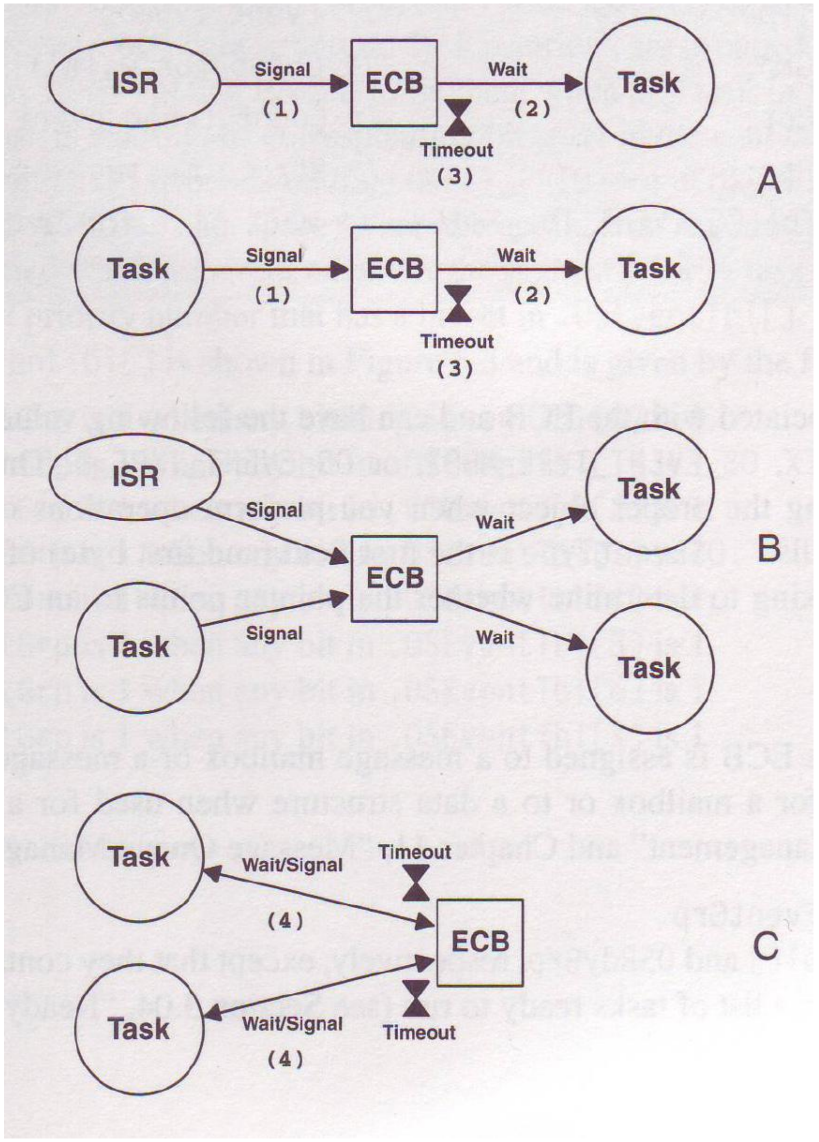
5 Event Control Blocks

Contents

- Placing a Task in the ECB Wait List
- Removing a Task from an ECB Wait List
- Finding the Highest Priority Task Waiting on an ECB
- List of Free ECBs
- Initializing an ECB: `OS_EventWaitListInit()`
- Making a Task Ready: `OS_EventTaskRdy()`
- Making a Task Wait for an Event: `OS_EventTaskWait()`
- Making a Task Ready Because of a Timeout: `OS_EventTO()`

