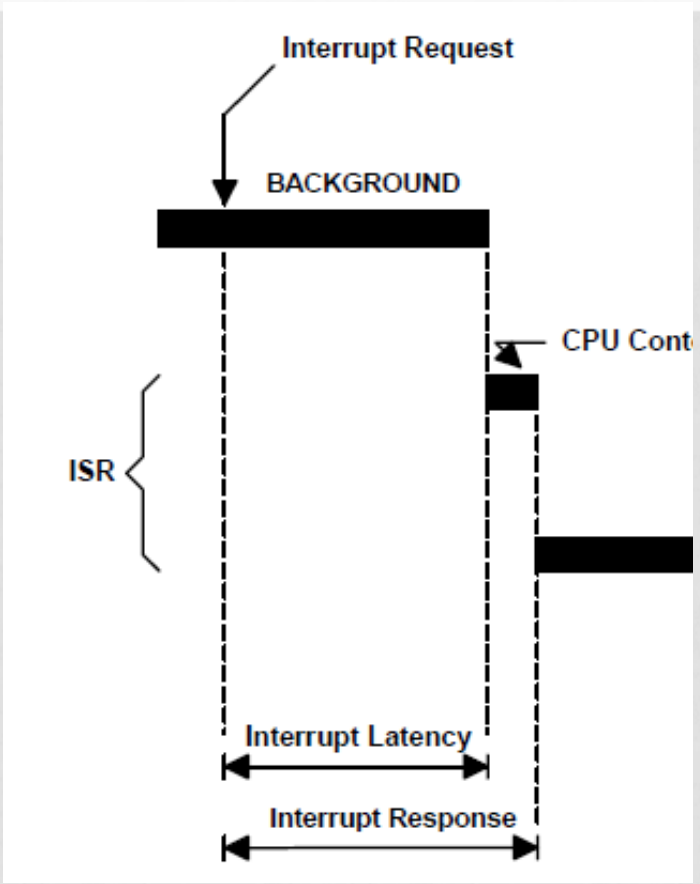
ISR – TIEMPOS DE RESPUESTA

- Una de las especificaciones mas importantes tiene relación con el tiempo en que las interrupciones están deshabilitadas.



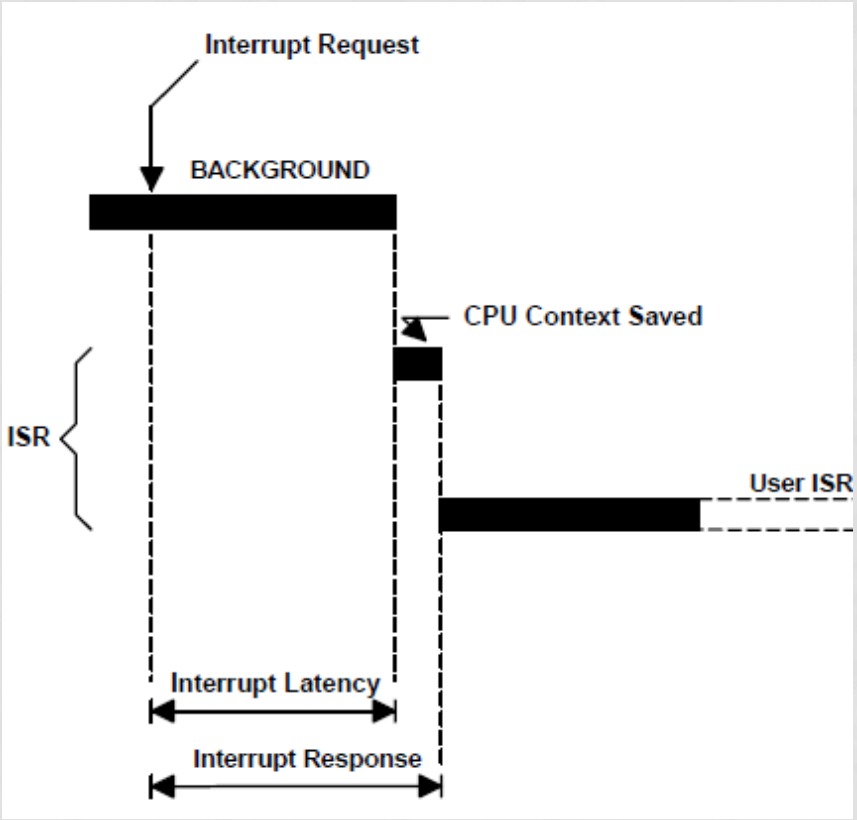
ISR – LATENCIA

- Es el tiempo durante el cual están deshabilitadas las interrupciones + mas el tiempo que se requiere para ejecutar la primer instrucción de la ISR.



