CH9
Event Flag Management

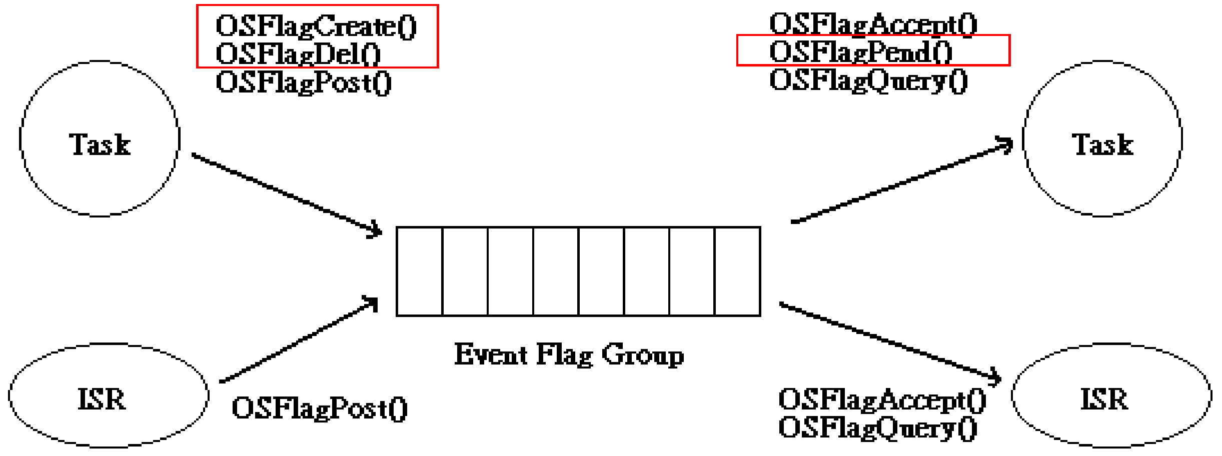
Date : 2007/08/16

Speaker:Ming-Shyong Tsai

BASICS

- μC/OS-II event flags consist of two elements
  - A series of bits hold the *current state* of events in the group
  - A list of tasks waiting for *combination*
- Task V.S. ISR
  - Differences
- Others
  - Function en
  - typedef INT

