Internal architecture of uC/OS-II

B. Miramond

Polytech Nice Sophia Antipolis

Outline

- Partie 1 : Scheduler
- Partie 2 : Synchronisation
- Partie 3 : Hook functions

Scheduling services in uC/OS-II

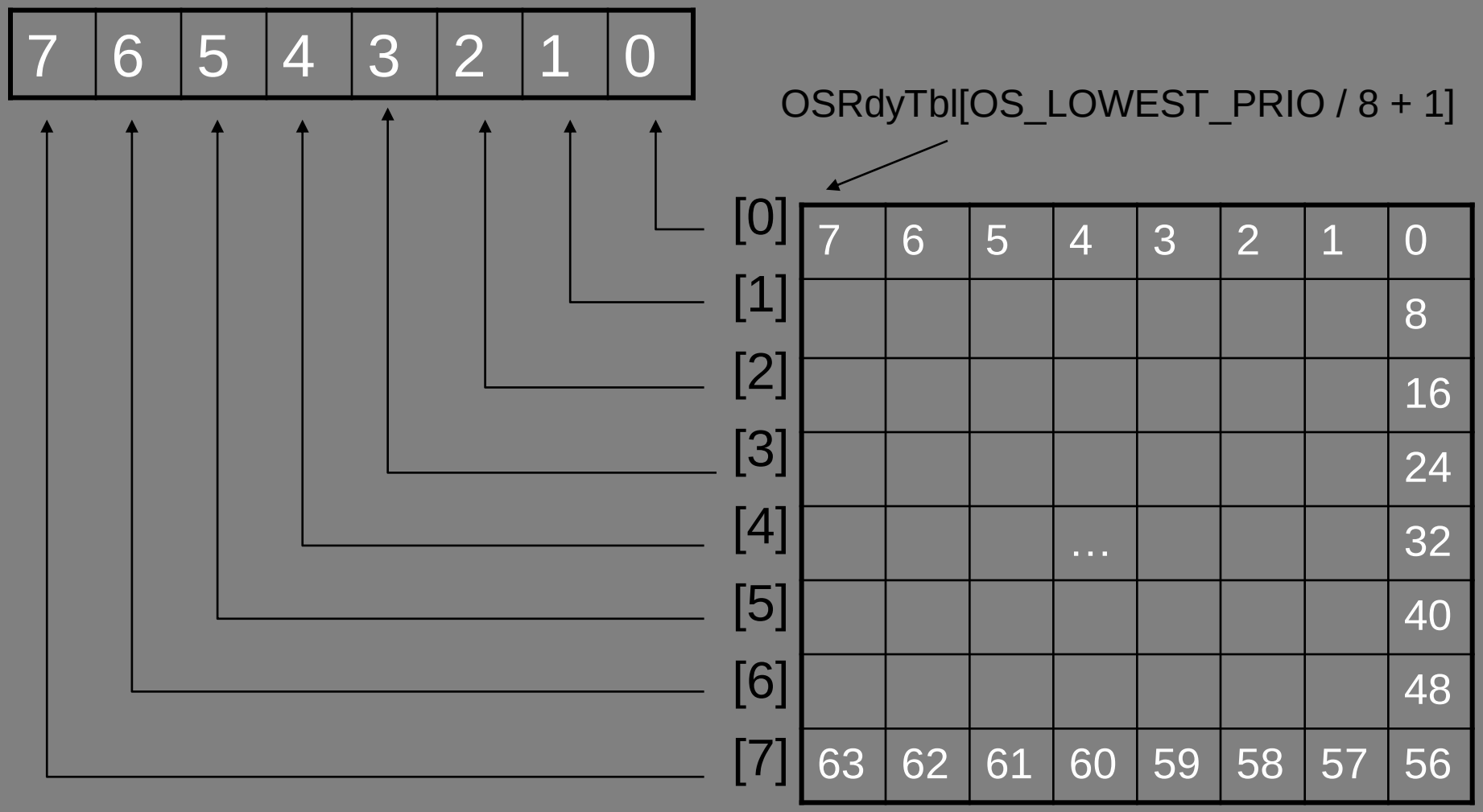
- Tasks list
  - List of Ready tasks
  - List of free TCB
- Scheduler
- Task level context switch
- Idle Task
- Statistics Task

Ready List

- It is a list of 2 variables:
  - *#define OS_RDY_TBL_SIZE ((OS_LOWEST_PRIO) / 8 + 1)*
    // Size of ready table
  - *OS_EXT INT8U OSRdyGrp; /* Ready list group */*
  - *OS_EXT INT8U OSRdyTbl[OS_RDY_TBL_SIZE];*
    / Table of tasks which are ready to run */*
- The tasks are grouped by priority (8 tasks per group).
- Each bit Gi of the OSRdyGrp byte indicates whether there is a READY task in group i.
- The OSRdyTbl[] byte array indicates which task(s) in the group is/are READY.
- The priorities are sorted in ascending order, so OSRdyTbl[0] is the highest priority task group and (OSRdyTbl[0] & 0x01) is the mask of the highest priority task of the group.
- Thus, 64 tasks can be placed waiting for an event in order of priority in this two-dimensional table.

Relation between OSRdyGrp and OSRdyTbl

INT8U OSRdyGrp



Summary

- Bit 0 of OSRdyGrp is at 1 if one of the bits of OSRdyTbl [0] is at 1,
- Bit 1 of OSRdyGrp is at 1 if one of the bits of OSRdyTbl [1] is at 1, …

Placement of a pending task

Ti(prio)

- Placing a task in the list :
  - Set the Gi bit of OSRdyGrp corresponding to prio to 1 (Rdy task)
  - Gi = prio / 8 + 1
  - Set the bit corresponding to prio in OSRdyTbl[prio / 8 +1]

Corresponding code

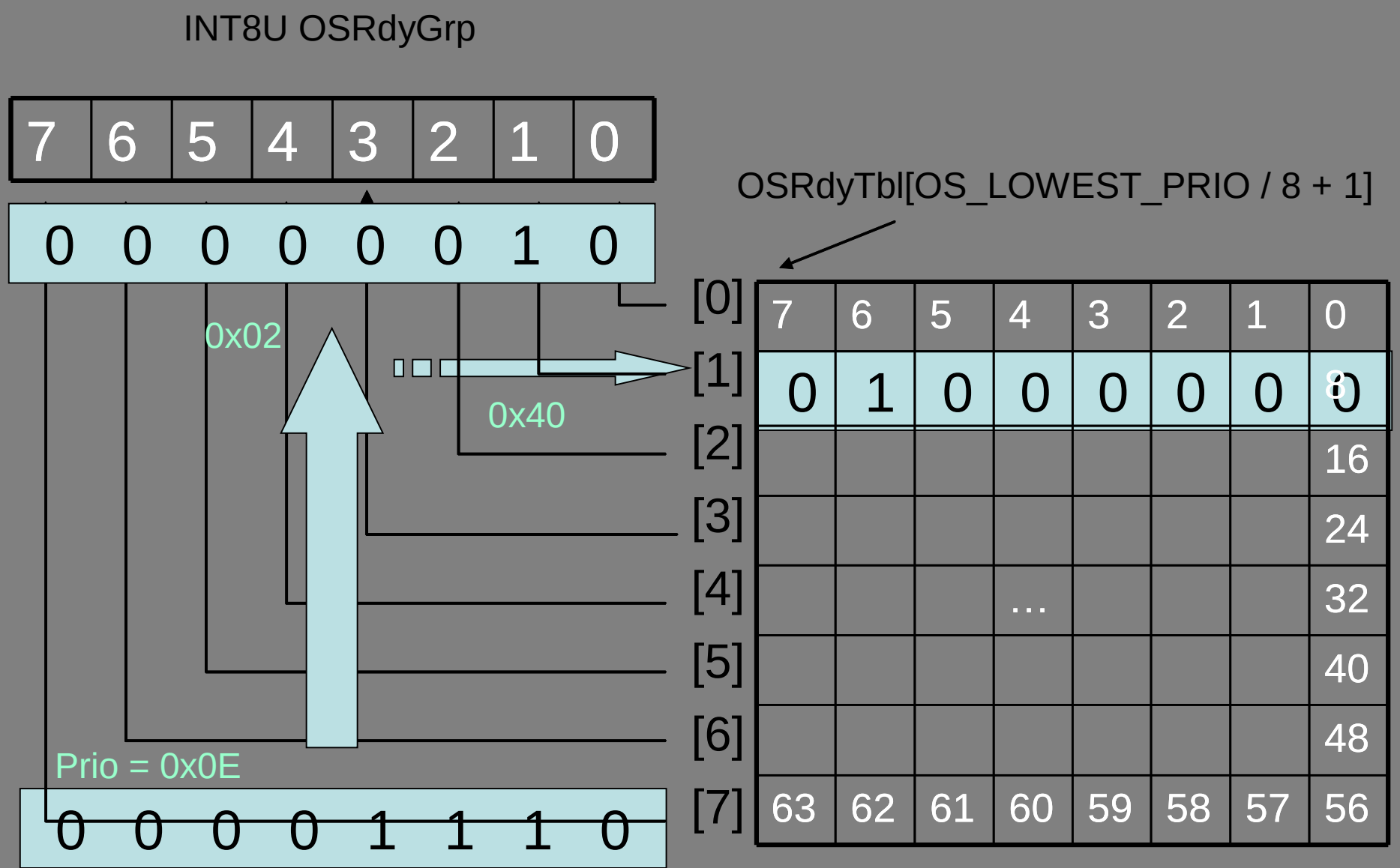
- *INT8U const OSMapTbl[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80}; // Priority->Bit Mask lookup table*

