Chapter 11
Message Queue Management

System & Network Lab

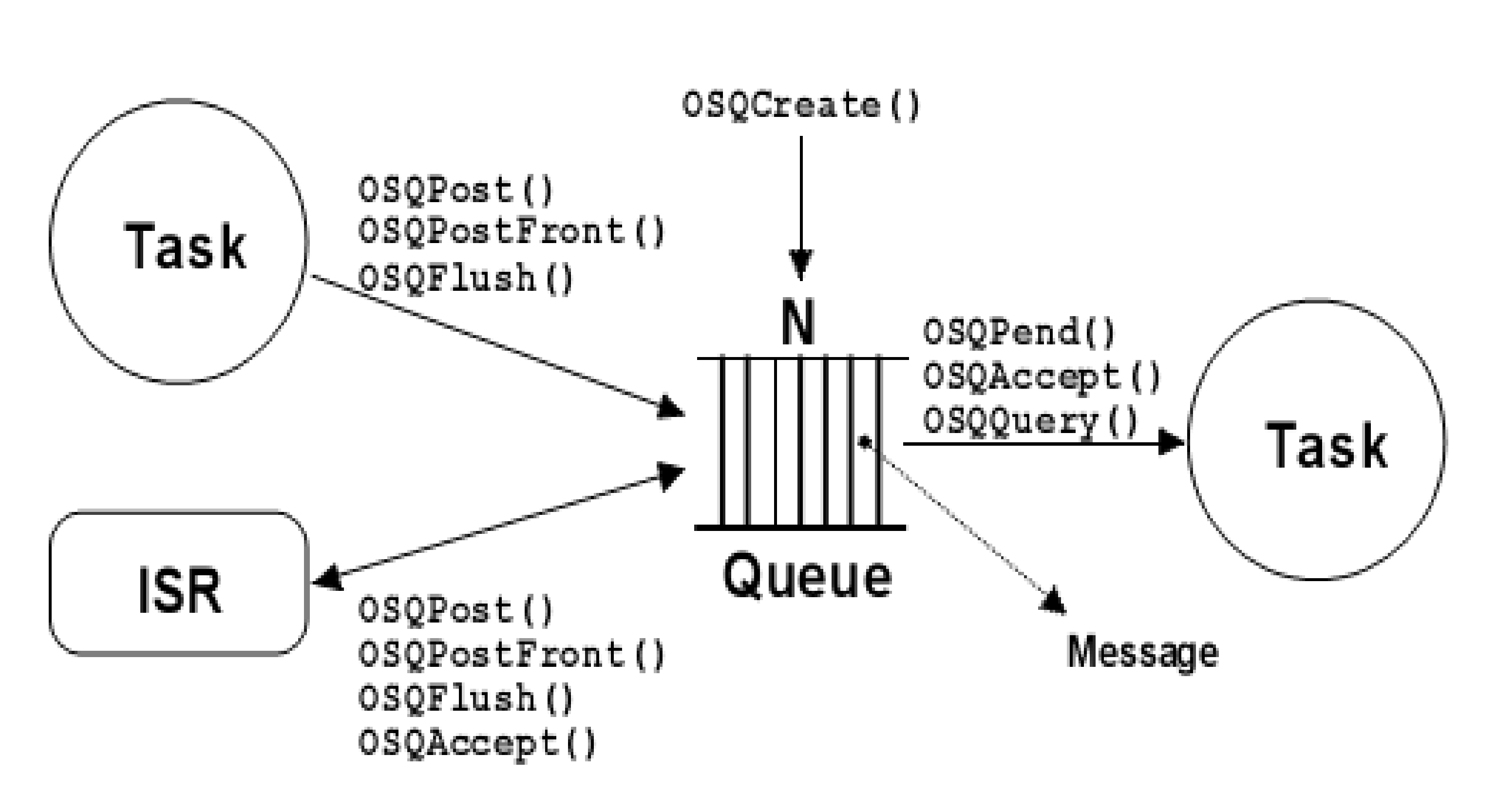
Tsung-Yu Ye

Outline

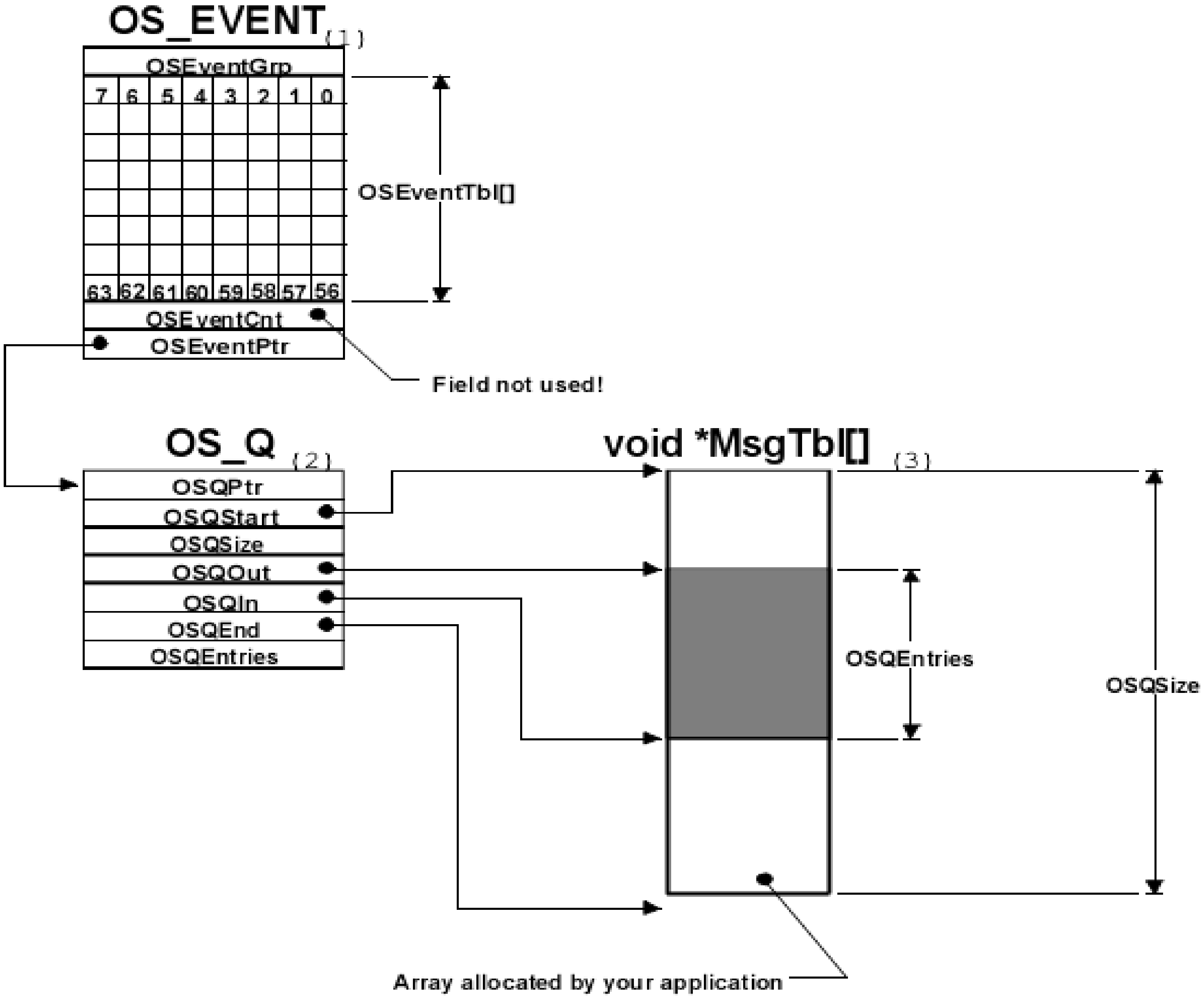
- Introduction
- Creating a Message Queue, OSQCreate()
- Deleting a Message Queue, OSQDel()
- Waiting for a message at a Queue, OSQPend()
- Sending a message to a queue, OSQPost(), OSQPostFront(), OSQPostOpt()
- Getting a message without waiting, OSQAccept()
- Flushing a queue, OSQFlush()
- Obtaining the status of a queue, OSQQuery()
- Using a message queue when reading analog inputs
- Using a queue as a counting semaphore

