uC/OS-II Part 7: Mutual Exclusion Semaphores

Real-Time Compting
Prof. Li-Pin Chang
ESSLab@NCTU

Mutual Exclusion Semaphores

- A mutex is used to synchronize tasks with priority inversion management
- If a LPT blocks a HPT, the priority of the LPT is boosted
  - Priority inheritance
  - With priority inheritance, multiple tasks might have the same priority

Mutual Exclusion Semaphores

- uC/OS-II uses an alternative approach to work around the priority duplication issue
  - Reserve a priority for a mutex
  - The reserved priority (for the mutex) must be higher than all tasks that use the mutex

Mutual Exclusion Semaphores

- Be careful about the priority for mutex!!
  - It should be immediately higher than the highest priority of all tasks use the mutex
  - To avoid affecting unrelated tasks

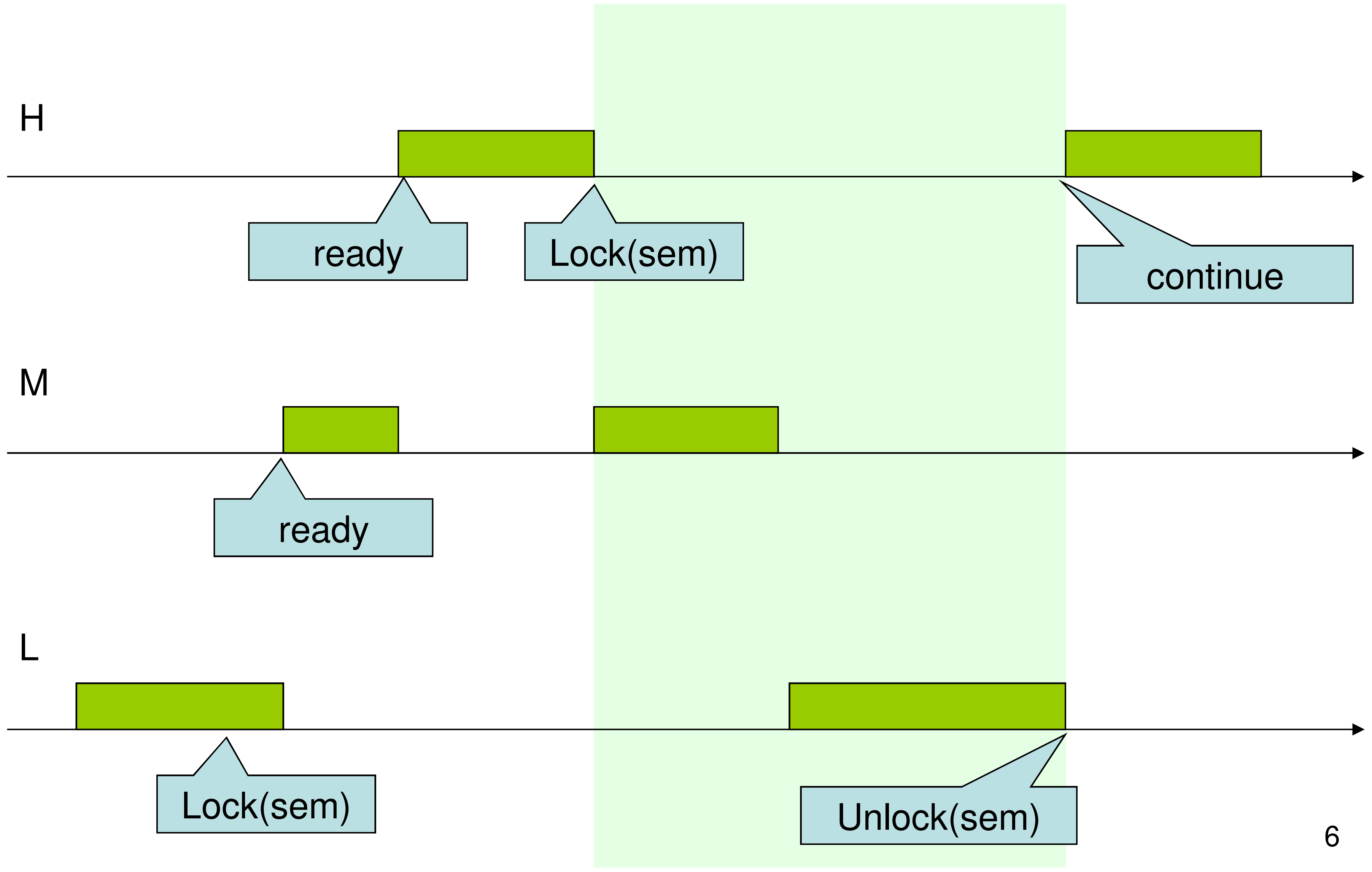
Example1

```
OSMutexCreate(VH, &err);

void taskPrioH {
        whle(1) {
                /*…*/
                OSMutexPend(mutex, 0, &err);
                /*…*/
                OSMutexPost(Mutex);
        }
}
```