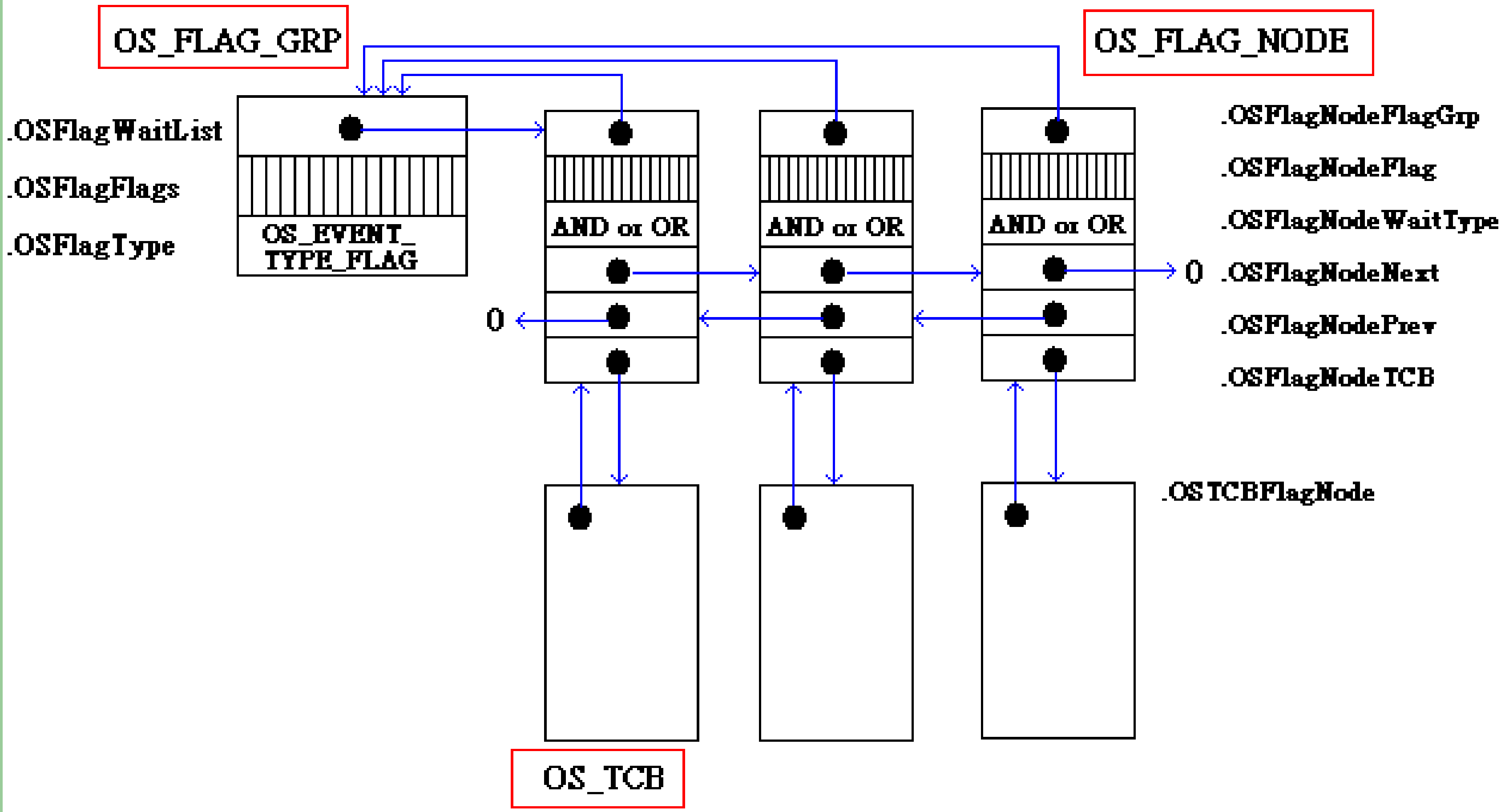
OSFlagCreate()
OSFlagDel()
OSFlagPend()
OSFlagAccept()
OSFlagPost()
OSFlagQuery()



Functions

- OSFlagCreate()
  - Creating an event flag group
- OSFlagDel()
  - Deleting an event flag group
- OSFlagPend()
  - Waiting for events
  - OS_FlagBlock()
- OSFlagAccept()
  - Looking for events
- OSFlagPost()
  - Setting or clearing events
  - OS_FlagTaskRdy()
- OSFlagQuery()
  - Querying an event flag group

Data Structure



Data Structure(cont.)

```
typedef struct {
void        *OSFlagNodeNext;
void        *OSFlagNodePrev;
void        *OSFlagNodeTCB;
void        *OSFlagNodeFlagGrp;
OS_FLAGS      OSFlagNodeFlags;
INT8U            OSFlagNodeWaitType;
} OS_FLAG_NODE;
```

OS_FLAG_WAIT_CLR_ALL
wait for ALL bits in 'mask' to be clear (0)

