~~~ Event Control Blocks in MicroC/OS ~~~

To use synchronization and communication services such as Semaphores, Mutex, Mailbox or Message Queues, uC/OS-II uses the Event Control Blocks. All of these services use the same structure where they are initialized, OS_EVENT.

```
typedef struct os_event {
   INT8U    OSEventType;                  /* Type of event control block (see OS_EVENT_TYPE_xxxx)    */
   void     *OSEventPtr;                  /* Pointer to message or queue structure                   */
   INT16U   OSEventCnt;                   /* Semaphore Count (not used if other EVENT type)          */
   OS_PRIO  OSEventGrp;                   /* Group corresponding to tasks waiting for event to occur */
   OS_PRIO  OSEventTbl[OS_EVENT_TBL_SIZE]; /* List of tasks waiting for event to occur               */

#if OS_EVENT_NAME_EN > 0u
   INT8U   *OSEventName;
#endif
} OS_EVENT;
```

OSEventType specifies the type of event control block (a sem, mutex, mailbox/message queue). The event type is as following:

```
#define  OS_EVENT_TYPE_UNUSED         0u
#define  OS_EVENT_TYPE_MBOX           1u
#define  OS_EVENT_TYPE_Q              2u
#define  OS_EVENT_TYPE_SEM            3u
#define  OS_EVENT_TYPE_MUTEX          4u
#define  OS_EVENT_TYPE_FLAG           5u
```

OSEventPtr is a pointer to message or a queue structure. It points to the data of task that needs to be manipulated. It is only used when the Event Control Block is assigned to a message mailbox or a message queue.

OSEvenCnt represents the count/value of the semaphore. It indicates the availability of resources associated with the semaphore. It is used when the ECB is uses for a semaphore or a mutex.

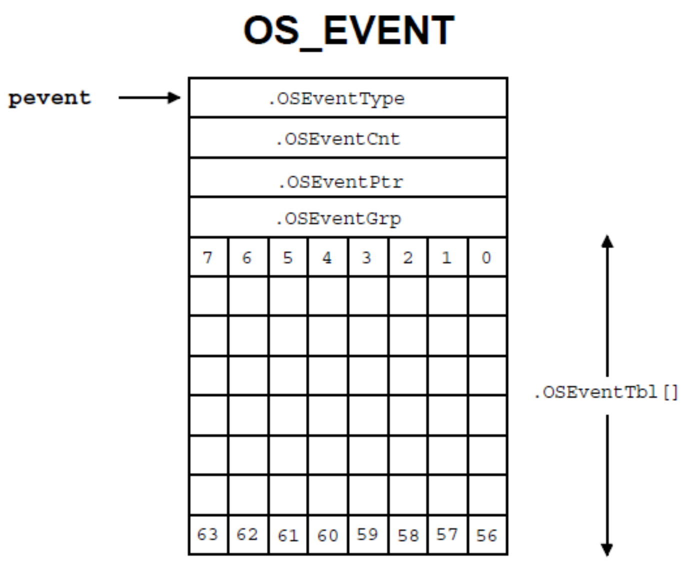
OSEventGrp represents the group of tasks that are waiting for the event to occur.

OSEventTbl holds the priority of the tasks waiting for the event.

OSEventName is a pointer to a string that represents the name of the event.

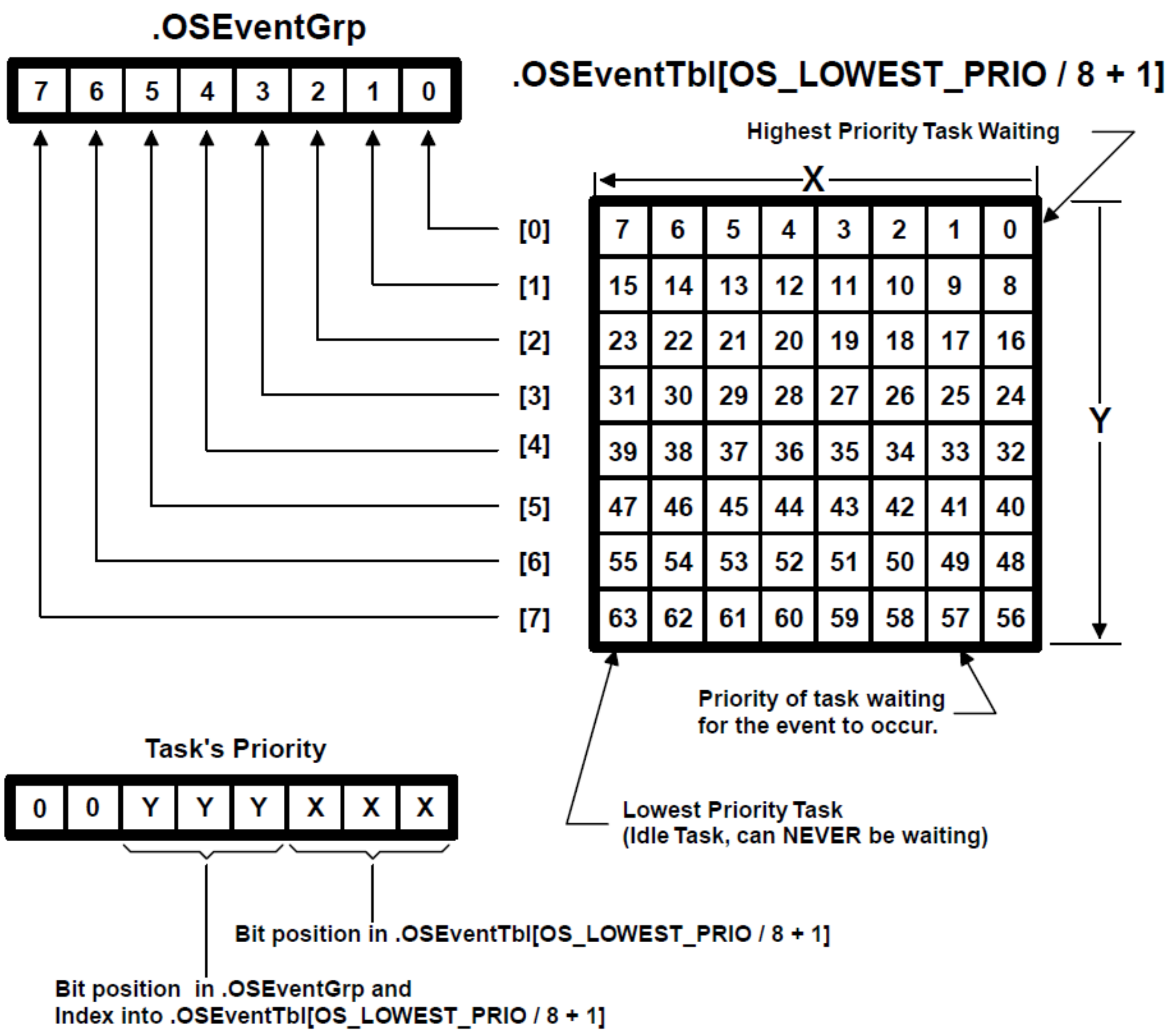
“Each bit in. **OSEvenGrp** is used to indicate when any task in a group is waiting for the event to occur. When a task is waiting, its corresponding bit is set in the wait tab, .**OSEventTbl**[].”

Figure 6.2 Event Control Block (ECB).



Bit 0 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[0]` is 1.
Bit 1 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[1]` is 1.
Bit 2 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[2]` is 1.