- *OSRdyGrp |= OSMapTbl[prio >> 3];*
- *OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];*

- *Exemple : Insertion de T(14)*
- *Prio = 0000 1110 = 0x 0E*
- *Prio >> 3 = 0000 0001 = 0x01 => OSRdyTbl[1]*
- *OSMapTbl[prio >> 3] = 0x02*
- *Donc OSRdyGrp |= 0000 0010, 2e bit à 1*

- *0000 1110 = prio*
- *0000 0111 = 0x07*
- *0000 0110 = prio & 0x07 = 6*
- *OSMapTbl[6] = 2^6 = 0x40*

Insertion of a task



Deleting a task

```
if(
   (OSRdyTbl[prio>> 3] &= ~OSMapTbl[prio& 0x07])
  == 0)
   OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

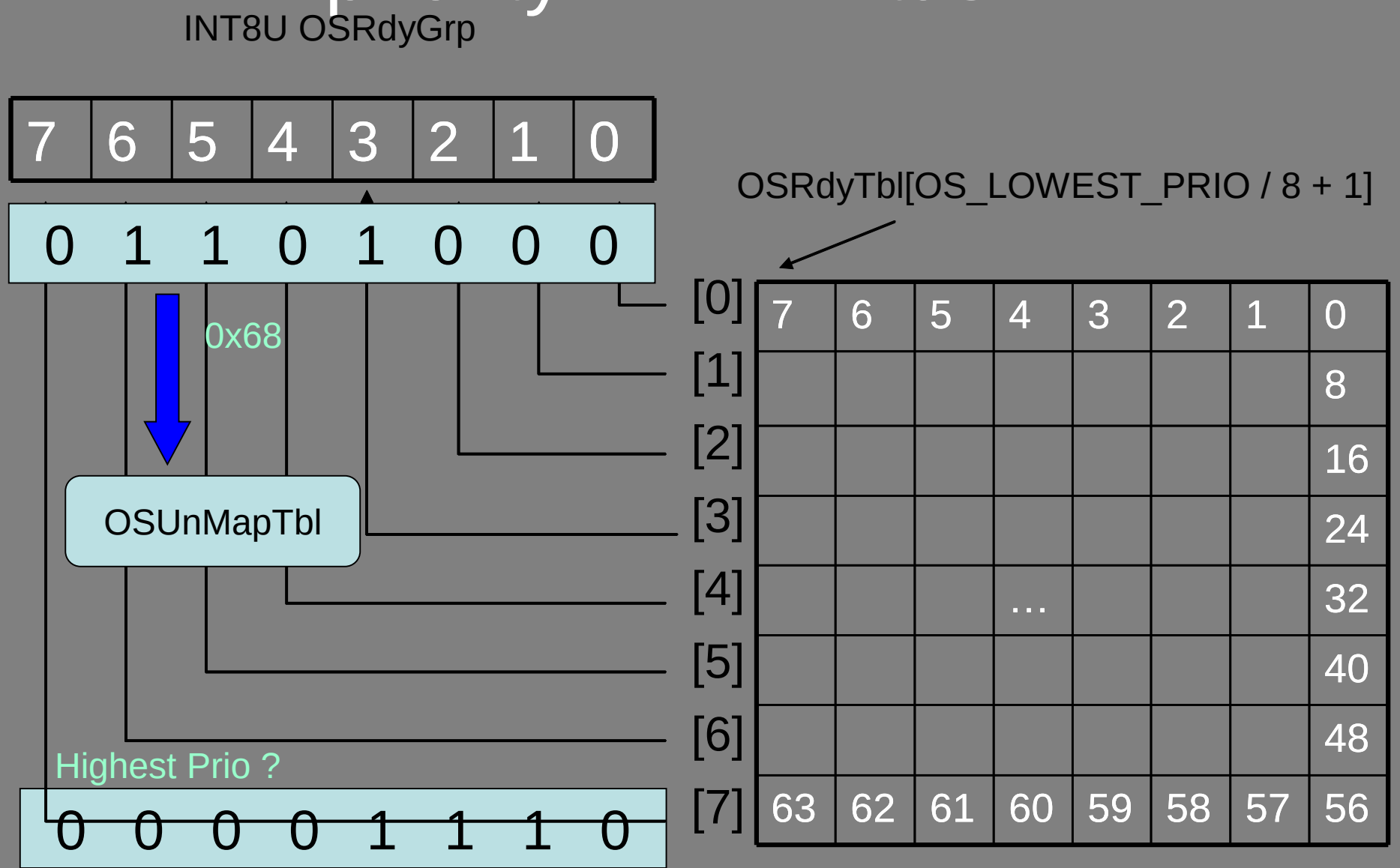
The bit of the group is set to zero if the whole row of the table is empty.

Finding the highest priority task in state READY

This is the reverse process:

1. Find the highest priority line from the group number
2. Find the least significant bit in the row = highest priority = Task Id

Example: what is the highest priority READY task?



OSUnMapTbl

OSRdyGrp = 0x68