Use of Event Control Blocks

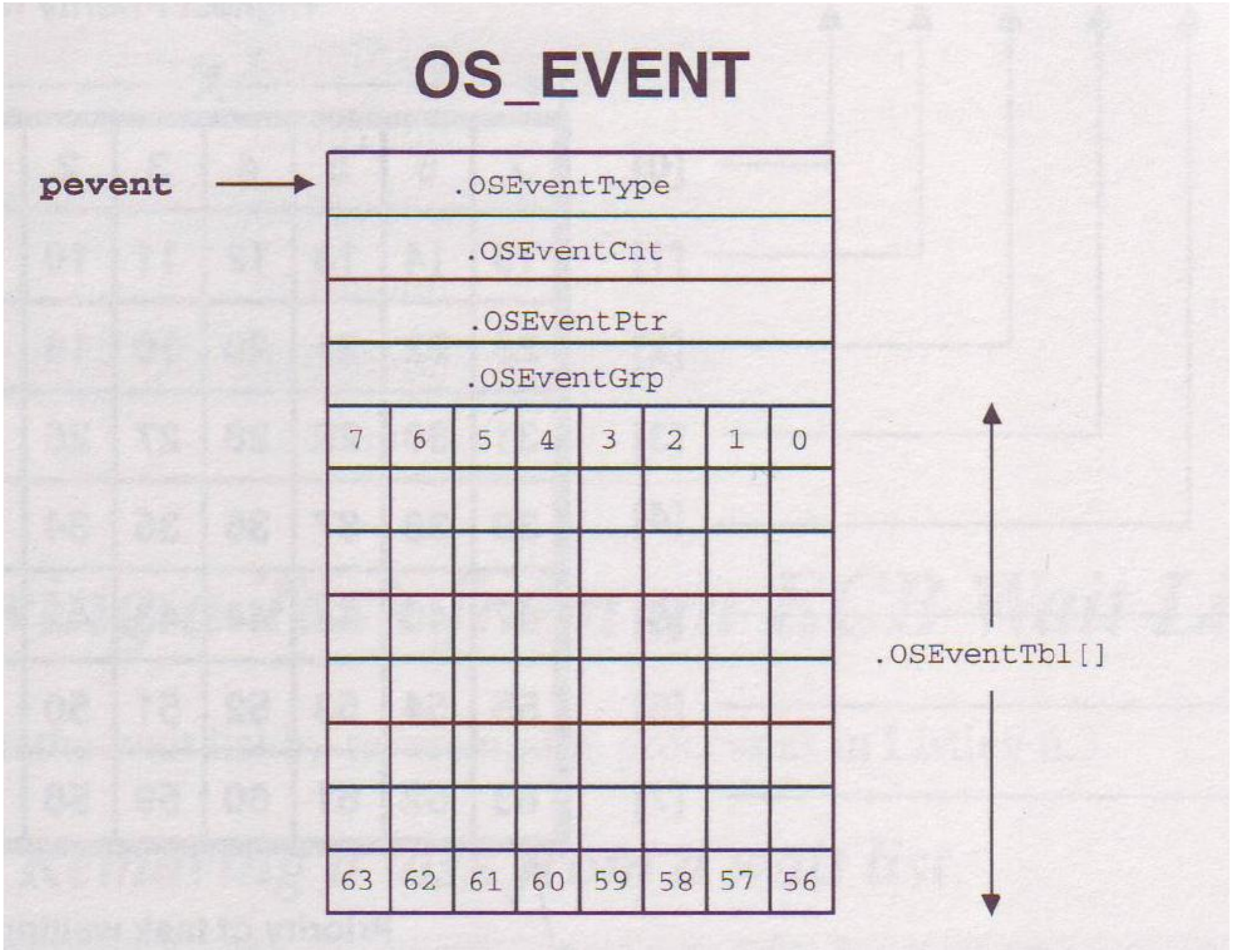


Event Control Block

```
typedef struct {
   INT8U OSEventType; /* Event type */
   INT8U OSEventGrp;  /* Group for waiting list */
   INT16U OSEventCnt; /* Count(event is a semaphore) */
   void *OSEventPtr;  /* Ptr to message or queue */
   INT8U OSEventTbl[OS_EVENT_TBL_SIZE];  /* Wait list */
} OS_EVENT;
```

- Event Types
  - OS_EVENT_TYPE_SEM
  - OS_EVENT_TYPE_MUTEX
  - OS_EVENT_TYPE_MBOX
  - OS_EVENT_TYPE_Q

Event Control Block



OSEventGrp and OSEventTbl[]

