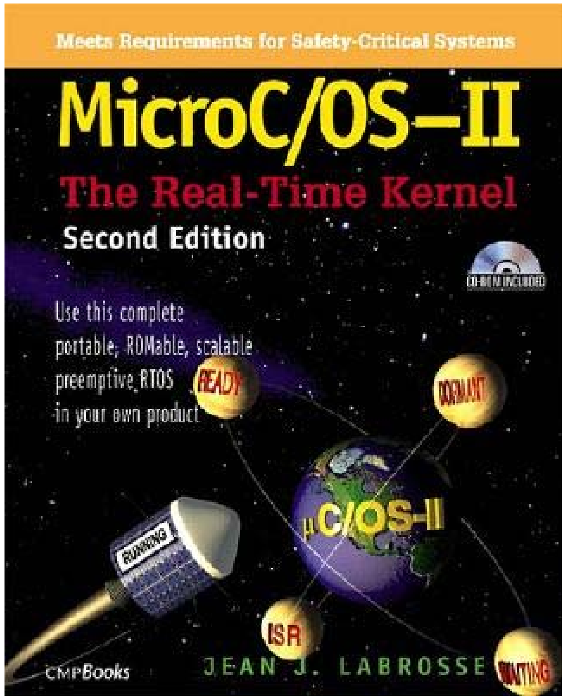
Chapter 6

Event Control Blocks


Meets Requirements for Safety-Critical Systems
MicroC/OS-II
The Real-Time Kernel
Second Edition
Use this complete portable, ROMable, scalable preemptive RTOS in your own product
READY
DORMANT
RUNNING
μC/OS-II
ISR
WAITING
CMPBooks
JEAN J. LABROSSE