```
INT8U  const  OSUnMapTbl[] = {
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F                    */
    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0
    6, 0, 1, 0, 2, 0, 1, 0
    4, 0, 1, 0, 2, 0
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0
};
```

3 = OSUnMapTbl[0x68];

Principe of the table = **Look Up Table**

Give the least significant bit (highest prio) for each value :

0x10 = 0001 0000 => bit 4

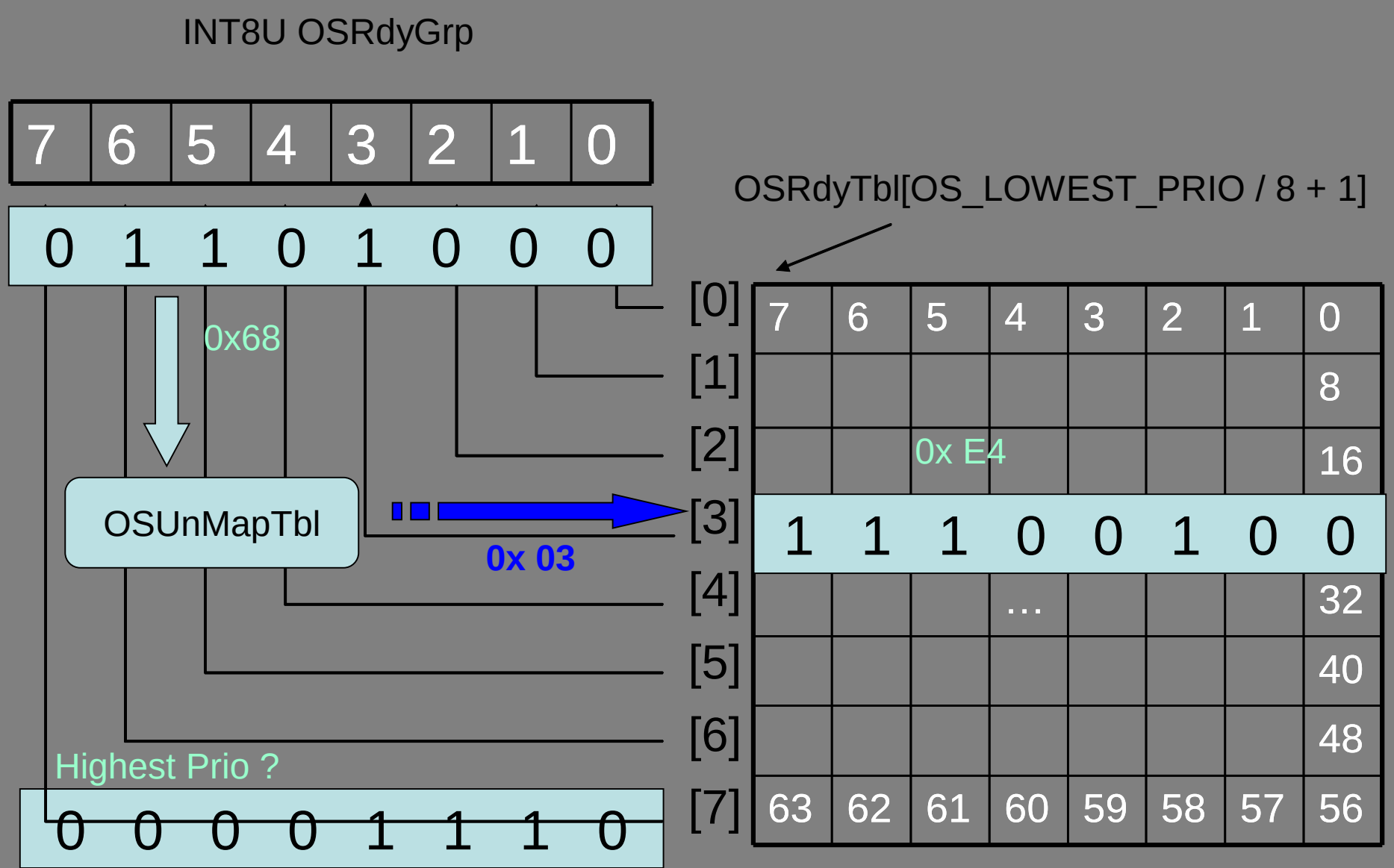
0x 68 = 0110 1000 => bit 3

D4 = 1101 0100 => bit 2 …

It is also the line number in OSRdyTbl!

All the columns are identical except the 1st one

Example


INT8U OSRdyGrp
7 6 5 4 3 2 1 0
0 1 1 0 1 0 0 0
0x68
OSUnMapTbl
0x 03
OSRdyTbl[OS_LOWEST_PRIO / 8 + 1]
[0] 7 6 5 4 3 2 1 0
[1] 8
[2] 16
0x E4
[3] 1 1 1 0 0 1 0 0
[4] ... 32
[5] 40
[6] 48
[7] 63 62 61 60 59 58 57 56
Highest Prio ?
0 0 0 0 1 1 1 0

OSUnMapTbl

OSRdyGrp = 0x68

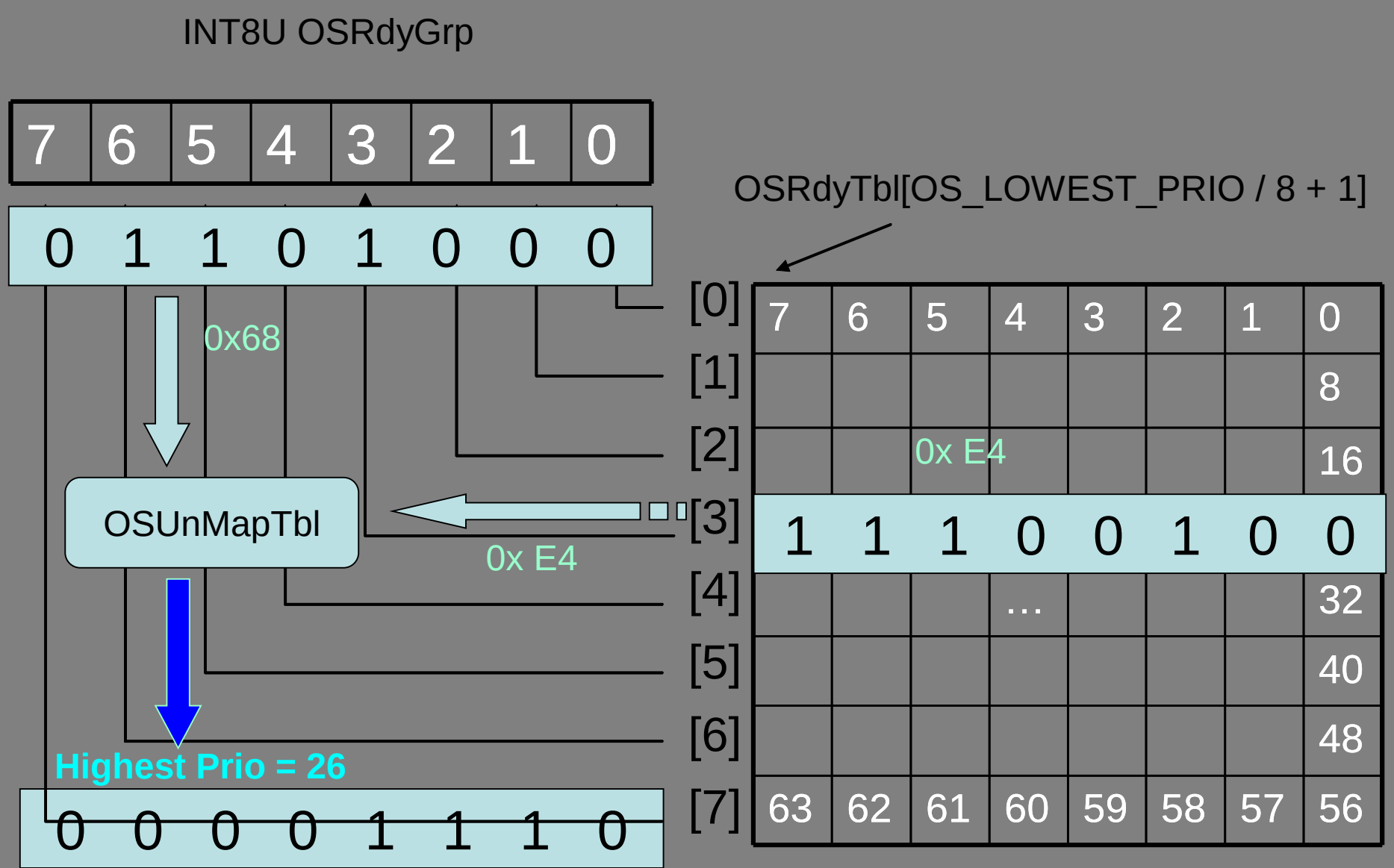
```
INT8U  const  OSUnMapTbl[] = {
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F                    */
    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x90 to 0x9F                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xA0 to 0xAF                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xB0 to 0xBF                    */
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xC0 to 0xCF                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xD0 to 0xDF                    */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xE0 to 0xEF                    */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
};
```

3 = OSUnMapTbl[0x68];

2 = OSUnMapTbl[E4];

26 = (3 << 3) + 2;

Example


INT8U OSRdyGrp
7 6 5 4 3 2 1 0
0 1 1 0 1 0 0 0
0x68
OSUnMapTbl
0x E4
Highest Prio = 26
0 0 0 0 1 1 1 0
OSRdyTbl[OS_LOWEST_PRIO / 8 + 1]
[0] 7 6 5 4 3 2 1 0
[1] 8
[2] 0x E4 16
[3] 1 1 1 0 0 1 0 0
[4] … 32
[5] 40
[6] 48
[7] 63 62 61 60 59 58 57 56

Related Code