Introduction – 1/3

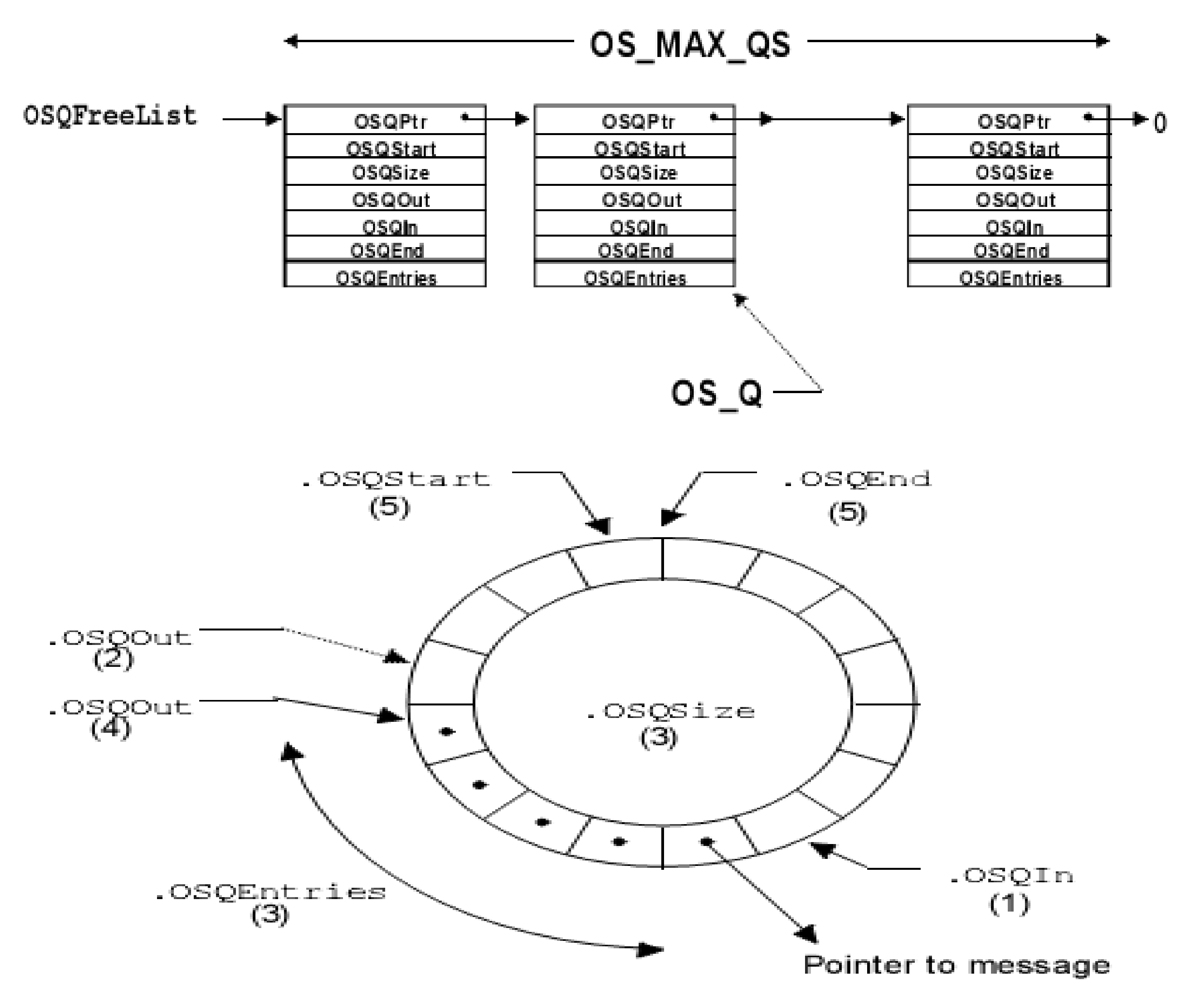
OSQpend can’t be used in ISR because ISR must finish without any wait.