Olli Wang <olliwang@ollix.com>

Powered by Ollix

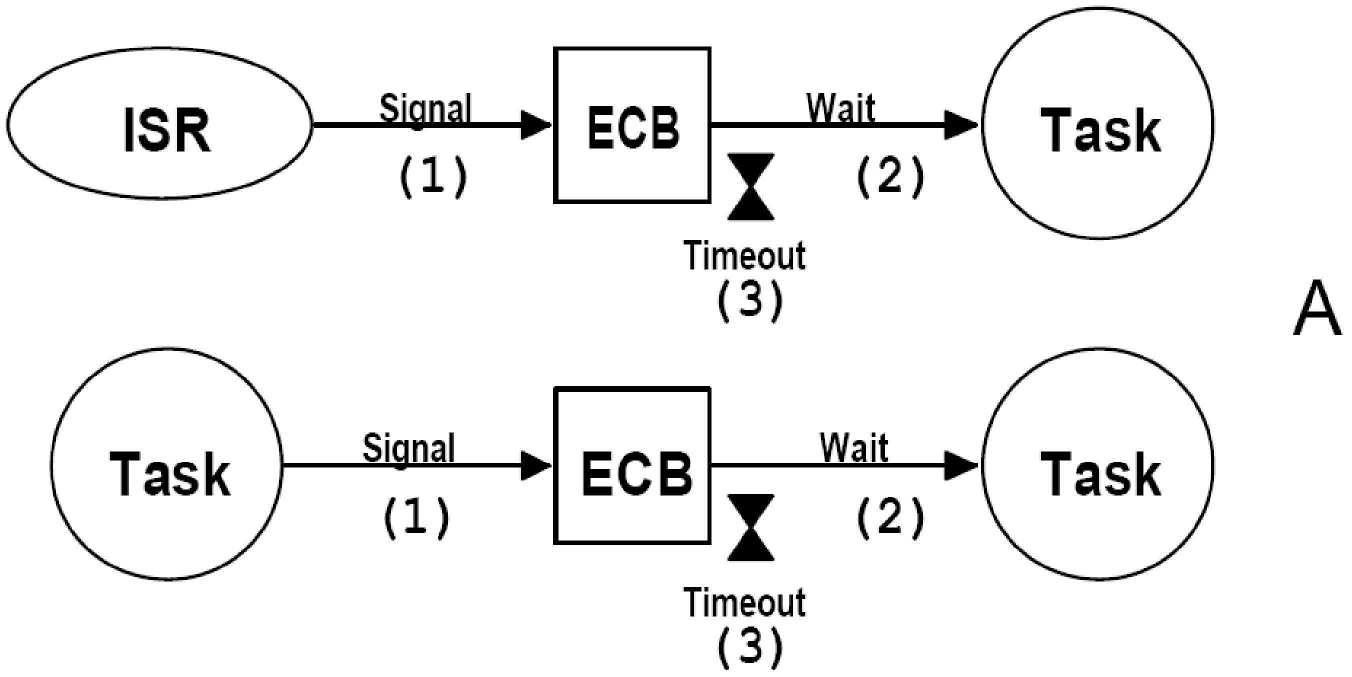
July 29th, 2008

Contents

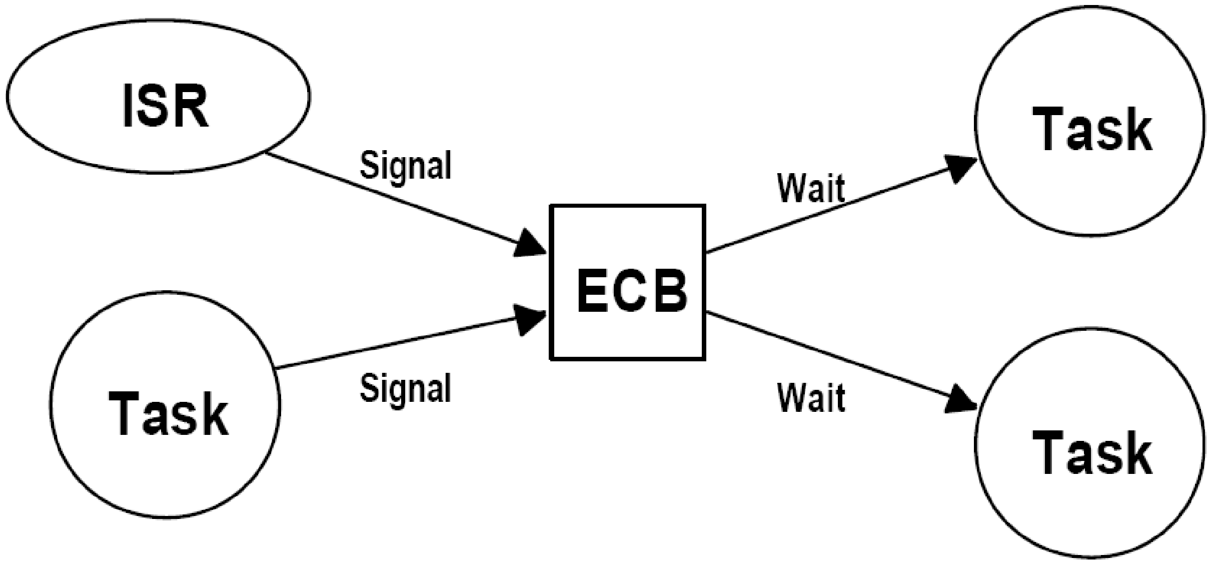
Section. 1

Event Control Blocks

Event Control Blocks

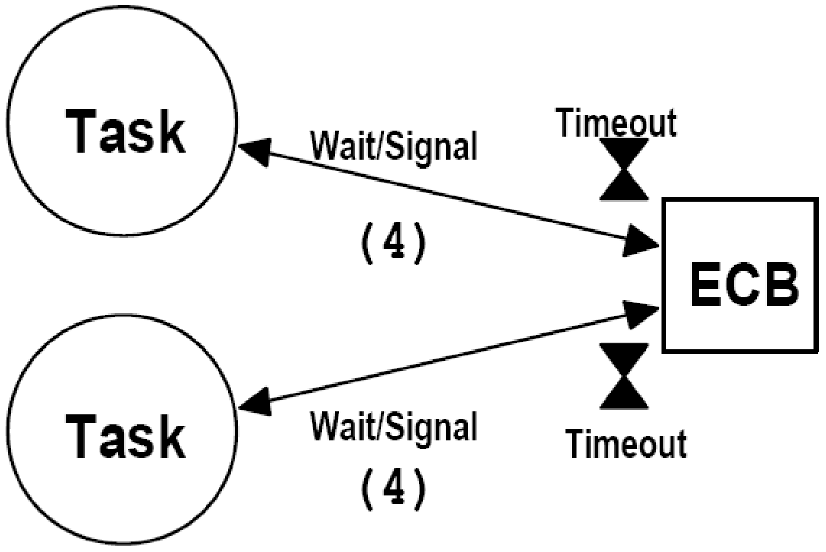


Event Control Blocks



B

Event Control Blocks



C

Semaphore

Event Control Blocks

An ECB is used as a building block to implement services:

1. Semaphore Management
2. Mutual Exclusion Semaphores
3. Message Mailbox Management
4. Message Queue Management

Event Control Blocks

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

Event Control Blocks

Each task that needs to wait for the event to occur is placed in the wait list which consists of the two variable:

1. `OSEventGrp`
2. `OSEventTbl[]`

Event Control Blocks

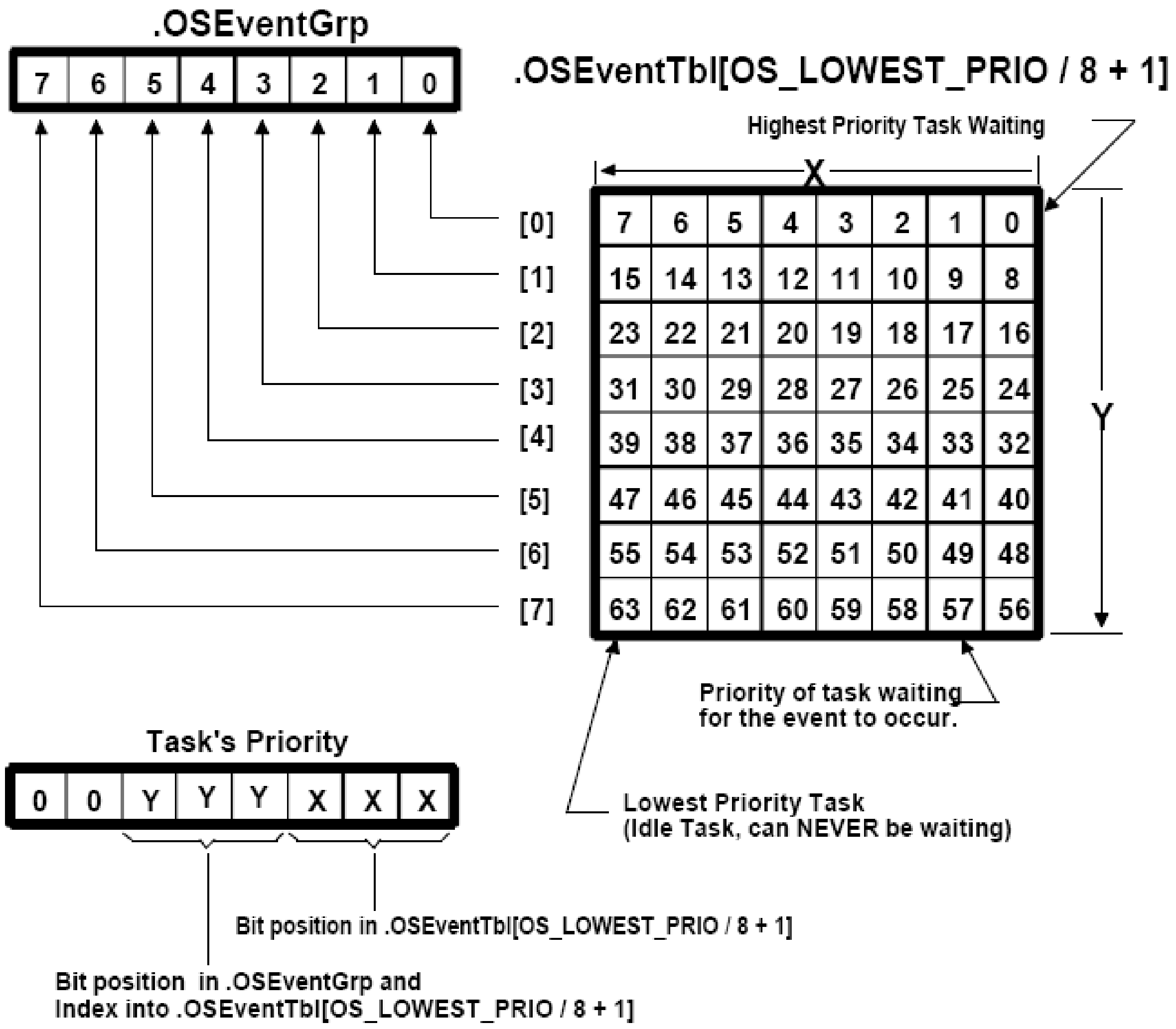

.OSEventGrp
7 6 5 4 3 2 1 0
.OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Highest Priority Task Waiting
X
Y
[0] [1] [2] [3] [4] [5] [6] [7]
7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8
23 22 21 20 19 18 17 16
31 30 29 28 27 26 25 24
39 38 37 36 35 34 33 32
47 46 45 44 43 42 41 40
55 54 53 52 51 50 49 48
63 62 61 60 59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

Figure 6-2, Wait list for task waiting for an event to occur.

Section. 2

Placing a Task in the ECB Wait List

Placing a Task in the ECB Wait List

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

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

Table 6.1, Contents of OSMapTbl[].