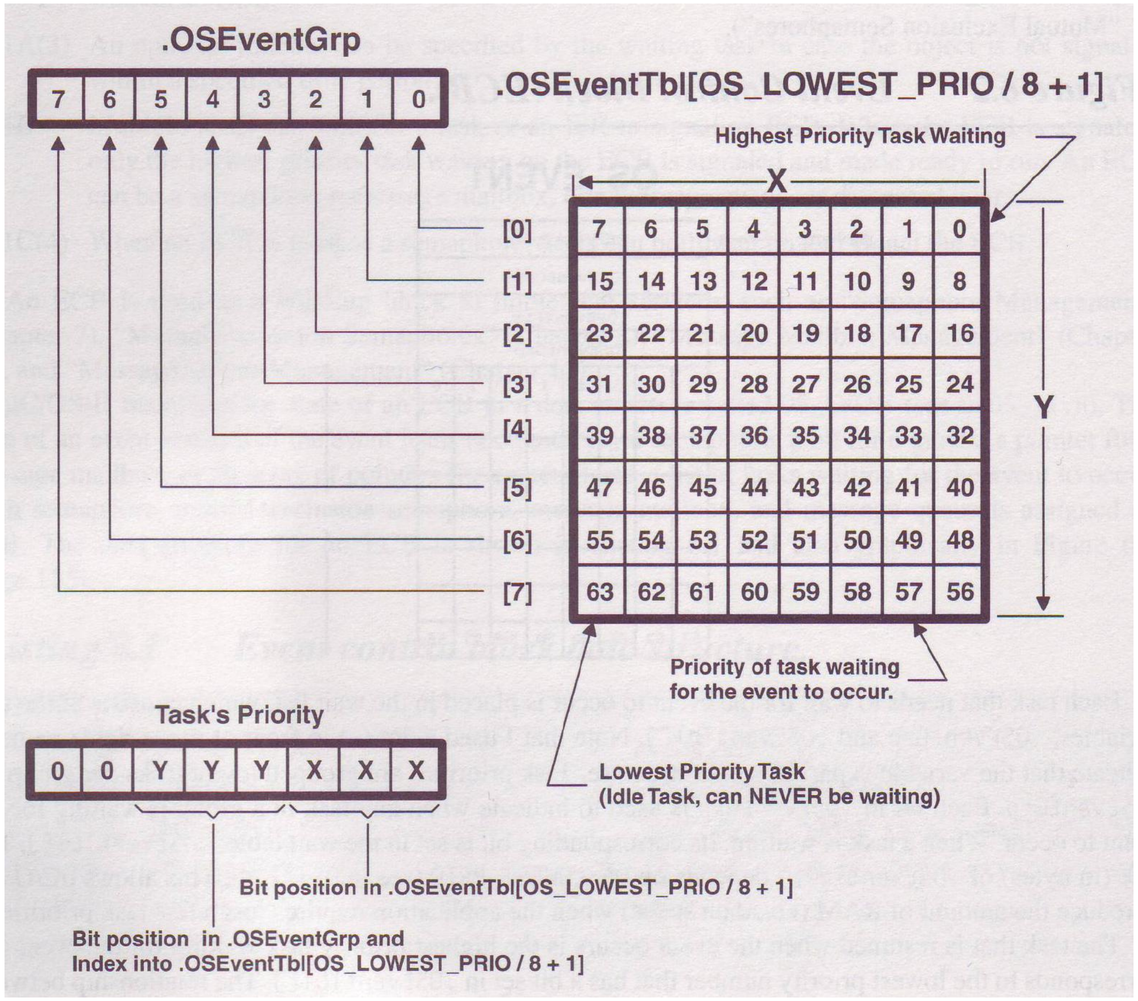
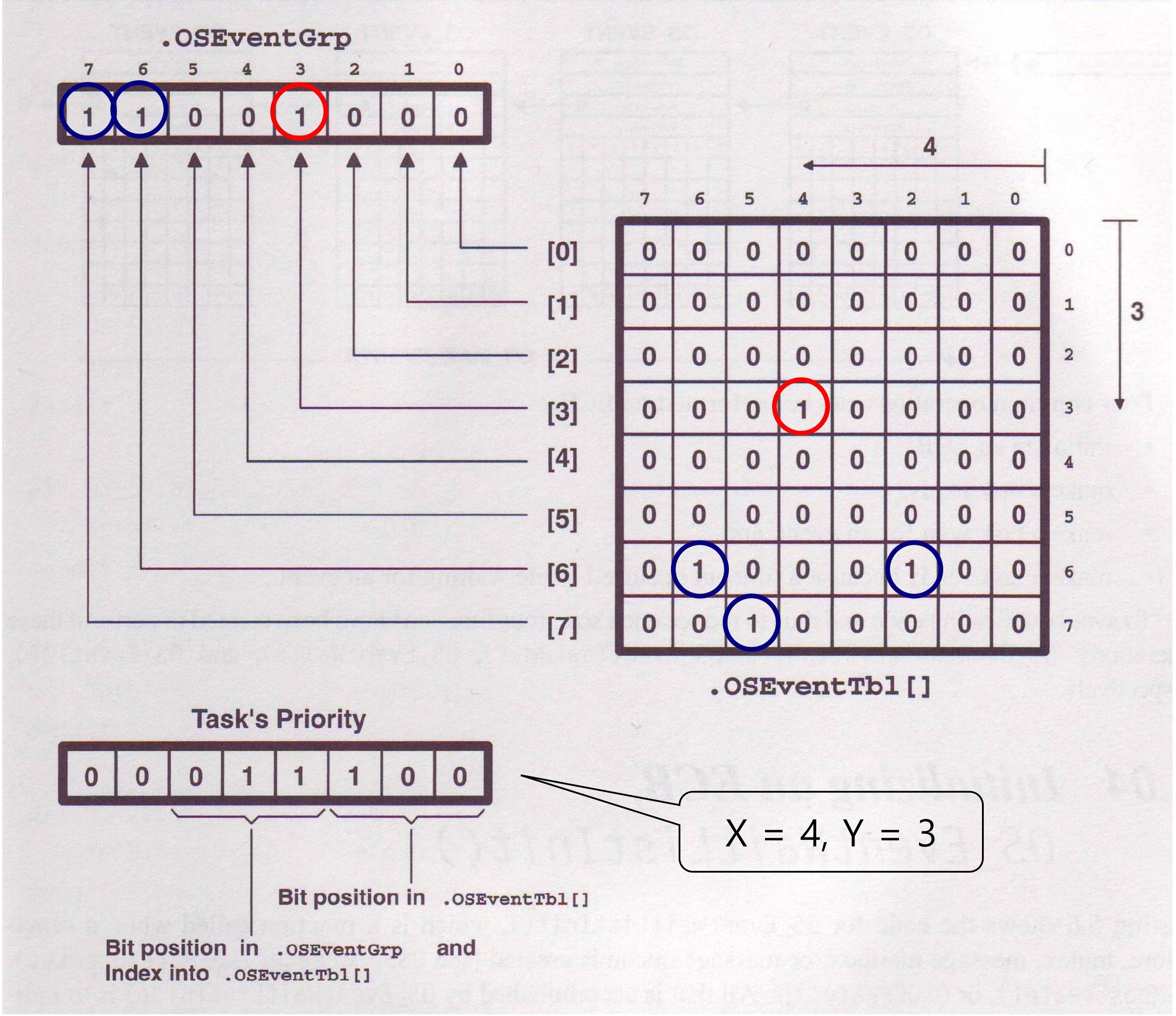
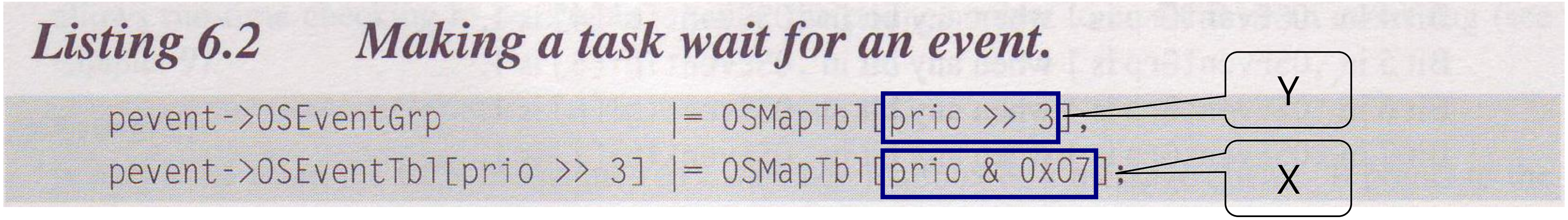