```
y = OSUnMapTbl[OSRdyGrp];
// determine y position in OSRdyTbl
x = OSUnMapTbl[OSRdyTbl[y]];
// determine x position in OSRdyTbl[y]
prio = (y << 3) + x;
```

We thus obtain the index to obtain the corresponding TCB in OSTCBPrioTbl[] !

uC/OS scheduler

- uC/OS-II always executes the highest priority task in the READY state.
- The function is OS_Sched()
- The time of this function is constant whatever the number of tasks created!
- **Determinism and Predictability**

OS_Sched()

```
void  OS_Sched (void)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr;
#endif
    INT8U      y;


    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0) {                                 /* Schedule only if all ISRs done and ...       */
        if (OSLockNesting == 0) {                            /* ... scheduler is not locked                  */
            y             = OSUnMapTbl[OSRdyGrp];            /* Get pointer to HPT ready to run              */
            OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
            if (OSPrioHighRdy != OSPrioCur) {                 /* No Ctx Sw if current task is highest rdy     */
                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
#if OS_TASK_PROFILE_EN > 0
                OSTCBHighRdy->OSTCBCtxSwCtr++;               /* Inc. # of context switches to this task      */
#endif
                OSCtxSwCtr++;                                /* Increment context switch counter             */
                OS_TASK_SW();                                /* Perform a context switch                     */
            }
        }
    }
    OS_EXIT_CRITICAL();
}
```

OS_TASK_SW()

- It is a macro that calls a microprocessor interrupt (software interrupt)
- Before the call, 2 variables must be placed:
  - OSTCBCur, the TCB pointer of the task to be interrupted
  - And OSTCBHighRdy, the TCB pointer of the task to execute

Idle Task

- OS_LOWEST_PRIO
- This task is directly used to calculate the CPU utilization rate.
- Its code is therefore :

```
void  OS_TaskIdle (void *pdata)
{
#if OS_CRITICAL_METHOD == 3 //Allocate storage for CPU status register
    OS_CPU_SR  cpu_sr;
#endif


    pdata = pdata;          /* Prevent compiler warning for not using 'pdata'     */
    for (;;) {
        OS_ENTER_CRITICAL();
        OSIdleCtr++;
        OS_EXIT_CRITICAL();
        OSTaskIdleHook();                   /* Call user definable HOOK              */
    }
}
```

Statistic Task

- Only created if FLAG OS_TASK_STAT_EN is set to 1 (OS_CFG.H).
- It runs every second and calculates the percentage of use of CPU: INT8U OSCPUUsage.
- To use it, you have to call the OSStatInit function before creating user tasks !

Computing the CPU use rate

- Every second TaskStat reads the counter value incremented by IdleTask.
- It then resets this counter to zero.

$$\tau_{CPU} = 100 \times \left(1 - \frac{OSIdleCtr}{OSIdleCtrMax} \right)$$

- Where OSIdleCtrMax is the maximum number of increments per IdleTask when no other task is running.

Dynamic priorities

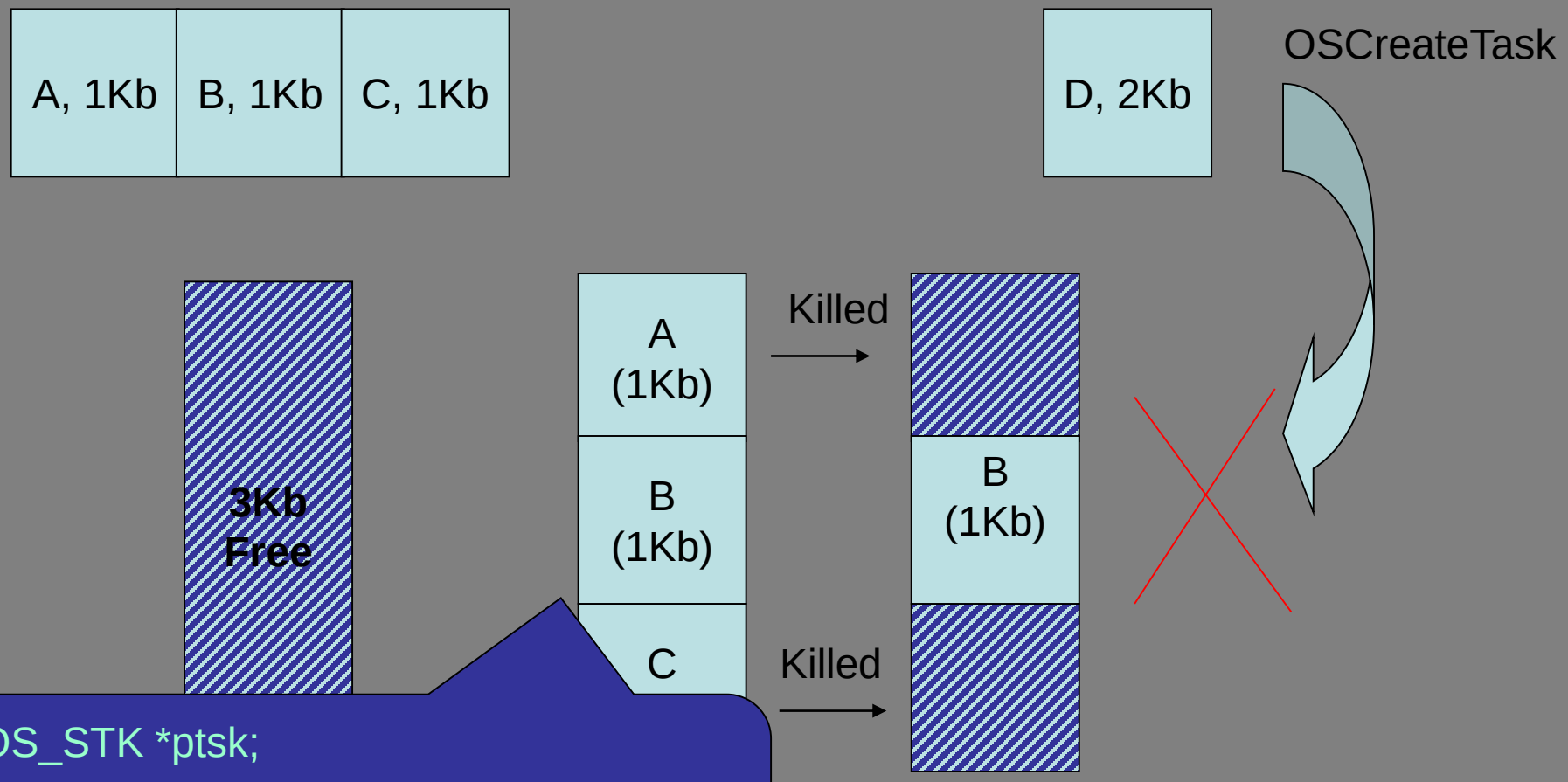
- As with the static priorities, uC/OS-II does not itself set dynamic priorities.
- But the user can call the following function OSTaskChangePrio() to avoid inversion of priorities in the case of resource sharing (cf. course on scheduling algorithms).

Task Stacks

OS_STK

- Continues memory spaces !
- Who can be allocated
  - Statically, or
  - Dynamically.
- In the last case, one must be wary of the fragmentation caused by the malloc function in the heap area (heap).
- Dynamic heap allocation is therefore only used when tasks are not destroyed.
- In practice is is preconized to avoid malloc function

Example of dynamic allocation




