Chap 7
Semaphore Management

Ming-Hsin Ho

2008/7/29

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

(1/2)

- MicroC/OS-II's semaphores consist of two elements:
  - a 16-bit unsigned integer used to hold the semaphore count
  - a list of tasks waiting for the semaphore
- MicroC/OS-II provides six services to access semaphores:
  - OSSemAccept()、OSSemCreate()、OSSemDel()、OSSemPend()、OSSemPost()、OSSemQuery()
- Interrupt Service Routine can not use:
  - OSSemCreate()、OSSemDel()、 OSSemPend()

Introduction

(2/2)

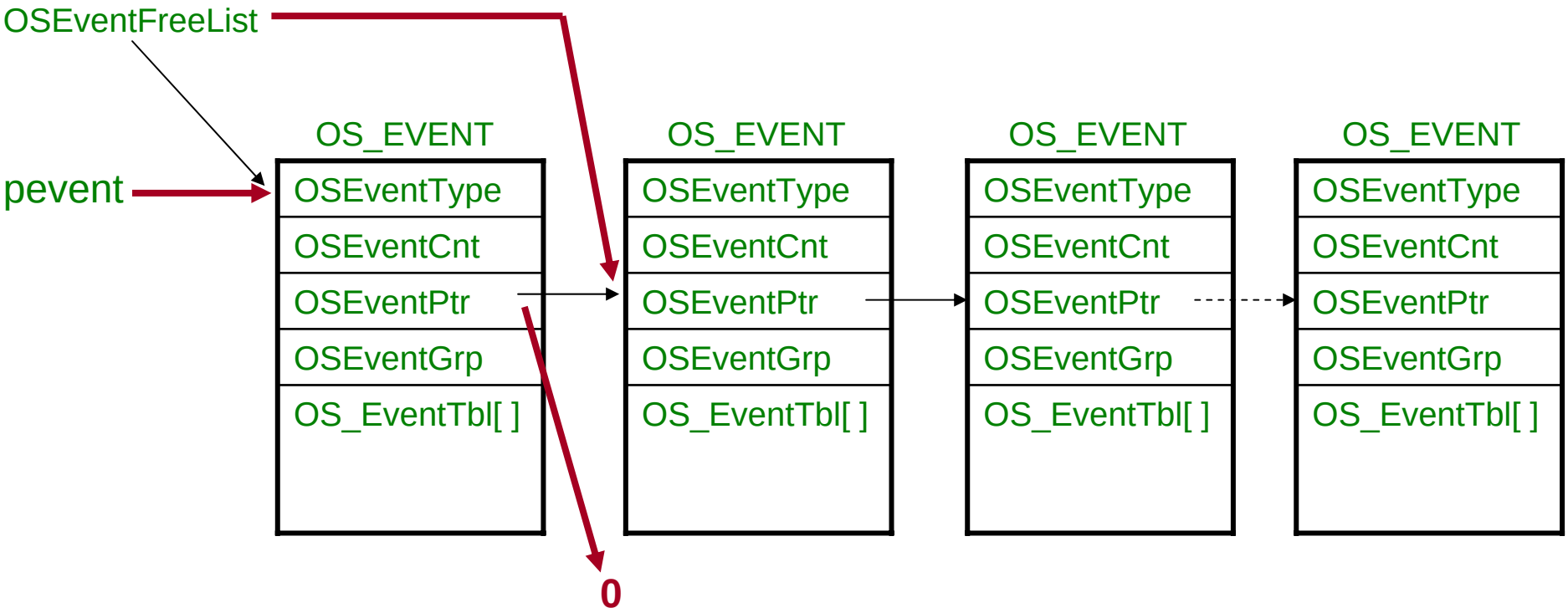
MicroC/OS-II Semaphore Service	Enable when set to 1 in OS_CFG.H
OSSemAccept()	OS_SEM_ACCEPT_EN
OSSemCreate()	
OSSemDel()	OS_SEM_DEL_EN
OSSemPend()	
OSSemPost()	
OSSemQuery()	OS_SEM_QUERY_EN

To enable MicroC/OS-II's semaphore services, you must set the configuration constant OS_SEM_EN to 1.

