Mutual Exclusion Semaphores – uC/OSII

Yu-Han Li

OSNET Lab. of C.S.

National Chung Hsing University

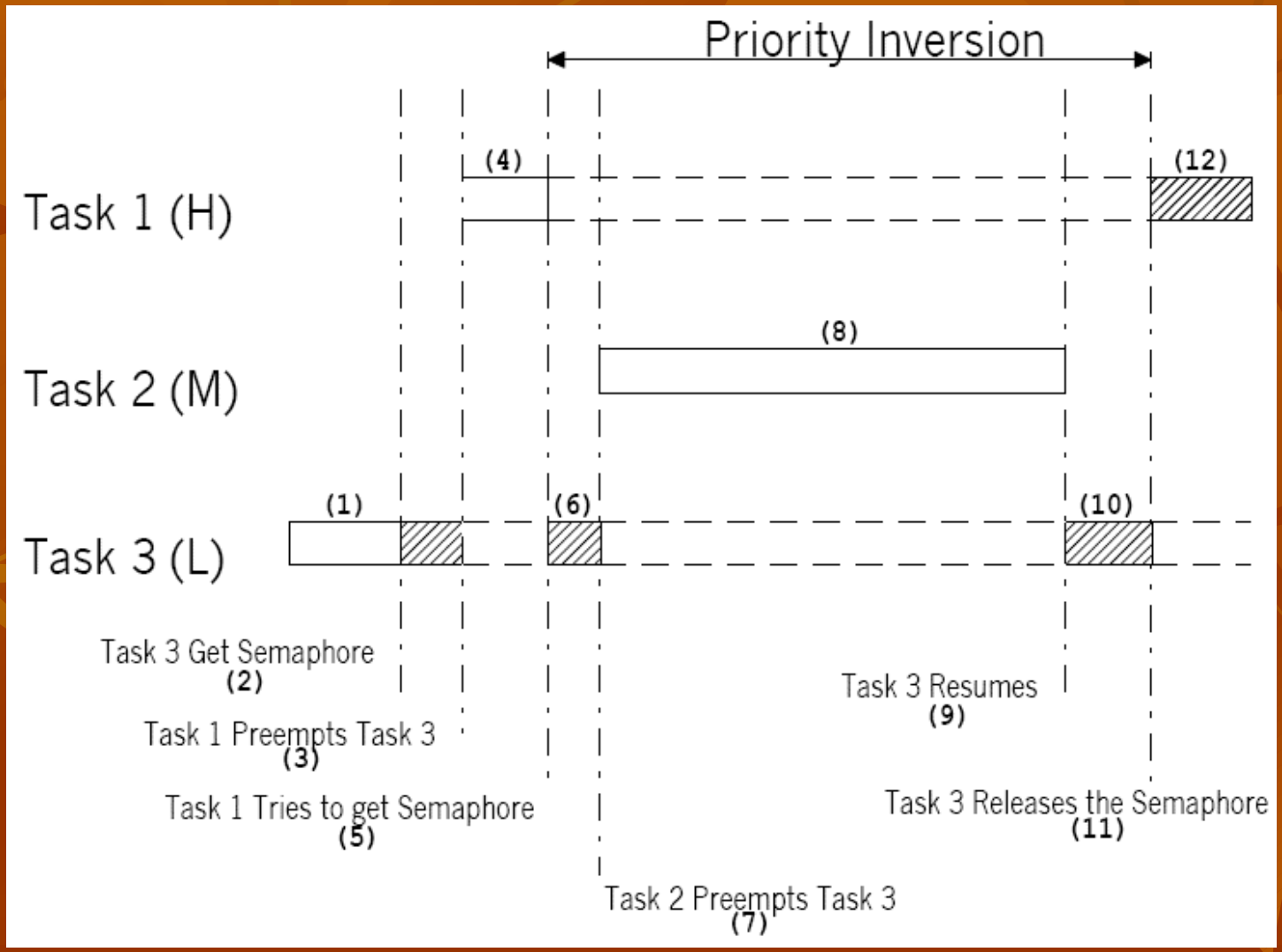
Contents

- Background
- Mutex use example
- OSMutexCreate()
- OSMutexDel()
- OSMutexPend() – (Blocking)
- OSMutexPost()
- OSMutexAccept() – (Non-blocking)
- OSMutexQuery()

Background (1/3)

- Mutual Exclusion Semaphores(**mutexes**) : to gain exclusive access to resources by tasks.
- Mutexes are ***binary semaphores***.
- A mutex is used by application to reduce the ***priority inversion*** *problem*.
- To reduce priority inversion, the kernel **increase** the priority of the ***lower priority task*** to the higher priority task until the lower priority task is down.