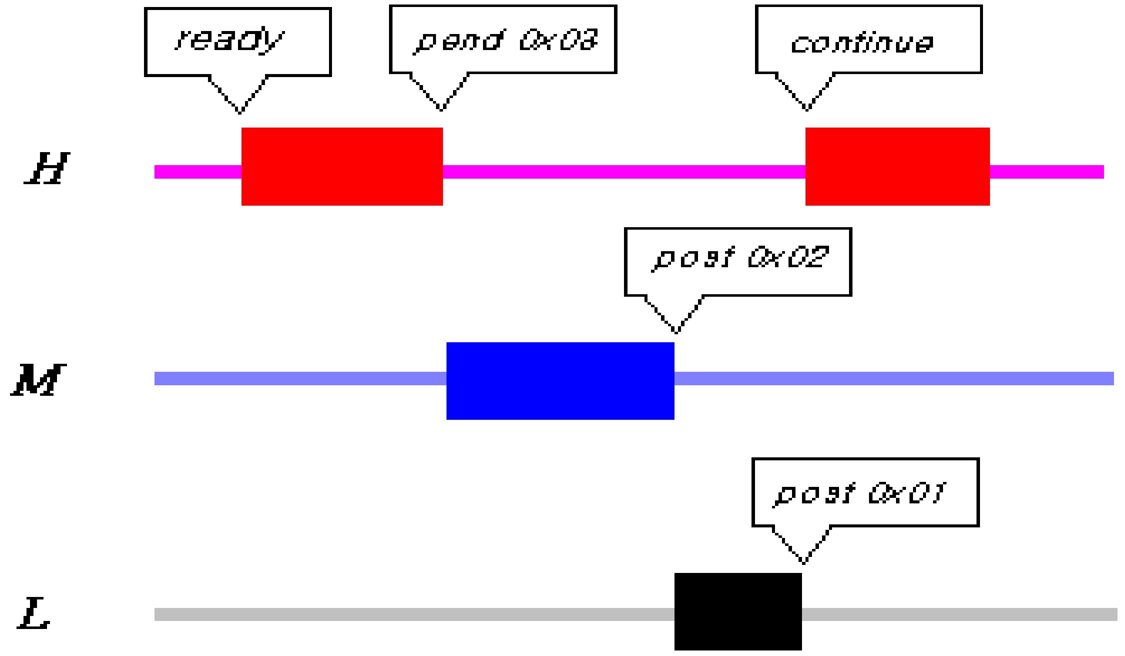
OS_FLAG_WAIT_CLR_ALL
OS_FLAG_WAIT_CLR_AND

OS_FLAG_WAIT_CLR_ANY
OS_FLAG_WAIT_CLR_OR

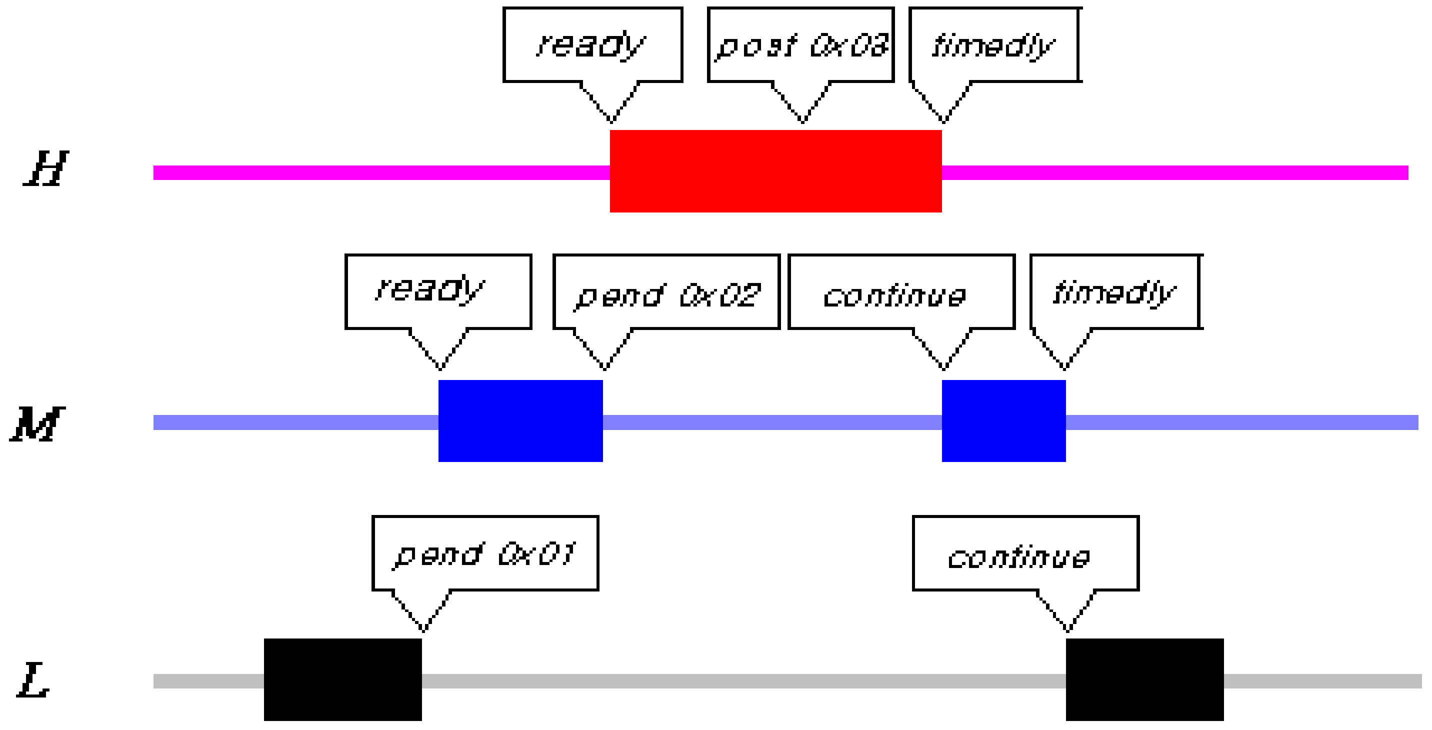
OS_FLAG_WAIT_SET_ALL
OS_FLAG_WAIT_SET_AND

OS_FLAG_WAIT_SET_ANY
OS_FLAG_WAIT_SET_OR

Example 1



Example 2

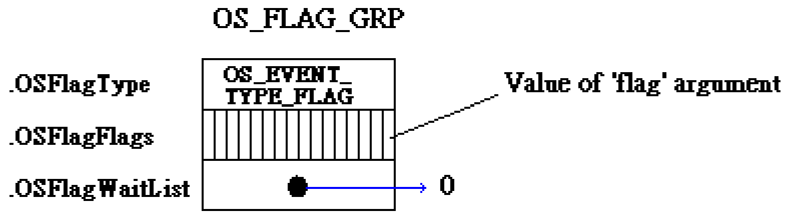

ready
post 0x03
timedly
H
ready
pend 0x02
continue
timedly
M
pend 0x01
continue
L

Creating an Event Flag Group ,OSFlagCreate()

```
#define  OS_EVENT_TYPE_UNUSED     0
#define  OS_EVENT_TYPE_MBOX       1
#define  OS_EVENT_TYPE_Q          2
#define  OS_EVENT_TYPE_SEM        3
#define  OS_EVENT_TYPE_MUTEX      4
#define  OS_EVENT_TYPE_FLAG       5
```

```
OS_FLAG_GRP  *OSFlagCreate
     (OS_FLAGS flags, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
   OS_CPU_SR    cpu_sr;
#endif
   OS_FLAG_GRP *pgrp;


   if (OSIntNesting > 0) {
       *err = OS_ERR_CREATE_ISR;
       return ((OS_FLAG_GRP *)0);
```



```
if (pgrp != (OS_FLAG_GRP *)0) {
     OSFlagFreeList       =
      (OS_FLAG_GRP*)
        OSFlagFreeList->OSFlagWaitList;
