Chapter 8
Mutual Exclusion Semaphores

79256030 張景和
s9256030@cs.nchu.edu.tw
中興大學資科所
系統與網路實驗室

- Creating a Mutex, OSMutexCreate()
- Deleting a Mutex, OSMutexDel()
- Waiting on a Mutex(Blocking), OSMutexPeng()
- Signaling on a Mutex, OSMutexPost()

Mutual Exclusion Semaphores (Mutex) are used by task to gain exclusive access to resources.

A Mutex is used by your application code to reduce the priority inversion problem.

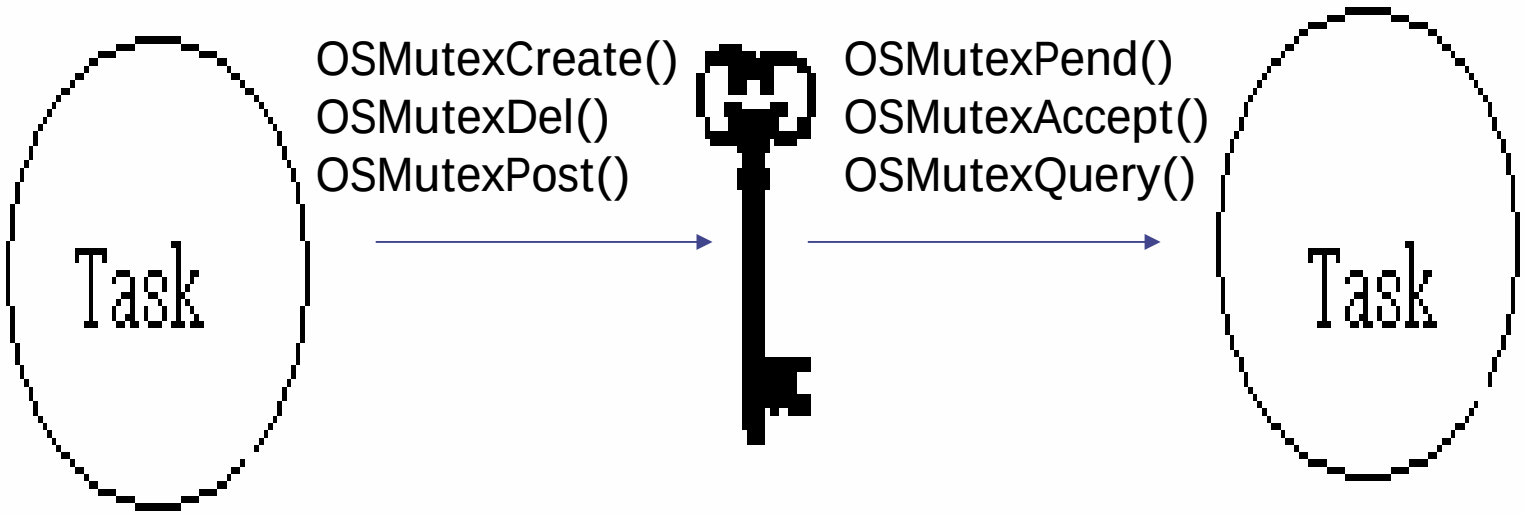
Mutex use example

```
Void main (void)
{
        INT8U err;
        osinit();
        ------------------ application Initialization ------------------
        OSMutexCreate(9,&err);
        OSTaskCreate(TaskPrio10, (void *)0,&TaskPrio10stk[999], 10);
        OSTaskCreate(TaskPrio15, (void *)0,&TaskPrio15stk[999], 15);
        OSTaskCreate(TaskPrio20, (void *)0,&TaskPrio20stk[999], 20);
        ------------------ application Initialization ------------------
        OSStart();
}
```

```
Void TaskPrio10 (void *pdata)
{
        INT8U err;
        pdata = pdata;
        While (1) {
                ------------------ application Initialization ------------------
                OSMutexPend(ResourceMutex , 0 ,&err) ;
                ------------------ application Initialization ------------------
                OSMutexPost(ResourceMutex);
                ------------------ application Initialization ------------------
                }
}
```

```
Void TaskPrio15 (void *pdata)
{
        INT8U err;
        pdata = pdata;
        While (1) {
                ------------------ application Initialization ------------------
                OSMutexPend(ResourceMutex , 0 ,&err) ;
                ------------------ application Initialization ------------------
                OSMutexPost(ResourceMutex);
                ------------------ application Initialization ------------------
                }
}
```

```
Void TaskPrio20 (void *pdata)
{
        INT8U err;
        pdata = pdata;
        While (1) {
                ------------------ application Initialization ----------------
                OSMutexPend(ResourceMutex , 0 ,&err) ;
                ------------------ application Initialization ----------------
                OSMutexPost(ResourceMutex);
                ------------------ application Initialization ----------------
                }
}
```