Bit 3 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[3]` is 1.
Bit 4 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[4]` is 1.
Bit 5 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[5]` is 1.
Bit 6 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[6]` is 1.
Bit 7 in `.OSEventGrp` is 1 when any bit in `.OSEventTbl[7]` is 1.

Figure 6.3 Wait list for task waiting for an event to occur.


.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]

The task waits for an even by making
pevent->OSEventGrp |= OSMapTbl[prio>>3];
pevent->OSEventTbl[prio>>3] |=OSMapTbl[prio & 0x07];

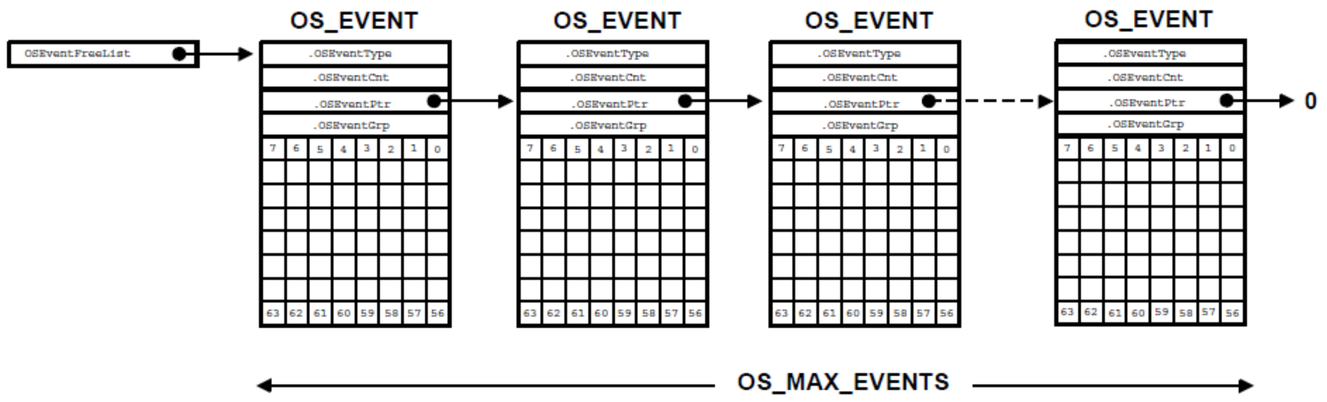
The task is removed from the wait list by making

If((pevent->OSEventTbl[prio>>3] = ~OSMapTbl[prio &0x07]) ==0)

Pevent ->OSEventGrp &= ~OSMapTabl[pro>>3]

The number of Event Control Blocks to be allocate depends on the number of semaphores, mutex, mailboxes and queues.

Figure 6.5 List of free ECBs.



The tasks performed on the ECB are:

Initialize and ECB,
Make a Task Ready,
Make ad Task wat for an event,
make a task ready as a timeout occurred while waiting for an event.

These activities are performed by using the following 4 functions:

OS_EventWaitList()
OS_EventTaskRdy()
OS_EventWait()
OS_EventTO().

List of the function for event management in uC/OS-II are:

OS_EventTaskRdy() – Responsible for marking a task as ready to run. When the event occurs, this function is called to make the task ready to be scheduled by the kernel.

OS_EventTaskWait() – Responsible to keep a task into a waiting state for specific event to occur. Called when a particular event is signaled. It will remain in the waiting state until the event occurs at which point it can be made ready again using **OS_EvernTaskRdy**().

OS_EventTaskRemove() – Responsible to remove a task from the waiting list of a specific even. Called when the timeout occurs or when the task is no longer needed in the event.

OS_EventTaskWaitMulti() – Allows a task to wait for multiple events simultaneously.

OS_EventTaskRemoveMulti() – Remove a ask from the waiting list of multiple events.

OS_EventWaitListInit() – Initializes the waiting list associated with an event. Initializes the data structure OS_EVENT that is required to manage the tasks waiting for the event and makes it ready for further use. The ECB is initialized by using this function.

Let us further understand how the above functions work in detail.

OS_EventTaskRdy():

