



Software en Tiempo Real

Msc. Ing. Carlos Centeno
Ingeniería Electrónica
UTN FRC

Año 2023

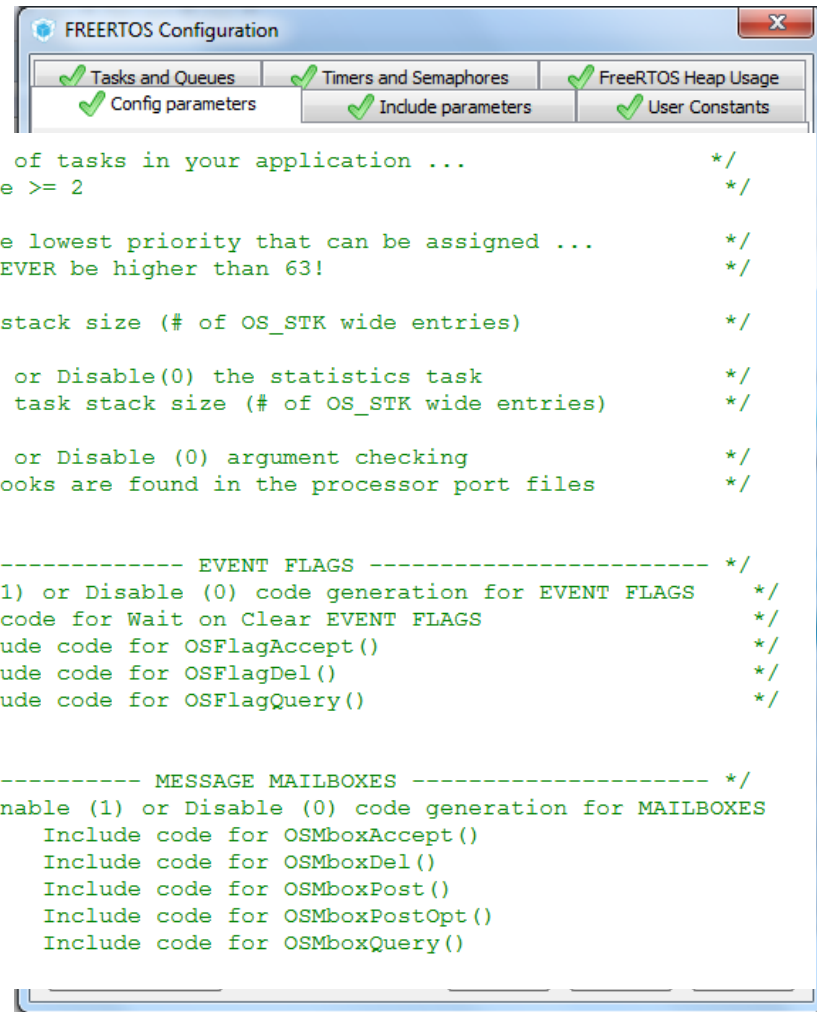
Solución - RTOS

- 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 RTOS

- 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 RTOS



```
#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 */

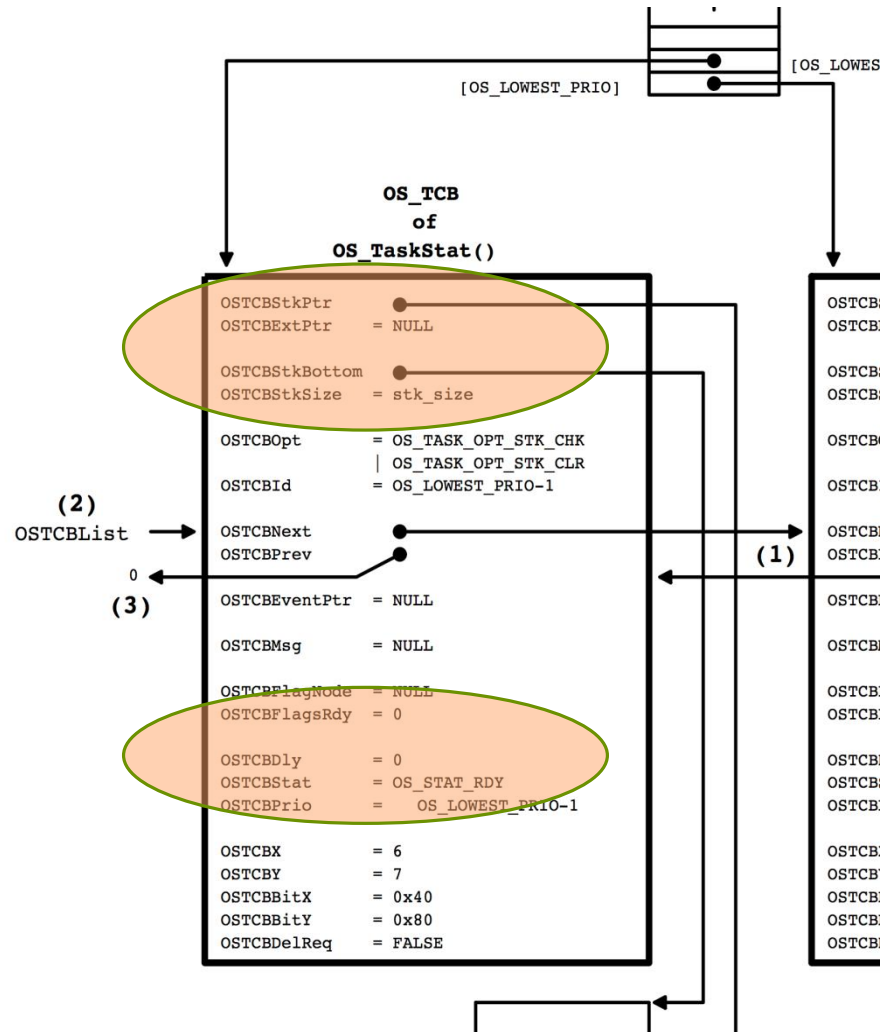
/* ----- EVENT FLAGS ----- */
#define OS_FLAG_EN              0       /* Enable (1) or Disable (0) code generation for EVENT FLAGS */