```
OS_STK *ptsk;

ptsk = (OS_STK *)malloc(stack_size);
If (ptsk != (OS_STK *)0)
        // Create the task
```

The creation task functions: OSTaskCreate() and OSTaskCreateExt()

- Rule 1: Tasks must be created before launching the OSStart() function that launches the multi-tasking management mechanism.
- Rule 2: Ext is an extended version used only for code analysis.
- The first one has 4 arguments, the second one 9.

Arguments de la fonction de création

- void (*task)(void*pd): un pointeur sur le code de la tâche
- Void * pdata : un pointeur sur une donnée passée en paramètre à la tâche à sa création
- OS_STK *ptos : pointeur sur le haut de pile de la tâche
- INT8U prio : La priorité désirée de la tâche (limitations à 64 possibilités)
- INT16U id : Un identifieur unique de la tâche (extension par rapport à la limitation 64, sinon id = prio)
- OS_STK *pbos : Bottom-of-stack
- INT32U stk_size : taille de la pile (utilisée pour le stack checking). Cf. Lab 2.
- void *pext : pointeur sur une donnée utilisateur pour étendre le TCB. Cf. Lab 2.
- INT16U opt : uCOS_ii.h contient la liste des options possibles (OS_TASK_OPT_STK_CHK, OS_TASK_OPT_TSK_CLR, OS_TASK_OPT_SAVE_FP…). Chaque constante est un FLAG binaire.

Rappel, PCB ou TCB

Identificateur processus
Etat courant du processus
Contexte processeur
Contexte mémoire
Ressources utilisées
Ordonnancement
Informations de comptabilisation

Fichier µCOS_II.h : TCB (1)

OS_STK = INT16U => affichage *2 en octets

```
*********************************************************************************************
*********************************************************************************************
*/

typedef struct os_tcb {
    OS_STK          *OSTCBStkPtr;        /* Pointer to current top of stack                    */

#if OS_TASK_CREATE_EXT_EN > 0
    void            *OSTCBExtPtr;        /* Pointer to user definable data for TCB extension  */
    OS_STK          *OSTCBStkBottom;     /* Pointer to bottom of stack                         */
    INT32U           OSTCBStkSize;       /* Size of task stack (in number of stack elements)   */
    INT16U           OSTCBOpt;           /* Task options as passed by OSTaskCreateExt()        */
    INT16U           OSTCBId;            /* Task ID (0..65535)                                 */
#endif

    struct os_tcb *OSTCBNext;            /* Pointer to next     TCB in the TCB list            */
    struct os_tcb *OSTCBPrev;            /* Pointer to previous TCB in the TCB list            */

#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0) || (OS_SEM_EN > 0) || (OS_MUTEX_EN > 0)
    OS_EVENT        *OSTCBEventPtr;      /* Pointer to event control block                     */
#endif
```

Voir exemple d'utilisation en TP

Utiliser pour la mesure dynamique de taille de pile

Non utilisé : ID = prio

Fichier µCOS_II.h : TCB (2)

```
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0)
    void          *OSTCBMsg;          /* Message received from OSMboxPost() or OSQPost()          */
#endif

#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (
#if OS_TASK_DEL_EN > 0
    OS_FLAG_NODE  *OSTCBFlagNode;     /* Pointer to event flag node                               */
#endif
    OS_FLAGS      OSTCBFlagsRdy;      /* Event flags that made task ready to run                  */
#endif

    INT16U        OSTCBDly;           /* Nbr ticks to delay task or, timeout waiting for event    */
    INT8U         OSTCBStat;          /* Task status                                              */
    INT8U         OSTCBPrio;          /* Task priority (0 == highest, 63 == lowest)               */

    INT8U         OSTCBX;             /* Bit position in group  corresponding to task priority (0..7) */
    INT8U         OSTCBY;             /* Index into ready table corresponding to task priority    */
    INT8U         OSTCBBitX;          /* Bit mask to access bit position in ready table           */
    INT8U         OSTCBBitY;          /* Bit mask to access bit position in ready group           */

#if OS_TASK_DEL_EN > 0
    BOOLEAN       OSTCBDelReq;        /* Indicates whether a task needs to delete itself          */
#endif
} OS_TCB;
```

Utilisé si l'option OSTCBOpt.OS_TASK_EN = 1

wait(timeout) ou wait(event, timeout)

Évite les calculs en-ligne, cf. chapitre suivant

OS_Init()

During this call, uC/OS-II initializes 5 data structures :

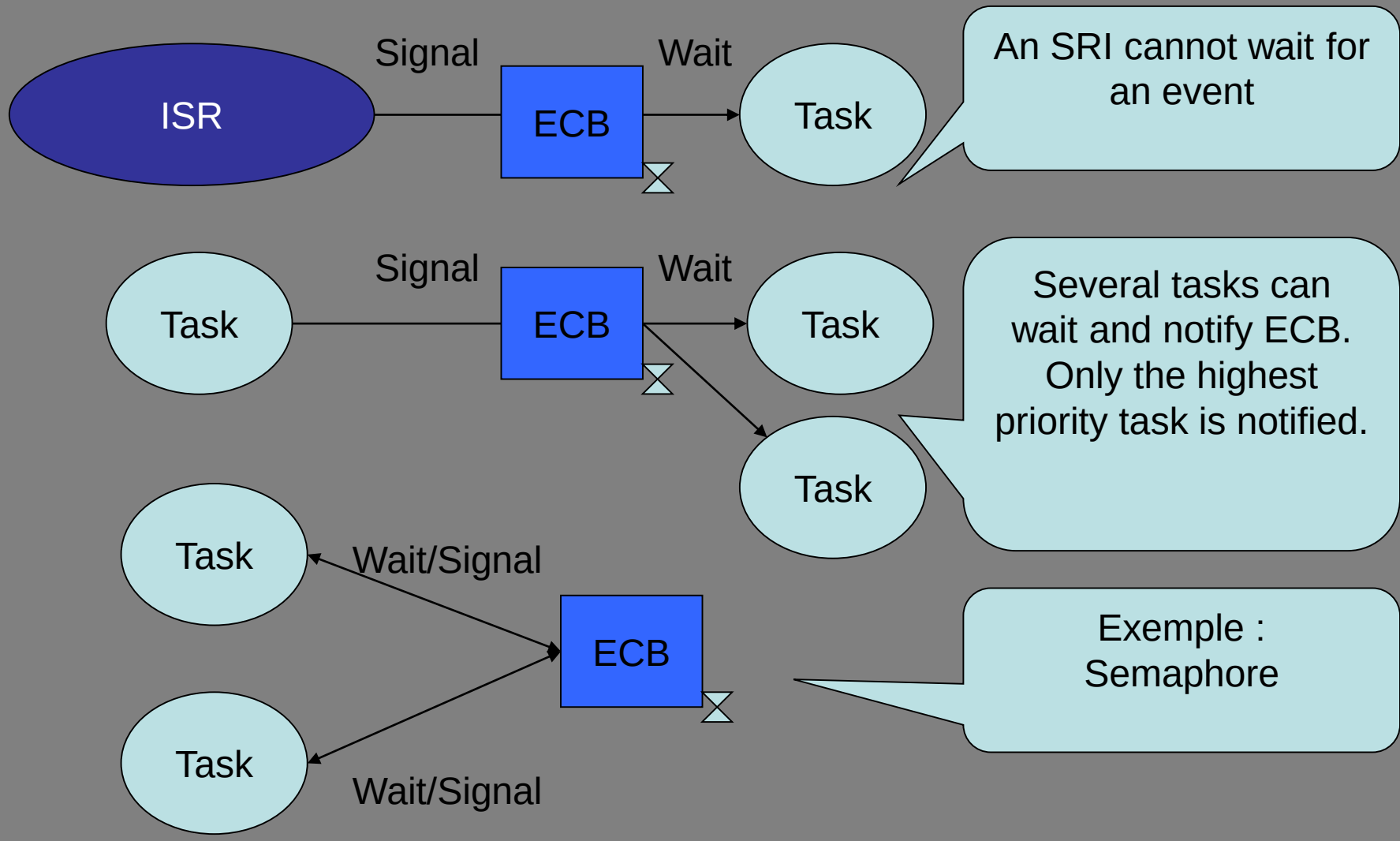
1. OSTCBFreeList for OS_TCB
2. OSEventFreeList for OS_EVENT
3. OSQFreeList for OS_Q
4. OSFlagFreeList for OS_FLAG_GRP
5. OSMemFreeList for OS_MEM

The length of these chained lists is specified in OS_CFG.H

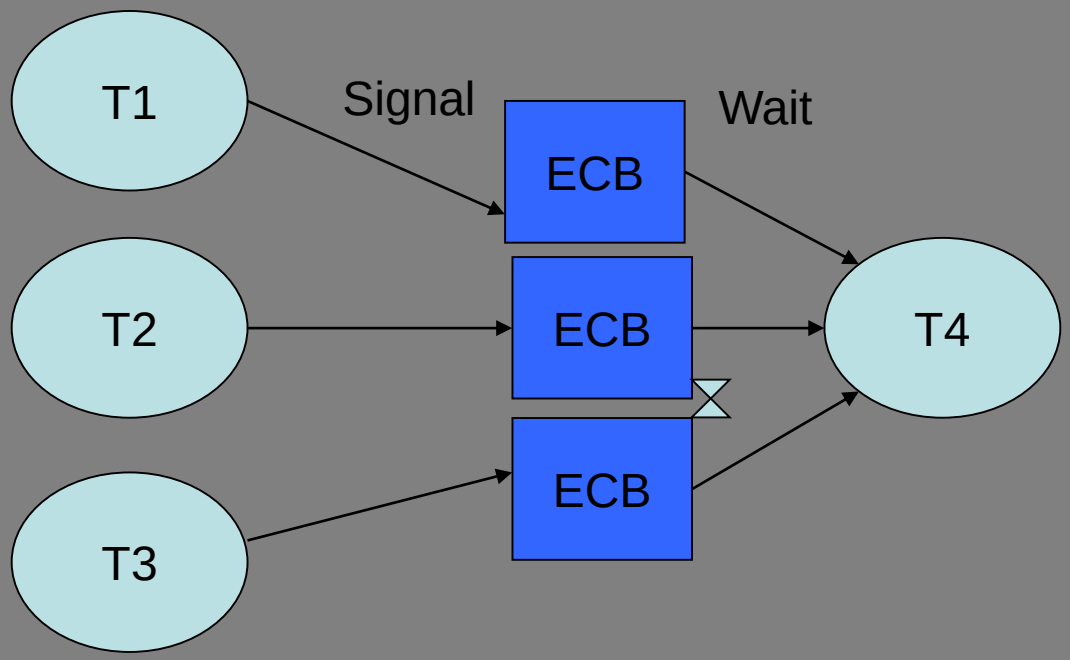
Synchronisation between tasks

- Event Control Block
- Type uC/OS-II ECB = OS_EVENT
- Liste of waiting tasks
- Electing one of the waiting tasks when the resource is released

Utilisation of ECB



Utilisation of ECB



ECB data Structure