.OSEventGrp
7 6 5 4 3 2 1 0
.OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Highest Priority Task Waiting
X
Y
[0] 7 6 5 4 3 2 1 0
[1] 15 14 13 12 11 10 9 8
[2] 23 22 21 20 19 18 17 16
[3] 31 30 29 28 27 26 25 24
[4] 39 38 37 36 35 34 33 32
[5] 47 46 45 44 43 42 41 40
[6] 55 54 53 52 51 50 49 48
[7] 63 62 61 60 59 58 57 56
Priority of task waiting for the event to occur.
Task's Priority
0 0 Y Y Y X X X
Lowest Priority Task (Idle Task, can NEVER be waiting)
Bit position in .OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Bit position in .OSEventGrp and Index into .OSEventTbl[OS_LOWEST_PRIO / 8 + 1]

Figure 6.4 Example of ECB wait list.



	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	1	0	0	0	1	0	0
7	0	0	1	0	0	0	0	0

Placing a Task in the ECB Wait List



Listing 6.2 ***Making a task wait for an event.***

```
pevent->OSEventGrp              |= OSMapTbl[prio >> 3];
pevent->OSEventTbl[prio >> 3]   |= OSMapTbl[prio & 0x07];
```

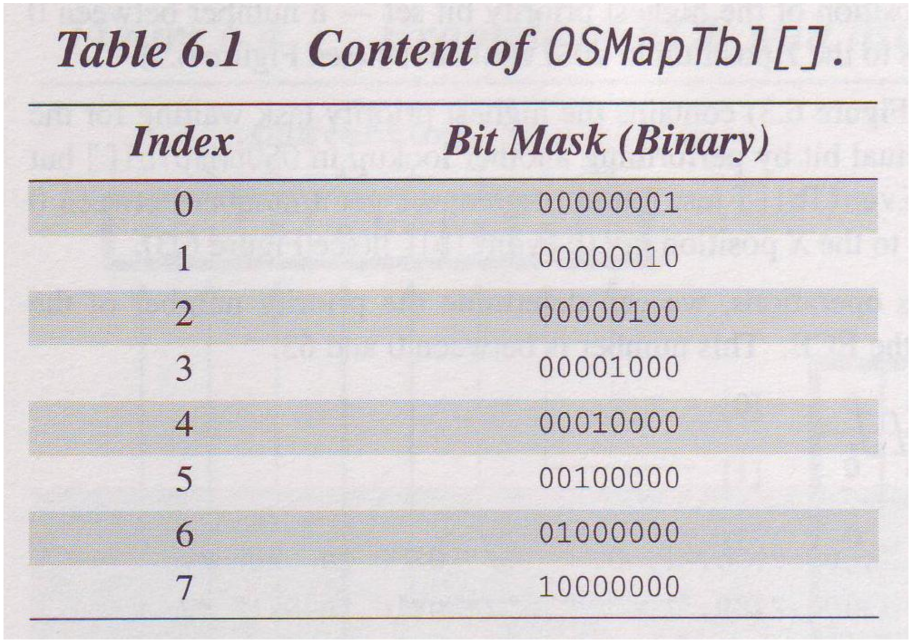


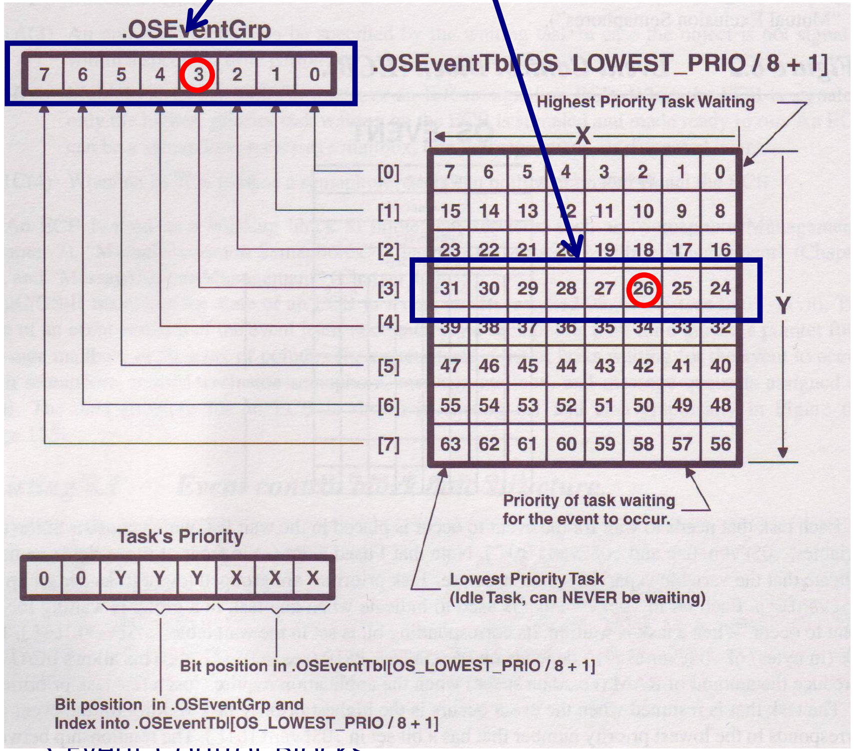
Table 6.1 ***Content of*** `OSMapTbl[]`.