Background – Priority Inversion (2/3)



Background (3/3)

- Problems of the MicroC/OS-II. – ***Priority***.
- A solution: A ***priority*** just above the highest priority task that needs to access the mutex was reversed by *mutex* to allow a lower priority task to be raised .
- An unused priority just above the highest task priority is reserved as the *priority inheritance priority(PIP)*.

Mutex use example – main() (1/4)

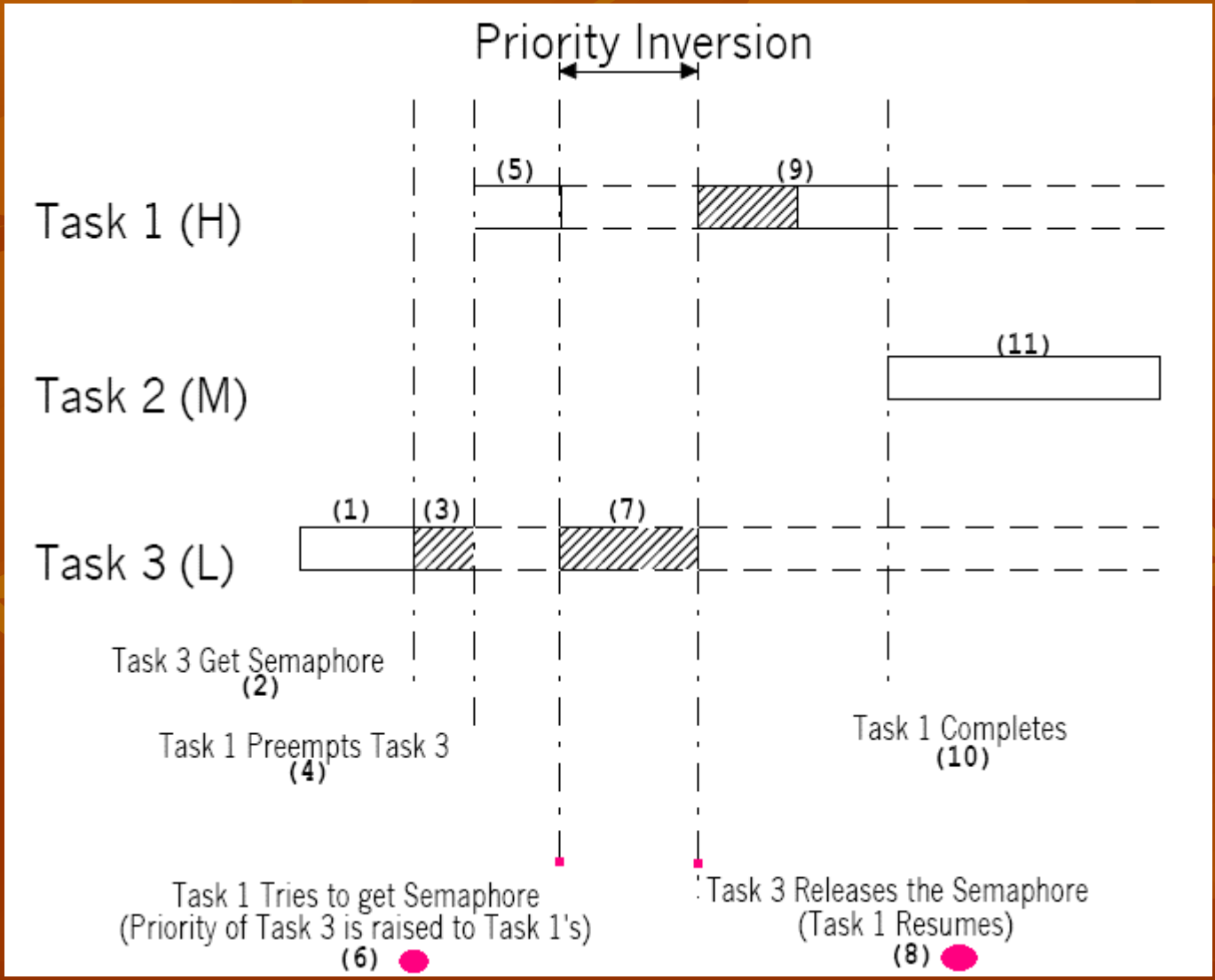
- ```
  OS_EVENT *ResourceMutex;
  OS_STK TaskPrio10Stk[1000];
  OS_STK TaskPrio15Stk[1000];
  OS_STK TaskPrio20Stk[1000];
  ```

- ```
  void main(void) {
      INT8U err;
      OSInit();
      ----- Application Initialization -----
      ResourceMutex = 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();
   }
  ```

Muxtex use example – TaskPrioNUM() (2/4)

- NUM = 10(task1) , 15(task2) ,and 20(task3) (Total: 3 funtions)
- void TaskPrioNUM(void *pdata){
  INT8U err;
  pdata = pdata;
  while(1){
  ------ Application code ------
  OSMutexPend(ResourceMutex , 0, &err);
  ------ Access common resource ------
  OSMutexPost(ResourceMutex);
  ------ Application code ------
  }
  }
- In this case, ***OSMutexPend()*** notices that a higher priority task needs the resource and raise the priority of task3 to ***9***,which forces a context switch back to task3.
- ***OSMutexPost()*** returns task3 to its original priority and notices task1 to access resource.