Section. 3

Removing a Task from an ECB Wait List

Removing a Task from an ECB Wait List

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

Section. 4

Finding the Highest Priority Task Waiting on an ECB

Finding the Highest Priority Task Waiting on an ECB

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

Section. 5

List of Free ECBs

List of Free ECBs

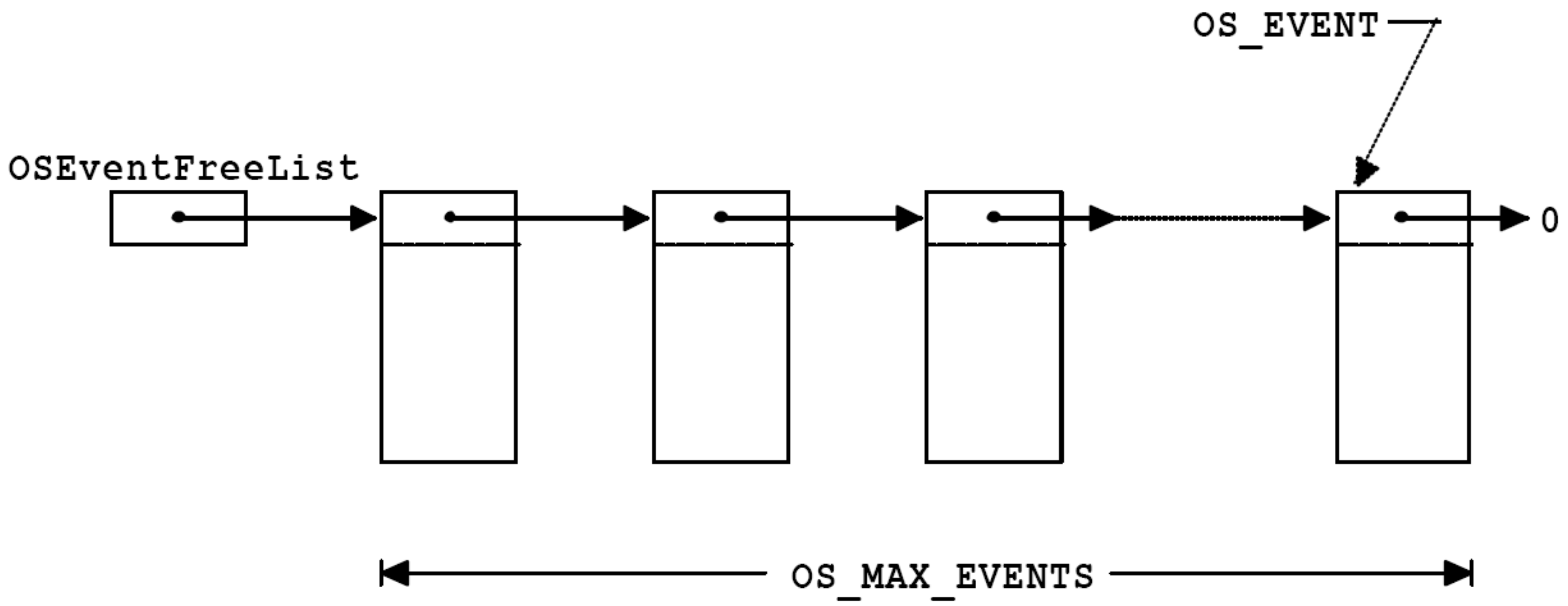


Figure 6-3, List of free ECBs.

List of Free ECBs

Four common operations can be performed on ECBs:

1. Initialize an ECB
2. Make a task ready
3. Make a task wait for an event
4. Make a task ready because of a timeout

Section. 6

Initializing an ECB

Initializing an ECB

`OS_EventWaitListInit()`

called when a semaphore, mutex, message mailbox, or message queue is created.

Initializing an ECB

```
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 > 1
    *ptbl++ = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 2
    *ptbl++ = 0x00;
#endif
#if OS_EVENT_TBL_SIZE > 3
    *ptbl++ = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 4
    *ptbl++ = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 5
    *ptbl++ = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 6
    *ptbl++ = 0x00;
#endif

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

Section. 7

Making a Task Ready

Making a Task Ready

`OS_EventTaskRdy()`

called by the POST functions for a service when an ECB is signaled and the HPT waiting on the ECB needs to be made ready to run.

Making a Task Ready

```
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;
    }
```

Making a Task Ready

```
    ptcb = OSTCBPrioTbl[prio];
    ptcb->OSTCBDly = 0;
    ptcb->OSTCBEventPtr = (OS_EVENT *)0;
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0)
    ptcb->OSTCBMsg = msg;
#else
    msg = msg;
#endif
    ptcb->OSTCBStat &= ~msk;
    if (ptcb->OSTCBStat == OS_STAT_RDY) {
        OSRdyGrp |= bity;
        OSRdyTbl[y] |= bitx;
    }
    return (prio);
}
```

Section. 8

Making a Task Wait for an Event

Making a Task Wait for an Event

`OS_EventTaskWait()`

called by the PEND functions of a service when a task must wait on an ECB.

Making a Task Wait for an Event

```c
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;
}
```

Section. 9

Making a Task Ready Because of a Timeout

Making a Task Ready Because of a Timeout

`OS_EventTo()`

called by `PEND` functions for a service when `OSTimeTick()` has readied a task to run.

Making a Task Ready Because of a Timeout

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

終わり
ありがとう