      pgrp->OSFlagType     =
        OS_EVENT_TYPE_FLAG;
      pgrp->OSFlagFlags    = flags;
      pgrp->OSFlagWaitList = (void *)0;
      OS_EXIT_CRITICAL();
       *err               = OS_NO_ERR;
   } else {
       OS_EXIT_CRITICAL();
       *err=OS_FLAG_GRP_DEPLETED;
   }     OS_FLAG_GRP_DEPLETED=154
   return (pgrp);
}
```

Initial value of event flags

All 0's for set bits or

all 1's for cleared bits

Deleting an Event Flag Group, OSFlagDel()

```
OS_FLAG_GRP  *OSFlagDel
     (OS_FLAG_GRP *pgrp, INT8U opt,
     INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR     cpu_sr;
#endif
    BOOLEAN       tasks_waiting;
    OS_FLAG_NODE *pnode;

    if (OSIntNesting > 0) {
        *err = OS_ERR_DEL_ISR;
        return (pgrp);
    }
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return (pgrp);
    }
if (pgrp->OSFlagType !=
      OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return (pgrp);
    }
#endif
 OS_ENTER_CRITICAL();
    if (pgrp->OSFlagWaitList != (void *)0) {
        tasks_waiting = TRUE;
    } else {
        tasks_waiting = FALSE;
    }
```