Creating a Semaphore
OSSemCreate() (1/3)

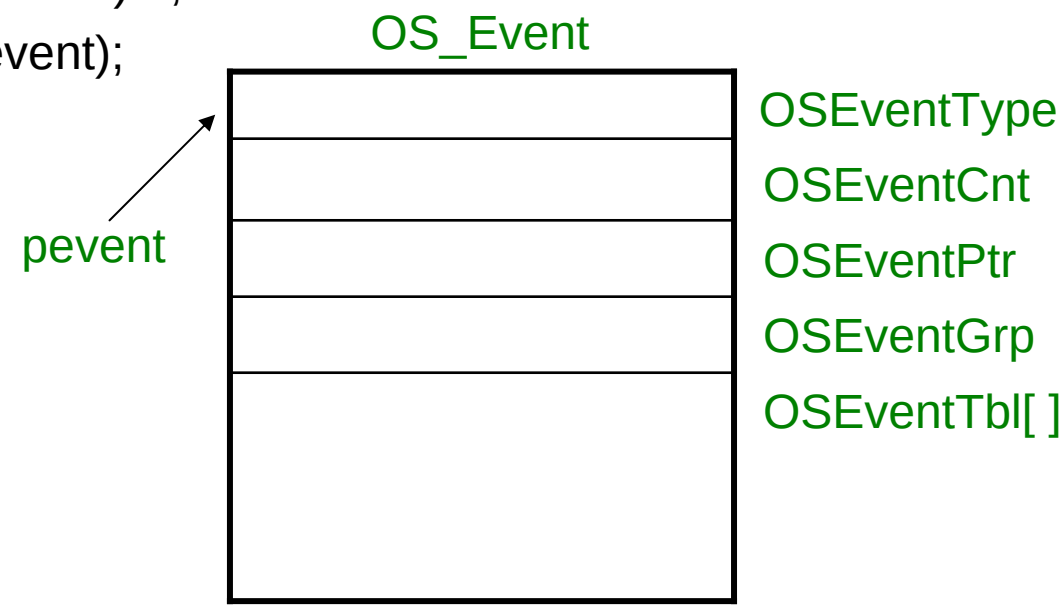
```
 1  OS_EVENT *OSSemCreate (INT16U cnt)
 2  {
 3  # if OS_CRITICAL_METHOD == 3
 4      OS_CPU_SR cpu_sr;
 5  #endif
 6      OS_EVENT *pevent;
 7      if (OSIntNesting > 0) {
 8          return ((OS_EVENT *) 0);
 9      }
10      OS_ENTER_CRITICAL();
11      pevent = OSEventFreeList;
12      if (OSEventFreeList != (OS_EVENT *) 0) {
13          OSEventFreeList = (OS_EVENT *) OSEventFreeList->OSEventPtr;
14      }
15      OS_EXIT_CRITICAL();
```

Creating a Semaphore
OSSemCreate() (2/3)



Creating a Semaphore
OSSemCreate() (3/3)

```
16      if (pevent != (OS_EVENT *) 0) {
17          pevent->OSEventType = OS_EVENT_TYPE_SEM;
18          pevent->OSEventCnt = cnt;
19          pevent->OSEventPtr = (void *) 0;
20          OS_EventWaitListInit(pevent);
21      }
22      return (pevent);
23  }
```



Deleting a Semaphore
OSSemDel() (1/5)

```
 1  OS_EVENT *OSSemDel (OS_EVENT *pevent, INT8U opt, INT8U *err)
 2  {
 3  #if OS_CRITICAL_METHOD == 3
 4      OS_CPU_SR cpu_sr;
 5  #endif
 6      BOOLEAN tasks_waiting;
 7      If (OSIntNesting > 0) {
 8          *err = OS_ERR_DEL_ISR;
 9          return (pevent);
10    }
11  #if OS_ARG_CHK_EN > 0
12      if (pevent == (OS_EVENT *) 0) {
13          *err = OS_ERR_PEVENT_NULL;
14          return (pevent);
15      }
```

OS_DEL_NO_PEND
OS_DEL_ALWAYS

Deleting a Semaphore
OSSemDel() (2/5)

```
16      if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
17          *err = OS_ERR_ENENT_TYPE;
18          return (pevent);
19      }
20  #endif
21      OS_ENTER_CRITICAL();
22      if (pevent->OSEventGrp != 0x00) {
23          tasks_waiting = TRUE;
24      } else {
25          tasks_waiting = FALSE;
26      }
```

Deleting a Semaphore
OSSemDel() (3/5)