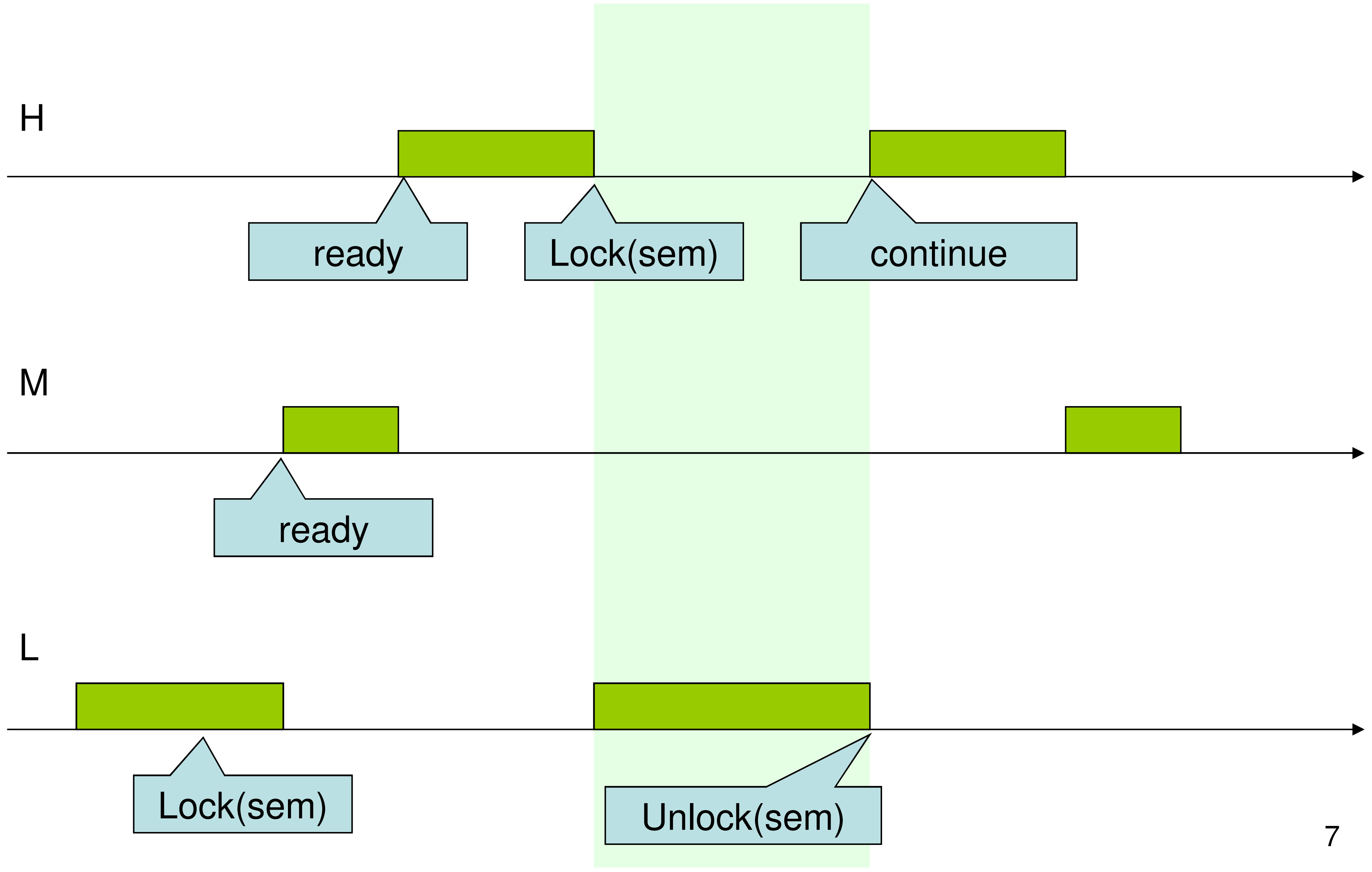
```
void taskPrioM {
        while(1) {
                /*…*/
        }
}
```

```
void taskPrioL {
        whle(1) {
                /*…*/
                OSMutexPend(mutex, 0, &err);
                /*…*/
                OSMutexPost(Mutex);
        }
}
```