Introduction – 2/3



Introduction – 3/3



OSQCreate() – 1/3

```
OS_EVENT  *OSQCreate (void **start, INT16U size)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register      */
  OS_CPU_SR  cpu_sr;
#endif
  OS_EVENT  *pevent;
  OS_Q      *pq;

  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();
```

OSQCreate() – 2/3

```
if (pevent != (OS_EVENT *)0) {                          /* See if we have an event control block    */
    OS_ENTER_CRITICAL();
    pq = OSQFreeList;                                   /* Get a free queue control block  */
    if (pq != (OS_Q *)0) {                              /* Were we able to get a queue control block ? */
      OSQFreeList       = OSQFreeList->OSQPtr;          /* Yes, Adjust free list pointer to next free*/
      OS_EXIT_CRITICAL();
      pq->OSQStart      = start;                        /*     Initialize the queue               */
      pq->OSQEnd        = &start[size];
      pq->OSQIn         = start;
      pq->OSQOut        = start;
      pq->OSQSize       = size;
      pq->OSQEntries     = 0;
      pevent->OSEventType = OS_EVENT_TYPE_Q;
      pevent->OSEventCnt  = 0;
      pevent->OSEventPtr  = pq;
      OS_EventWaitListInit(pevent);                     /*     Initalize the wait list            */
    }
```

OSQCreate() – 3/3

```
else {
            pevent->OSEventPtr = (void *)OSEventFreeList;           /* No,  Return event control block on error  */
            OSEventFreeList    = pevent;
            OS_EXIT_CRITICAL();
            pevent = (OS_EVENT *)0;
        }
    }
    return (pevent);
}
```

OSQDel()

```
OS_EVENT  *OSQDel (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;
   OS_Q      *pq;

   if (OSIntNesting > 0) {                           /* See if called from ISR ...               */
      *err = OS_ERR_DEL_ISR;                         /* ... can't DELETE from an ISR             */
      return ((OS_EVENT *)0);
   }
#if OS_ARG_CHK_EN > 0
        PSEUDO CODE:    Validate 'pevent‘ and event block type, return error code if  it’s illegal.
#endif
    OS_ENTER_CRITICAL();
```

```
if (pevent->OSEventGrp != 0x00) {                          /* See if any tasks waiting on queue       */
    tasks_waiting = TRUE;                                    /* Yes                                   */
  } else {
    tasks_waiting = FALSE;                                   /* No                                    */
  }
  switch (opt) {
    case OS_DEL_NO_PEND:                                     /* Delete queue only if no task waiting    */
       if (tasks_waiting == FALSE) {
         pq              = (OS_Q *)pevent->OSEventPtr;  /* Return OS_Q to free list        */
         pq->OSQPtr        = OSQFreeList;
         OSQFreeList       = pq;
         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);                       /* Queue has been deleted                  */
       } else {
         OS_EXIT_CRITICAL();
         *err = OS_ERR_TASK_WAITING;
         return (pevent);
       }
```

```
        case OS_DEL_ALWAYS:                                       /* Always delete the queue          */
            while (pevent->OSEventGrp != 0x00) {                  /* Ready ALL tasks waiting for queue      */
                OS_EventTaskRdy(pevent, (void *)0, OS_STAT_Q);
            }
            pq   = (OS_Q *)pevent->OSEventPtr;                    /* Return OS_Q to free list      */
            pq->OSQPtr        = OSQFreeList;
            OSQFreeList       = pq;
            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);                               /* Queue has been deleted              */
        default:
            OS_EXIT_CRITICAL();
            *err = OS_ERR_INVALID_OPT;
            return (pevent);
    }
}
```