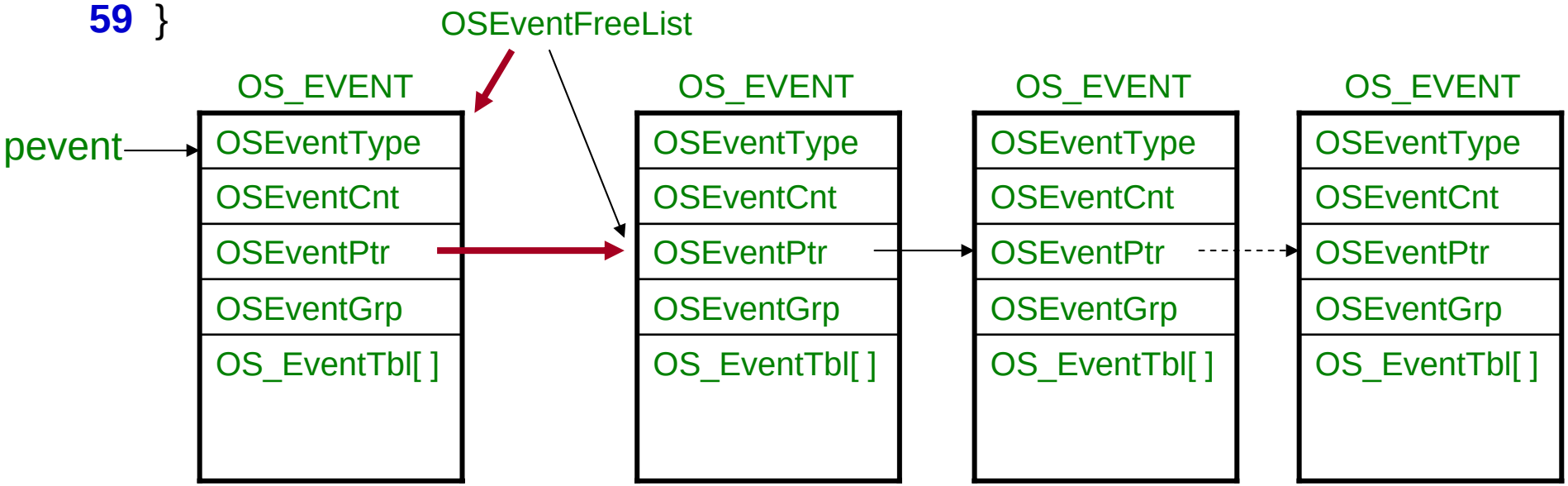
```
27      switch (opt) {
28          case OS_DEL_NO_PEND:
29              if (task_waiting == FALSE) {
30                  pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
31                  pevent->OSEventPtr = OSEventFreeList;
32                  OSEventFreeList = pevent;
33                  OS_EXIT_CRITICAL();
34                  *err = OS_NO_ERR;
35                  return ((OS_EVENT *) 0);
36              } else {
37                  OS_EXIT_CRITICAL();
38                  *err = OS_ERR_TASK_WAITING;
39                  return (pevent);
40              }
```

Deleting a Semaphore
OSSemDel() (4/5)

```
41          case OS_DEL_ALWAYS:
42              while (pevent->OSEventGrp != 0x00) {
43                  OS_EventTaskRdy(pevent, (void *) 0, OS_STAT_SEM);
44              }
45              pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
46              pevent->OSEventPtr = OSEventFreeList;
47              OSEventFreeList = pevent;
48              OS_EXIT_CRITICAL();
49              if (tasks_waiting == TRUE) {
50                  OS_Sched();
51              }
52              *err = OS_NO_ERR;
53              return ((OS_EVENT *) 0);
```

Deleting a Semaphore
OSSemDel() (5/5)

```
54          default:
55              OS_EXIT_CRITICAL();
56              *err = OS_ERR_INVALID_OPT;
57              return (pevent);
58      }
59  }
```



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

```
1  void OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
2  {
3  #if OS_CRITICAL_METHOD == 3
4      OS_CPU_SR cpu_sr;
5  #endif
6      if (OSIntNesting > 0) {
7          *err = OS_ERR_PEND_ISR;
8          return;
9      }
10 #if OS_ARG_CHK_EN > 0
11     if (pevent == (OS_EVENT *) 0) {
12         *err = OS_ERR_PEVENT_NULL;
13         return;
14     }
```