#define OS_FLAG_WAIT_CLR_EN    1       /* Include code for Wait on Clear EVENT FLAGS */
#define OS_FLAG_ACCEPT_EN      1       /* Include code for OSFlagAccept() */
#define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

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 ejecución.
- **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      *OSTCBEvtPtr;
    #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;

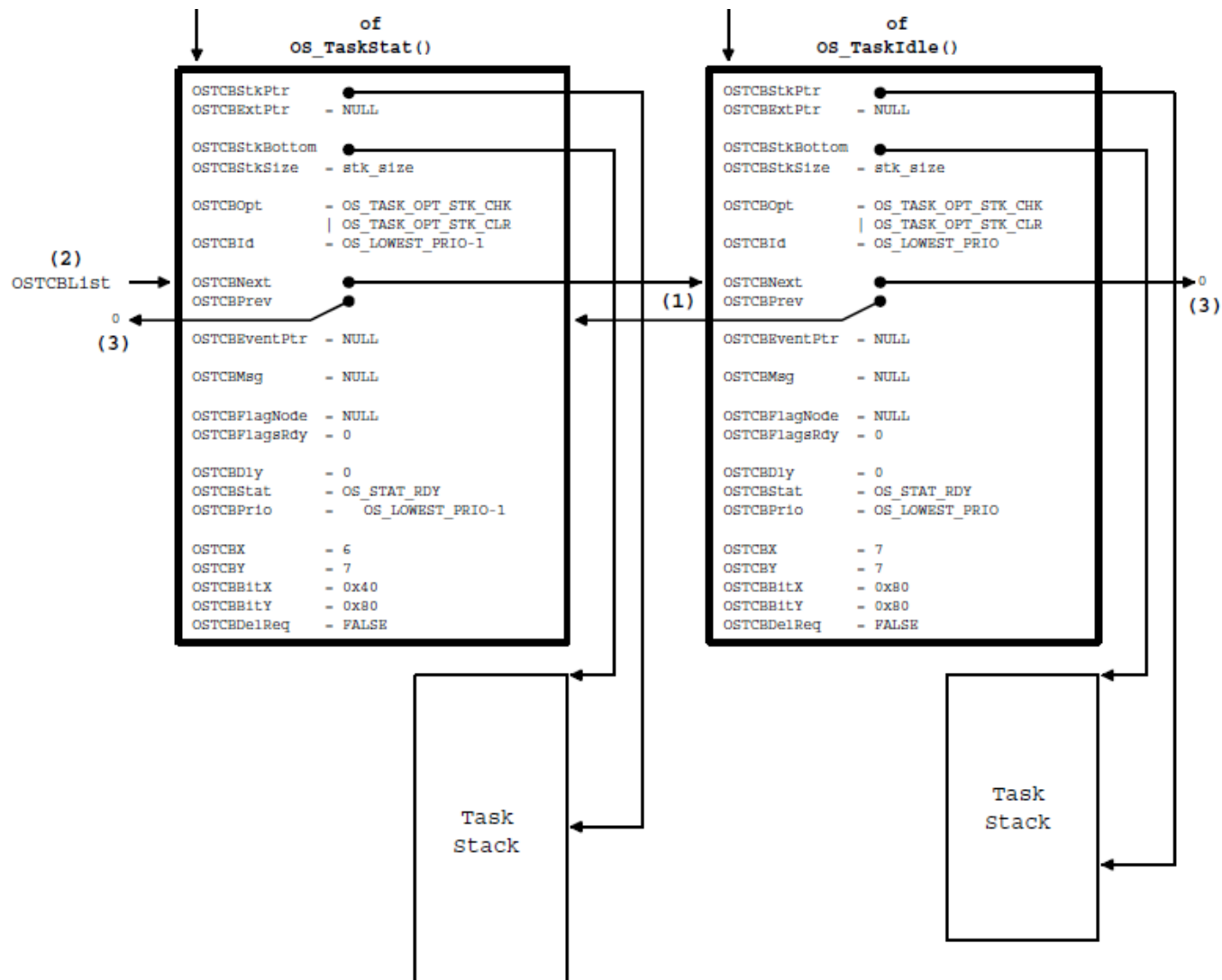
    #if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