OSQPend() –1/3

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

    PSEUDO CODE : validate OSIntNesting , return error code if  it’s bigger than zero

#if OS_ARG_CHK_EN > 0

    PSEUDO CODE:    Validate 'pevent‘ and event block type, return error code if  it’s illegal.

#endif
```

OSQPend() - 2/3

```
OS_ENTER_CRITICAL();
pq = (OS_Q *)pevent->OSEventPtr;            /* Point at queue control block                */
if (pq->OSQEntries > 0) {                   /* See if any messages in the queue             */
    msg = *pq->OSQOut++;                    /* Yes, extract oldest message from the queue   */
    pq->OSQEntries--;                       /* Update the number of entries in the queue    */
    if (pq->OSQOut == pq->OSQEnd) {         /* Wrap OUT pointer if we are at the end of the queue */
        pq->OSQOut = pq->OSQStart;
    }
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
    return (msg);                           /* Return message received                      */
}
OSTCBCur->OSTCBStat |= OS_STAT_Q;           /* Task will have to pend for a message to be posted  */
OSTCBCur->OSTCBDly   = timeout;             /* Load timeout into TCB                        */
OS_EventTaskWait(pevent);                   /* Suspend task until event or timeout occurs   */
OS_EXIT_CRITICAL();
OS_Sched();                                 /* Find next highest priority task ready to run */
OS_ENTER_CRITICAL();
msg = OSTCBCur->OSTCBMsg;
```

OSQPend() – 3/3

```
if (msg != (void *)0) {                                     /* Did we get a message?                    */
    OSTCBCur->OSTCBMsg      = (void *)0;                     /* Extract message from TCB (Put there by QPost)   */
    OSTCBCur->OSTCBStat     = OS_STAT_RDY;
    OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;                 /* No longer waiting for event                  */
    OS_EXIT_CRITICAL();
    *err                    = OS_NO_ERR;
    return (msg);                                           /* Return message received                  */
  }
  OS_EventTO(pevent);                                       /* Timed out                               */
  OS_EXIT_CRITICAL();
  *err = OS_TIMEOUT;                                        /* Indicate a timeout occured                 */
  return ((void *)0);                                       /* No message received                      */
}
```

OSQPost() – 1/2

```
INT8U  OSQPost (OS_EVENT *pevent, void *msg)
{
#if OS_CRITICAL_METHOD == 3                      /* Allocate storage for CPU status register          */
    OS_CPU_SR  cpu_sr;
#endif
    OS_Q      *pq;

#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {               /* Validate 'pevent'                            */
        return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {                      /* Make sure we are not posting a NULL pointer   */
        return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {           /* Validate event block type    */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

OSQPost() - 2/2

```
OS_ENTER_CRITICAL();
  if (pevent->OSEventGrp != 0x00) {                   /* See if any task pending on queue            */
     OS_EventTaskRdy(pevent, msg, OS_STAT_Q);         /* Ready highest priority task waiting on event  */
     OS_EXIT_CRITICAL();
     OS_Sched();                                      /* Find highest priority task ready to run      */
     return (OS_NO_ERR);
  }
  pq = (OS_Q *)pevent->OSEventPtr;                    /* Point to queue control block                 */
  if (pq->OSQEntries >= pq->OSQSize) {                /* Make sure queue is not full                  */
     OS_EXIT_CRITICAL();
     return (OS_Q_FULL);
  }
  *pq->OSQIn++ = msg;                                 /* Insert message into queue                    */
  pq->OSQEntries++;                                   /* Update the nbr of entries in the queue       */
  if (pq->OSQIn == pq->OSQEnd) {                      /* Wrap IN ptr if we are at end of queue        */
     pq->OSQIn = pq->OSQStart;
  }
  OS_EXIT_CRITICAL();
  return (OS_NO_ERR);
}
```