```
INT8U  OS_EventTaskRdy (OS_EVENT  *pevent,
                        void      *pmsg,
                        INT8U      msk,
                        INT8U      pend_stat)
```

```
{
    OS_TCB   *ptcb;
    INT8U     y;
    INT8U     x;
    INT8U     prio;
#if OS_LOWEST_PRIO > 63u
    OS_PRIO  *ptbl;
#endif


#if OS_LOWEST_PRIO <= 63u
    y    = OSUnMapTbl[pevent->OSEventGrp];              /* Find HPT waiting for message                */
    x    = OSUnMapTbl[pevent->OSEventTbl[y]];
    prio = (INT8U)((y << 3u) + x);                      /* Find priority of task getting the msg       */
#else
    if ((pevent->OSEventGrp & 0xFFu) != 0u) {           /* Find HPT waiting for message                */
        y = OSUnMapTbl[ pevent->OSEventGrp & 0xFFu];
    } else {
        y = OSUnMapTbl[(OS_PRIO)(pevent->OSEventGrp >> 8u) & 0xFFu] + 8u;
    }
    ptbl = &pevent->OSEventTbl[y];
    if ((*ptbl & 0xFFu) != 0u) {
        x = OSUnMapTbl[*ptbl & 0xFFu];
    } else {
        x = OSUnMapTbl[(OS_PRIO)(*ptbl >> 8u) & 0xFFu] + 8u;
    }
    prio = (INT8U)((y << 4u) + x);                      /* Find priority of task getting the msg       */
#endif