Mutex use example – Priority inheritance (3/4)

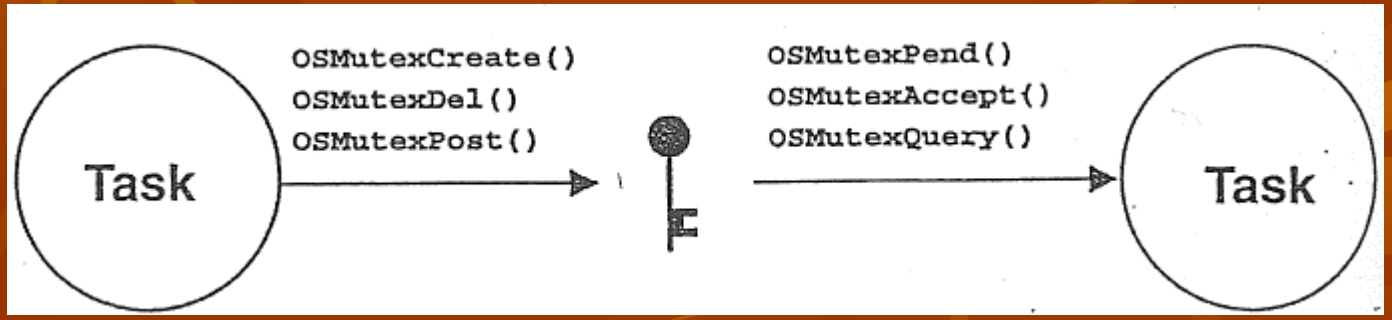


Mutex use example – Configuration (4/4)

- *None* of the mutex services are *enabled* when OS_MUTEX_EN is set to 0 in **OS_CFG.H**
- **Mutex config constants in OS_CFG.H**

µC/OS-II mutex service	*Enabled when set to* 1 *in* OS_CFG.H
OSMutexAccept()	OS_MUTEX_ACCEPT_EN
OSMutexCreate()	
OSMutexDel()	OS_MUTEX_DEL_EN
OSMutexPend()	
OSMutexPost()	
OSMutexQuery()	OS_MUTEX_QUERY_EN