OSQPostFront() – 1/2

```
INT8U  OSQPostFront (OS_EVENT *pevent, void *msg)
{
#if OS_CRITICAL_METHOD == 3                     /* Allocate storage for CPU status register          */
    OS_CPU_SR  cpu_sr;
#endif
    OS_Q      *pq;

#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {                     /* Validate 'pevent'                              */
        return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {                              /* Make sure we are not posting a NULL pointer   */
        return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {     /* Validate event block type                     */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
    OS_ENTER_CRITICAL();
```

OSQPostFront() – 2/2

```
    if (pevent->OSEventGrp != 0x00) {                    /* See if any task pending on queue            */
        OS_EventTaskRdy(pevent, msg, OS_STAT_Q);         /* Ready highest priority task waiting on event  */
        OS_EXIT_CRITICAL();
        OS_Sched();                                      /* Find highest priority task ready to run       */
        return (OS_NO_ERR);
    }
    pq = (OS_Q *)pevent->OSEventPtr;                     /* Point to queue control block                 */
    if (pq->OSQEntries >= pq->OSQSize) {                 /* Make sure queue is not full                  */
        OS_EXIT_CRITICAL();
        return (OS_Q_FULL);
    }
    if (pq->OSQOut == pq->OSQStart) {                    /* Wrap OUT ptr if we are at the 1st queue entry */
        pq->OSQOut = pq->OSQEnd;
    }
    pq->OSQOut--;
    *pq->OSQOut = msg;                                   /* Insert message into queue                    */
    pq->OSQEntries++;                                    /* Update the nbr of entries in the queue       */
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

OSQPostOpt() – 1/3

```
INT8U  OSQPostOpt (OS_EVENT *pevent, void *msg, INT8U opt)
{
#if OS_CRITICAL_METHOD == 3                      /* Allocate storage for CPU status register          */
    OS_CPU_SR  cpu_sr;
#endif
    OS_Q      *pq;

#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {               /* Validate 'pevent'                                 */
        return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {                      /* Make sure we are not posting a NULL pointer   */
        return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {     /* Validate event block type                   */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

OSQPostOpt() - 2/3

```
OS_ENTER_CRITICAL();
  if (pevent->OSEventGrp != 0x00) {                          /* See if any task pending on queue           */
    if ((opt & OS_POST_OPT_BROADCAST) != 0x00) {   /* Do we need to post msg to ALL waiting tasks ? */
      while (pevent->OSEventGrp != 0x00) {                  /* Yes, Post to ALL tasks waiting on queue      */
        OS_EventTaskRdy(pevent, msg, OS_STAT_Q);
      }
    } else {
      OS_EventTaskRdy(pevent, msg, OS_STAT_Q);        /* No,  Post to HPT waiting on queue            */
    }
    OS_EXIT_CRITICAL();
    OS_Sched();                                                  /* Find highest priority task ready to run      */
    return (OS_NO_ERR);
  }
  pq = (OS_Q *)pevent->OSEventPtr;                          /* Point to queue control block                */
  if (pq->OSQEntries >= pq->OSQSize) {                     /* Make sure queue is not full                 */
    OS_EXIT_CRITICAL();
    return (OS_Q_FULL);
  }
```