ISR – TIEMPO DE RESPUESTA

- Es el que requiere el micro desde que recibe el pedido de interrupción hasta que ejecuta la primer instrucción del la ISR luego de salvar el contexto.
- Dependerá del tipo de sistema desarrollado, foreground/background o RTOS preemptive o nonpreemtive.

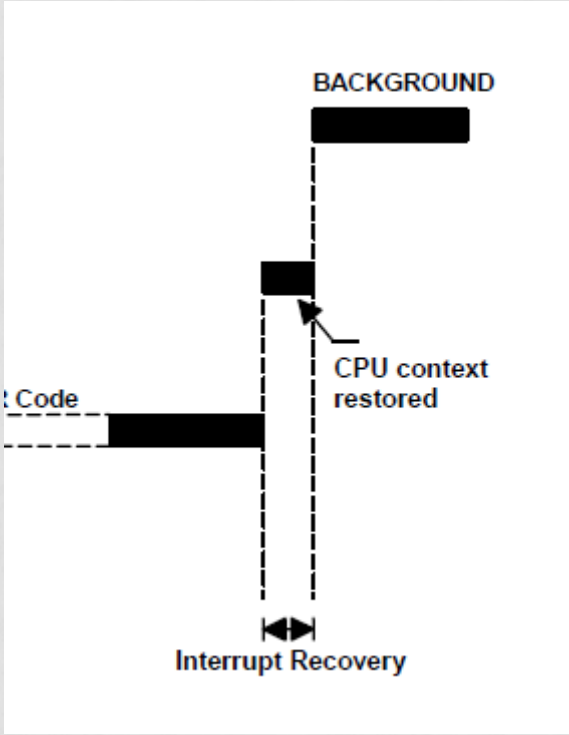


ISR – TIEMPO DE RESPUESTA

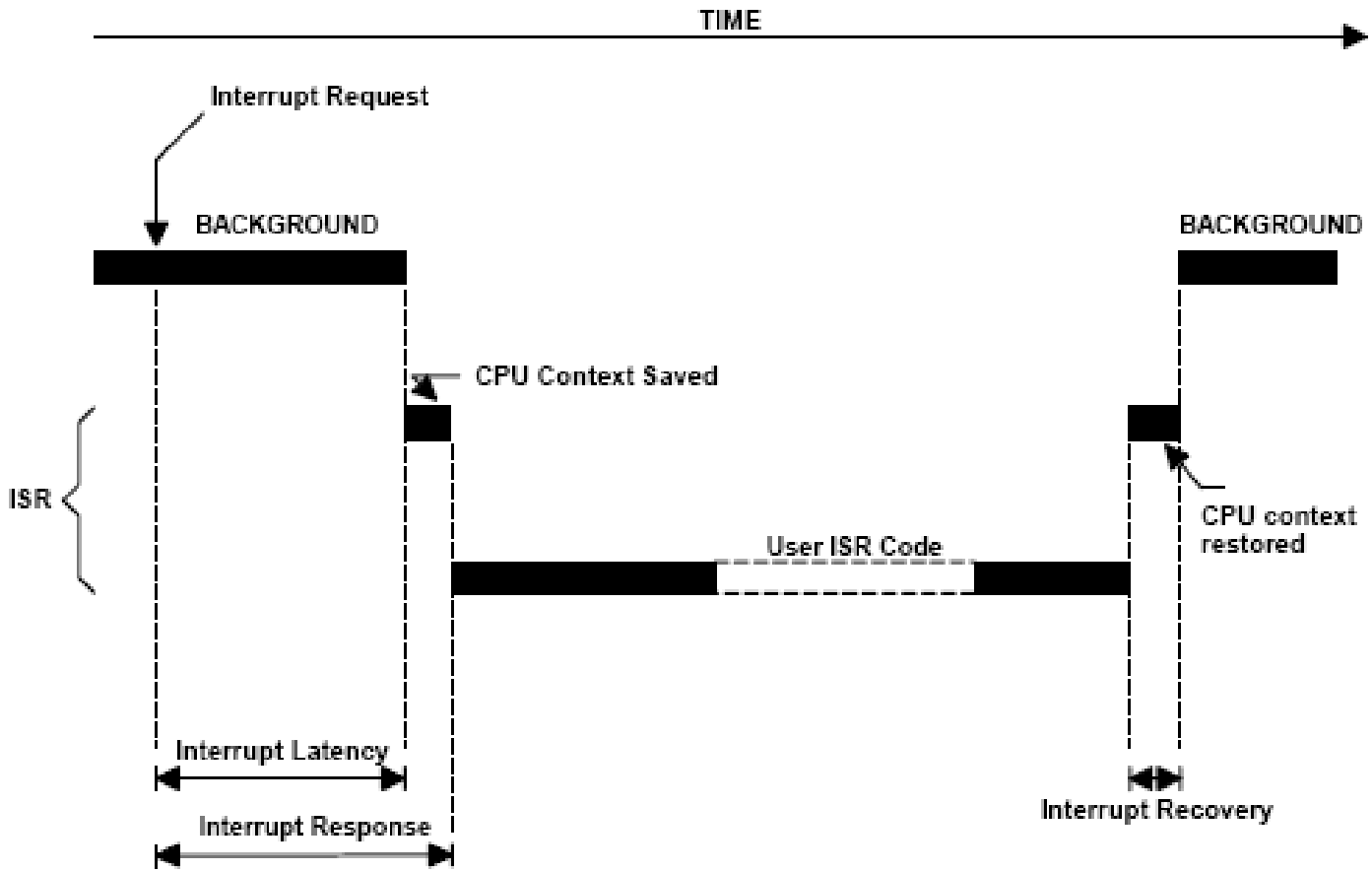
- Se debe considerar como tiempo de respuesta el peor caso como tiempo de respuesta del sistema.
  - Siempre responde en 50uSeg pero alguna vez en 250uS, entonces el tiempo se definirá como de 250uS.