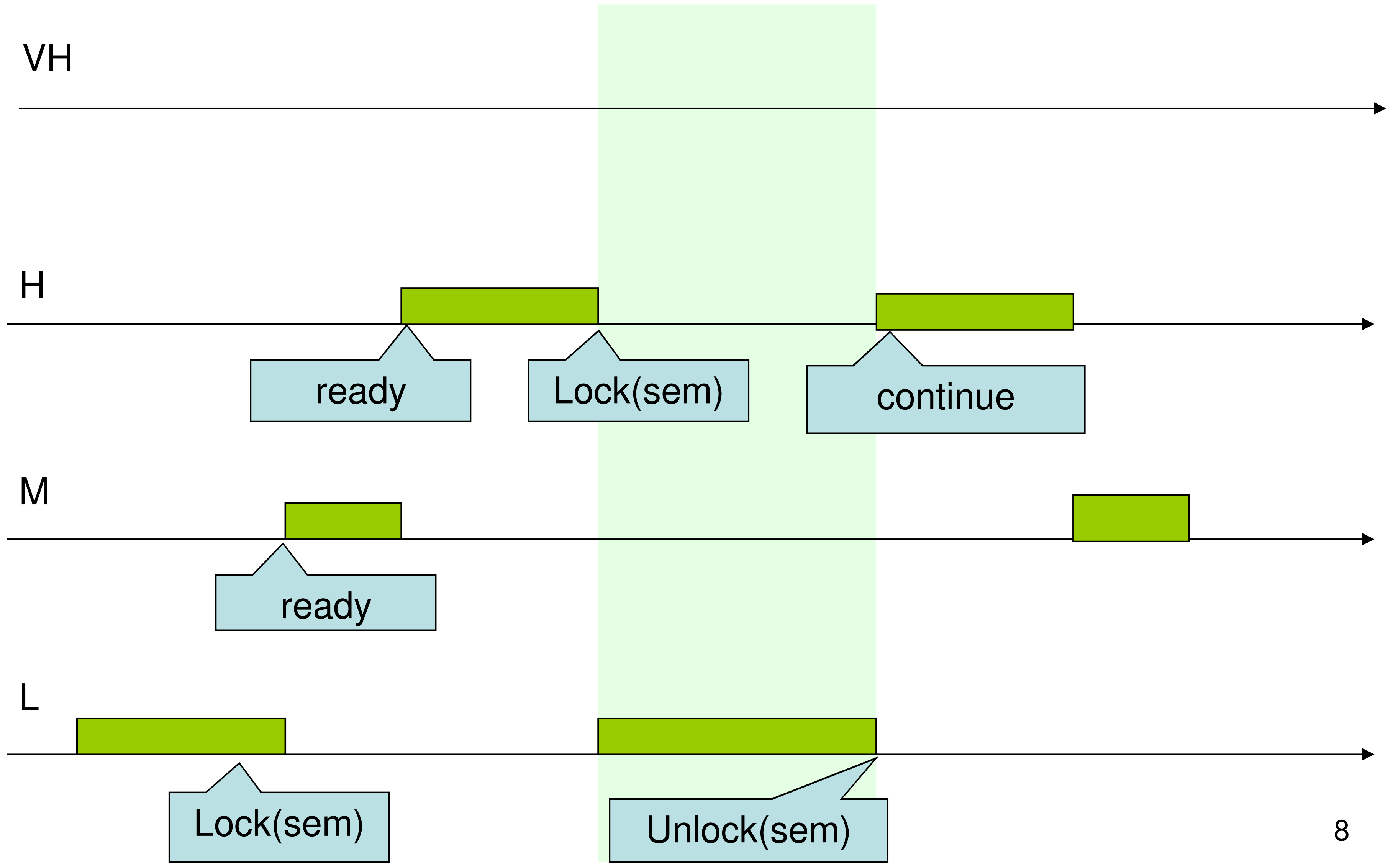
Example - without PIP



Example - with PIP



Example - µC/OS-II



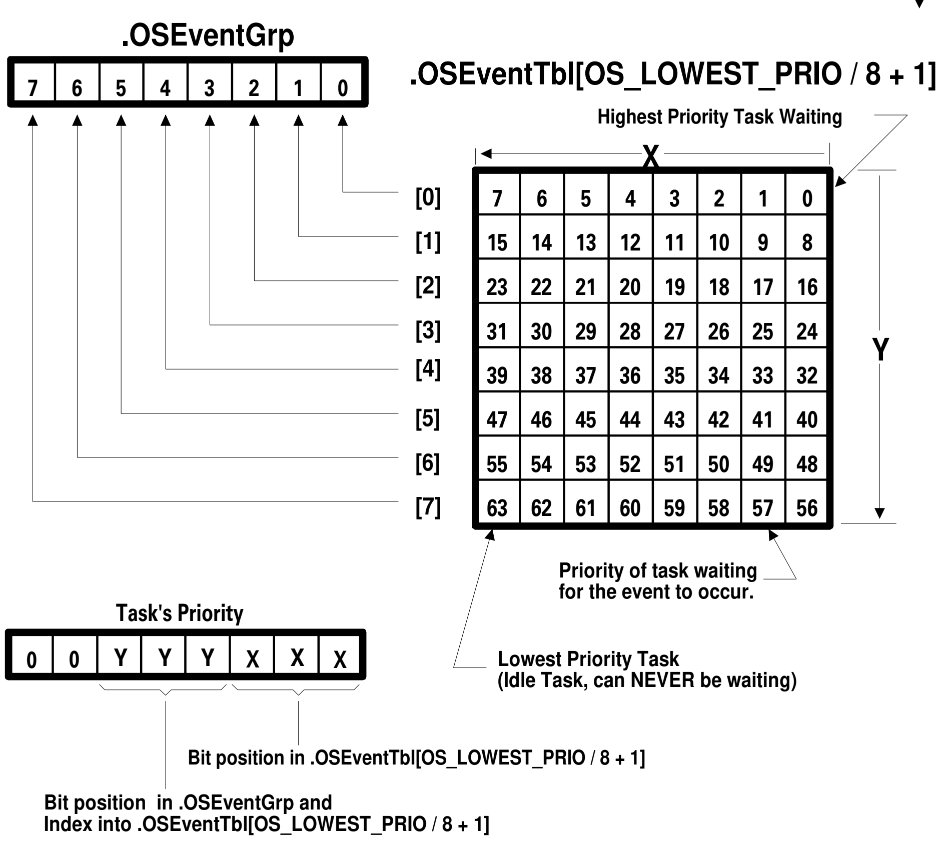
Mutual Exclusion Semaphores

- uC/OS-II
  - It differs from NPCS
    - The response is not that bad (not non-preemptible)
  - It differs from ceiling priority protocol (CPP)
    - CPP raises priority when resource lock is acquired
    - uC/OS-II raises priority when LPT blocks HPT
  - It differs from priority-inheritance protocol (PIP)
    - uc/OS-II has no incremental priority inheritance