Index	*Bit Mask (Binary)*
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

Removing a Task from an ECB Wait List

Listing 6.3 Removing a task from a wait list.

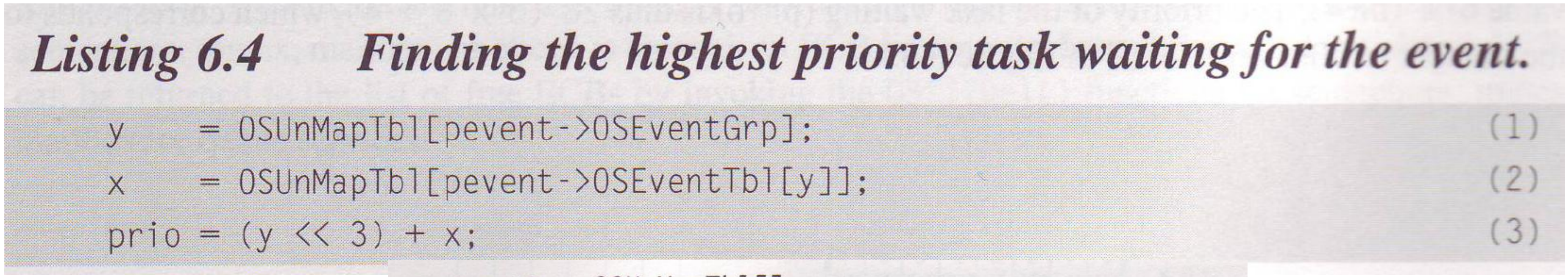
```
if ((pevent->OSEventTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07]) == 0) {
    pevent->OSEventGrp &= ~OSMapTbl[prio >> 3];
}
```


.OSEventGrp
7 6 5 4 3 2 1 0
.OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Highest Priority Task Waiting
X
Y
[0] [1] [2] [3] [4] [5] [6] [7]
7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8
23 22 21 20 19 18 17 16
31 30 29 28 27 26 25 24
39 38 37 36 35 34 33 32
47 46 45 44 43 42 41 40
55 54 53 52 51 50 49 48
63 62 61 60 59 58 57 56
Priority of task waiting for the event to occur.
Task's Priority
0 0 Y Y Y X X X
Lowest Priority Task (Idle Task, can NEVER be waiting)
Bit position in .OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Bit position in .OSEventGrp and Index into .OSEventTbl[OS_LOWEST_PRIO / 8 + 1]

