MicroC/OS-II Chapter 7

中興資科所 楊智閔
學號:79256029

CHAPTER 7 Semaphore Management

- Outline

Ch7
Semaphore Management

A semaphore

- allows a task to synchronize with either an ISR or a task
  (you initialize the semaphore to 0)
- gains exclusive access to a resource
  (you initialize the semaphore to a value greater than 0)
- signals the occurrence of an event
  (you initialize the semaphore to 0)

Table7.1
Semaphore configuration constants in OS_CFG.H

μC/OS-II Semaphore Service	Enabled when set to 1 in S_CFH.H
OSSemAccept()	OS_SEM_ACCEPT_EN
OSSemCreate()	
OSSemDel()	OS_SEM_DEL
OSSemPend()	
OSSemPost()	
OSSemQuery()	OS_SEM_QUERY_EN

7.00 Creating a Semaphore, [OSSemCreate()]

- You create a semaphore by calling OSSemCreate() and specifying the initial count of the semaphore.
- The initial value of a semaphore can be between 0 and 65,536
- If you use the semaphore to signal the occurrence of one or more events, you initialize the semaphore to 0
- If you use the semaphore to access a single shared resource, you need to initialize the semaphore to 1
- If the semaphore allows your application to obtain any one of n identical resources, initialize the semaphore to n and use it as a counting semaphore

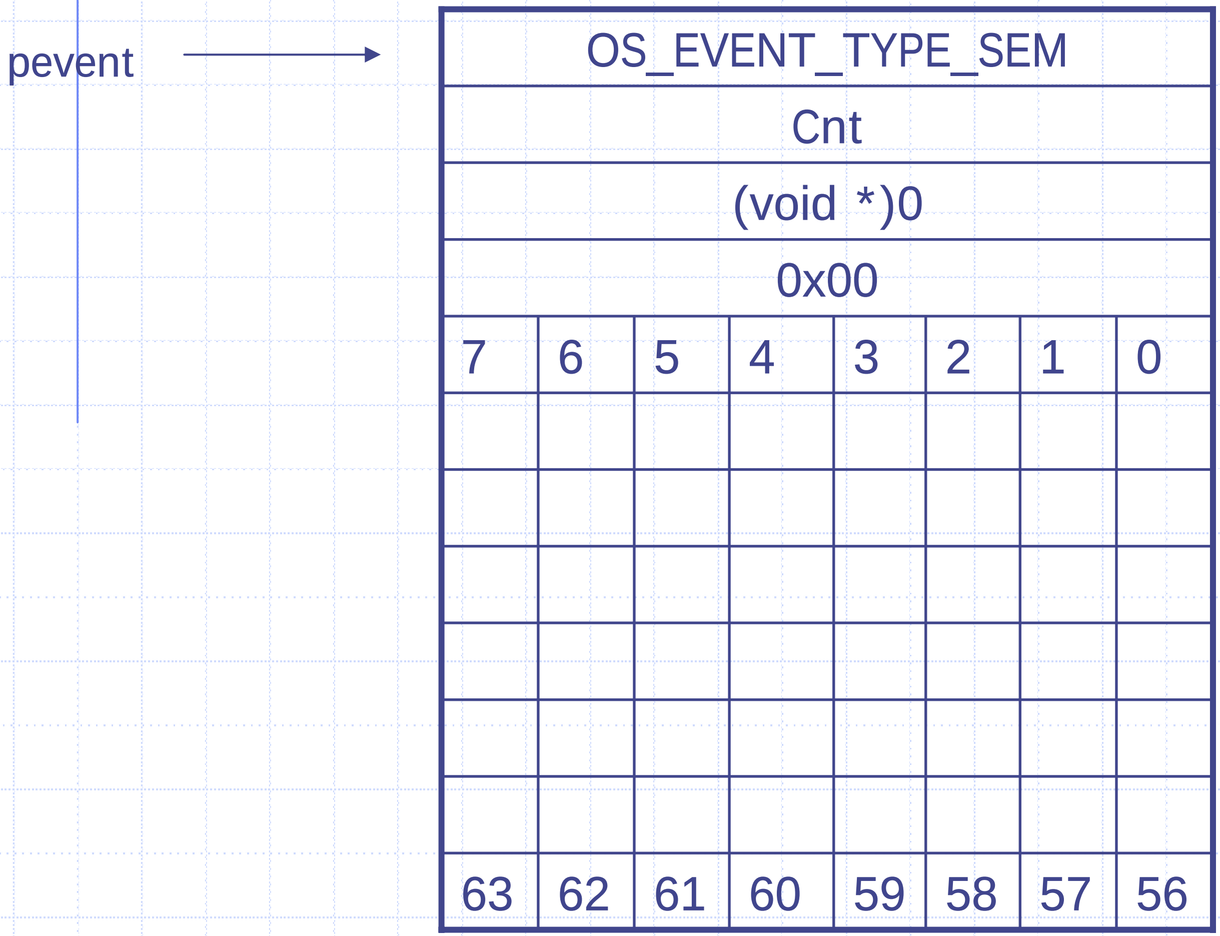
Listing7.1 Creating a Semaphore,[OSSemCreate]

```
OS_EVENT  *OSSemCreate (INT16U cnt)
{
#if  OS_CRITICAL_METHOD == 3
      OS_CPU_SR  cpu_sr;
#endif

    OS_EVENT  *pevent;

    if (OSIntNesting > 0) {
        return ((OS_EVENT *)0);
    }
    OS_ENTER_CRITICAL();
    pevent = OSEventFreeList;
   if (OSEventFreeList != (OS_EVENT *)0) {
 OSEventFreeList = (OS_EVENT *)OSEventFreeList->OSEventPtr;
   }
   OS_EXIT_CRITICAL();
```