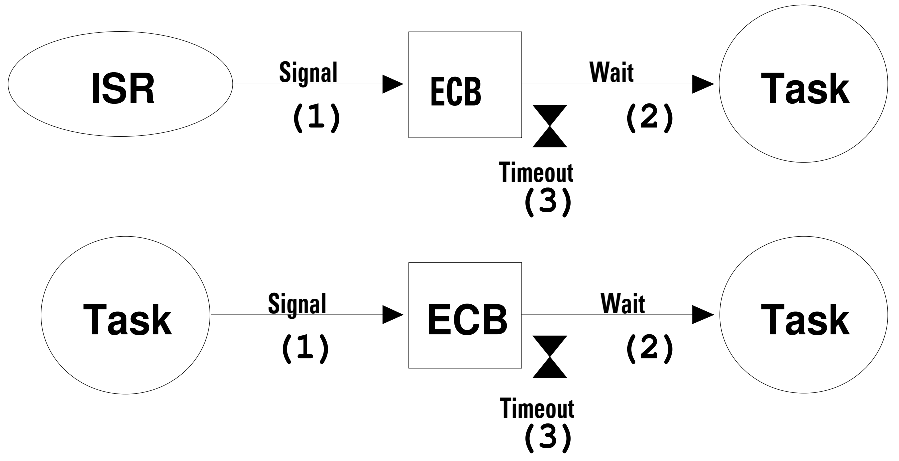
Functions

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

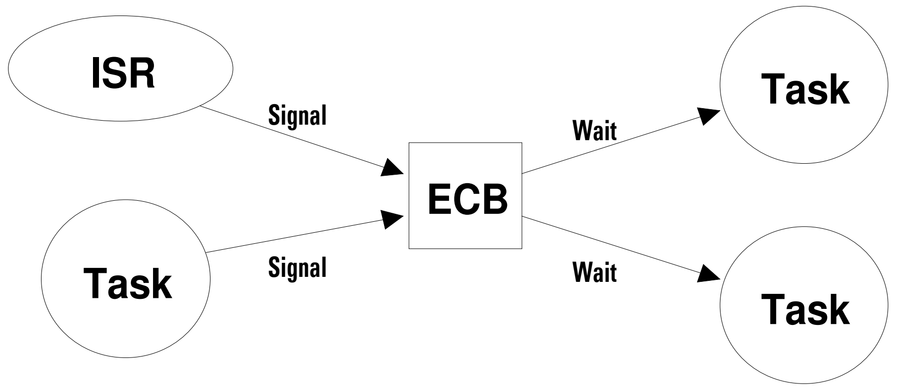
```
typedef struct {
   INT8U   OSEventType;               /* Event type                          */
   INT8U   OSEventGrp;                /* Group for wait list                 */
   INT16U  OSEventCnt;                /* Count (when event is a semaphore) */
   void   *OSEventPtr;                /* Ptr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



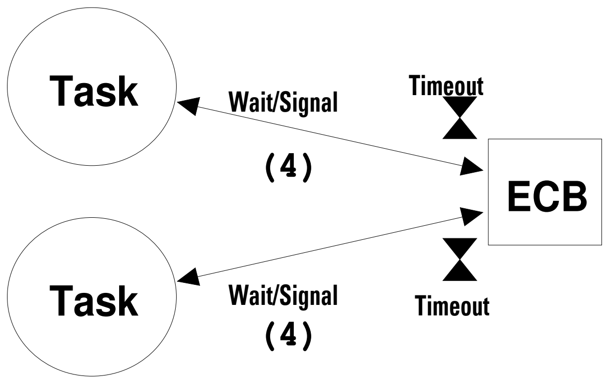
.OSEventGrp
7 6 5 4 3 2 1 0
.OSEventTbl[OS_LOWEST_PRIO / 8 + 1]
Highest Priority Task Waiting
X
Y
[0] [1] [2] [3] [4] [5] [6] [7]