Deleting an Event Flag Group, OSFlagDel() (cont.)

```
switch (opt) {
      case OS_DEL_NO_PEND:
          if (tasks_waiting == FALSE) {
              pgrp->OSFlagType     = OS_EVENT_TYPE_UNUSED;
              pgrp->OSFlagWaitList = (void *)OSFlagFreeList;
              OSFlagFreeList       = pgrp;
              OS_EXIT_CRITICAL();
              *err                 = OS_NO_ERR;
              return ((OS_FLAG_GRP *)0);
          } else {
              OS_EXIT_CRITICAL();
              *err                 = OS_ERR_TASK_WAITING;
              return (pgrp);
          }
```

Deleting an Event Flag Group, OSFlagDel() (cont.)

```
case OS_DEL_ALWAYS:
     pnode = (OS_FLAG_NODE*)pgrp->OSFlagWaitList;
     while (pnode != (OS_FLAG_NODE *)0) {
      OS_FlagTaskRdy(pnode, (OS_FLAGS)0);
      pnode = (OS_FLAG_NODE *)pnode->OSFlagNodeNext;
      }
     pgrp->OSFlagType     = OS_EVENT_TYPE_UNUSED;
     pgrp->OSFlagWaitList = (void *)OSFlagFreeList;
     OSFlagFreeList       = pgrp;
     OS_EXIT_CRITICAL();
     if (tasks_waiting == TRUE) {
         OS_Sched();
     }
     *err = OS_NO_ERR;
     return ((OS_FLAG_GRP *)0);

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

OS_FLAGS OSFlagPend (OSFlagMyGrp, (OS_FLAGS) 0x01, ***FLAG_WAIT_SET_ANY****+* ***OS_FLAG_COMSUME****, 0, &err)*

Waiting for Event(s) of an Event Flag Group, OSFlagPend()

```
OS_FLAGS  OSFlagPend
      (OS_FLAG_GRP *pgrp, OS_FLAGS
      flags, INT8U wait_type, INT16U
      timeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR     cpu_sr;
#endif
    OS_FLAG_NODE  node;
    OS_FLAGS      flags_cur;
    OS_FLAGS      flags_rdy;
    BOOLEAN       consume;

    if (OSIntNesting > 0) {
        *err = OS_ERR_PEND_ISR;
        return ((OS_FLAGS)0);
    }
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType !=
      OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
    if (wait_type & OS_FLAG_CONSUME) {
        wait_type &= ~OS_FLAG_CONSUME;
        consume    = TRUE;
    } else {
        consume    = FALSE;
    }
```

Waiting for Event(s) of an Event Flag Group, OSFlagPend() (cont.)

```
OS_ENTER_CRITICAL();
   switch (wait_type) {
       case OS_FLAG_WAIT_SET_ALL:
            flags_rdy = pgrp->OSFlagFlags & flags;         // 0001 = 0011 & 0001
            if (flags_rdy == flags) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags &= ~flags_rdy;      //0010 = 0011 & 1110
                }
                flags_cur = pgrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err      = OS_NO_ERR;
                return (flags_cur);
            } else {
                OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
                OS_EXIT_CRITICAL();
            }
            break;
```

```
#define OS_FLAG_WAIT_CLR_ALL  0
#define OS_FLAG_WAIT_CLR_ANY 1
#define OS_FLAG_WAIT_SET_ALL  2
#define OS_FLAG_WAIT_SET_ANY 3
```

Waiting for Event(s) of an Event Flag Group, OSFlagPend() (cont.)

```
case OS_FLAG_WAIT_SET_ANY:
            flags_rdy = pgrp->OSFlagFlags & flags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags &= ~flags_rdy;
                }
                flags_cur = pgrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err      = OS_NO_ERR;
                return (flags_cur);
            } else {
                OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
                OS_EXIT_CRITICAL();
            }
            break;
```

Waiting for Event(s) of an Event Flag Group, OSFlagPend() (cont.)