- **Relationship between tasks and a mutex.**



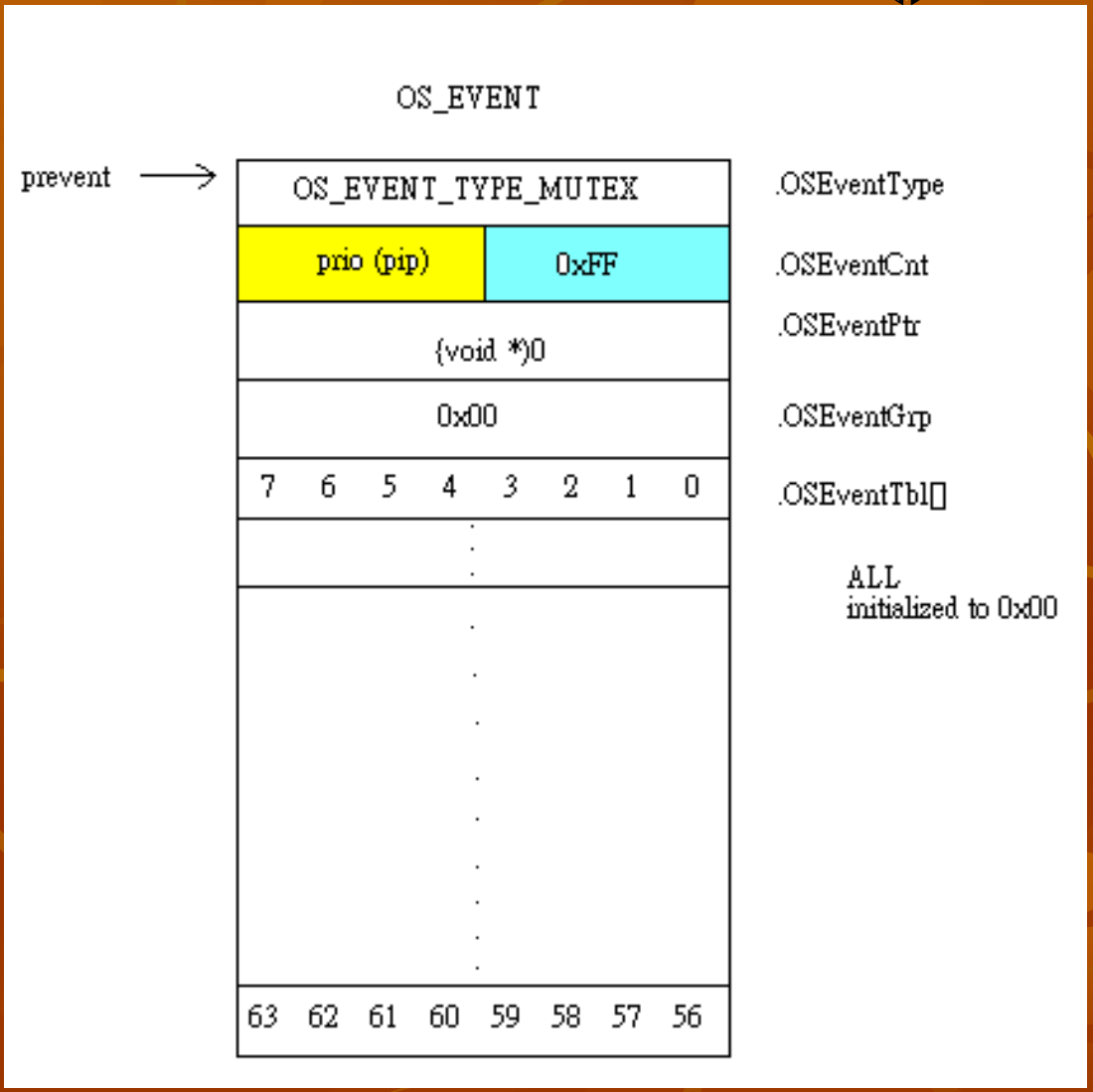
OSMutexCreate() (1/3)

```
OS_EVENT  *OSMutexCreate (INT8U prio, INT8U *err){
#if OS_CRITICAL_METHOD == 3
    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);
    }
```

OSMutexCreate() (2/3)

```
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);
}
```

OSMutexCreate() (3/3)



OSMutexDel() (1/3)

```
OS_EVENT  *OSMutexDel (OS_EVENT *pevent, INT8U opt, INT8U *err){
#if OS_CRITICAL_METHOD == 3
    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  */
    }
```

OSMutexDel() (2/3)

```
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);
        }
```

OSMutexDel() (3/3)

```
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);
  }
}
```

OSMutexPend() – successfully (1/3)

```
void  OSMutexPend (OS_EVENT *pevent, INT16U timeout, INT8U *err){
   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 <clean>*/
      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;  }
```

OSMutexPend() – unsuccessfully (2/3)

```
pip   = (INT8U)(pevent->OSEventCnt >> 8);                          /* No, Get PIP from mutex */
mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);
                                                                   /*  Get priority of mutex owner   */
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;  }
```

OSMutexPend() (3/3)

```
    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;    /*it's ok. don't need to wait */
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
}
```

OSMutexPost() (1/2)

```
INT8U  OSMutexPost (OS_EVENT *pevent) {
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    INT8U      pip;                                  /* Priority inheritance priority  */
    INT8U      prio;
    if (OSIntNesting > 0) {
        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'  */
```

OSMutexPost() (2/2)

```
        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;                 /* No task needs it now. */
      OS_EXIT_CRITICAL();
      return (OS_NO_ERR);  }
```

OSMutexAccept()

```
INT8U  OSMutexAccept (OS_EVENT *pevent, INT8U *err){

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 (clean) */
      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);  }
```

OSMutexQuery() (1/2)

- To allocate a variable of type OS_MUTEX_DATA.
- OS_MUTEX_DATA contains 1 .OSMutexPIP 2 .OSOwnerPrio 3. OSValue. 4 .else waiting info.

```
INT8U  OSMutexQuery (OS_EVENT *pevent, OS_MUTEX_DATA *pdata)  {
    INT8U     *psrc;
    INT8U     *pdest;
    if (OSIntNesting > 0) {return (OS_ERR_QUERY_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;
    }
```

OSMutexQuery() (2/2)

```
    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 > 7
    *pdest             = *psrc;
#endif
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Appendix

A. #define OS_MUTEX_AVAILABLE 0x00FF

B. .OSTCBY = priority >> 3;
//to determine in which group !

C. .OSTCBBitX = OSMapTbl[priority & 0x07];
//to determine in which bit of group !

- <BACK>

Thank you !!