        OS_EVENT      *OSTCBEvtPtr;
    #endif

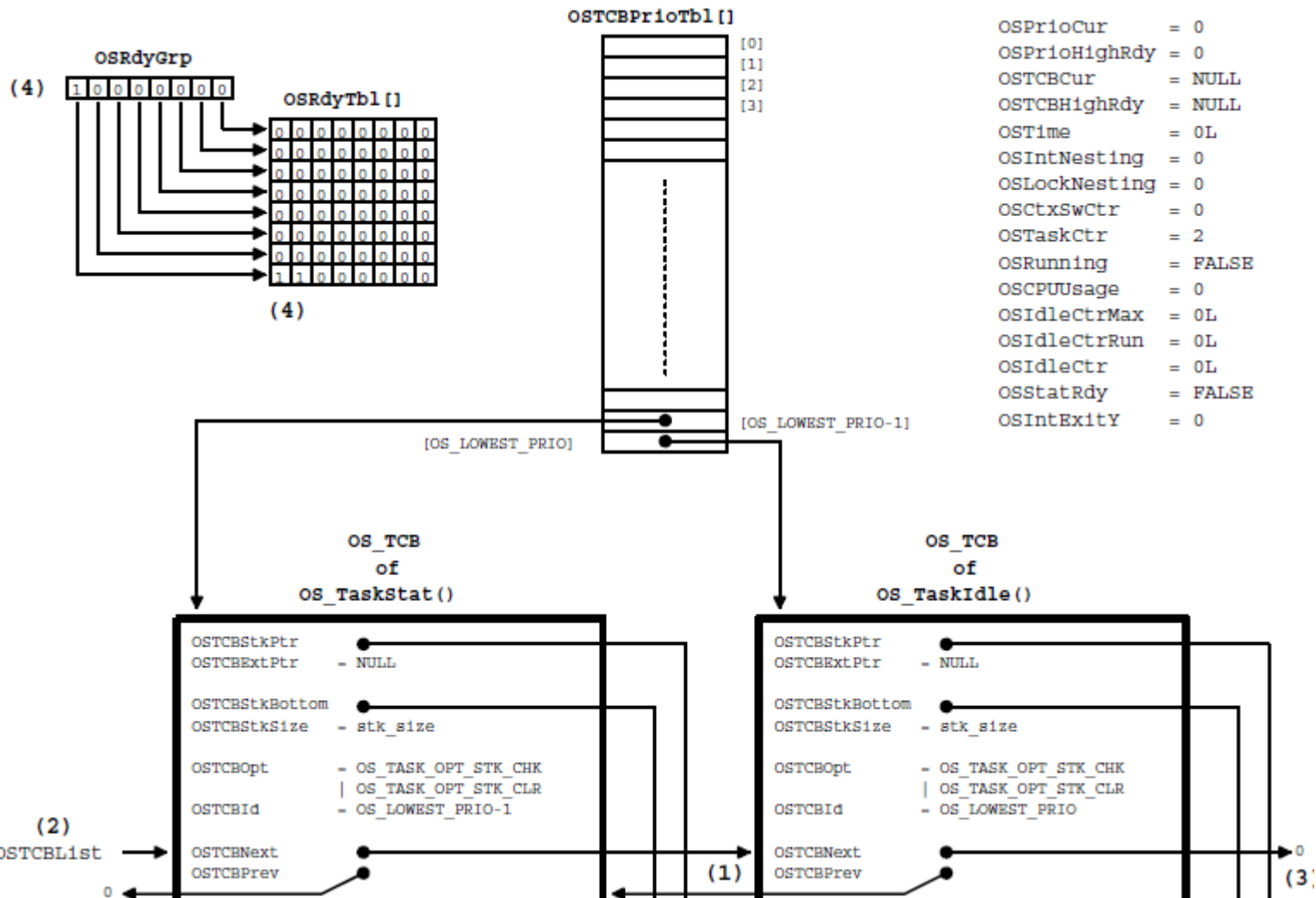
    #if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
        void          *OSTCBMsg;
    #endif

        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

- 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.

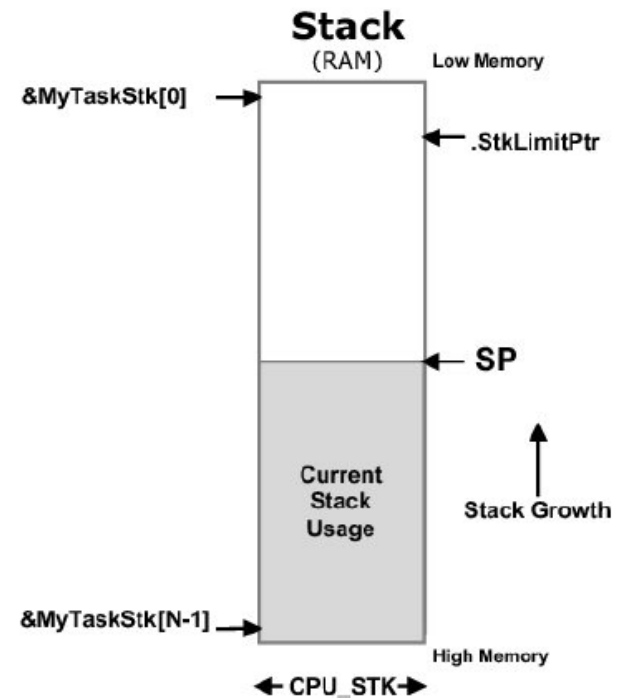
Stack - Definición

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.

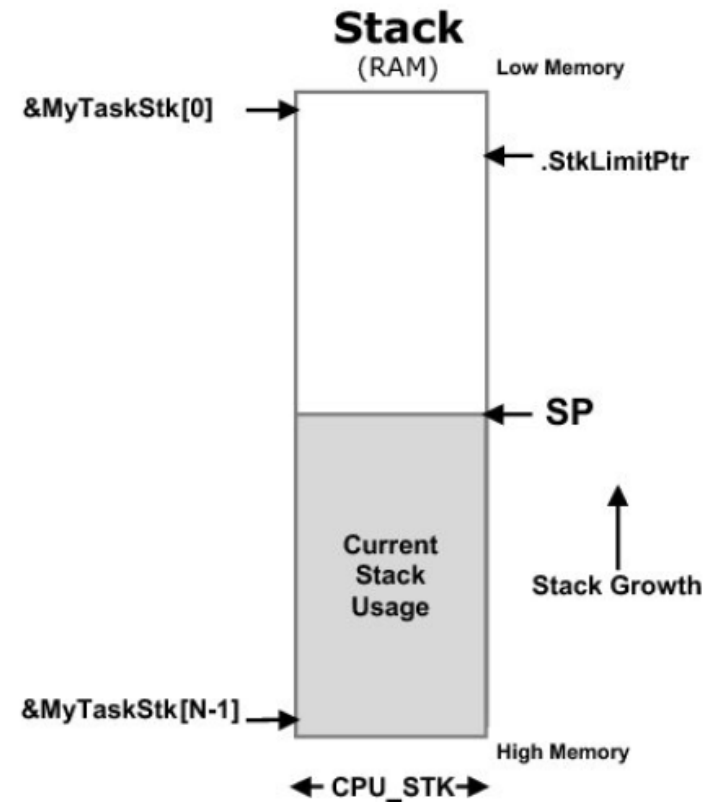
Mecanismos protección Stack