Task
OSMutexCreate()
OSMutexDel()
OSMutexPost()
OSMutexPend()
OSMutexAccept()
OSMutexQuery()
Task

Creating a Mutex, OSMutexCreate()

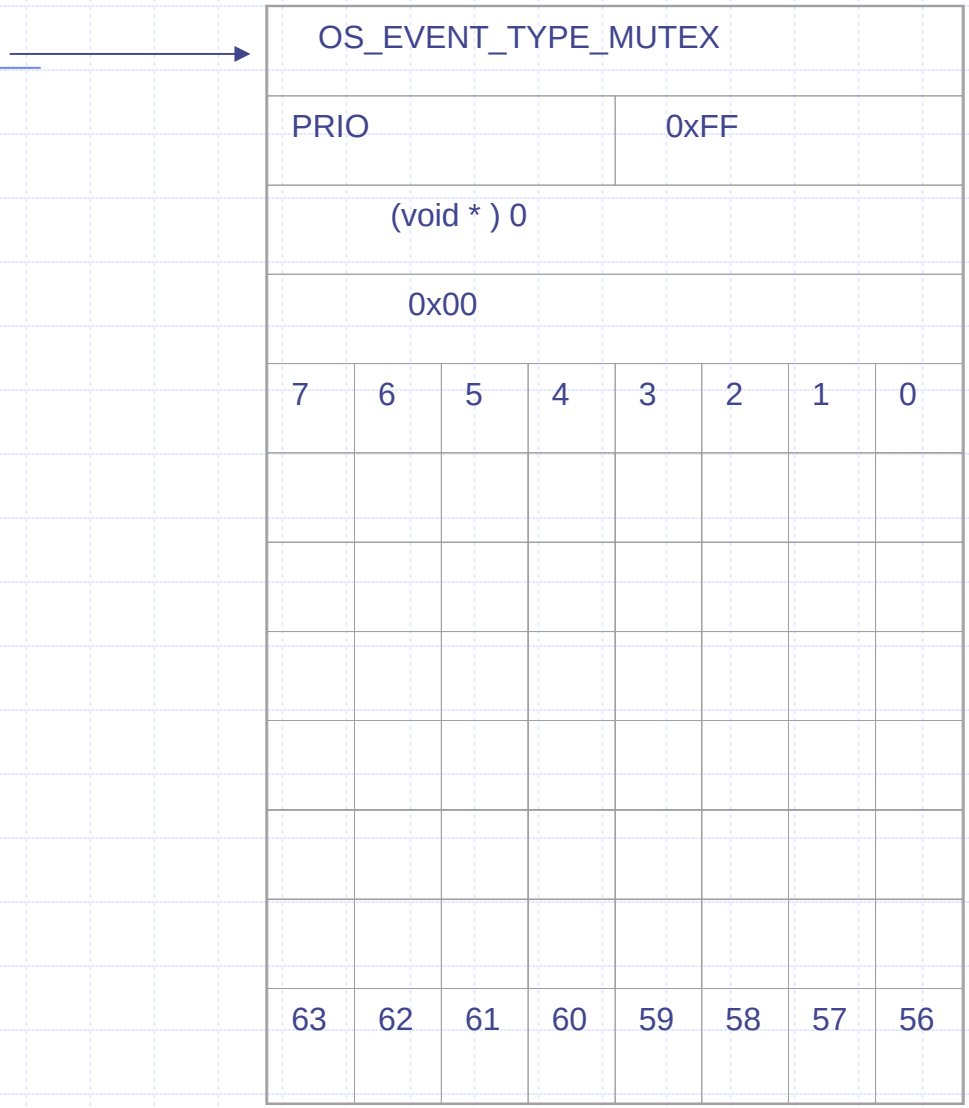
```
OS_EVENT  *OSMutexCreate (INT8U prio, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_EVENT  *pevent;


    if (OSIntNesting > 0) {                          /* See if called from ISR ...               */
        *err = OS_ERR_CREATE_ISR;                    /* ... can't CREATE mutex from an ISR       */
        return ((OS_EVENT *)0);
    }
#if OS_ARG_CHK_EN > 0
    if (prio >= OS_LOWEST_PRIO) {                    /* Validate PIP                             */
        *err = OS_PRIO_INVALID;
        return ((OS_EVENT *)0);
    }
#endif
    OS_ENTER_CRITICAL();
```

```
if (OSTCBPrioTbl[prio] != (OS_TCB *)0) {              /* Mutex priority must not already exist    */
        OS_EXIT_CRITICAL();                           /* Task already exist at priority ...       */
        *err = OS_PRIO_EXIST;                         /* ... inheritance priority                 */
        return ((OS_EVENT *)0);
    }

OSTCBPrioTbl[prio] = (OS_TCB *)1;                     /* Reserve the table entry                  */
    pevent             = OSEventFreeList;             /* Get next free event control block        */
    if (pevent == (OS_EVENT *)0) {                    /* See if an ECB was available              */
        OSTCBPrioTbl[prio] = (OS_TCB *)0;             /* No, Release the table entry              */
        OS_EXIT_CRITICAL();
        *err               = OS_ERR_PEVENT_NULL;      /* No more event control blocks             */
        return (pevent);
    }
    OSEventFreeList     = (OS_EVENT *)OSEventFreeList->OSEventPtr;   /* Adjust the free list          */
    OS_EXIT_CRITICAL();
    pevent->OSEventType = OS_EVENT_TYPE_MUTEX;
    pevent->OSEventCnt  = (prio << 8) | OS_MUTEX_AVAILABLE;/* Resource is available                  */
    pevent->OSEventPtr  = (void *)0;                  /* No task owning the mutex                 */
    OS_EventWaitListInit(pevent);
    *err                = OS_NO_ERR;
    return (pevent);
}
```

pevent



- OSEventType
- OSEventCnt
- OSEnentPtr
- OSEventGrp
- OSEventTbl[]
  All initialized to 0x00