    ptcb                  =  OSTCBPrioTbl[prio];        /* Point to this task's OS_TCB                 */
    ptcb->OSTCBDly        =  0u;                        /* Prevent OSTimeTick() from readying task     */
#if ((OS_Q_EN > 0u) && (OS_MAX_QS > 0u)) || (OS_MBOX_EN > 0u)
    ptcb->OSTCBMsg        =  pmsg;                      /* Send message directly to waiting task       */
#else
    pmsg                  =  pmsg;                      /* Prevent compiler warning if not used        */
#endif
    ptcb->OSTCBStat      &= (INT8U)~msk;                /* Clear bit associated with event type        */
    ptcb->OSTCBStatPend   =  pend_stat;                 /* Set pend status of post or abort            */
                                                        /* See if task is ready (could be susp'd)      */
    if ((ptcb->OSTCBStat &   OS_STAT_SUSPEND) == OS_STAT_RDY) {
        OSRdyGrp         |=  ptcb->OSTCBBitY;           /* Put task in the ready to run list           */
        OSRdyTbl[y]      |=  ptcb->OSTCBBitX;
        OS_TRACE_TASK_READY(ptcb);
    }

    OS_EventTaskRemove(ptcb, pevent);                   /* Remove this task from event   wait list     */
#if (OS_EVENT_MULTI_EN > 0u)
    if (ptcb->OSTCBEventMultiPtr != (OS_EVENT **)0) {   /* Remove this task from events' wait lists    */
        OS_EventTaskRemoveMulti(ptcb, ptcb->OSTCBEventMultiPtr);
        ptcb->OSTCBEventPtr       = (OS_EVENT  *)pevent;/* Return event as first multi-pend event ready*/
    }
#endif

    return (prio);
}
```

When the priority is less than 63, the priority calculation takes place using x and y. X and Y variables are used to calculate the Highest priority task waiting for the message and **prio** is used to find the priority of the task that is getting the message.

Breaking down of the calculation

```
 if ((pevent->OSEventGrp & 0xFFu) != 0u)
```

Checks if the lower 8 bits of pevent->OSEventGrp are not zero. It determines whether there are high-priority tasks waiting for the message.

```
y = OSUnMapTbl[ pevent->OSEventGrp & 0xFFu];
```

If there are HPTs waiting for the message (condition is true), y is set to the value obtained from OSUnMapTbl using the lower 8 bits of pevent->OSEventGrp as an index. This mapping provides the corresponding priority value.

```
y = OSUnMapTbl[(OS_PRIO)(pevent->OSEventGrp >> 8u) & 0xFFu] + 8u
```

If there are no HPTs waiting for the message (condition is false), y is calculated differently. The upper 8 bits of pevent->OSEventGrp are shifted right by 8 bits, and the resulting value is treated as an OS_PRIO type. This value is then used as an index into OSUnMapTbl to obtain the corresponding priority value. The additional + 8u adjustment is made to account for priorities greater than 63.

```
ptbl = &pevent->OSEventTbl[y]
```

ptbl is set to the address of the element in pevent->OSEventTbl corresponding to the calculated y index.

```
((*ptbl & 0xFFu) != 0u)
```

Checks if the lower 8 bits of the value pointed to by ptbl are not zero. It determines whether there are tasks waiting for the message at the given priority level.

```
x = OSUnMapTbl[*ptbl & 0xFFu]
```

If there are tasks waiting for the message (condition is true), x is set to the value obtained from OSUnMapTbl using the lower 8 bits of the value pointed to by ptbl as an index.