ISR – TIEMPO DE RECUPERACIÓN

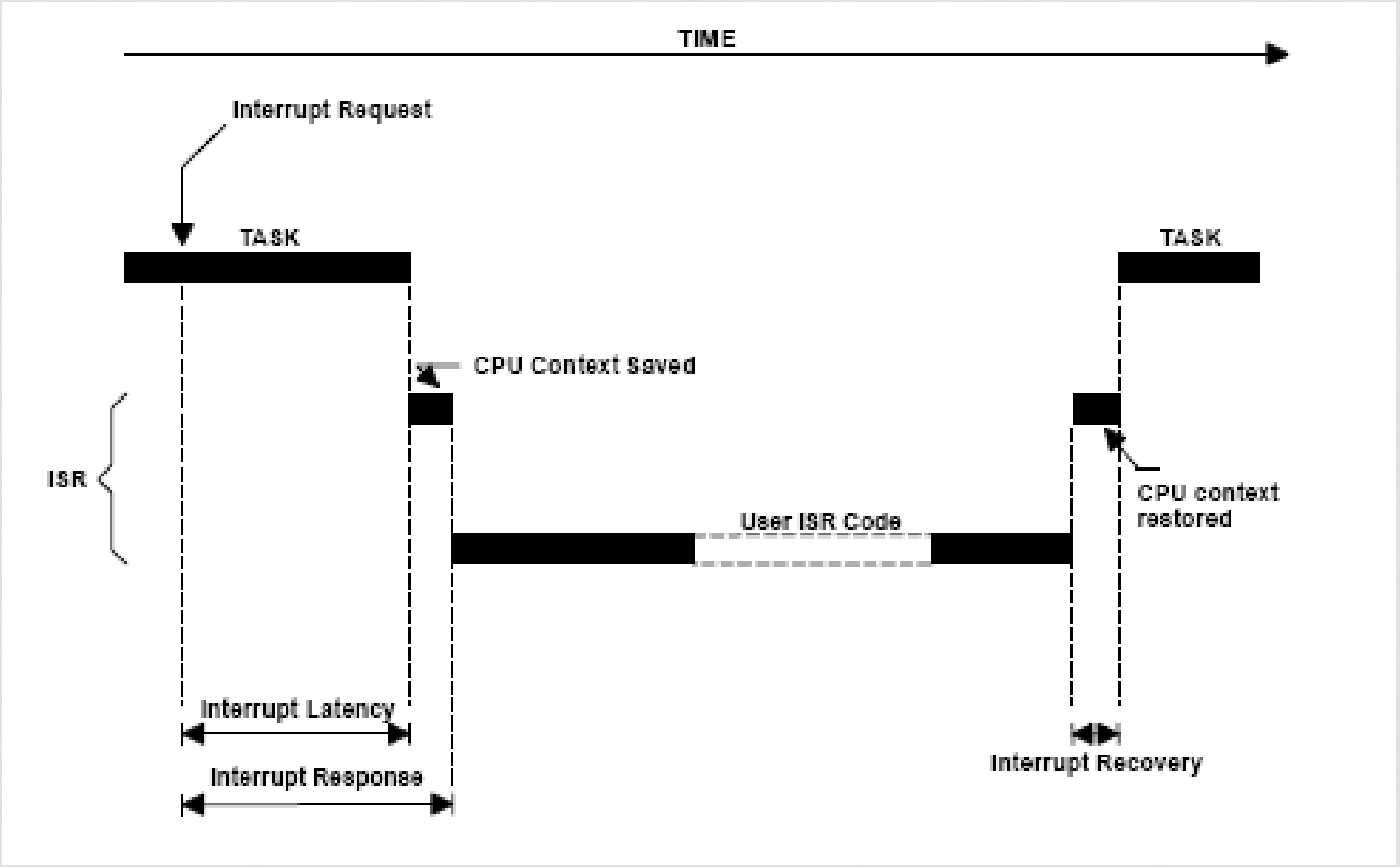
- Es el que se necesita para volver al punto de interrupción.
- Dependiendo del sistema desarrollado será mas o menos el tiempo requerido.
- Foreground/background
- RTOS non preemptive
- RTOS preemptive