Deleting a Mutex, OSMutexDel()

```
OS_EVENT  *OSMutexDel (OS_EVENT *pevent, INT8U opt, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                  /* Allocate storage for CPU status register          */
    OS_CPU_SR  cpu_sr;
#endif
    BOOLEAN    tasks_waiting;
    INT8U      pip;


    if (OSIntNesting > 0) {                  /* See if called from ISR ...                        */
        *err = OS_ERR_DEL_ISR;               /* ... can't DELETE from an ISR                      */
        return (pevent);
    }
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {           /* Validate 'pevent'                                 */
        *err = OS_ERR_PEVENT_NULL;
        return ((OS_EVENT *)0);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {      /* Validate event block type           */
        *err = OS_ERR_EVENT_TYPE;
        return (pevent);
    }
#endif
```

```
OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0x00) {                    /* See if any tasks waiting on mutex        */
        tasks_waiting = TRUE;                            /* Yes                                      */
    } else {
        tasks_waiting = FALSE;                           /* No                                       */
    }
    switch (opt) {
        case OS_DEL_NO_PEND:                             /* Delete mutex only if no task waiting     */
            if (tasks_waiting == FALSE) {
                pip                 = (INT8U)(pevent->OSEventCnt >> 8);
                OSTCBPrioTbl[pip]   = (OS_TCB *)0;       /* Free up the PIP                          */
                pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
                pevent->OSEventPtr  = OSEventFreeList;   /* Return Event Control Block to free list  */
                OSEventFreeList     = pevent;
                OS_EXIT_CRITICAL();
                *err = OS_NO_ERR;
                return ((OS_EVENT *)0);                  /* Mutex has been deleted                   */
            } else {
                OS_EXIT_CRITICAL();
                *err = OS_ERR_TASK_WAITING;
                return (pevent);
            }
```

```
case OS_DEL_ALWAYS:                                /* Always delete the mutex                 */
            while (pevent->OSEventGrp != 0x00) {          /* Ready ALL tasks waiting for mutex       */
                OS_EventTaskRdy(pevent, (void *)0, OS_STAT_MUTEX);
            }
            pip                 = (INT8U)(pevent->OSEventCnt >> 8);
            OSTCBPrioTbl[pip]   = (OS_TCB *)0;            /* Free up the PIP                         */
            pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
            pevent->OSEventPtr  = OSEventFreeList;        /* Return Event Control Block to free list */
            OSEventFreeList     = pevent;                 /* Get next free event control block       */
            OS_EXIT_CRITICAL();
            if (tasks_waiting == TRUE) {                  /* Reschedule only if task(s) were waiting */
                OS_Sched();                               /* Find highest priority task ready to run */
            }
            *err = OS_NO_ERR;
            return ((OS_EVENT *)0);                       /* Mutex has been deleted                  */

        default:
            OS_EXIT_CRITICAL();
            *err = OS_ERR_INVALID_OPT;
            return (pevent);
    }
}
#endif
```

Waiting on a Mutex(Blocking),OSMutexPend()

```
void  OSMutexPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    INT8U      pip;                                  /* Priority Inheritance Priority (PIP)      */
    INT8U      mprio;                                /* Mutex owner priority                     */