OSQPostOpt() – 3/3

```
 if ((opt & OS_POST_OPT_FRONT) != 0x00) {        /* Do we post to the FRONT of the queue?        */
        if (pq->OSQOut == pq->OSQStart) {           /* Yes, Post as LIFO, Wrap OUT pointer if we ... */
            pq->OSQOut = pq->OSQEnd;                /*      ... are at the 1st queue entry          */
        }
        pq->OSQOut--;
        *pq->OSQOut = msg;                          /*      Insert message into queue               */
    } else {                                        /* No,  Post as FIFO                            */
        *pq->OSQIn++ = msg;                         /*      Insert message into queue               */
        if (pq->OSQIn == pq->OSQEnd) {              /*      Wrap IN ptr if we are at end of queue   */
            pq->OSQIn = pq->OSQStart;
        }
    }
    pq->OSQEntries++;                               /* Update the nbr of entries in the queue       */
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

OSQAccept() – 1/2

```
void  *OSQAccept (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3                               /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr;
#endif
    void      *msg;
    OS_Q      *pq;

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

OSQAccept() – 2/2

```
OS_ENTER_CRITICAL();
    pq = (OS_Q *)pevent->OSEventPtr;          /* Point at queue control block                   */
    if (pq->OSQEntries > 0) {                 /* See if any messages in the queue               */
        msg = *pq->OSQOut++;                  /* Yes, extract oldest message from the queue     */
        pq->OSQEntries--;                     /* Update the number of entries in the queue      */
        if (pq->OSQOut == pq->OSQEnd) {       /* Wrap OUT pointer if we are at the end of the queue */
            pq->OSQOut = pq->OSQStart;
        }
    } else {
        msg = (void *)0;                      /* Queue is empty                                 */
    }
    OS_EXIT_CRITICAL();
    return (msg);                             /* Return message received (or NULL)              */
}
```

OSQFlush()

```
INT8U  OSQFlush (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3                               /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr;
#endif
    OS_Q      *pq;
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {                        /* Validate 'pevent'                            */
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {     /* Validate event block type                    */
        return (OS_ERR_EVENT_TYPE);
    }
#endif
    OS_ENTER_CRITICAL();
    pq             = (OS_Q *)pevent->OSEventPtr;          /* Point to queue storage structure             */
    pq->OSQIn      = pq->OSQStart;
    pq->OSQOut     = pq->OSQStart;
    pq->OSQEntries = 0;
    OS_EXIT_CRITICAL();
```

OSQQuery() – 1/4

```
INT8U  OSQQuery (OS_EVENT *pevent, OS_Q_DATA *pdata)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register      */
   OS_CPU_SR  cpu_sr;
#endif
   OS_Q      *pq;
   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_Q) {     /* Validate event block type              */
      return (OS_ERR_EVENT_TYPE);
   }
#endif
```