Listing7.1 Creating a Semaphore,[OSSemCreate]

```
  if (pevent != (OS_EVENT *)0) {
      pevent->OSEventType = OS_EVENT_TYPE_SEM;
      pevent->OSEventCnt  = cnt;
      pevent->OSEventPtr  = (void *)0;
      OS_EventWaitListInit(pevent);
      }
     return (pevent);
}
```

Figure7.2 ECB just before OSSemCreate() returns.



pevent

OS_EVENT_TYPE_SEM

Cnt

(void *)0

0x00

7 6 5 4 3 2 1 0

63 62 61 60 59 58 57 56

.OSEventType

.OSEventCnt

.OSEventPtr

.OSEventGrp

ALL
initialized
to
0x00

7.01 Deleting a Semaphore,[OSSemDel()]

- OSSemDel() is used to delete a semaphore.
- This function is dangerous to use because multiple tasks could attempt to access a deleted semaphore.
- Before you delete a semaphore, you must first delete all the tasks that can access the semaphore.

Notes：

- Interrupts are disabled when pended tasks are readied, which means that interrupt latency depends on the number of task that are waiting on the semaphore.

Listing7.2 Deleting a Semaphore, [OSSemDel()]

```
OS_EVENT  *OSSemDel (OS_EVENT *pevent, INT8U opt, INT8U *err)
{
#if  OS_CRITICAL_METHOD == 3
      OS_CPU_SR  cpu_sr;
#endif

    BOOLEAN    tasks_waiting;
    if (OSIntNesting > 0) {
        *err = OS_ERR_DEL_ISR;
        return (pevent);
    }

#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        *err = OS_ERR_PEVENT_NULL;
        return (pevent);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
        *err = OS_ERR_EVENT_TYPE;
        return (pevent);
    }
#endif
```

Listing7.2 Deleting a Semaphore, [OSSemDel()]

```
OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0x00) {
        tasks_waiting = TRUE;
    } else {
        tasks_waiting = FALSE;
    }
    switch (opt) {
        case OS_DEL_NO_PEND:
            if (tasks_waiting == FALSE) {
                pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
                pevent->OSEventPtr  = OSEventFreeList;
                OSEventFreeList     = pevent;
                OS_EXIT_CRITICAL();
                *err = OS_NO_ERR;
                return ((OS_EVENT *)0);
            }
```

Listing7.2 Deleting a Semaphore, [OSSemDel()]

```
   else {
           OS_EXIT_CRITICAL();
             *err = OS_ERR_TASK_WAITING;
             return (pevent);
         }
case OS_DEL_ALWAYS:
         while (pevent->OSEventGrp != 0x00) {
             OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM);
         }
         pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
         pevent->OSEventPtr  = OSEventFreeList;
         OSEventFreeList     = pevent;
         OS_EXIT_CRITICAL();
```

Listing7.2 Deleting a Semaphore, [OSSemDel()]

```
 if (tasks_waiting == TRUE) {
                 OS_Sched();
             }
             *err = OS_NO_ERR;
             return ((OS_EVENT *)0);

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

7.02 Waiting on a Semaphore, [OSSemPend()]

- OSSemPend() is used when task want exclusive access to a resource, needs to synchronize its activities with an ISR or a task, or is waiting until an event occurs.
- If a task calls OSSemPend() and the value of the semaphore is greater than 0, OSSemPend() decrements the semaphore and returns to its caller.
- If the values of the semaphore is 0, OSSemPend() places the calling task in the waiting list for the semaphore.
- The task waits until a task or an ISR signals the semaphore or the specified timeout expires.
- If the semaphore is signal, μC/OS-II resumes the highest priority task waiting for the semaphore

Listing7.3 Waiting on a Semaphore, [OSSemPend()]