    BOOLEAN    rdy;                                  /* Flag indicating task was ready           */
    OS_TCB    *ptcb;


    if (OSIntNesting > 0) {                          /* See if called from ISR ...               */
        *err = OS_ERR_PEND_ISR;                      /* ... can't PEND from an ISR               */
        return;
    }
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {                   /* Validate 'pevent'                        */
        *err = OS_ERR_PEVENT_NULL;
        return;
    }
```

```
if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {      /* Validate event block type            */
    *err = OS_ERR_EVENT_TYPE;
    return;
}
#endif
OS_ENTER_CRITICAL();                                   /* Is Mutex available?                  */
if ((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
    pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;       /* Yes, Acquire the resource            */
    pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;         /*      Save priority of owning task    */
    pevent->OSEventPtr  = (void *)OSTCBCur;            /*      Point to owning task's OS_TCB   */
    OS_EXIT_CRITICAL();
    *err  = OS_NO_ERR;
    return;
}
pip   = (INT8U)(pevent->OSEventCnt >> 8);              /* No, Get PIP from mutex               */
mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);  /*     Get priority of mutex own
ptcb  = (OS_TCB *)(pevent->OSEventPtr);                /*      Point to TCB of mutex owner     */
if (ptcb->OSTCBPrio != pip && mprio > OSTCBCur->OSTCBPrio) {  /*     Need to promote prio of owner?*
    if ((OSRdyTbl[ptcb->OSTCBY] & ptcb->OSTCBBitX) != 0x00) { /*     See if mutex owner is ready   */
                                                       /*      Yes, Remove owner from Rdy ...*/
                                                       /*           ... list at current prio */
        if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00) {
            OSRdyGrp &= ~ptcb->OSTCBBitY;
        }
        rdy = TRUE;
    } else {
        rdy = FALSE;                                   /* No                                   */
    }
```

```
ptcb->OSTCBPrio        = pip;                  /* Change owner task prio to PIP          */
        ptcb->OSTCBY           = ptcb->OSTCBPrio >> 3;
        ptcb->OSTCBBitY        = OSMapTbl[ptcb->OSTCBY];
        ptcb->OSTCBX           = ptcb->OSTCBPrio & 0x07;
        ptcb->OSTCBBitX        = OSMapTbl[ptcb->OSTCBX];
        if (rdy == TRUE) {                     /* If task was ready at owner's priority ...*/
            OSRdyGrp          |= ptcb->OSTCBBitY;     /* ... make it ready at new priority.       */
            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        }
        OSTCBPrioTbl[pip]      = (OS_TCB *)ptcb;
    }
    OSTCBCur->OSTCBStat |= OS_STAT_MUTEX;          /* Mutex not available, pend current task       */
    OSTCBCur->OSTCBDly   = timeout;                /* Store timeout in current task's TCB          */
    OS_EventTaskWait(pevent);                      /* Suspend task until event or timeout occurs   */
    OS_EXIT_CRITICAL();
    OS_Sched();                                    /* Find next highest priority task ready        */
    OS_ENTER_CRITICAL();
    if (OSTCBCur->OSTCBStat & OS_STAT_MUTEX) {     /* Must have timed out if still waiting for event*/
        OS_EventTO(pevent);
        OS_EXIT_CRITICAL();
        *err = OS_TIMEOUT;                         /* Indicate that we didn't get mutex within TO  */
        return;
    }
    OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
}
```