```
typedef struct{
INT8U      OSEventType;
INT16U     OSEventCnt;    // for semaphores
void       *OSEventPtr;   // message
INT8U      OSEventGrp;
INT8U OSEventTbl[OS_EVENT_TBL_SIZE];
} OS_EVENT;
```

Wait List

- Each task waiting for an event is placed in the Waiting list.
- This list, like the Ready list, consists of 2 variables :
  - OSEventGrp
  - OSEvntTbl[]
- They are used as before
  - The group is used to indicate whether a task is pending in group Gi.
  - If a process is pending in this group of 8, the corresponding bit is set to 1 in the table

OS_EVENT

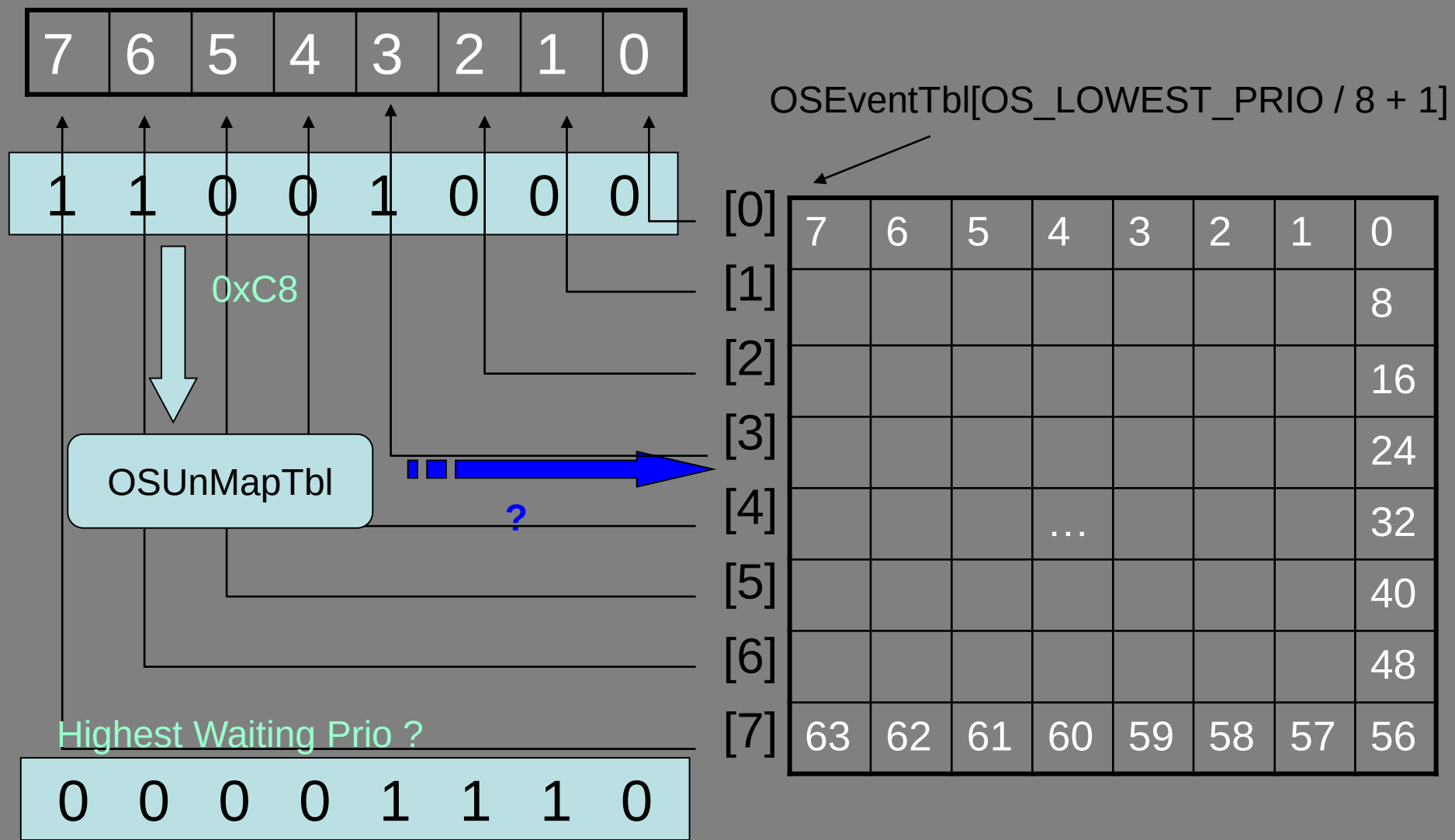
	.OSEventType							
	.OSEventCnt							
	.OSEventPtr							
	.OSEventGrp							
	7	6	5	4	3	2	1	0
.OSEventTbl[]								
	63	62	61	60	59	58	57	56

List management

- The same methods are used as for the RDY list for :
  - Placement of a pending task
  - Removing a task from the list
  - The search for the highest priority task (HPT) pending in a given ECB
- So there are as many Waiting lists as OS_EVENT declared !!!

Waiting list in an OS_EVENT

INT8U OSEventGrp



OSUnMapTbl

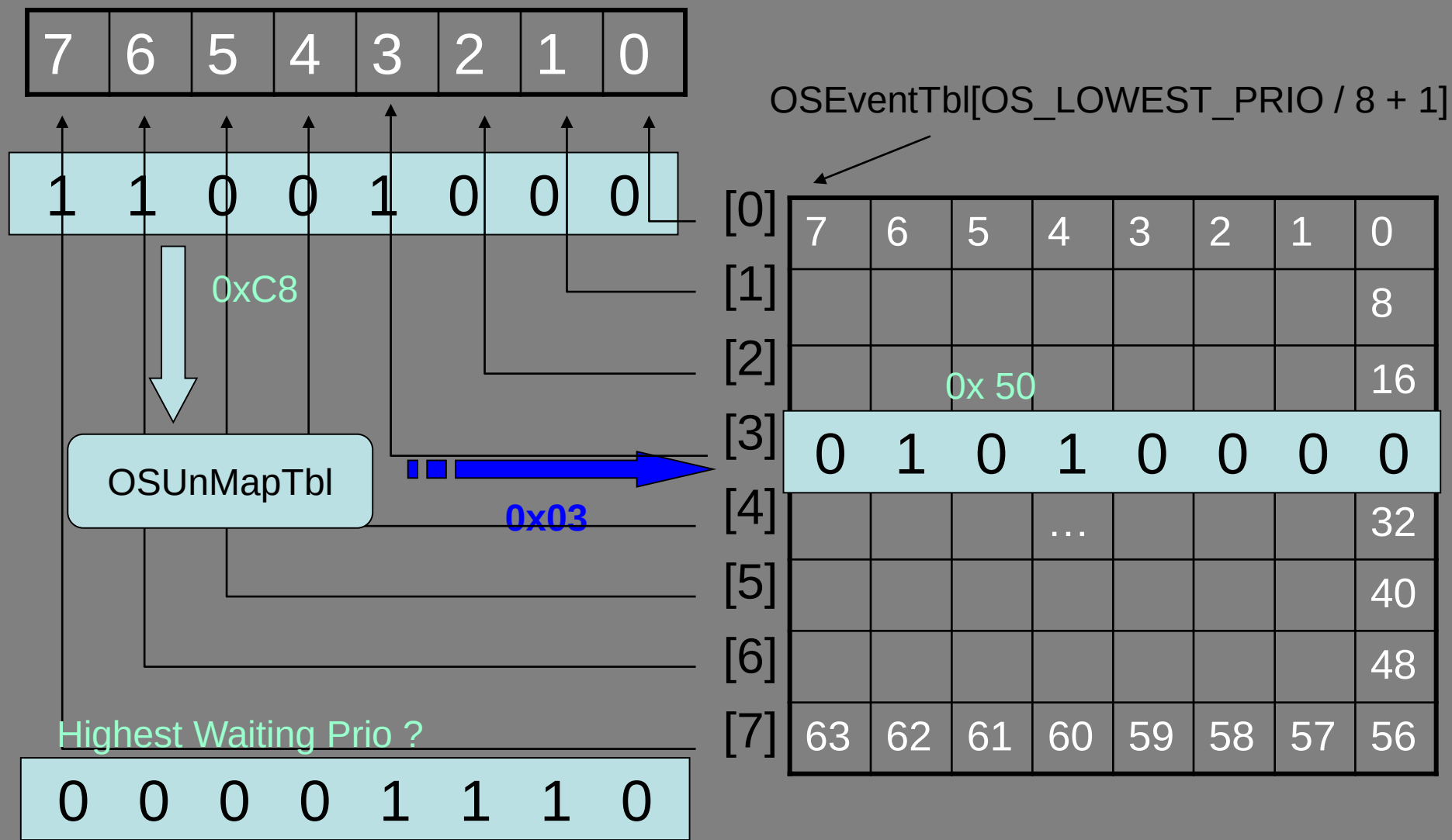
OSEventGrp = 0xC8

