6 Semaphore Management

Contents

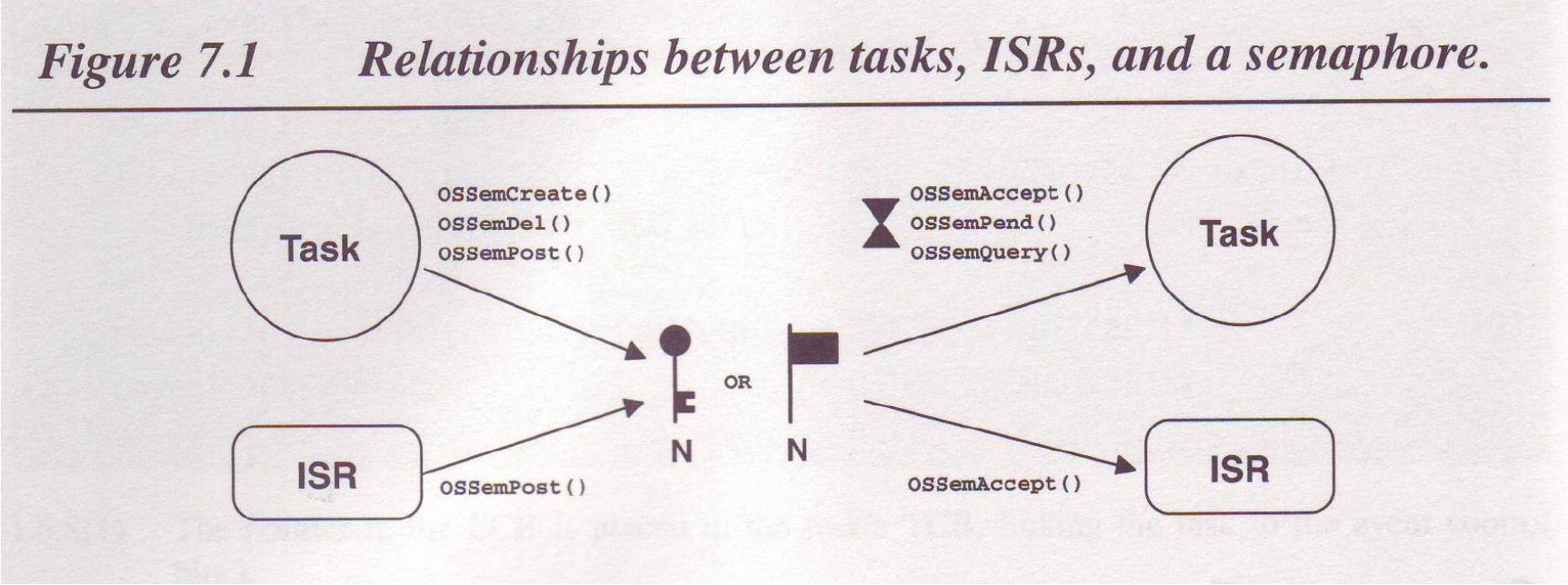
- Creating a Semaphore: `OSSemCreate()`
- Deleting a Semaphore: `OSSemDel()`
- Waiting on a Semaphore (Blocking): `OSSemPend()`
- Signaling a Semaphore: `OSSemPost()`
- Getting a Semaphore without Waiting (Non-blocking): `OSSemAccept()`

Semaphore Configuration

Table 7.1 Semaphore configuration constants in OS_CFG.H.

µC/OS-II Semaphore Service	*Enabled 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

Relationship between Tasks, ISRs, and a Semaphore



Creating a Semaphore: OSSemCreate()

```
OS_EVENT *OSSemCreate(INT16U cnt)
{
  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();
  if (pevent != (OS_EVENT *)0) {
    pevent->OSEventType = OS_EVENT_TYPE_SEM;
    pevent->OSEventCnt  = cnt;
    pevent->OSEventPtr  = (void *)0;
    OS_EventWaitListInit(pevent);
  }
  return (pevent);
}
```