```
x = OSUnMapTbl[(OS_PRIO)(*ptbl >> 8u) & 0xFFu] + 8u
```

If there are no tasks waiting for the message (condition is false), x is calculated differently. The upper 8 bits of the value pointed to by **ptbl** are shifted right by 8 bits, and the resulting value is treated as an **OS_PRIO** type. This value is then used as an index into **OSUnMapTbl** to obtain the corresponding priority value. The additional + 8u adjustment is made to account for priorities greater than 63.

Once the calculation is done, the processes TCB is pointed to the **OS_TCB**. **OSTCBDly** is set to 0 to prevent the task from being readied by the **OSTimeTick**. The message, **pmsg**, is assigned directly to the **OSTCBMsg**. he **OSTCBStat** field has the associated event type bit cleared using the mask value, and the **OSTCBStatPend** field is set to **pend_stat**.

This code checks if the task is ready (not suspended) by comparing the **OSTCBStat** field with **OS_STAT_RDY**. If it is ready, the task is added to the ready to run list by setting the appropriate bits in **OSRdyGrp** and **OSRdyTbl**. The OS_TRACE_TASK_READY macro is called for tracing purposes.

Once it's done, the task from the event wait list is removed by using OS_EventTaskRemove().

OS_EventTaskWait():

```
void  OS_EventTaskWait (OS_EVENT *pevent)
{
   INT8U  y;


   OSTCBCur->OSTCBEventPtr              = pevent;                /* Store ptr to ECB in TCB        */

   (1) pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;    /* Put task in waiting list        */
   (2) pevent->OSEventGrp                  |= OSTCBCur->OSTCBBitY;

  (3) y            =  OSTCBCur->OSTCBY;           /* Task no longer ready                            */
   (4) OSRdyTbl[y]  &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
   (5)OS_TRACE_TASK_SUSPENDED(OSTCBCur);
   if (OSRdyTbl[y] == 0u) {                     /* Clear event grp bit if this was only task pending */
     (6)  OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
   }
}
```

Store the Event Control Block in the Task Control Block by assigning the current TCB with pevent. The OSTCBBitx is the current task's X bit position in the event table and OSTCBY is the Y bit position in the event table. The (1) sets the bit corresponding to the current task's X bit position in the event table indicating that the task is now waiting for the event. (2) sets the bit corresponding to current task's Y bit position in the event group indicating that there is at least one task waiting for the event. (1) and (2) updates the event table and event group to reflect that the current task is waiting for the event.

(3) Y is then updated indicating that the task is no longer ready.

(4)STCBCur->OSTCBBitX is the bit mask corresponding to the current task's X bit position in the ready table. ~OSTCBCur->OSTCBBitX performs a bitwise complement operation on OSTCBCur->OSTCBBitX, inverting all its bits. It is then type casted to OS_PRIO ensuring compatibility with OSRdyTbl. The bitwise '&' operation clears the bit corresponding to the current task X bit position n the ready table.

(5) Traces the current task that is being suspended.

(6) the condition of OSRdyTbl is cheked if the entire ready table is zero that indicated that no other task is ready to run in the same priority group as the current task. The same operation as (4) is performed on the ready group. This clears the bit corresponding to the current tasks Y bit position in the ready group if it was the only task pending at that priority level.

Overall, this function updates the ready table, ready group and does the tracing when a task is suspended.

OS_EventTaskRemove():

```
void  OS_EventTaskRemove (OS_TCB   *ptcb,
                          OS_EVENT *pevent)
{
    INT8U  y;


    y                     =  ptcb->OSTCBY;
    (1) pevent->OSEventTbl[y]  &= (OS_PRIO)~ptcb->OSTCBBitX;    /* Remove task from wait list               */
    if (pevent->OSEventTbl[y] == 0u) {
    (2)    pevent->OSEventGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
    }
    ptcb->OSTCBEventPtr     = (OS_EVENT  *)0;              /* Unlink OS_EVENT from OS_TCB              */
}
```

(1) ptcb->OSTCBBitX performs a bitwise complement operation on ptcb->OSTCBBitX, inverting all its bits. &= performs a bitwise AND operation between the value in OSEventTbl[y] and (OS_PRIO)~ptcb->OSTCBBitX, and stores the result back into OSEventTbl[y]. This operation removes the task from the wait list for the event by clearing the bit corresponding to the task's X bit position.

(2) If there is no other task waiting on the event at the same priority level as the removed task, procced to clear Y. same operation as (1) is performed but this operation clears the bit corresponding to the task's Y bit position in the event group if it was the only task waiting on the event at that priority level.

The event pointer is set to NULL unlinking the **OS_EVENT** from the TCB indicating that the task is no longer waiting for any event.

Overall, this function removes as task from the event wait list and clears the corresponding bit in the events wait list and unlinks the event from the TCB.

OS_EventTaskWaitMulti():