7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8
23 22 21 20 19 18 17 16
31 30 29 28 27 26 25 24
39 38 37 36 35 34 33 32
47 46 45 44 43 42 41 40
55 54 53 52 51 50 49 48
63 62 61 60 59 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]



A



B

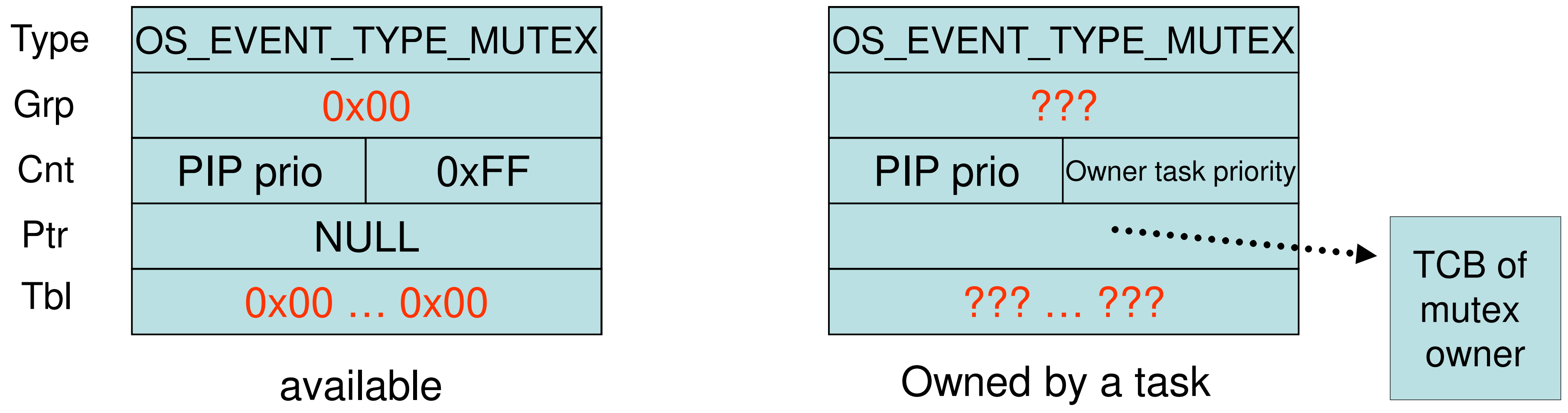


C

No pending in ISR

OSMutexCreate()

- OSMutexCreate() stores the PIP priority in OSEventCnt
  - OSEventCnt stores a value if the event is a semaphore



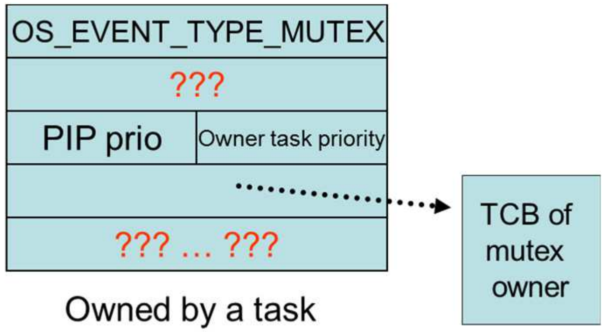
```
OS_EVENT  *OSMutexCreate (INT8U prio, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                                /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_EVENT  *pevent;