OSQQuery() – 2/4

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

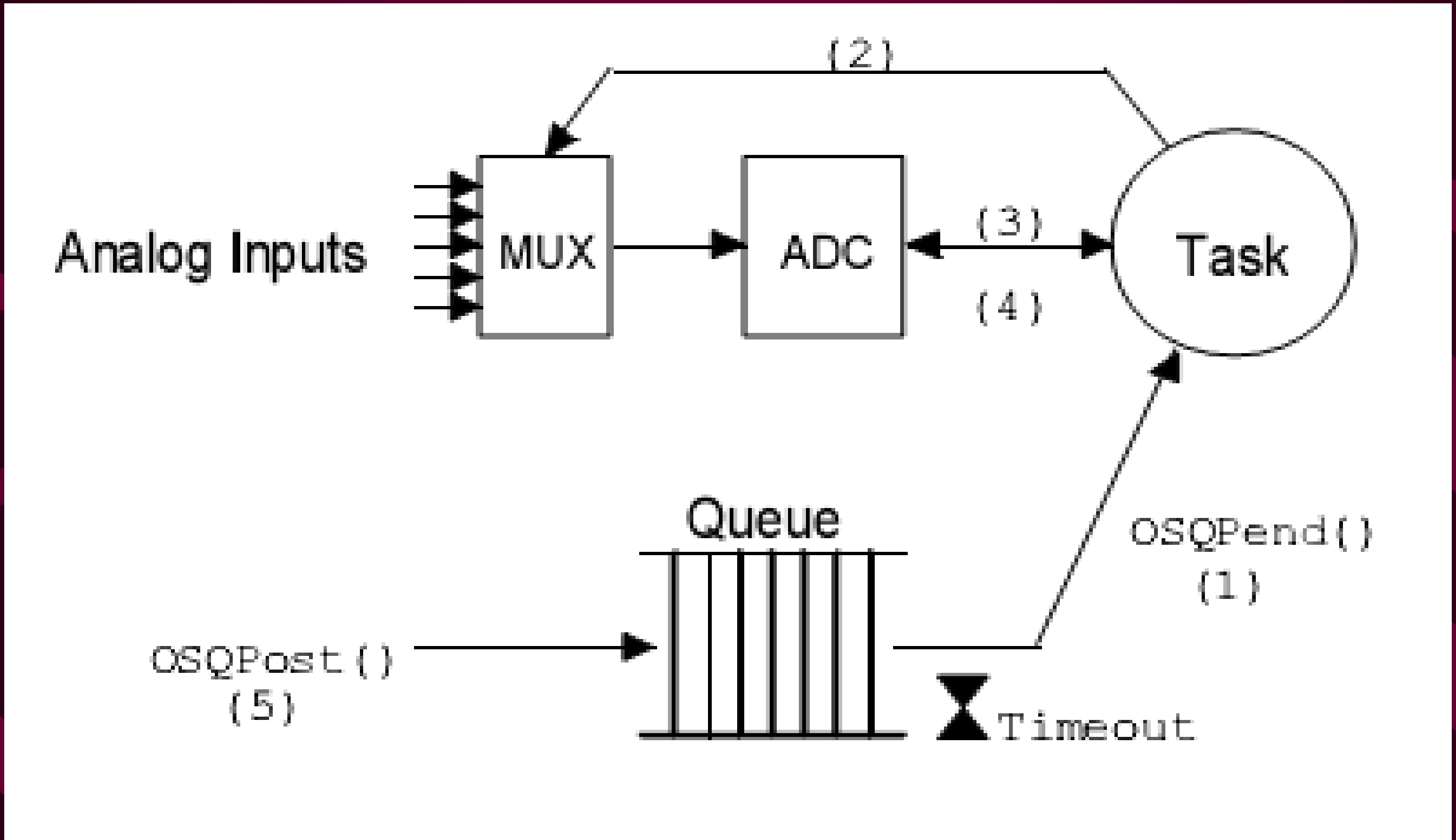
OSQQuery() – 3/4

```
#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
```

OSQQuery() – 4/4

```
 pq = (OS_Q *)pevent->OSEventPtr;
    if (pq->OSQEntries > 0) {
        pdata->OSMsg = *pq->OSQOut;                    /* Get next message to return if available  */
    } else {
        pdata->OSMsg = (void *)0;
    }
    pdata->OSNMsgs = pq->OSQEntries;
    pdata->OSQSize = pq->OSQSize;
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Using a message queue when reading analog inputs



Using a queue as a counting semaphore – 1/2

```
void main (void)
{
   OSInit();
   .
   QSem = OSQCreate(&QMsgTbl[0], N_RESOURCES);
   for (i = 0; i < N_RESOURCES; i++) {
      OSQPost(Qsem, (void *)1);
   }
   .
   OSTaskCreate(Task1, .., .., ..);
   .
   OSStart();
}
```

Using a queue as a counting semaphore – 2/2

```
void Task1 (void *pdata)
{
   INT8U err;
    for (;;) {
      OSQPend(&QSem, 0, &err);              /* Obtain access to resource(s)  */
      .
      .   /* Task has semaphore, access resource(s)                */
      .
      OSMQPost(QSem, (void )1);             /* Release access to resource(s) */
   }
}
```

The End