```
void  OS_EventTaskWaitMulti (OS_EVENT **pevents_wait)
{
   OS_EVENT **pevents;
   OS_EVENT  *pevent;
   INT8U      y;


(1)  OSTCBCur->OSTCBEventPtr      = (OS_EVENT  *)0;
(2)  OSTCBCur->OSTCBEventMultiPtr = (OS_EVENT **)pevents_wait;      /* Store ptr to ECBs in TCB        */

   pevents =  pevents_wait;
   pevent  = *pevents;
   while (pevent != (OS_EVENT *)0) {                              /* Put task in waiting lists       */
(3)      pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;
(4)      pevent->OSEventGrp                   |= OSTCBCur->OSTCBBitY;
       pevents++;
(5)      pevent = *pevents;
   }

(6)  y             =  OSTCBCur->OSTCBY;           /* Task no longer ready                              */
(7)  OSRdyTbl[y]  &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
(8)  OS_TRACE_TASK_SUSPENDED(OSTCBCur);
   if (OSRdyTbl[y] == 0u) {                      /* Clear event grp bit if this was only task pending */
(9)      OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
   }
}
```

The **OSTCBEventPtr** of the current tasks TCB is set to NULL representing that the task is not waiting for any event (1).

The (2) sets the current task TCB to the address of **pevents_wait** which contains pointers to multiple OS_EVENT structures.

(3) updates the wait list for the current **pevent** by setting the bit specified by the current tasks Y bit position and X bit position to indicate that the task is waiting on this event.

(4) updates the event group of the current **pevent** by setting the bit specified by the current tasks Y bit position to indicate that there is at least one task waiting on this event at this priority level.

(5) -Move the pevents pointer to the next element in the pevents_wait array and update the pevent pointer to value pointed to by pevents which is the next OS_EVENT structure in the array.

(6) Stores the current task Y bit position in y. (7) clears the bit in the ready table corresponding to the current tasks X position at the Y bit position indicating that task is no longer ready to run.

(8) Generates a trace event indicating that the current task has been suspended. (9) Clears the event group bit if the current task is the only task pending.

Overall, this function is responsible for making a task wait on multiple events specified in the pevents_wait array.

OS_EventTaskRemoveMulti():

```
void  OS_EventTaskRemove (OS_TCB   *ptcb,
                          OS_EVENT *pevent)
{
    INT8U  y;


    y                       =  ptcb->OSTCBY;
    pevent->OSEventTbl[y]  &= (OS_PRIO)~ptcb->OSTCBBitX;    /* Remove task from wait list              */
    if (pevent->OSEventTbl[y] == 0u) {
        pevent->OSEventGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
    }
    ptcb->OSTCBEventPtr     = (OS_EVENT  *)0;               /* Unlink OS_EVENT from OS_TCB             */
}
```

Remove the current event from the task wait list by clearing the X and Y bits of the current event and unlink the **OS_EVENT** from the **OS_TCB**.

OS_EventWaitListInit():

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


    pevent->OSEventGrp = 0u;                     /* No task waiting on event                           */
    for (i = 0u; i < OS_EVENT_TBL_SIZE; i++) {
        pevent->OSEventTbl[i] = 0u;
    }
}
```

This function initializes the wait list and event group of an event. It iterates through each entry in the event table and sets it to 0 indicating that no tasks are currently blocked on this event for each priority level and no task is currently waiting on this event.

Few other event management related functions in MicroC/OS-II are OSEventNameGet(), OSEventNameSet(), OSEventPendMulti(). These will be described in further articles.

~~~~~~

References: MicroC/OS-II The Real-Time Kernel, Second Edition, Jean J. Labrosse. Open Source code.

Article Written By: Yashwanth Naidu Tikkisetty

Follow **#oswithyash** to get updates related to Embedded Systems, Space and Technology.

~~~~~~