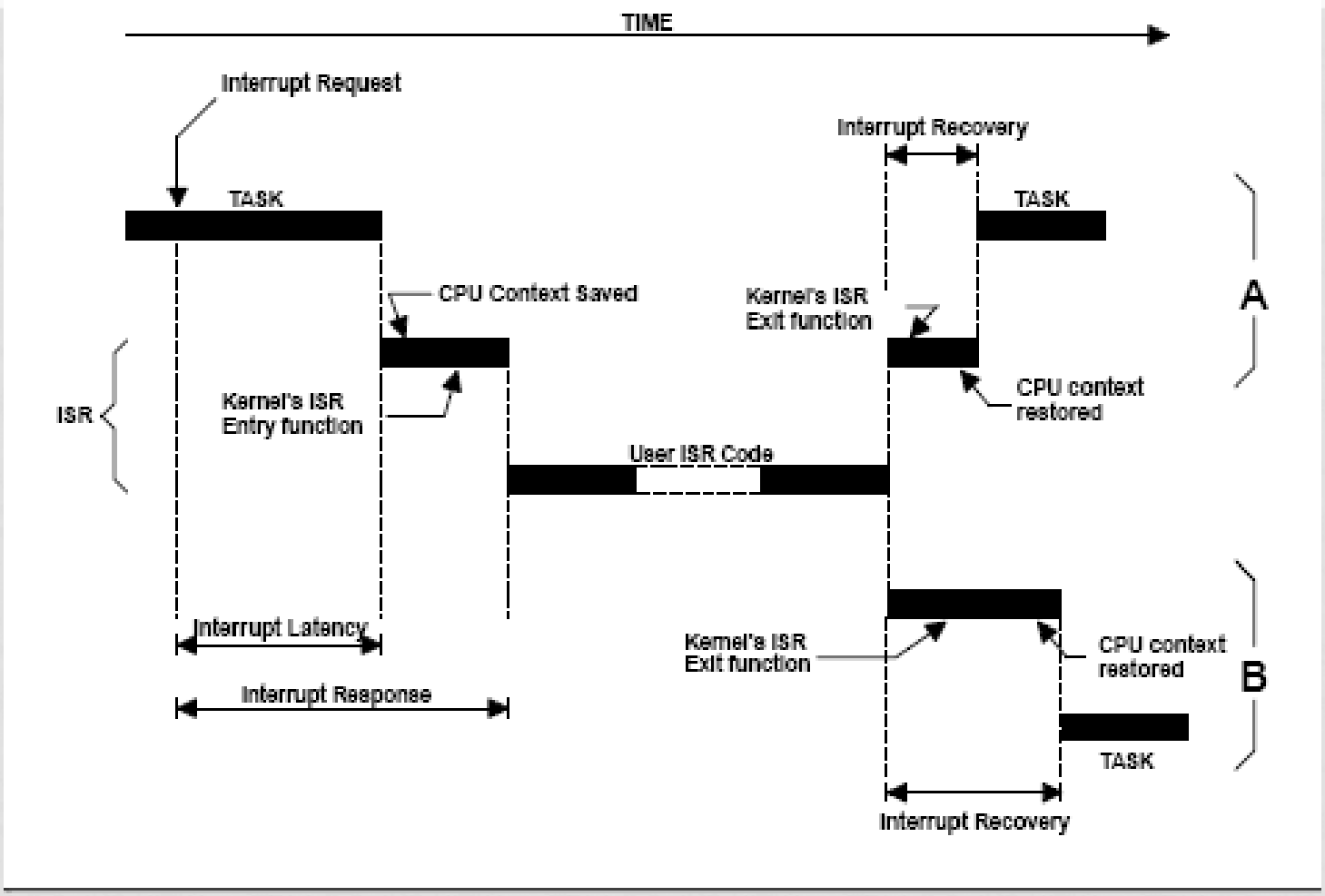
ISR – TIEMPO TOTAL

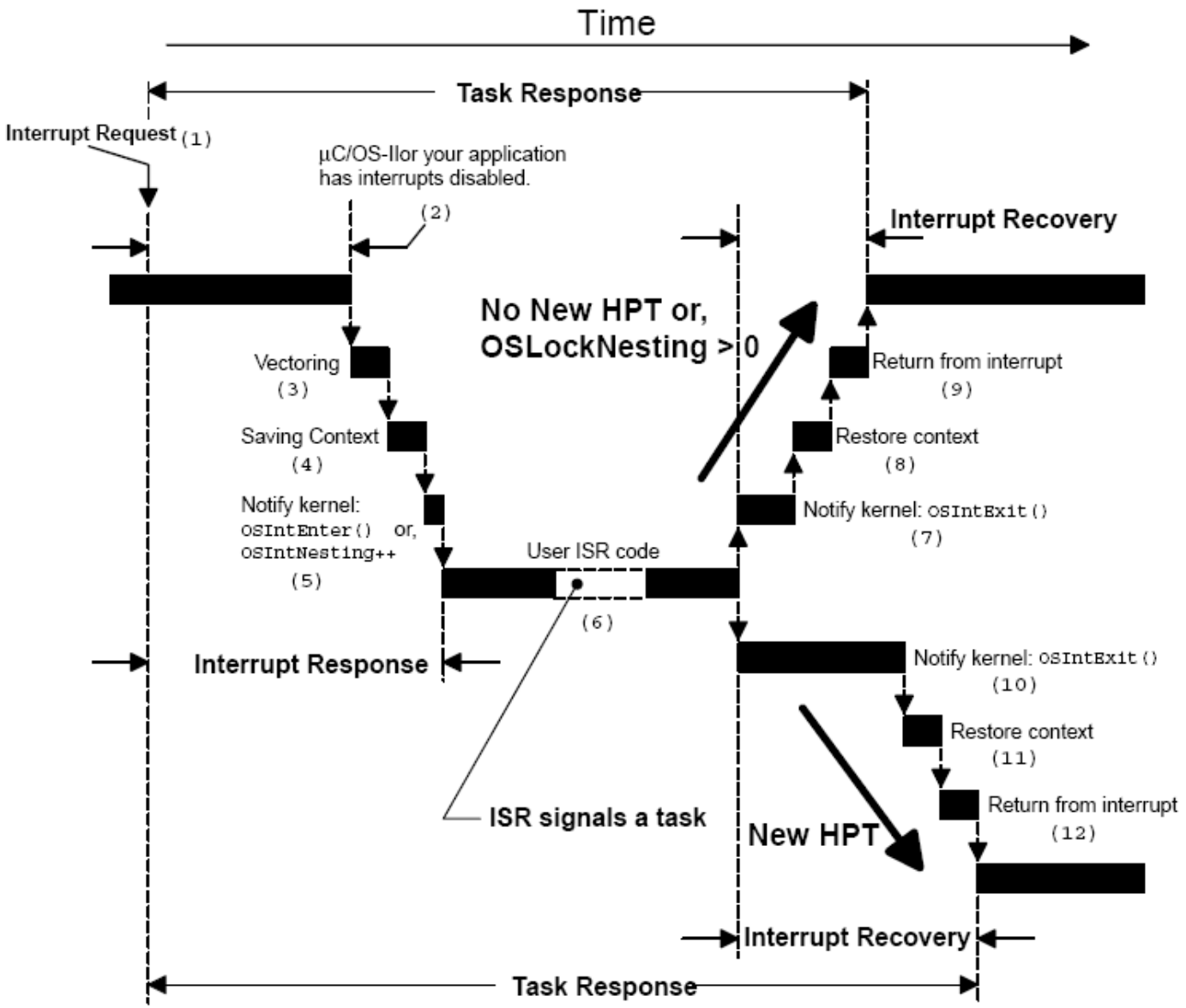


ISR – TIEMPO TOTAL



ISR – TIEMPO TOTAL




Time
Task Response
Interrupt Request (1)
µC/OS-IIor your application has interrupts disabled.
(2)
Interrupt Recovery
No New HPT or, OSLockNesting > 0
Vectoring (3)
Return from interrupt (9)
Saving Context (4)
Restore context (8)
Notify kernel: OSIntEnter() or, OSIntNesting++ (5)
Notify kernel: OSIntExit() (7)
User ISR code
(6)
Interrupt Response
Notify kernel: OSIntExit() (10)
Restore context (11)
Return from interrupt (12)
ISR signals a task
New HPT
Interrupt Recovery
Task Response

ASPECTOS RELEVANTES - RESUMEN

- Se debe disponer de una plataforma que posea RAM y ROM acorde al problema.
- Capacidad de administrar en forma dinámica el STACK
- Requerimos un dispositivo que permita controlar el tiempo del sistema.
  - TIMER
- Se debe crear una tarea antes de lanzar el proceso multitarea.

ASPECTOS RELEVANTES - RESUMEN

- Se deben realizar un análisis del sistema para la correcta definición de las Tareas.
- Los Tareas definidas requieren de asignación de prioridades.
- Es posible reasignar prioridades en forma dinámica.
- Establecer el modo de trabajo del RTOS.
  - Preemptive - Non Premtive

ASPECTOS RELEVANTES - RESUMEN

- El uso de Eventos para Sincronización requiere de recursos de ROM y RAM.
- Se debe configurar el RTOS antes de comenzar
  - Cantidad de Tareas, Eventos, etc.

PREGUNTAS?

Contacto: MSc. Carlos Centeno
ccenteno@gmail.com

G.In.T.E.A – FRC UTN
http://www.investigacion.frc.utn.edu.ar/gintea/

FUENTES UTILIZADAS

- MPLAB 8.53
- C18 3.02
- uCOS-II
- freeRTOS 8.0.0
- https://www.freertos.org
- https://doc.micrium.com/display/osiidoc
- https://www.st.com/en/development-tools/sw4stm32.html