Signaling a Mutex, OSMutexPost()

```
INT8U  OSMutexPost (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register      */
    OS_CPU_SR  cpu_sr;
#endif
    INT8U      pip;                            /* Priority inheritance priority                 */
    INT8U      prio;


    if (OSIntNesting > 0) {                    /* See if called from ISR ...                    */
        return (OS_ERR_POST_ISR);              /* ... can't POST mutex from an ISR              */
    }
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {             /* Validate 'pevent'                             */
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) { /* Validate event block type              */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

```
OS_ENTER_CRITICAL();
    pip  = (INT8U)(pevent->OSEventCnt >> 8);          /* Get priority inheritance priority of mutex     */
    prio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);  /* Get owner's original priority  */
    if (OSTCBCur->OSTCBPrio != pip &&
        OSTCBCur->OSTCBPrio != prio) {                /* See if posting task owns the MUTEX           */
        OS_EXIT_CRITICAL();
        return (OS_ERR_NOT_MUTEX_OWNER);
    }
    if (OSTCBCur->OSTCBPrio == pip) {                 /* Did we have to raise current task's priority? */
                                                      /* Yes, Return to original priority              */
                                                      /*      Remove owner from ready list at 'pip'    */
        if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX) == 0) {
            OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
        }
        OSTCBCur->OSTCBPrio         = prio;
        OSTCBCur->OSTCBY            = prio >> 3;
        OSTCBCur->OSTCBBitY         = OSMapTbl[OSTCBCur->OSTCBY];
        OSTCBCur->OSTCBX            = prio & 0x07;
        OSTCBCur->OSTCBBitX         = OSMapTbl[OSTCBCur->OSTCBX];
        OSRdyGrp                   |= OSTCBCur->OSTCBBitY;
        OSRdyTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;
        OSTCBPrioTbl[prio]          = (OS_TCB *)OSTCBCur;
    }
```