```
INT8U  const  OSUnMapTbl[] = {
   0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x20 to 0x2F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x30 to 0x3F                    */
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x40 to 0x4F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F                    */
   7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x90 to 0x9F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xA0 to 0xAF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xB0 to 0xBF                    */
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xC0 to 0xCF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xD0 to 0xDF                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xE0 to 0xEF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0       /* 0xF0 to 0xFF                    */
};
```

y

Waiting list in an OS_EVENT

INT8U OSEventGrp



OSUnMapTbl

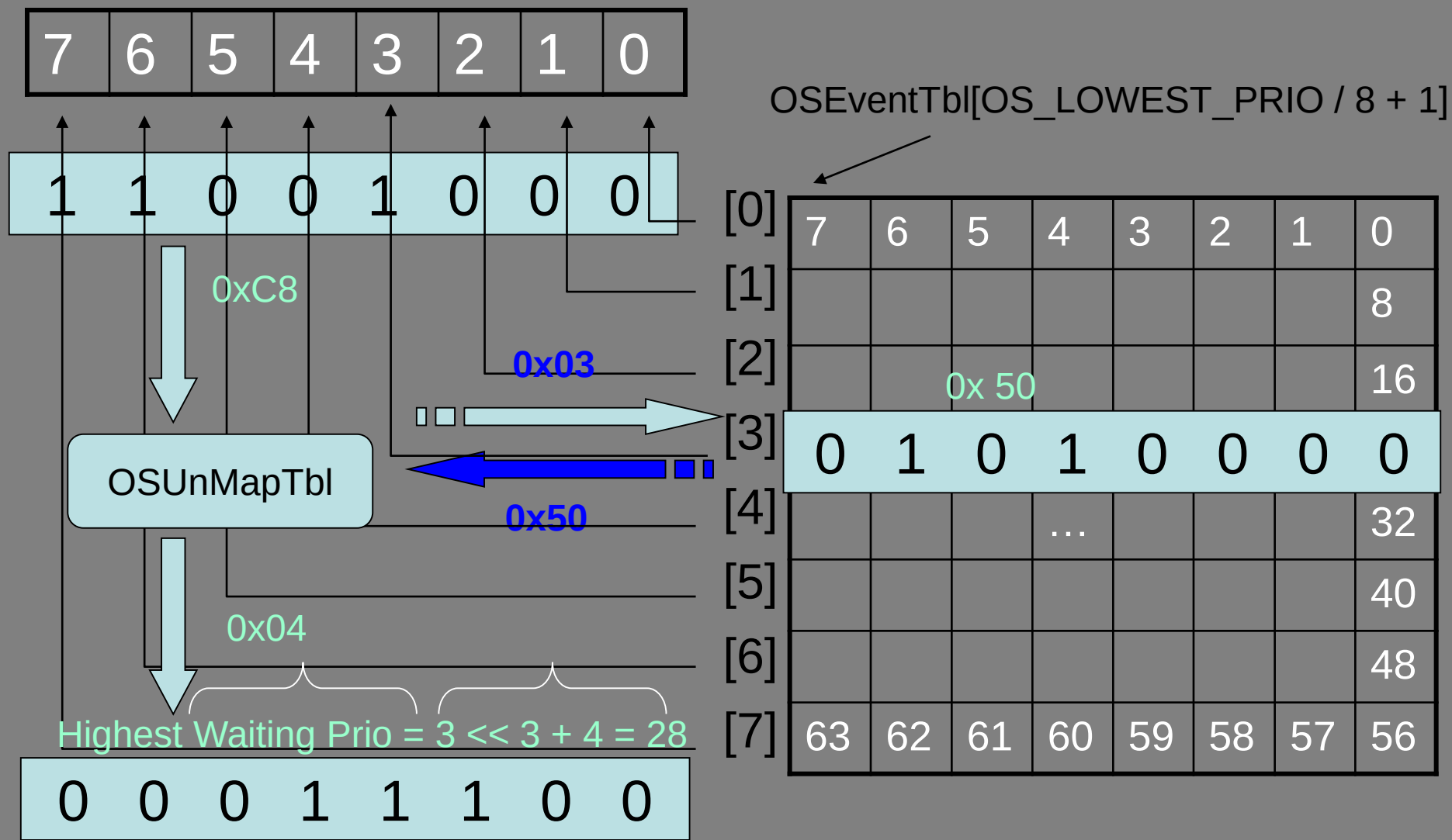
OSEventTbl[OSEventGrp] = 0x50