    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);
}
```

Reserve a priority by placing a non-null value

OSMutexPend()

```
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;
    }
    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 owner   */
    ptcb  = (OS_TCB *)(pevent->OSEventPtr);                     /*     Point to TCB of mutex owner   */
```

If the mutex is free…



```
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;
}
```

- If the owner's priority has not been raised, do it.
- If the owner's is current ready, change its bit from the ready-list bitmap

Set the ECB bitmap, clear itself from RdyMap

Locked the mutex without timed-out

pip = mutex's PIP priority

mprio = mutex owner's priority (original)

ptcb→OSTCBPrio = mutex owner's priority (current)

OSMutexPost()

```
INT8U  OSMutexPost (OS_EVENT *pevent) {
    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            */
    }
    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);
    }
```

The task posting the mutex must be the task that owns (has acquired) the mutex!!

OSMutexPost()

```
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);
}
```

Move the ready bit of the current task back to its original position

Release the highest waiting task and transfer the ownership of the mutex to the task

OSMutexDel()

```
OS_EVENT  *OSMutexDel (OS_EVENT *pevent, INT8U opt, INT8U *err) {
    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);
    }
    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);
             }
```

OSMutexDel()

```
        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);
    }
}
```

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);
    }
    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);
}
```

Summary

- In realistic systems, compromise exists between simplicity and performance
  - [PCP]→[PIP]→[CPP]→[NPCS]
- With uc/OS-II’s mutex, how is
  - Priority inversion management?
  - deadlock avoidance?