```
#if OS_FLAG_WAIT_CLR_EN > 0
        case OS_FLAG_WAIT_CLR_ALL:
             flags_rdy = ~pgrp->OSFlagFlags & flags;        ~pgrp
             if (flags_rdy == flags) {
                 if (consume == TRUE) {
                     pgrp->OSFlagFlags |= flags_rdy;        |=
                 }
                 flags_cur = pgrp->OSFlagFlags;
                 OS_EXIT_CRITICAL();
                 *err      = OS_NO_ERR;
                 return (flags_cur);
             } else {
                 OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
                 OS_EXIT_CRITICAL();
             }
             break;
```

Waiting for Event(s) of an Event Flag Group, OSFlagPend() (cont.)

```
 case OS_FLAG_WAIT_CLR_ANY:
            flags_rdy = ~pgrp->OSFlagFlags & flags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags |= flags_rdy;
                }
                flags_cur = pgrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err      = OS_NO_ERR;
                return (flags_cur);
            } else {
                OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
                OS_EXIT_CRITICAL();
            }
            break;
#endif
```

Waiting for Event(s) of an Event Flag Group, OSFlagPend() (cont.)

```
   default:
        OS_EXIT_CRITICAL();
        flags_cur = (OS_FLAGS)0;
        *err      =
   OS_FLAG_ERR_WAIT_TYPE;
        return (flags_cur);
}
OS_Sched();
OS_ENTER_CRITICAL();
if (OSTCBCur->OSTCBStat &
   OS_STAT_FLAG) {
    OS_FlagUnlink(&node);
    OSTCBCur->OSTCBStat =
   OS_STAT_RDY;
    OS_EXIT_CRITICAL();
    flags_cur        = (OS_FLAGS)0;
    *err             = OS_TIMEOUT;
} else {
  if (consume == TRUE) {
```

OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);

The task is still waiting for event flags

```
     switch (wait_type) {
     case OS_FLAG_WAIT_SET_ALL:
     case OS_FLAG_WAIT_SET_ANY:
                 pgrp->OSFlagFlags &=
      ~OSTCBCur->OSTCBFlagsRdy;
           break;
#if  OS_FLAG_WAIT_CLR_EN > 0
     case OS_FLAG_WAIT_CLR_ALL:
     case OS_FLAG_WAIT_CLR_ANY:
                 pgrp->OSFlagFlags |=
      OSTCBCur->OSTCBFlagsRdy;
            break;
#endif
        }
       }
       flags_cur = pgrp->OSFlagFlags;
       OS_EXIT_CRITICAL();
       *err      = OS_NO_ERR;
    }
    return (flags_cur);
}
```

Adding a task to the event flag group wait list ,OS_FlagBlock()

```
static  void  OS_FlagBlock (OS_FLAG_GRP *pgrp, OS_FLAG_NODE *pnode, OS_FLAGS
      flags, INT8U wait_type, INT16U timeout)
{
    OS_FLAG_NODE  *pnode_next;

