Chapter 7
Semaphore Management

System & Network Lab

Tsung-Yu Yeh

Outline

- Introduction
- Creating a Semaphore, OSSemCreate()
- Deleting a Semaphore, OSSemDel()
- Waiting on a Semaphore, OSSemPend()
- Signaling a Semaphore, OSSemPost()
- Getting a Semaphore without waiting, OSSemAccept()
- Obtaining the status of a Semaphore, OSSemQuery()

Introduction

- Semaphore consist two element – 16bits unsigned interger and a list of tasks waiting for the Semaphore count to be greater than 0.
- OSSemPend() ,OSSemDel() can’t be called by Interrup Service Routine .
- OSSemCreate(), OSSemPend(), OSSemPost() can’t be disabled individually.

Creating a Semaphore, OSSemCreate()

```
OS_EVENT  *OSSemCreate (INT16U cnt){
#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 ...               */
        return ((OS_EVENT *)0);                /* ... can't CREATE from an ISR             */
    }
    OS_ENTER_CRITICAL();
    pevent = OSEventFreeList;                  /* Get next free event control block        */
    if (OSEventFreeList != (OS_EVENT *)0) {    /* See if pool of free ECB pool was empty   */
        OSEventFreeList = (OS_EVENT *)OSEventFreeList->OSEventPtr;
    }
    OS_EXIT_CRITICAL();
    if (pevent != (OS_EVENT *)0) {             /* Get an event control block               */
        pevent->OSEventType = OS_EVENT_TYPE_SEM;
        pevent->OSEventCnt  = cnt;             /* Set semaphore value                      */
        pevent->OSEventPtr  = (void *)0;       /* Unlink from ECB free list                */
        OS_EventWaitListInit(pevent);          /* Initialize to 'nobody waiting' on sem.   */
    }
    return (pevent);
}
```

Deleting a Semaphore, OSSemDel() --1/3

```
OS_EVENT  *OSSemDel (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;
     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 (pevent);   }
     if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {       /* 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 semaphore    */
        tasks_waiting = TRUE;                    /* Yes                                      */
     } else {
      tasks_waiting = FALSE;                     /* No                                       */
     }
```

Deleting a Semaphore, OSSemDel() -- 2/3

```
switch (opt) {
        case OS_DEL_NO_PEND:                    /* Delete semaphore only if no task waiting */
            if (tasks_waiting == FALSE) {
                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();
                *err = OS_NO_ERR;
                return ((OS_EVENT *)0);                   /* Semaphore has been deleted               */
            } else {
                OS_EXIT_CRITICAL();
                *err = OS_ERR_TASK_WAITING;
                return (pevent);
            }
```

Deleting a Semaphore, OSSemDel() -- 3/3

```
case OS_DEL_ALWAYS:                          /* Always delete the semaphore            */
        while (pevent->OSEventGrp != 0x00) {         /* Ready ALL tasks waiting for semaphore    */
          OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM);
        }
        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);                      /* Semaphore has been deleted             */

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

Waiting on a Semaphore, OSSemPend() – 1/3

```
void  OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register      */
    OS_CPU_SR  cpu_sr;
#endif


    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_SEM) {   /* Validate event block type              */
        *err = OS_ERR_EVENT_TYPE;
        return;
    }
#endif
```

Waiting on a Semaphore, OSSemPend() – 2/3

```
OS_ENTER_CRITICAL();
  if (pevent->OSEventCnt > 0) {                /* If sem. is positive, resource available ...   */
     pevent->OSEventCnt--;                     /* ... decrement semaphore only if positive.     */
     OS_EXIT_CRITICAL();
     *err = OS_NO_ERR;
     return;
  }
                                               /* Otherwise, must wait until event occurs       */
  OSTCBCur->OSTCBStat |= OS_STAT_SEM;          /* Resource not available, pend on semaphore     */
  OSTCBCur->OSTCBDly   = timeout;              /* Store pend timeout in 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_SEM) {     /* Must have timed out if still waiting for event*/
     OS_EventTO(pevent);
     OS_EXIT_CRITICAL();
     *err = OS_TIMEOUT;                        /* Indicate that didn't get event within TO      */
     return;
  }
```

Waiting on a Semaphore, OSSemPend() – 3/3

```
OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
}
```

Signaling a Semaphore, OSSemPost() – 1/2

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


#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {             /* Validate 'pevent'                */
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {       /* Validate event block type
*/
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Signaling a Semaphore, OSSemPost() – 2/2

```
    OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0x00) {                    /* See if any task waiting for semaphore    */
        OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM);  /* Ready highest prio task waiting on
event */
        OS_EXIT_CRITICAL();
        OS_Sched();                                      /* Find highest priority task ready to run  */
        return (OS_NO_ERR);
    }
    if (pevent->OSEventCnt < 65535) {                    /* Make sure semaphore will not overflow        */
        pevent->OSEventCnt++;                            /* Increment semaphore count to register event   */
        OS_EXIT_CRITICAL();
        return (OS_NO_ERR);
    }
    OS_EXIT_CRITICAL();                                  /* Semaphore value has reached its maximum      */
    return (OS_SEM_OVF);
}
```

Getting a Semaphore without waiting, OSSemAccept() – 1/2

```
INT16U  OSSemAccept (OS_EVENT *pevent)
{
  #if OS_CRITICAL_METHOD == 3                  /* Allocate storage for CPU status register      */
     OS_CPU_SR  cpu_sr;
  #endif
     INT16U     cnt;
  #if OS_ARG_CHK_EN > 0
     if (pevent == (OS_EVENT *)0) {            /* Validate 'pevent'                             */
        return (0);
     }
     if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {   /* Validate event block type             */
       return (0);
     }
  #endif
```

Getting a Semaphore without waiting, OSSemAccept() – 2/2

```
    OS_ENTER_CRITICAL();
    cnt = pevent->OSEventCnt;
    if (cnt > 0) {                          /* See if resource is available              */
        pevent->OSEventCnt--;               /* Yes, decrement semaphore and notify caller  */
    }
    OS_EXIT_CRITICAL();
    return (cnt);                           /* Return semaphore count                    */
}
```

Obtaining the status of a Semaphore, OSSemQuery() – 1/3

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


#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {             /* Validate 'pevent'                        */
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {    /* Validate event block type        */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Obtaining the status of a Semaphore, OSSemQuery() – 2/3

```
 OS_ENTER_CRITICAL();
   pdata->OSEventGrp = pevent->OSEventGrp;          /* Copy message mailbox 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
```

Obtaining the status of a Semaphore, OSSemQuery() – 3/3

```
#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
    pdata->OSCnt     = pevent->OSEventCnt;          /* Get semaphore count                */
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

- The End