```
INT8U  const  OSUnMapTbl[] = {
   0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x20 to 0x2F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x30 to 0x3F                    */
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x40 to 0x4F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F                    */
   7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x90 to 0x9F                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xA0 to 0xAF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xB0 to 0xBF                    */
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xC0 to 0xCF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xD0 to 0xDF                    */
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xE0 to 0xEF                    */
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0       /* 0xF0 to 0xFF                    */
};
```

x

Waiting list in an OS_EVENT

INT8U OSEventGrp



Utilisation of task priorities

Highest Waiting Prio = 3 << 3 + 4 = 28

0	0	0	1	1	1	0	0

Bit position in OSEventTbl[]

- Bit position in OSEventTbl[
- Index into OSEventTbl[]]

List of free ECB

- The number of ECB allocated is known at compilation, it is specified by
  - #define OS_MAX_EVENTS dans OS_CFG.H
- When OS_Init is called, all ECBs (like TCBs) are linked in a simply chained list
- When a semaphore, mutex, mailbox or queue is created, an ECB is removed from the free list.
- The ECB can return to it at the end of its use (OSSEMDel()...).

Setting a waiting task READY

- Once the priority calculation of the HPT
- TCB recovery
  - OS_TCB *ptcb = OSTCBPrioTbl[prio];
- Reset the task timeout to zero
  - ptcb->OSTCBDly = 0;
- Reset the pointer to event
  - ptcb->OSTCBEventPtr = (OS_EVENT *)0;
- If ECB corresponds to the sending of a message, placement of this message
  - ptcb->OSTCBMsg = msg;
- Updating the status of the process
  - Ptcb->OSTCBStat &= ~msk;
  - // msk = clear bits corresponding to OS_STAT_SEM, OS_STAT_MUTEX, OS_STAT_MBOX, OS_STAT_Q
- Placing the task in the Ready list
  - If (ptcb->OSTCBStat == OS_STAT_RDY){
  - OSRdyGrp |= bity; // mask du numéro de groupe (8bits)
  - OSRdyTbl[y] |= bitx; // mask du numéro de tâche (8bits)
  - }

OS_EventTaskRdy(pevent, msg, msk)

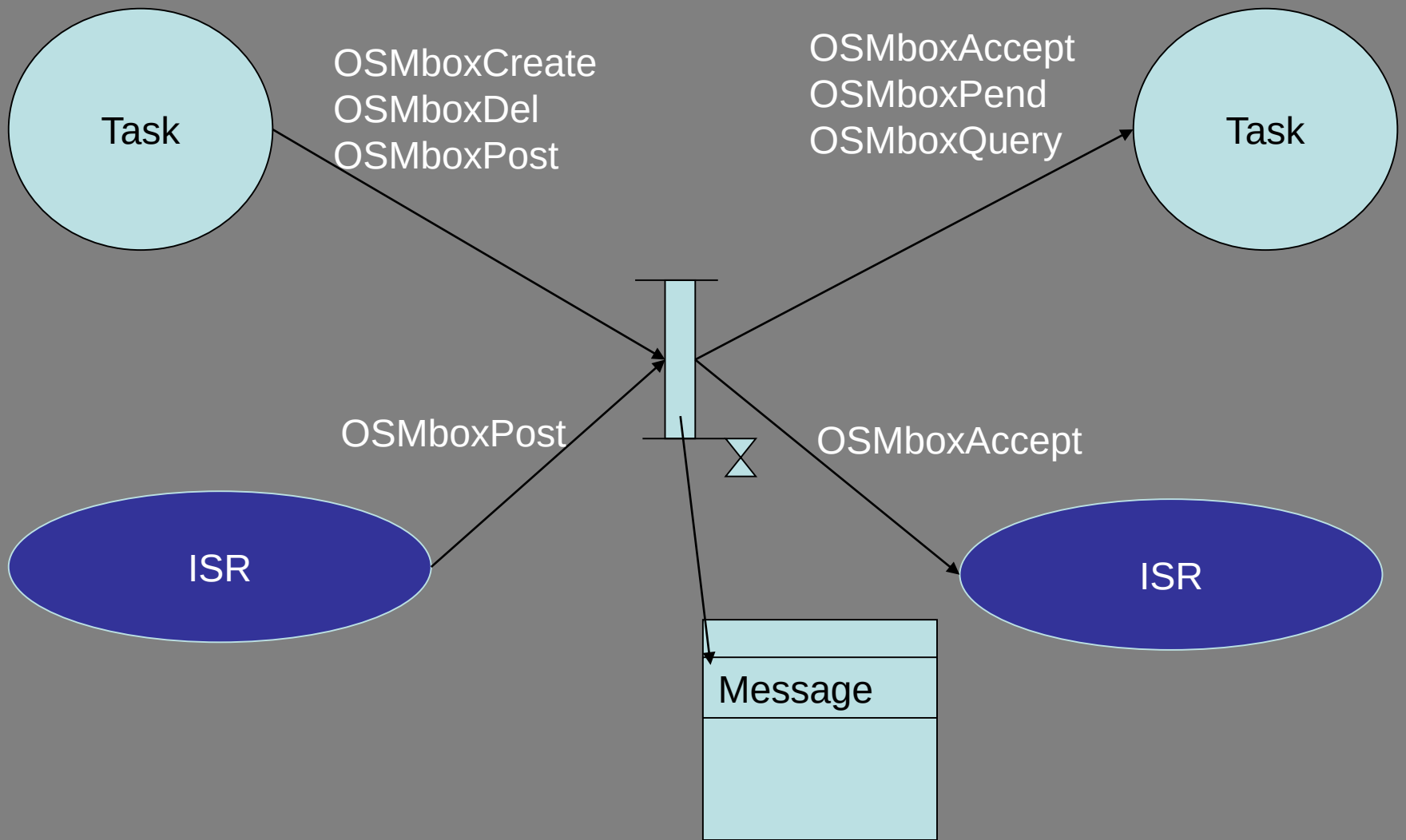
Setting a READY tasks in the waiting state

```
void OS_EventTaskWait(OS_EVENT *pevent){
// Mémorisation de l’evt à attendre dans le TCB
OSTBCur->OSTCBEventPtr = pevent;
// Enlever la tâche de la liste Ready
    If ((OSRdyTbl[OSTCBCur->OSTCBY] &=~OSTCBCur->OSTCBBitX)
            == 0x00){
            OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
    }
// Placement de la tâche en liste d’attente
pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;
// Placement dans le groupe d’attente
Pevent->OSEventGrp |= OSTCBCur->OSTCBBitY;
}
```

Example with mailbox

- Used to send a pointer to a variable.
- Six services :
  - OSMboxCreate()
  - OSMboxPend() // blocking call
  - OSMboxPost()
  - OSMboxPostOpt()
  - OSMboxAccept()
  - OSMboxQuery() // non blocking call

Utilisation



Creation OSMboxCreate()

```
OS_EVENT *OSMboxCreate(void *msg){
    OS_EVENT *pevent;
    OS_ENTER_CRITICAL()
    pevent = OSEventFreeList;
    If (OSEventFreeList != (OS_EVENT *) 0) // point to next
        OSEventFreeList = OSEventFreeList->OSEventPtr;
    OS_EXIT_CRITICAL()
    If (pevent != (OS_EVENT *) 0){
        pevent->OSEventType     = OS_EVENT_TYPE_MBOX;
        pevent->OSEventCnt      = 0;
        pevent->OSEventPtr      = msg;
        OS_EventWaitListInit(pevent);
    }
    return pevent;
}
```