- 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.

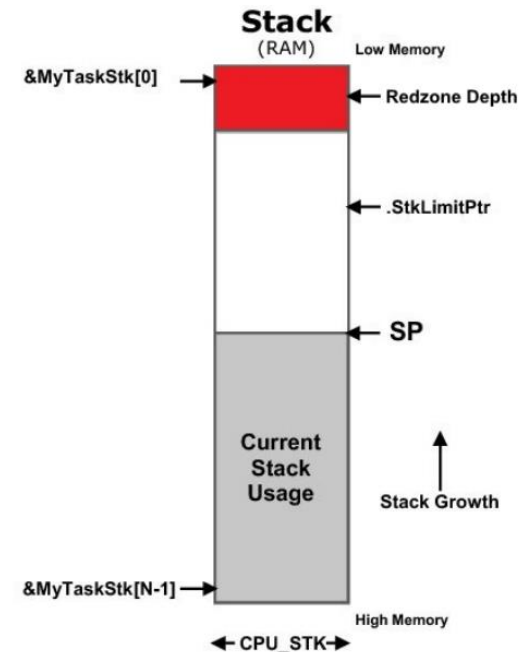
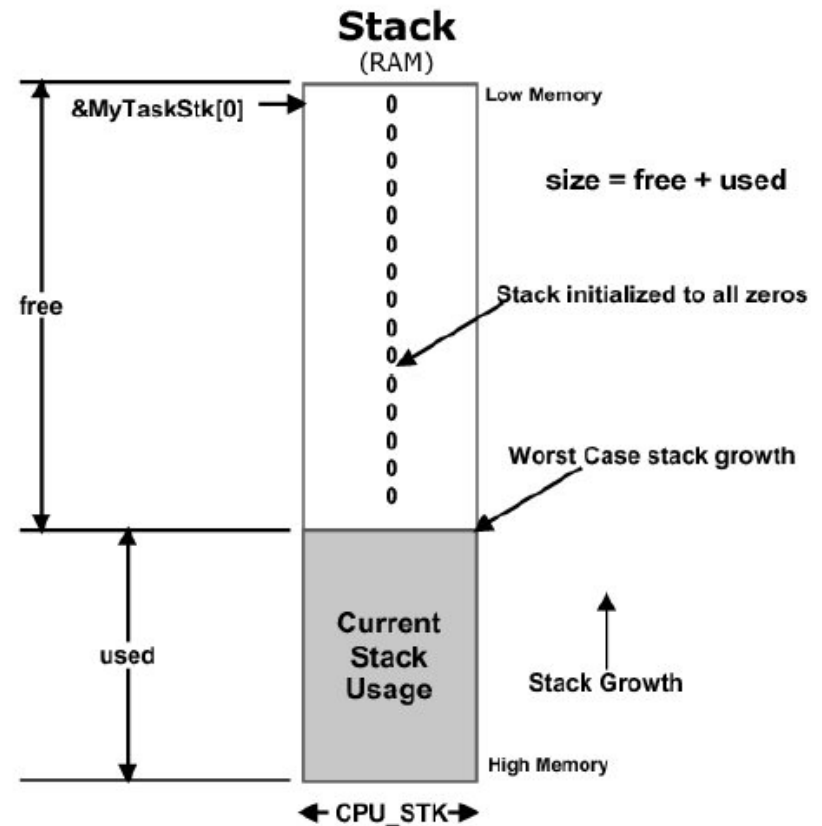


Figure - Redzone Stack Overflow detection

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 código.

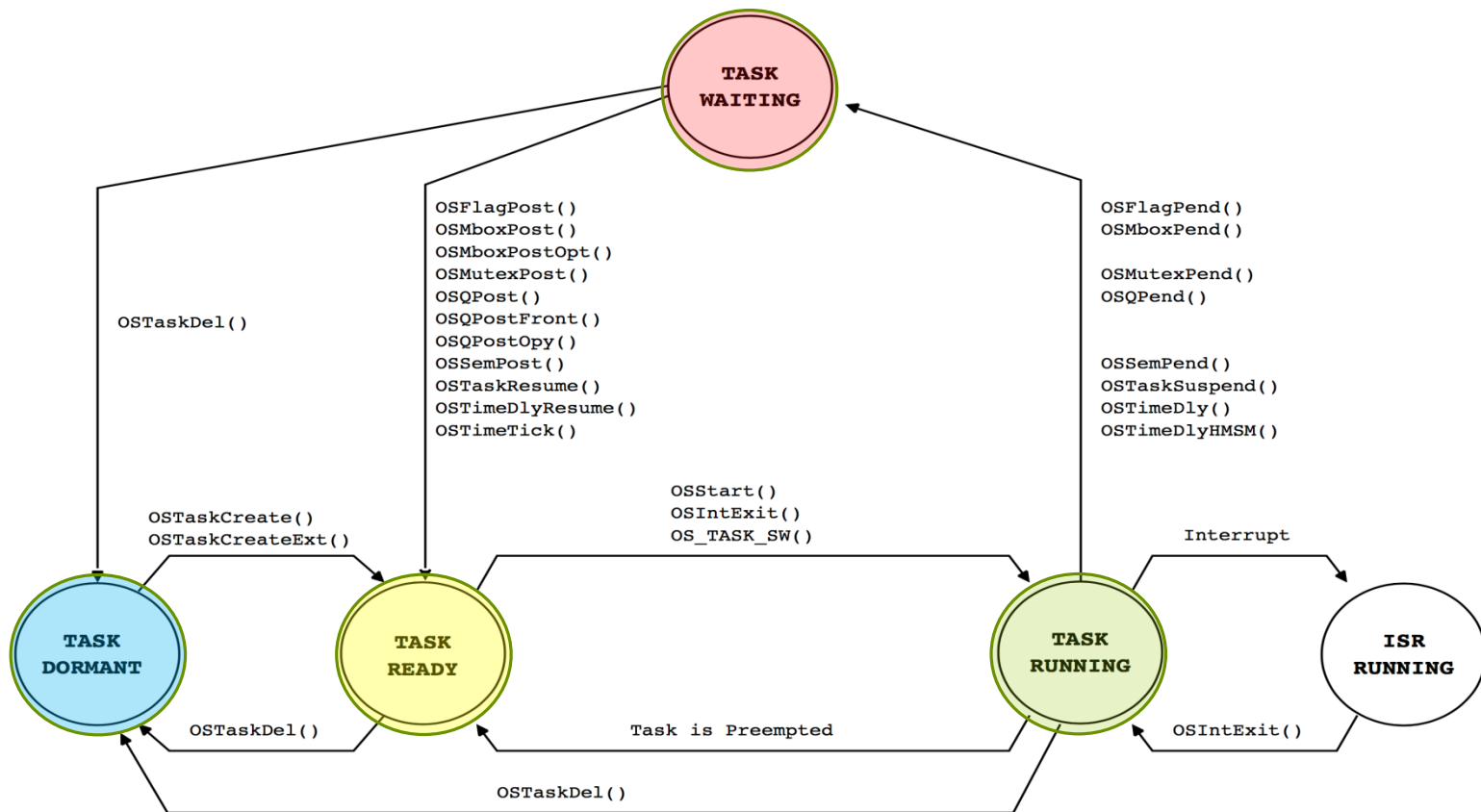


Estados Definidos

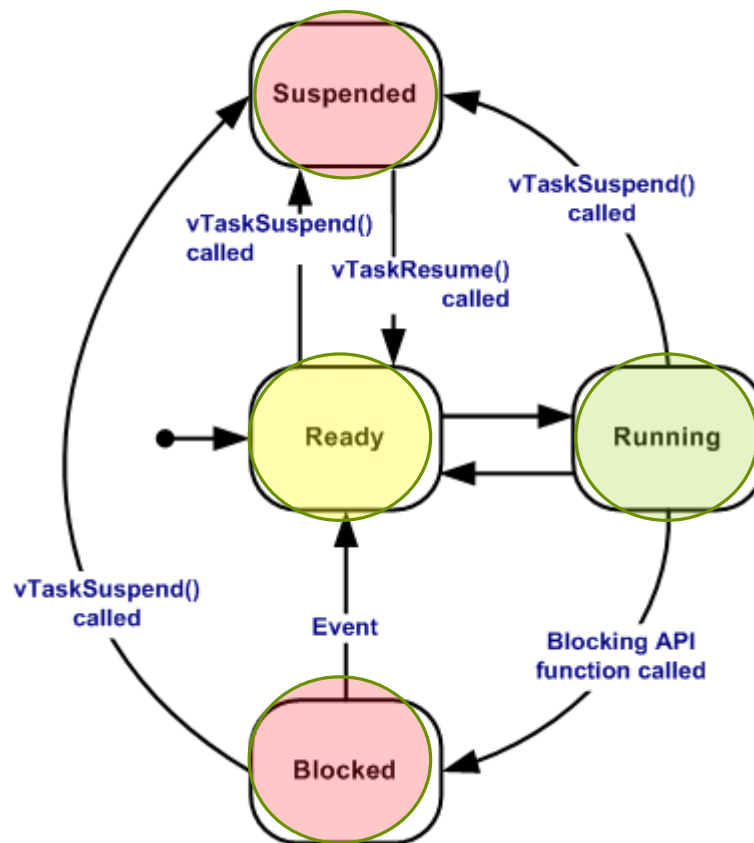
Estados Definidos RTOS

- Es deseable conocer como el 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 sincronizarse 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