0
<=65535 ticks

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

```
15      if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
16          *err = OS_ERR_EVENT_TYPE;
17          return;
18      }
19  #endif
20      OS_ENTER_CRITICAL();
21      if (pevent->OSEventCnt > 0) {
22          pevent->OSEventCnt--;
23          OS_EXIT_CRITICAL();
24          *err = OS_NO_ERR;
25          return;
26      }
27      OSTCBCur->OSTCBStat |= OS_STAT_SEM;
28      OSTCBCur->OSTCBDly = timeout;
```

OSTimeTick()

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

```
29      OS_EventTaskWait(pevent);
30      OS_EXIT_CRITICAL();
31      OS_Sched();
32      OS_ENTER_CRITICAL();
33      if (OSTCBCur->OSTCBStat & OS_STAT_SEM) {
34          OS_EventTO(pevent);
35          OS_EXIT_CRITICAL();
36          *err = OS_TIMEOUT;
37          return;
38      }
39      OSTCBCur->OSTCBEventPtr = (OS_EVENT *) 0;
40      OS_EXIT_CRITICAL();
41      *err = OS_NO_ERR;
42  }
```

OS_EventTaskWait()
OSTCBCur->OSTCBEventPtr = pevent

Signaling a Semaphore
OSSemPost() (1/2)

```
 1  INT8U OSSemPost (OS_EVENT *pevent)
 2  {
 3  #if OS_CRITICAL_METHOD == 3
 4      OS_CPU_SR cpu_sr;
 5  #endif
 6  #if OS_ARG_CHK_EN > 0
 7      if (pevent == (OS_EVENT *) 0) {
 8          return (OS_ERR_PEVENT_NULL);
 9      }
10      if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
11          return (OS_ERR_EVENT_TYPE);
12      }
13  #endif
14      OS_ENTER_CRITICAL();
```

Signaling a Semaphore
OSSemPost() (2/2)

```
15      if (pevent->OSEventGrp != 0x00) {
16          OS_EventTaskRdy(pevent, (void *) 0, OS_STAT_SEM);
17          OS_EXIT_CRITICAL();
18          OS_Sched();
19          return (OS_NO_ERR);
20      }
21      if (pevent->OSEventCnt < 65535) {
22          pevent->OSEventCnt++;
23          OS_EXIT_CRITICAL();
24          return (OS_NO_ERR);
25      }
26      OS_EXIT_CRITICAL();
27      return (OS_SEM_OVF);
```

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

```
 1  INT16U OSSemAccept (OS_EVENT *pevent)
 2  {
 3  #if OS_CRITICAL_METHOD == 3
 4      OS_CPU_SR cpu_sr;
 5  #endif
 6      INT16U cnt;
 7  #if OS_ARG_CHK_EN > 0
 8      if (pevent == (OS_EVENT *) 0) {
 9          return (0);
10      }
11      if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
12          return (0);
13      }
14  #endif
```

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

```
14      OS_ENTER_CRITICAL();
15      cnt = pevent->OSEventCnt;
16      if (cnt > 0) {
17          pevent->OSEventCnt--;
18      }
19      OS_EXIT_CRITICAL();
20      return (cnt);
21  }
```

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

```
 1 INT8U OSSemQuery (OS_EVENT *pevent, OS_SEM_DATA *pdata)
 2 {
 3 #if OS_CRITICAL_METHOD == 3
 4     OS_CPU_SR cpu_sr;
 5 #endif
 6     INT8U *psrc;
 7     INT8U *pdest;
 8 #if OS_ARG_CHK_EN > 0
 9     if (pevent == (OS_EVENT *) 0) {
10         return (OS_ERR_PEVENT_NULL);
11     }
12     if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {
13         return (OS_ERR_EVENT_TYPE);
14     }
15 #endif
```