    OSTCBCur->OSTCBStat      |= OS_STAT_FLAG;                    (1)
    OSTCBCur->OSTCBDly        = timeout;
#if OS_TASK_DEL_EN > 0
    OSTCBCur->OSTCBFlagNode   = pnode;                           (2)
#endif
    pnode->OSFlagNodeFlags    = flags;                           (3)
    pnode->OSFlagNodeWaitType = wait_type;
    pnode->OSFlagNodeTCB      = (void *)OSTCBCur;                (4)
    pnode->OSFlagNodeNext     = pgrp->OSFlagWaitList;            (5)
    pnode->OSFlagNodePrev     = (void *)0;                       (6)
    pnode->OSFlagNodeFlagGrp  = (void *)pgrp;                    (7)
    pnode_next              = (OS_FLAG_NODE *)pgrp->OSFlagWaitList;
```

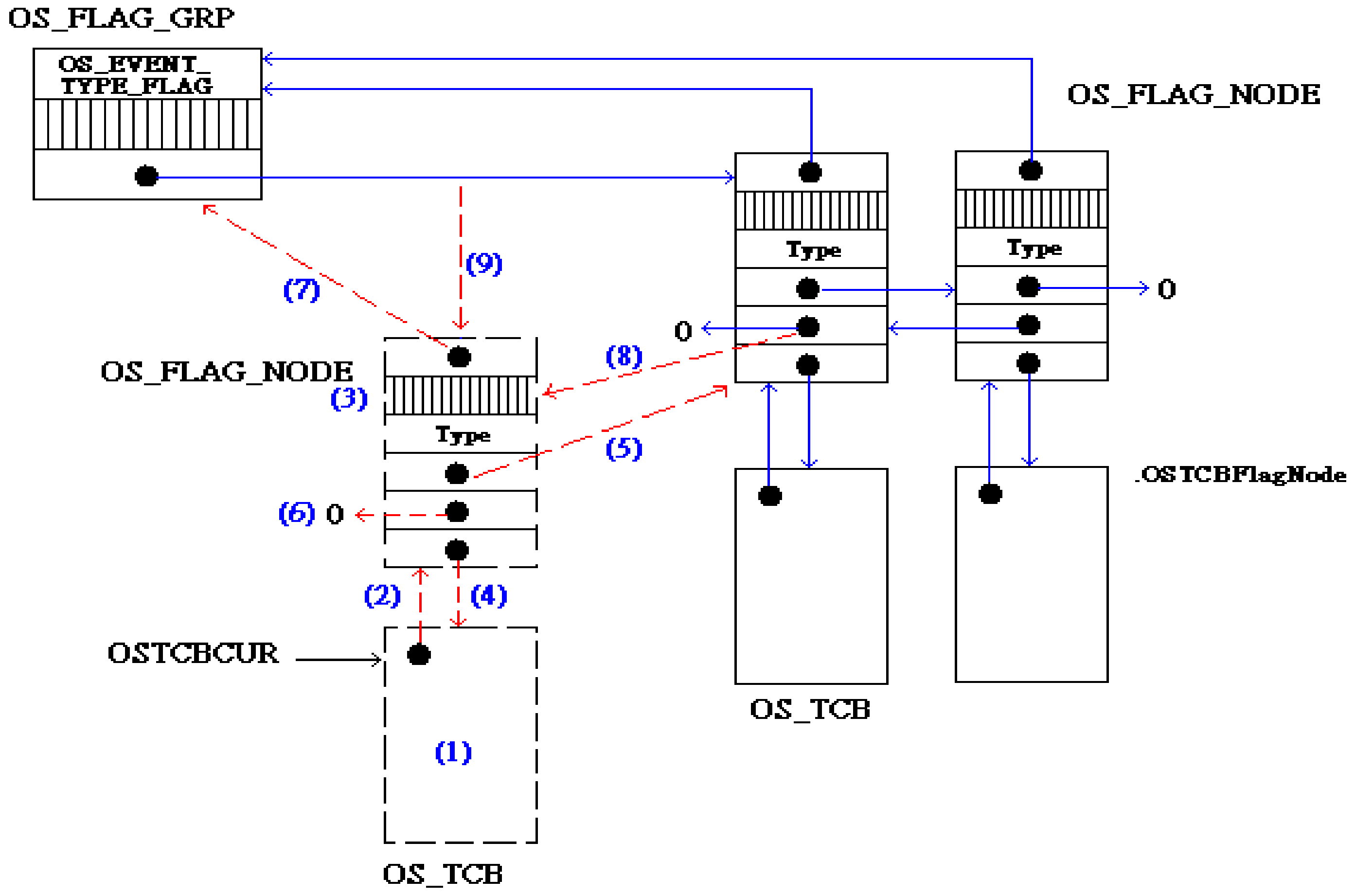
```
#define  OS_STAT_RDY      0x00
#define  OS_STAT_SEM      0x01
#define  OS_STAT_MBOX     0x02
#define  OS_STAT_Q        0x04
#define  OS_STAT_SUSPEND  0x08
#define  OS_STAT_MUTEX    0x10
#define  OS_STAT_FLAG     0x20
```

Adding a task to the event flag group wait list ,OS_FlagBlock() (cont.)