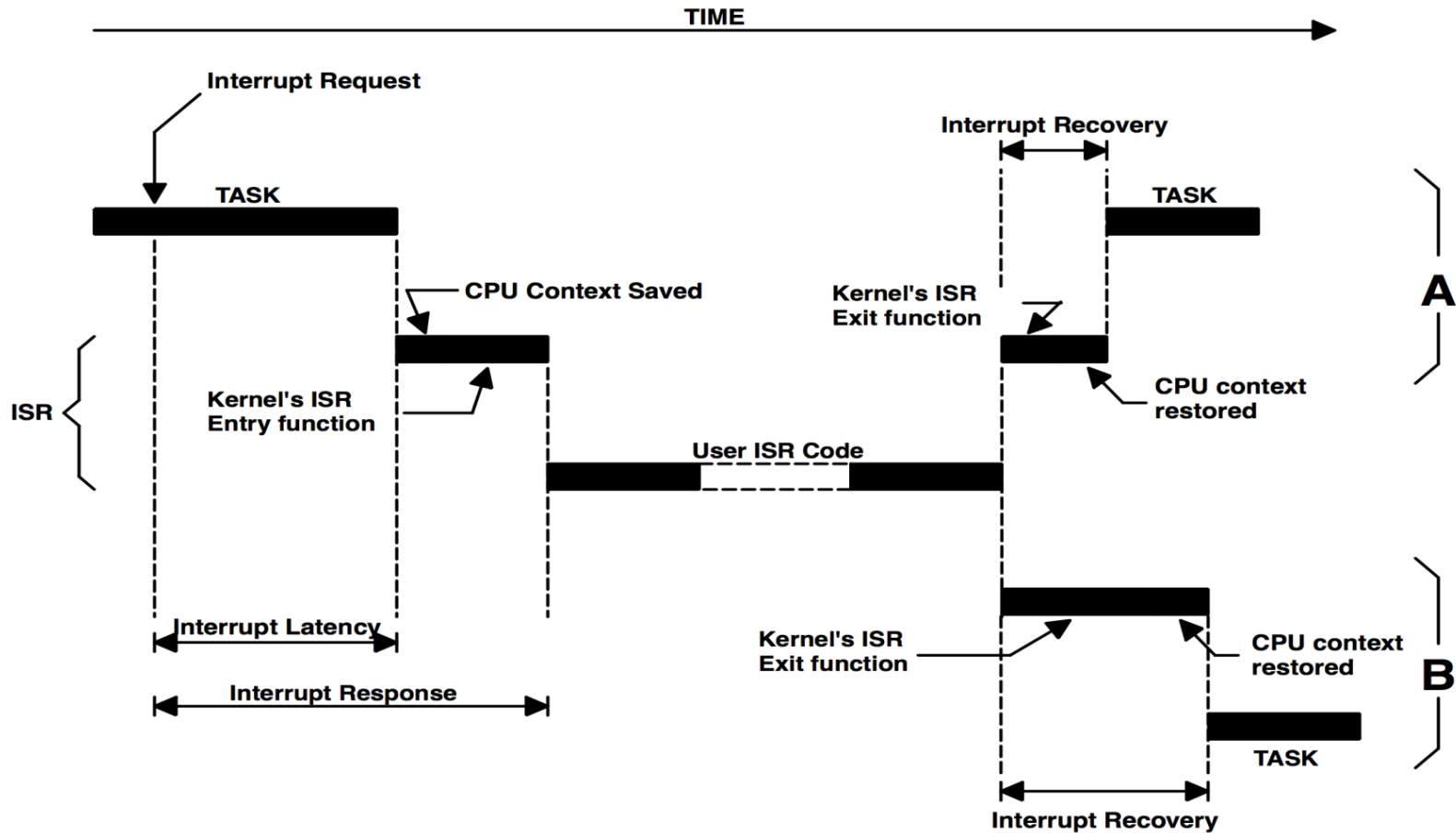
Prioridades

- 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 por 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

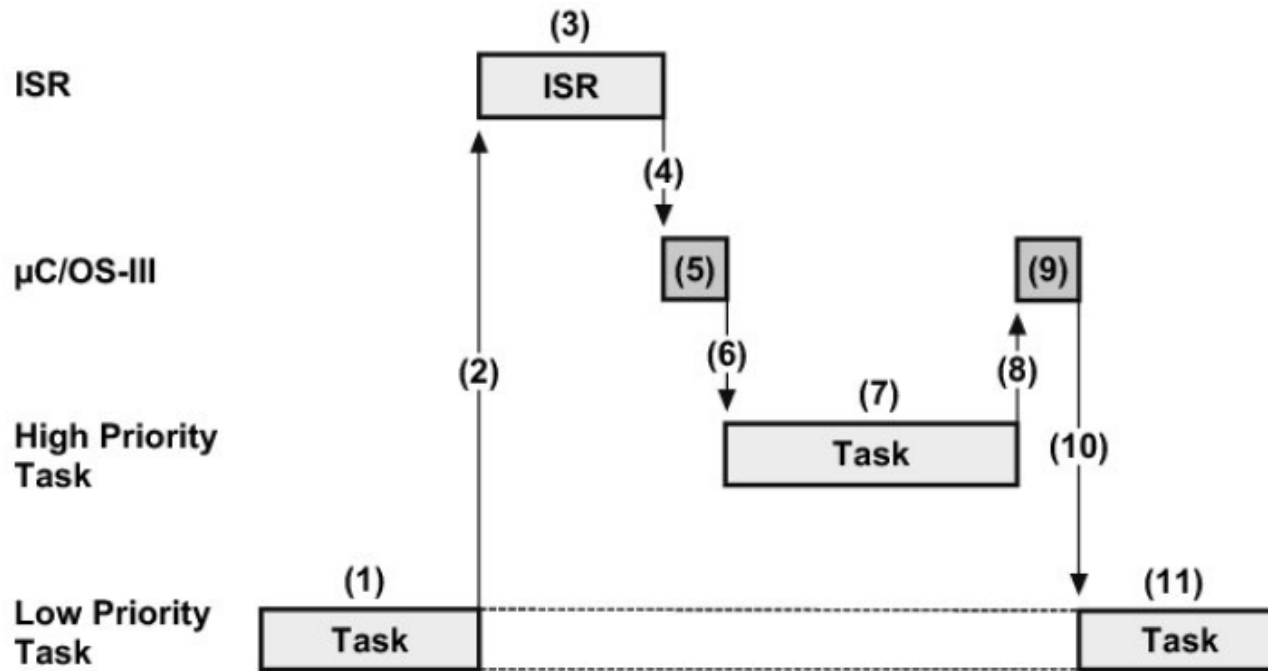


Figure - Preemptive scheduling

Round Robin Context Switch

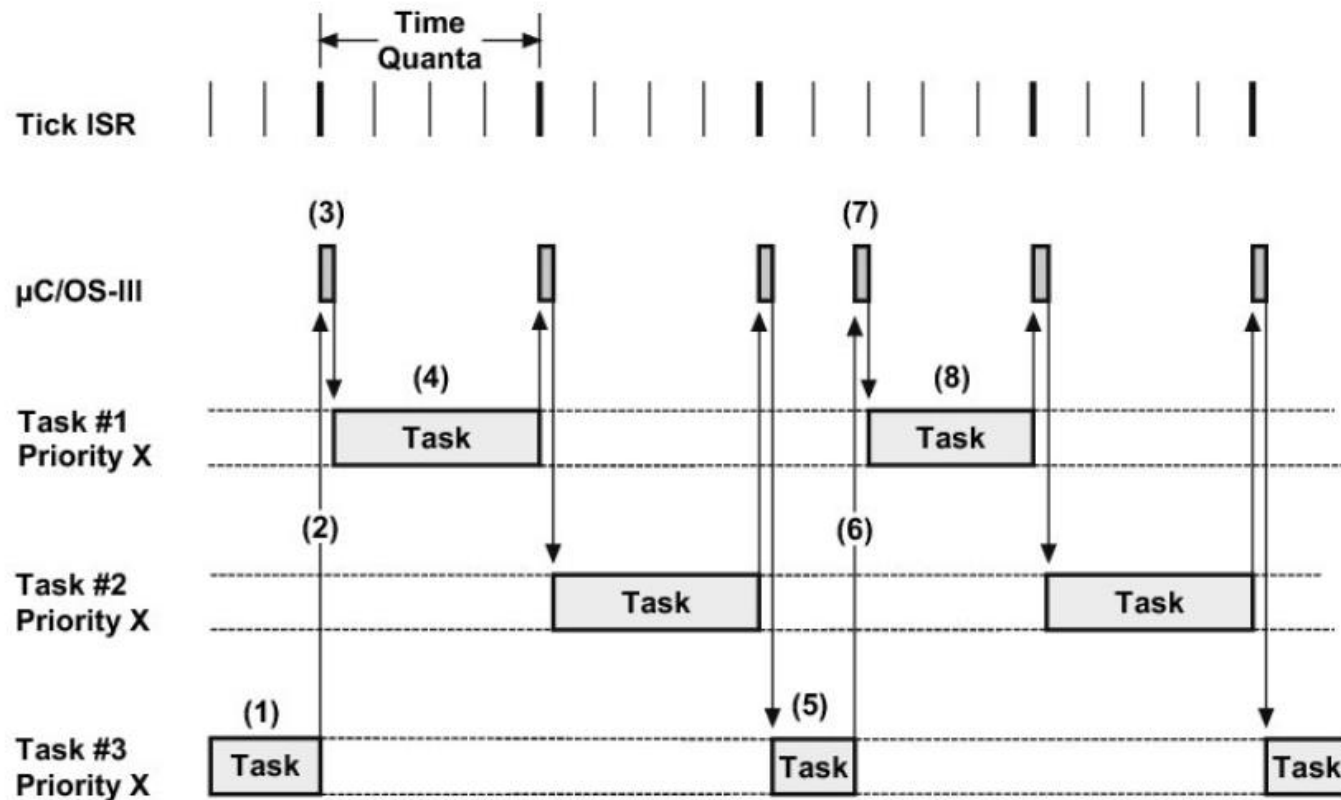
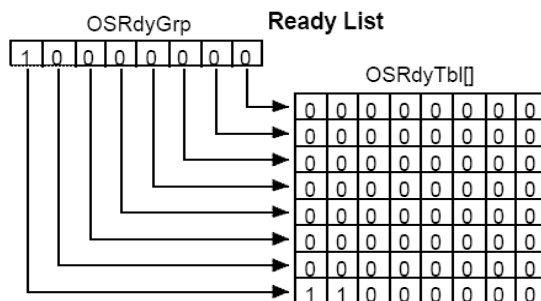


Figure - Round Robin Scheduling

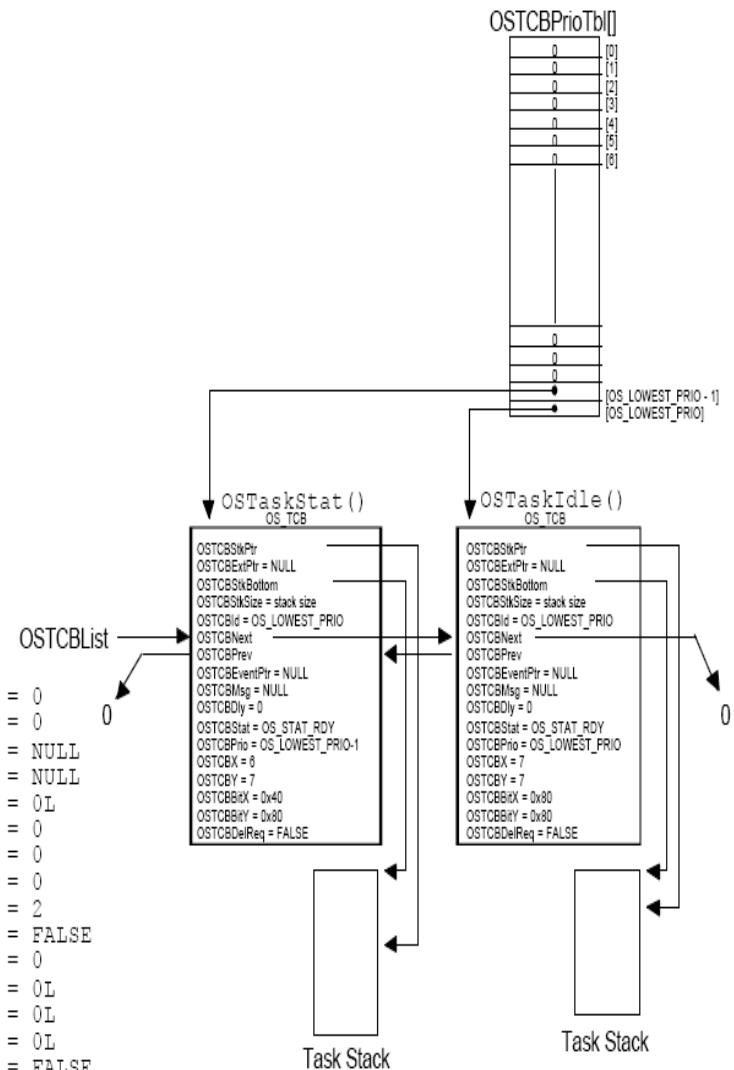
Inicialización del sistema

- 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