```
void  OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
{
#if  OS_CRITICAL_METHOD == 3
      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) {
          *err = OS_ERR_PEVENT_NULL;
          return (pevent);
     }
     if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
          *err = OS_ERR_EVENT_TYPE;
          return (pevent);
     }
#endif
```

Listing7.3 Waiting on a Semaphore, [OSSemPend()]

```
OS_ENTER_CRITICAL();

    if (pevent->OSEventCnt > 0) {
        pevent->OSEventCnt--;
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return;
    }
    OSTCBCur->OSTCBStat |= OS_STAT_SEM;
    OSTCBCur->OSTCBDly    = timeout;
    OS_EventTaskWait(pevent);
    OS_EXIT_CRITICAL();
    OS_Sched();                                            /* Find next highest priority
    task ready */

    OS_ENTER_CRITICAL();
    if (OSTCBCur->OSTCBStat & OS_STAT_SEM) {
        OS_EventTO(pevent);
        OS_EXIT_CRITICAL();
        *err = OS_TIMEOUT;
        return;
    }
```

Listing7.3 Waiting on a Semaphore, [OSSemPend()]

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

7.03 Signaling on a Semaphore, [OSSemPost()]

- A semaphore is signaled by calling OSSemPost().
- If the semaphore value is 0 or more, it is incremented.
- If the task are waiting for the semaphore to be signaled, OSSemPost() removes the highest priority task from the waiting list and make this task ready to run

Listing7.4 Signaling on a Semaphore, [OSSemPend()]

```
INT8U  OSSemPost (OS_EVENT *pevent)
{
#if  OS_CRITICAL_METHOD == 3
      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

        OS_ENTER_CRITICAL();
        if (pevent->OSEventGrp != 0x00) {
        OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM);
        OS_EXIT_CRITICAL();
        OS_Sched();
        return (OS_NO_ERR);
       }
```

Listing7.4 Signaling on a Semaphore, [OSSemPend()]

```
if (pevent->OSEventCnt < 65535) {
        pevent->OSEventCnt++;
        OS_EXIT_CRITICAL();
        return (OS_NO_ERR);
    }
    OS_EXIT_CRITICAL();
    return (OS_SEM_OVF);
}
```

7.04, Getting a Semaphore Without Waiting, [OSSemAccept()]

- OSSemAccept() cgecjs to see if a resource is available or an event has occurred.
- But it does not suspend the calling task if the resource is not available.

```
Example
OS_EVENT *DispSem;

Void Task (void *pdata)
{
   INT16U value;
   pdata =pdata;
  for(;;){
   value =OSSemAccept(DispSem);
     if (value > 0){                                  /*Resource available,  process..*/
     }
     …
  }

}
```

Listing 7.5

Getting a Semaphore Without Waiting, [OSSemAccept()]

```
INT16U  OSSemAccept (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3
     OS_CPU_SR  cpu_sr;
#endif
     INT16U     cnt;
#if   OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        return (0);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
    return (0);
    }
#endif
```

Listing 7.5

Getting a Semaphore Without Waiting, [OSSemAccept()]

```
    OS_ENTER_CRITICAL();
    cnt = pevent->OSEventCnt;
    if (cnt > 0){
    pevent->OSEventCnt--;
    }
    OS_EXIT_CRITICAL();
    return (cnt);
}
```

7.05
Obtaining the Status of a Semaphore, [OSSemQuery]

- OSSemQuery() receives tow arguments：
  pevent contains a pointer to the semaphore.
  pdata is a pointer to a data structure (OS_SEM_DATA) that hold information about the semaphore.

  pdata: contains the following fields

  INT16U OSCnt;
  INT8U OSEvenTbl[OS_EVENT_TBL_SIZE];
  INT8U OSEventGrp;

Listing 7.6
Obtaining the Status of a Semaphore, [OSSemQuery]

```
INT8U  OSSemQuery (OS_EVENT *pevent, OS_SEM_DATA *pdata)
{
    #if OS_CRITICAL_METHOD == 3
    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) {
      return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Listing 7.6
Obtaining the Status of a Semaphore, [OSSemQuery]

```
OS_ENTER_CRITICAL();
    pdata->OSEventGrp = pevent->OSEventGrp;
    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
```

Listing 7.6

Obtaining the Status of a Semaphore, [OSSemQuery]

```
#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

    pdata->OSCnt      = pevent->OSEventCnt;
    OS_EXIT_CRITICAL();

    return (OS_NO_ERR);
}
```