```
OSTCBPrioTbl[pip] = (OS_TCB *)1;                    /* Reserve table entry                          */
  if (pevent->OSEventGrp != 0x00) {                 /* Any task waiting for the mutex?              */
                                                    /* Yes, Make HPT waiting for mutex ready        */
    prio                = OS_EventTaskRdy(pevent, (void *)0, OS_STAT_MUTEX);
    pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;  /*      Save priority of mutex's new owner      */
    pevent->OSEventCnt |= prio;
    pevent->OSEventPtr  = OSTCBPrioTbl[prio];     /*      Link to mutex owner's OS_TCB            */
    OS_EXIT_CRITICAL();
    OS_Sched();                                   /*      Find highest priority task ready to run  */
    return (OS_NO_ERR);
  }
  pevent->OSEventCnt |= OS_MUTEX_AVAILABLE;       /* No,  Mutex is now available                  */
  pevent->OSEventPtr  = (void *)0;
  OS_EXIT_CRITICAL();
  return (OS_NO_ERR);
```

Getting a Mutex without Waiting (Non Blocking), OSMutexAccept()

```
INT8U  OSMutexAccept (OS_EVENT *pevent, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr;
#endif


    if (OSIntNesting > 0) {                    /* Make sure it's not called from an ISR        */
        *err = OS_ERR_PEND_ISR;
        return (0);
    }
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {             /* Validate 'pevent'                            */
        *err = OS_ERR_PEVENT_NULL;
        return (0);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {  /* Validate event block type            */
        *err = OS_ERR_EVENT_TYPE;
        return (0);
    }
#endif
```

```
OS_ENTER_CRITICAL();                          /* Get value (0 or 1) of Mutex                 */
    if ((pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
        pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;   /*      Mask off LSByte (Acquire Mutex)        */
        pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;     /*      Save current task priority in LSByte   */
        pevent->OSEventPtr  = (void *)OSTCBCur;        /*      Link TCB of task owning Mutex          */
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return (1);
    }
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
    return (0);
}

#endif
```

Obtaining the Status of a Mutex,OSMutexQuery()

```
INT8U  OSMutexQuery (OS_EVENT *pevent, OS_MUTEX_DATA *pdata)
{
#if OS_CRITICAL_METHOD == 3                  /* Allocate storage for CPU status register          */
    OS_CPU_SR  cpu_sr;
#endif
    INT8U     *psrc;
    INT8U     *pdest;


    if (OSIntNesting > 0) {                          /* See if called from ISR ...                */
        return (OS_ERR_QUERY_ISR);                   /* ... can't QUERY mutex from an ISR         */
    }
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {                   /* Validate 'pevent'                         */
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {      /* Validate event block type           */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

```
OS_ENTER_CRITICAL();
    pdata->OSMutexPIP  = (INT8U)(pevent->OSEventCnt >> 8);
    pdata->OSOwnerPrio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);
    if (pdata->OSOwnerPrio == 0xFF) {
        pdata->OSValue = 1;
    } else {
        pdata->OSValue = 0;
    }
    pdata->OSEventGrp  = pevent->OSEventGrp;              /* Copy wait list                         */
    psrc               = &pevent->OSEventTbl[0];
    pdest              = &pdata->OSEventTbl[0];
#if OS_EVENT_TBL_SIZE > 0
    *pdest++           = *psrc++;
#endif
```

```
#if OS_EVENT_TBL_SIZE > 1
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 2
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 3
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 4
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 5
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 6
    *pdest++           = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 7
    *pdest             = *psrc;
#endif
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
#endif                                  /* OS_MUTEX_QUERY_EN                */
```

OS_MUTEX_DATA

```
typedef struct {
    INT8U   OSEventTbl[OS_EVENT_TBL_SIZE];  /* List of tasks waiting for event to occur  */
    INT8U   OSEventGrp;                     /* Group corresponding to tasks waiting for event to occur */
    INT8U   OSValue;                        /* Mutex value (0 = used, 1 = available)                   */
    INT8U   OSOwnerPrio;                    /* Mutex owner's task priority or 0xFF if no owner         */
    INT8U   OSMutexPIP;                     /* Priority Inheritance Priority or 0xFF if no owner       */
} OS_MUTEX_DATA;
```