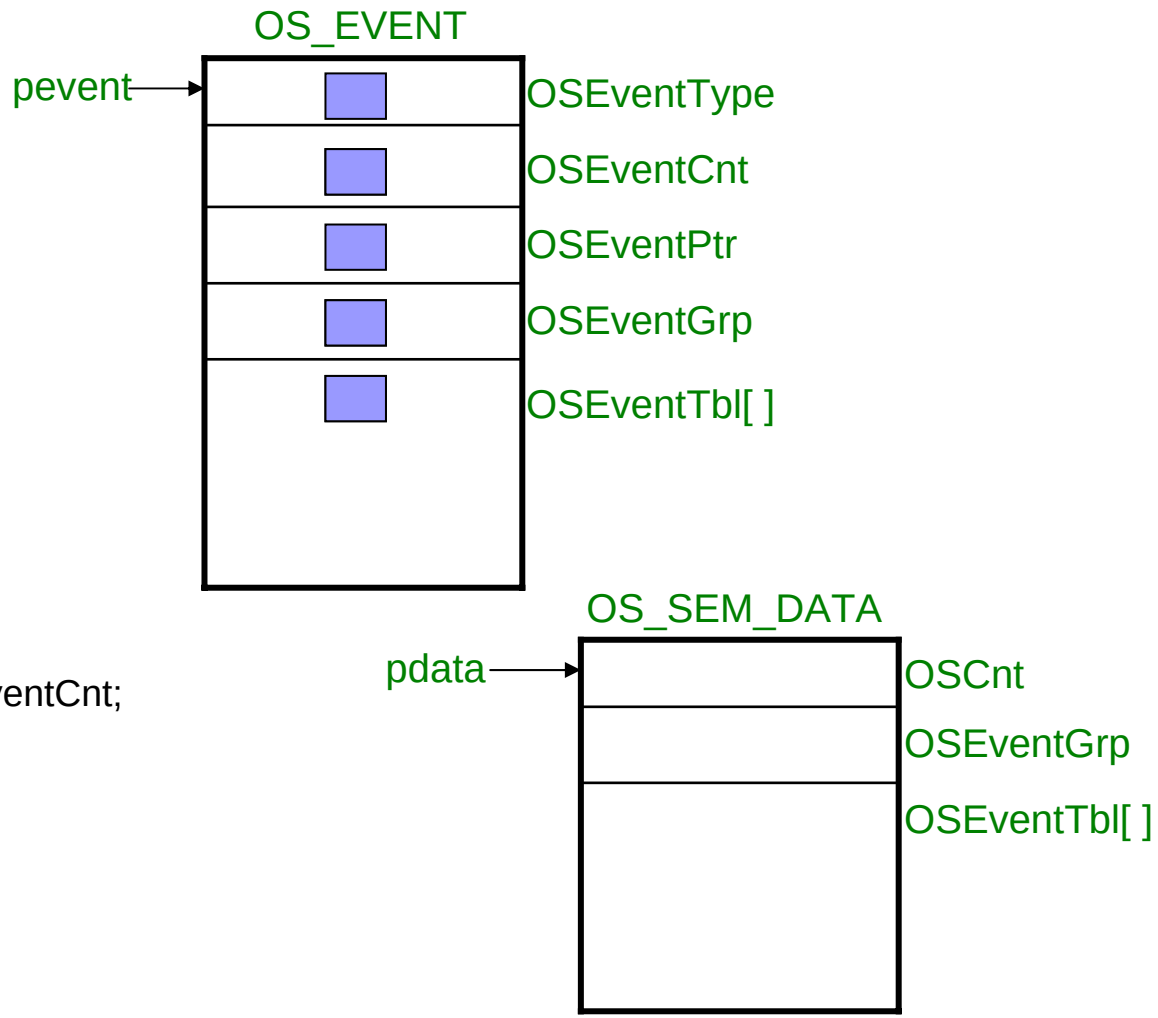
OSCnt
OSEventGrp
OSEventTbl[]

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

```
16      OS_ENTER_CRITICAL();
17      pdata->OSEventGrp = pevent->OSEventGrp;
18      psrc = &pevent->OSEventTbl[0];
19      pdest = &pdata->OSEventTbl[0];
20  #if OS_EVENT_TBL_SIZE > 0
21      *pdest++ = *psrc++;
22  #endif
23  # if OS_EVENT_TBL_SIZE > 1
24      *pdest++ = *psrc++;
25  #endif
26  # if OS_EVENT_TBL_SIZE > 2
27      *pdest++ = *psrc++;
28  #endif
29  #if OS_EVENT_TBL_SIZE > 3
30      *pdest++ = *psrc++;
31  #endif
```

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

```
32  #if OS_EVENT_TBL_SIZE > 4
33      *pdest++ = *psrc++;
34  #endif
35  #if OS_EVENT_TBL_SIZE > 5
36      *pdest++ = *psrc++;
37  #endif
38  #if OS_EVENT_TBL_SIZE > 6
39      *pdest++ = *psrc++;
40  #endif
41  #if OS_EVENT_TBL_SIZE > 7
42      *pdest++ = *psrc++;
43  #endif
44      pdata->OSCnt = pevent->OSEventCnt;
45      OS_EXIT_CRITICAL();
46      return (OS_NO_ERR);
47  }
```



The End