ISR can not call this function

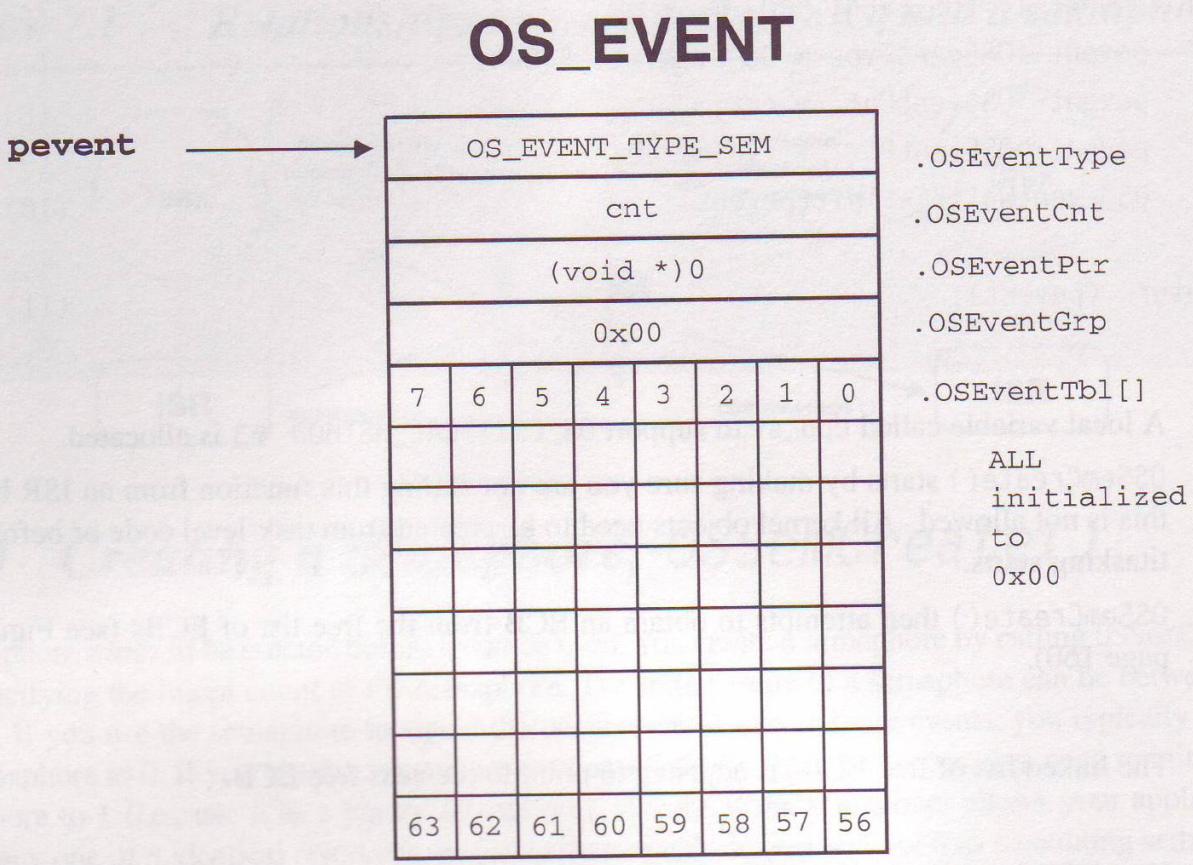
Get an ECB from OSEventFreeList

Initialize the ECB

Return the ECB

ECB Created by `OSSemCreate()`

Figure 7.2 ***ECB just before `OSSemCreate()` returns.***



Deleting a Semaphore: OSSemDel()

```
OS_EVENT  *OSSemDel (OS_EVENT *pevent, INT8U opt, INT8U
  *err)
{
  BOOLEAN tasks_waiting;
  ...
  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);
    } else {
      OS_EXIT_CRITICAL();
      *err = OS_ERR_TASK_WAITING;
      return (pevent);
    }
```

check if any task is waiting

delete if there's no pending task

if there's no waiting task then delete the semaphore

if any task is waiting then it is an error

Deleting a Semaphore: `OSSemDel()`

```
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();
   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);
 }
}
```

delete always

make all waiting tasks ready

delete the semaphore and call scheduler

Waiting on a Semaphore (Blocking): OSSemPend()

```
void  OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err) {
  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();
  OS_ENTER_CRITICAL();
  if (OSTCBCur->OSTCBStat & OS_STAT_SEM) {
    OS_EventTO(pevent);
    OS_EXIT_CRITICAL();
    *err = OS_TIMEOUT;
    return;
  }
  OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;
  OS_EXIT_CRITICAL();
  *err = OS_NO_ERR;
}
```

if the semaphore is available, the count is decremented

set some fields of TCB and make the task wait the event and call scheduler

if the timeout period expires (OS_STAT_SEM), calls OS_EventTO()

else (the semaphore is signaled) make the TCB with no event and returns

Signaling a Semaphore: `OSSemPost()`

```
INT8U  OSSemPost (OS_EVENT *pevent)
{
  OS_ENTER_CRITICAL();
  if (pevent->OSEventGrp != 0x00) {
    OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM);
    OS_EXIT_CRITICAL();
    OS_Sched();
    return (OS_NO_ERR);
  }
  if (pevent->OSEventCnt < 65535) {
    pevent->OSEventCnt++;
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
  }
  OS_EXIT_CRITICAL();
  return (OS_SEM_OVF);
}
```

if there's any task waiting, calls OS_EventTaskRdy() and scheduler

otherwise increment the semaphore count and returns

Getting a Semaphore without Waiting (Non-blocking): OSSemAccept()

```
INT16U  OSSemAccept (OS_EVENT *pevent)
{
  OS_ENTER_CRITICAL();
  cnt = pevent->OSEventCnt;
  if (cnt > 0) {
    pevent->OSEventCnt--;
  }
  OS_EXIT_CRITICAL();
  return (cnt);
}
```

if semaphore count > 0 then decrement it

return the count value