```

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
    
```



Inicialización del sistema

- Tareas Internas que son creadas según como se configure el proyecto
 - IDLE
 - Estadísticas
 - Ticks
 - Timers

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

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

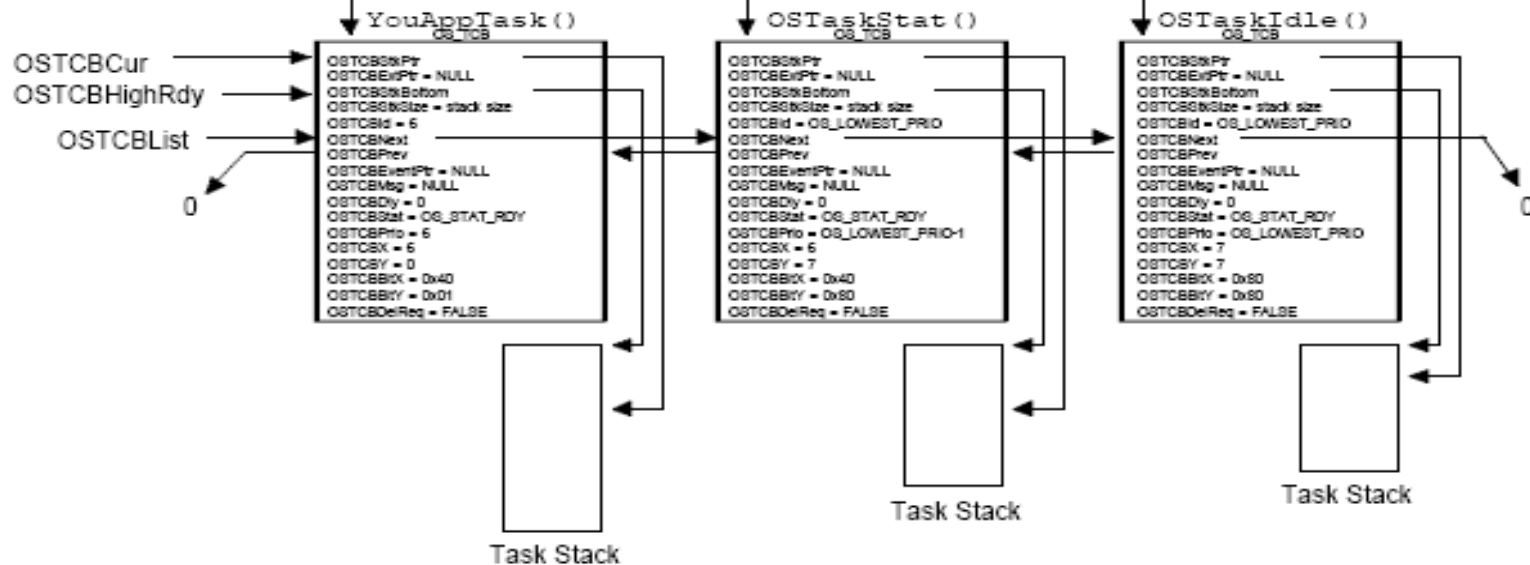
Ready List

OSRdyTbl[]

0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0

OSTCBPrioTbl[]

0	[0]
0	[1]
0	[2]
0	[3]
0	[4]
0	[5]
0	[6]
0	[7]
0	[8]
0	[9]
0	[10]
0	[11]
0	[12]
0	[13]
0	[14]
0	[15]
0	[16]
0	[17]
0	[18]
0	[19]
0	[20]
0	[21]
0	[22]
0	[23]
0	[24]
0	[25]
0	[26]
0	[27]
0	[28]
0	[29]
0	[30]
0	[31]
0	[32]
0	[33]
0	[34]
0	[35]
0	[36]
0	[37]
0	[38]
0	[39]
0	[40]
0	[41]
0	[42]
0	[43]
0	[44]
0	[45]
0	[46]
0	[47]
0	[48]
0	[49]
0	[50]
0	[51]
0	[52]
0	[53]
0	[54]
0	[55]
0	[56]
0	[57]
0	[58]
0	[59]
0	[60]
0	[61]
0	[62]
0	[63]
0	[64]
0	[65]
0	[66]
0	[67]
0	[68]
0	[69]
0	[70]
0	[71]
0	[72]
0	[73]
0	[74]
0	[75]
0	[76]
0	[77]
0	[78]
0	[79]
0	[80]
0	[81]
0	[82]
0	[83]
0	[84]
0	[85]
0	[86]
0	[87]
0	[88]
0	[89]
0	[90]
0	[91]
0	[92]
0	[93]
0	[94]
0	[95]
0	[96]
0	[97]
0	[98]
0	[99]



Tarea de Ejemplo

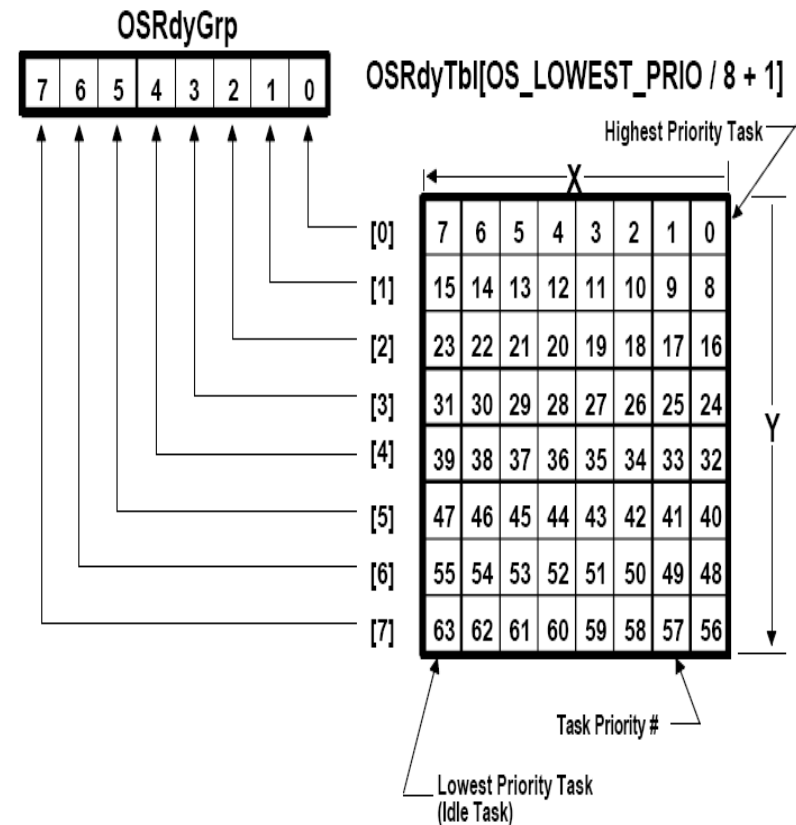
- Son bucles infinitos que no retornan ningún valor.
- Dentro de ellas se llaman a funciones que pueden o no retornar algún valor.