Let `prio` is 26 then

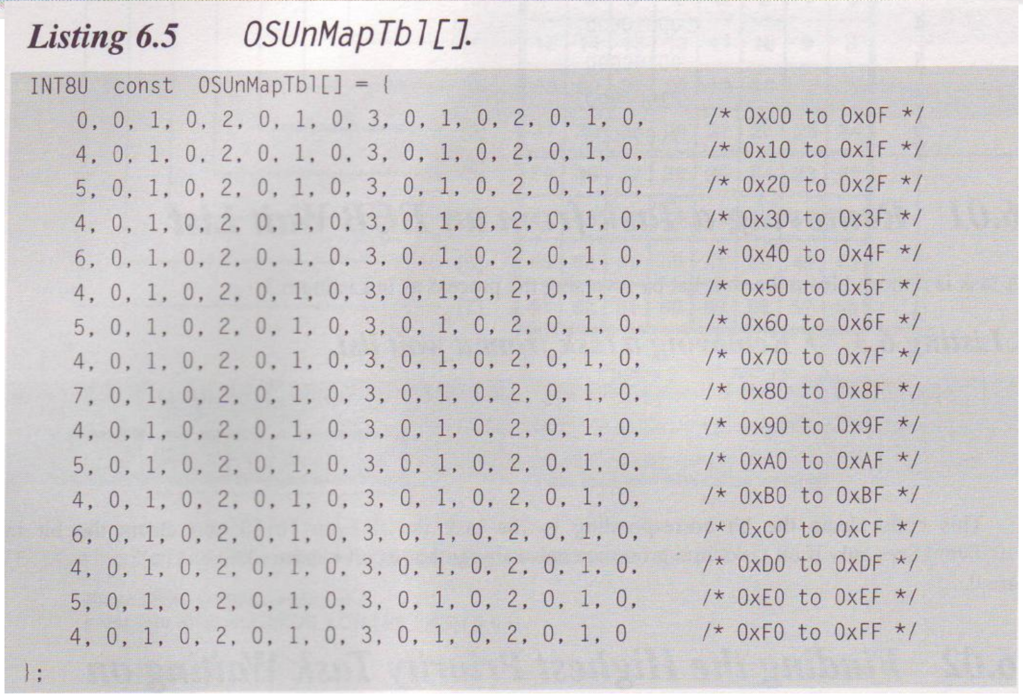
```
prio>>3 = Y = 3
prio&0x07 = X = 2
OSMapTbl[prio&0x07] = 00000100
~ OSMapTbl[prio&0x07] = 11111011
OSMapTbl[prio>>3] = 00001000
~ OSMapTbl[prio>>3]  = 11110111
```

Finding the Highest Priority Task Waiting on an ECB



Listing 6.4 Finding the highest priority task waiting for the event.

```
y    = OSUnMapTbl[pevent->OSEventGrp];                        (1)
x    = OSUnMapTbl[pevent->OSEventTbl[y]];                     (2)
prio = (y << 3) + x;                                          (3)
```

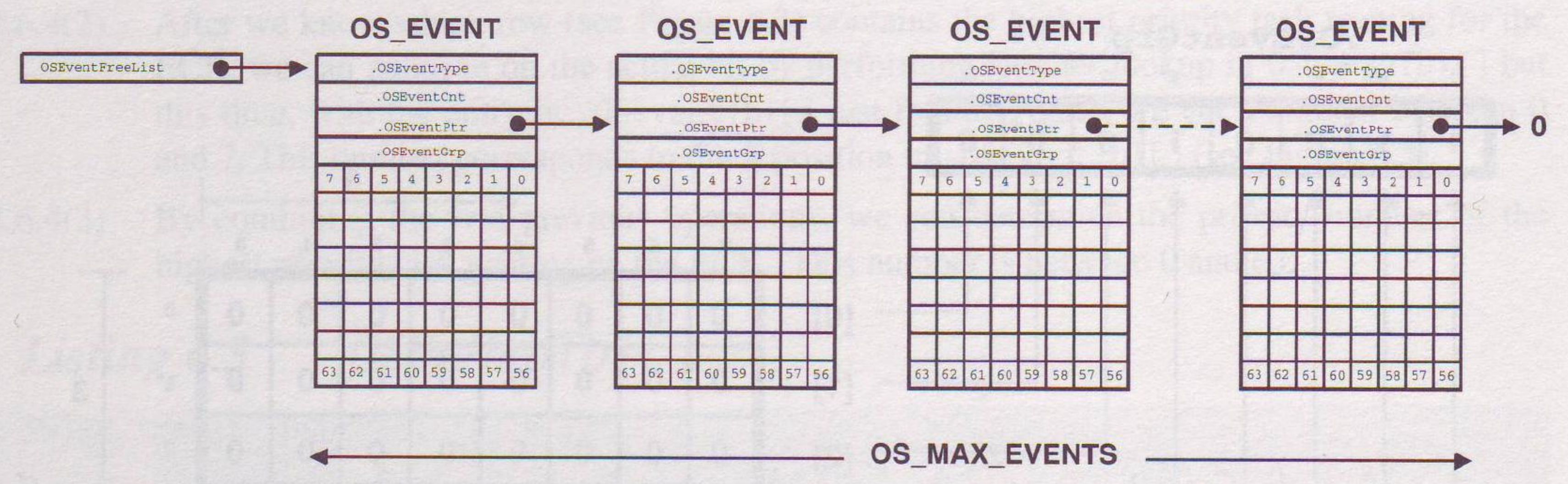


Listing 6.5 `OSUnMapTbl[]`.

```
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 */
};
```

List of Free ECBs

Figure 6.5 List of free ECBs.