```
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[].



Tareas

Administración de Tareas

- Cada tarea o hilo podrá en su vida util:
 - Ser creada
 - Se debe Definir el Stack.
 - Ser Borrada
 - Cambiar su prioridad
 - Ser Suspendida
 - Ser Resumida

Creación de Tareas

- ◉ Debe estar definida en la configuración inicial la cantidad de Tareas
 - ◉ Esto hace la reserva inicial de TCBs.
- ◉ Se deben definir las prioridades.
- ◉ Se debe establecer el tamaño del STACK de cada Tarea.
- ◉ La cantidad de tareas que se pueden manejar depende del RTOS elegido.
 - ◉ El kernel usa como identificador de cada tarea su prioridad.

Creación de Tareas

- Funciones para Creación de Tareas
 - OSTaskCreate()
 - OSTaskCreateExt() “incorpora características adicionales”
- UNA tarea debe ser creada siempre antes de iniciar el proceso de multitarea, OSStart();
- No se debe crear una tarea dentro de la ISR.

OSTaskCreate()

- OSTaskCreate(task, pdata, ptos, prio)
- Requiere 4 argumentos
 - Task → Puntero a función
 - Pdata → Puntero que permite pasar un dato cuando la tarea esta corriendo
 - Ptos → puntero al inicio del stack asignado para esta tarea.
 - Prio → prioridad asignada a la tarea

RETORNO

OS_NO_ERR : tarea creada

OS_PRIO_EXIT : prioridad ya asignada

OS_PRIO_INVALID : prioridad invalida (\geq OS_LOWEST_PRIO)

OSTaskCreate()

- Secuencia Temporal del proceso de creación de una tarea
- Se verifica que **prio** este dentro de los valores definidos
 - \geq OS_LOWEST_PRIO “**RETORNA error**”
 - #define OS_LOWEST_PRIO 12
- Se verifica la prioridad no este asignada a otra tarea antes creada.
 - Reserva la prioridad a esta tarea

OSTaskCreate()

- Se llama a la función que inicializa el stack asociado a esta tarea.
- La función retorna el puntero al inicio del stack.
 - SE DEBE conocer como el procesador maneja la memoria de stack.
 - Manejo de memoria de forma creciente o decreciente.
- Se llama a la función que inicializa el TCB asociado a la tarea
 - Si no se produce error se incrementa el numero de tareas creadas
 - Se invoca al **Scheduler** para verificar que tarea es de mayor prioridad.
- Si se produce error al inicializar el stack
 - Se libera la prioridad para un futuro llamado.

```

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

```

OSTCBList

0

OSTaskStat()
OS TCB

```

OSTCBSStkPtr
OSTCBExtPtr = NULL
OSTCBSStkBottom
OSTCBSStkSize = stack size
OSTCBId = OS_LOWEST_PRIO
OSTCBNext
OSTCBPrev
OSTCBEvtPtr = NULL
OSTCBMsg = NULL
OSTCBDly = 0
OSTCBStat = OS_STAT_RDY
OSTCBPrio = OS_LOWEST_PRIO-1
OSTCBX = 6
OSTCBY = 7
OSTCBBitX = 0x40
OSTCBBitY = 0x80
OSTCBDelReq = FALSE

```

Task Stack

OSTCBPrioTbl[]

```

0 [0]
0 [1]
0 [2]
0 [3]
0 [4]
0 [5]
0 [6]

```

[OS_LOWEST_PRIO - 1]
[OS_LOWEST_PRIO]

OSTaskIdle()
OS TCB

```

OSTCBSStkPtr
OSTCBExtPtr = NULL
OSTCBSStkBottom
OSTCBSStkSize = stack size
OSTCBId = OS_LOWEST_PRIO
OSTCBNext
OSTCBPrev
OSTCBEvtPtr = NULL
OSTCBMsg = NULL
OSTCBDly = 0
OSTCBStat = OS_STAT_RDY
OSTCBPrio = OS_LOWEST_PRIO
OSTCBX = 7
OSTCBY = 7
OSTCBBitX = 0x80
OSTCBBitY = 0x80
OSTCBDelReq = FALSE