Initializing an ECB: OS_EventWaitListInit()

```
void  OS_EventWaitListInit (OS_EVENT *pevent)
{
    INT8U  *ptbl;

    pevent->OSEventGrp = 0x00;
    ptbl               = &pevent->OSEventTbl[0];

#if OS_EVENT_TBL_SIZE > 0
    *ptbl++            = 0x00;
#endif

...

#if OS_EVENT_TBL_SIZE > 7
    *ptbl              = 0x00;
#endif
}
```

```
#define OS_LOWEST_PRIO 63
#define OS_EVENT_TBL_SIZE ((OS_LOWEST_PRIO) / 8 + 1)
```

Making a Task Ready: `OS_EventTaskRdy()`

- Remove the highest priority task from the wait list of the ECB and make this task ready to run
- This function is called by the POST functions for a semaphore, a mutex, a message mailbox, or a message queue when an ECB is signaled and the highest priority task waiting for the ECB need to be made ready to run

Making a Task Ready: OS_EventTaskRdy()

```
INT8U  OS_EventTaskRdy (OS_EVENT *pevent, void *msg, INT8U msk)
{
  OS_TCB *ptcb; INT8U   x; INT8U y; INT8U bitx; INT8U bity; INT8U prio;

  y    = OSUnMapTbl[pevent->OSEventGrp];
  bity = OSMapTbl[y];
  x    = OSUnMapTbl[pevent->OSEventTbl[y]];
  bitx = OSMapTbl[x];
  prio = (INT8U)((y << 3) + x);
  if ((pevent->OSEventTbl[y] &= ~bitx) == 0x00)
    pevent->OSEventGrp &= ~bity;
  ptcb                 =  OSTCBPrioTbl[prio];
  ptcb->OSTCBDly       =  0;
  ptcb->OSTCBEventPtr  = (OS_EVENT *)0;
  ptcb->OSTCBMsg       = msg;
  ptcb->OSTCBStat     &= ~msk;
  if (ptcb->OSTCBStat == OS_STAT_RDY) {
    OSRdyGrp        |=  bity;
    OSRdyTbl[y]     |=  bitx;
  }
  return (prio);
}
```

Remove the task from an ECB wait list

The message is stored in the task's TCB and stat bit field is cleared

If the task is in ready state, insert it to ready list

Making a Task Wait for an Event: `OS_EventTaskWait()`

- Remove the current task from the ready list and place it in the wait list of ECB
- This function is called by the PEND functions for a semaphore, a mutex, a message mailbox, or a message queue when a task must wait on an ECB

Making a Task Wait for an Event: OS_EventTaskWait()

```
void  OS_EventTaskWait (OS_EVENT *pevent)
{
  OSTCBCur->OSTCBEventPtr = pevent;
  if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX)
       == 0x00) {
    OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
  }
  pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;
  pevent->OSEventGrp                   |= OSTCBCur->OSTCBBitY;
}
```

Remove the task from the ready list

Place the task in wait list of the ECB

Making a Task Ready Because of a Timeout: `OS_EventTO()`

- This function is called by the PEND functions for a semaphore, a mutex, a message mailbox, or a message queue when ECB was not signaled within the specified timeout period

Making a Task Ready Because of a Timeout: `OS_EventTO()`

```
void  OS_EventTO (OS_EVENT *pevent)
{
  if ((pevent->OSEventTbl[OSTCBCur->OSTCBY]
        &= ~OSTCBCur->OSTCBBitX) == 0x00) {
    pevent->OSEventGrp &= ~OSTCBCur->OSTCBBitY;
  }
  OSTCBCur->OSTCBStat     = OS_STAT_RDY;
  OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;
}
```

Remove the task from an ECB wait list

Make the task ready to run