```

Task Stack

0

```

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

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

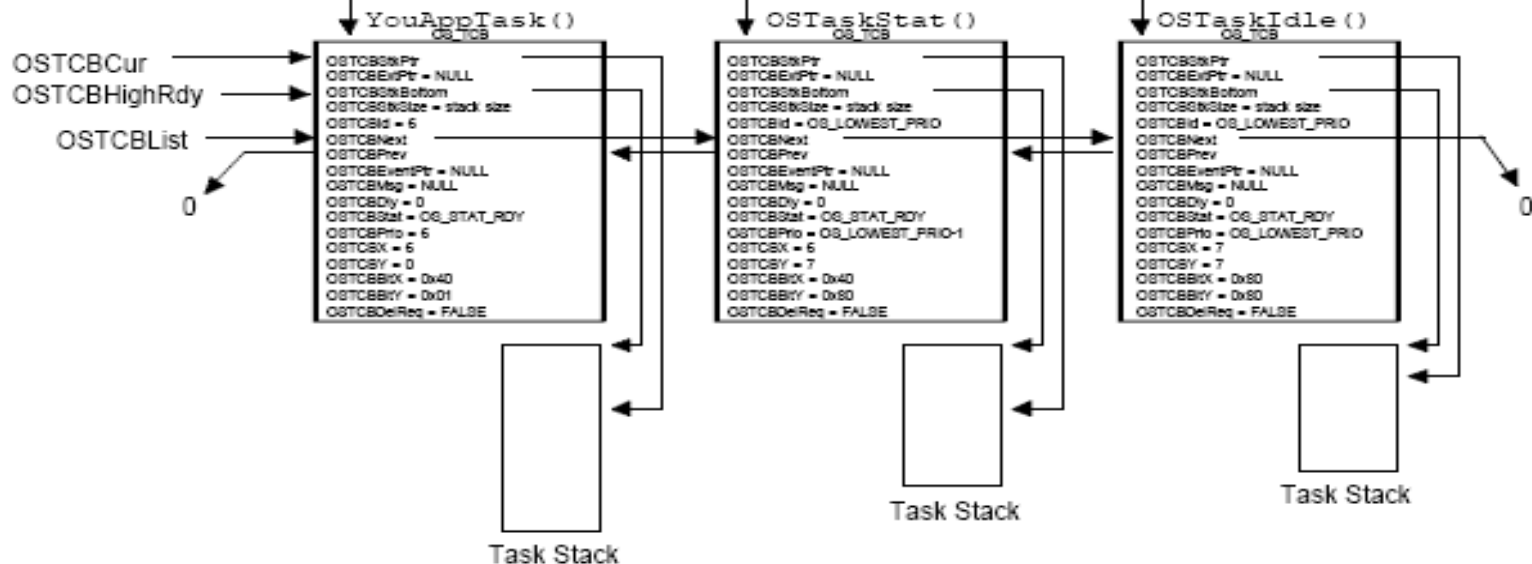
Ready List

OSRdyTbl[]

0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0

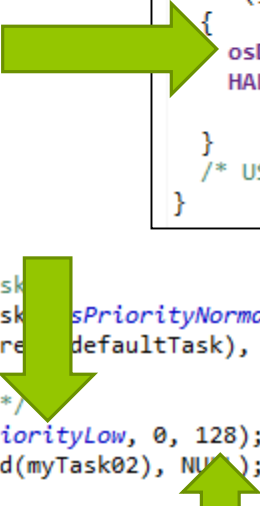
OSTCBPrioTbl[]

0	[0]
0	[1]
0	[2]
0	[3]
0	[4]
0	[5]
0	[6]
0	[7]
0	[8]
0	[9]
0	[10]
0	[11]
0	[12]
0	[13]
0	[14]
0	[15]
0	[16]
0	[17]
0	[18]
0	[19]
0	[20]
0	[21]
0	[22]
0	[23]
0	[24]
0	[25]
0	[26]
0	[27]
0	[28]
0	[29]
0	[30]
0	[31]
0	[32]
0	[33]
0	[34]
0	[35]
0	[36]
0	[37]
0	[38]
0	[39]
0	[40]
0	[41]
0	[42]
0	[43]
0	[44]
0	[45]
0	[46]
0	[47]
0	[48]
0	[49]
0	[50]
0	[51]
0	[52]
0	[53]
0	[54]
0	[55]
0	[56]
0	[57]
0	[58]
0	[59]
0	[60]
0	[61]
0	[62]
0	[63]
0	[64]
0	[65]
0	[66]
0	[67]
0	[68]
0	[69]
0	[70]
0	[71]
0	[72]
0	[73]
0	[74]
0	[75]
0	[76]
0	[77]
0	[78]
0	[79]
0	[80]
0	[81]
0	[82]
0	[83]
0	[84]
0	[85]
0	[86]
0	[87]
0	[88]
0	[89]
0	[90]
0	[91]
0	[92]
0	[93]
0	[94]
0	[95]
0	[96]
0	[97]
0	[98]
0	[99]



Creación de Tareas

```
...
28 /* definition and creation of myBinarySem02 */
29 osSemaphoreDef(myBinarySem02);
30 myBinarySem02Handle = osSemaphoreCreate(osSemaphoreDef(myBinarySem02), 1);
31
32 /* USER CODE BEGIN RTOS_SEMAPHORES */
33 /* add semaphores, ... */
34 /* USER CODE END RTOS_SEMAPHORES */
35
36 /* USER CODE BEGIN RTOS_TIMERS */
37 /* start timers, add new ones, ... */
38 /* USER CODE END RTOS_TIMERS */
39
40 /* Create the thread(s) */
41 /* definition and creation of defaultTask */
42 osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
43 defaultTaskHandle = osThreadCreate(osThreadDef(defaultTask), NULL);
44
45 /* definition and creation of myTask02 */
46 osThreadDef(myTask02, StartTask02, osPriorityLow, 0, 128);
47 myTask02Handle = osThreadCreate(osThreadDef(myTask02), NULL);
48
49 /* definition and creation of myTask03 */
50 osThreadDef(myTask03, StartTask03, osPriorityBelowNormal, 0, 128);
51 myTask03Handle = osThreadCreate(osThreadDef(myTask03), NULL);
52
```



```
/* 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);  
    }  
}
```



Borrado de Tareas

- Se envía la tarea al estado DORMANT.
- No se puede borrar la tarea IDLE.
- No se puede eliminar una tarea desde la ISR
- En el proceso de borrado el kernel:
 - Verifica que no se intente borrar la tarea IDLE.
 - Verifica que la prioridad esté dentro del intervalo definido previamente.
 - Para borrarse a si misma la tarea se debe pasar como argumento `OS_PRIO_SELF`.
- El proceso de borrado se divide en dos partes para disminuir el tiempo en que no se atiende una posible interrupción.
- Si esta READY se elimina

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 los 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 de Diseño

Ejemplo de Diseño

- Implementar el Control de un Tablero Electrónico indicador multideportes, basado en un sistema embebido con RTOS.
- Características Generales:
 - Indicador de tiempo
 - Minutos
 - Segundos
 - Décimas de Segundo
 - Indicador de Puntuación
 - 3 dígitos – Local y Visitante
 - Indicador de Faltas
 - 1 Dígito – Local y Visitante
 - Consola de Comando con repetición de la información del tablero
 - Conexión Wireless con el tablero.