```
if (pnode_next != (void *)0) {
        pnode_next->OSFlagNodePrev = pnode;                            (8)
    }
    pgrp->OSFlagWaitList = (void *)pnode;                              (9)

    if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX) == 0) {
        OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
   //calling task is made “ not ready to run”.
    }
}
```

Adding a task to the event flag group wait list ,OS_FlagBlock() (cont.)



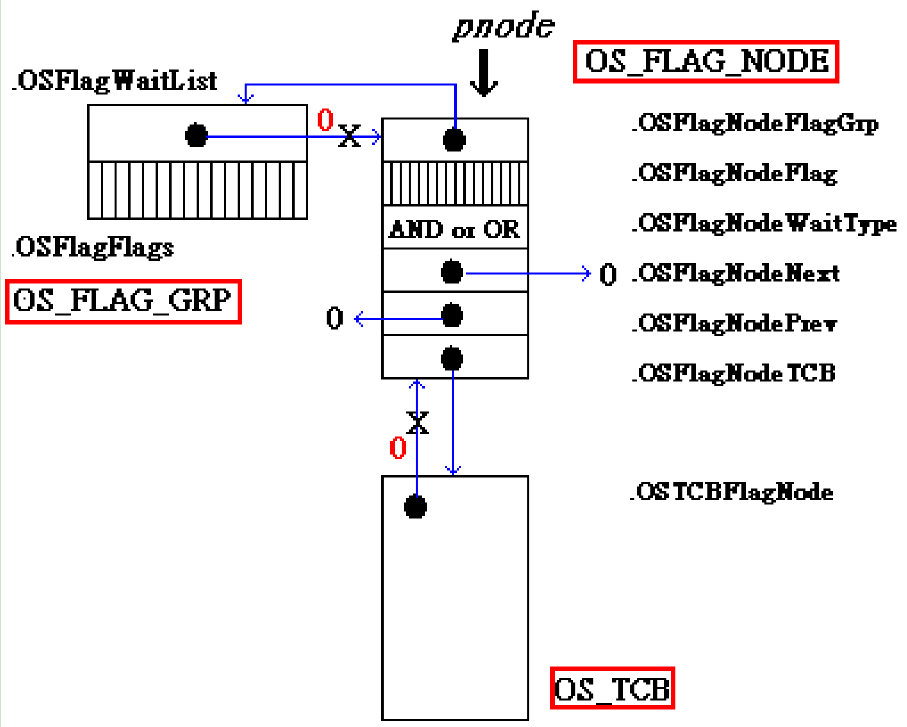
Setting or Clearing Event(s) in an Event Flag Group, OSFlagPost()

```
OS_FLAGS  OSFlagPost (OS_FLAG_GRP
      *pgrp, OS_FLAGS flags, INT8U opt,
      INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR     cpu_sr;
#endif
    OS_FLAG_NODE *pnode;
    BOOLEAN       sched;
    OS_FLAGS      flags_cur;
    OS_FLAGS      flags_rdy;

#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType !=
       OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
```

Setting or Clearing Event(s) in an Event Flag Group, OSFlagPost() (cont.)

```
OS_ENTER_CRITICAL();
switch (opt) {
   // OSFlagFlags = 0011 , flag  1111
    case OS_FLAG_CLR:
```



```
sched = FALSE;
   pnode = (OS_FLAG_NODE *)pgrp-
      >OSFlagWaitList;
   while (pnode != (OS_FLAG_NODE *)0) {
      switch (pnode->OSFlagNodeWaitType) {
         case OS_FLAG_WAIT_SET_ALL:
            flags_rdy = pgrp->OSFlagFlags
                & pnode->OSFlagNodeFlags;
            if (flags_rdy ==
=0001          pnode->OSFlagNodeFlags) {
               if (OS_FlagTaskRdy(pnode,
                  flags_rdy) == TRUE) {
                  sched = TRUE;
               }
            }
            break;
         …
      }
   pnode = (OS_FLAG_NODE *)
               pnode- >OSFlagNodeNext;
```

Setting or Clearing Event(s) in an Event Flag Group, OSFlagPost() (cont.)

```
case OS_FLAG_WAIT_SET_ANY:
     flags_rdy =
    pgrp->OSFlagFlags &
    pnode->OSFlagNodeFlags;
        if (flags_rdy != (OS_FLAGS)0) {
            if (OS_FlagTaskRdy(pnode,
                   flags_rdy) == TRUE) {
                     sched = TRUE;
                 }
         }
        break;
```

```
#if OS_FLAG_WAIT_CLR_EN > 0
     case OS_FLAG_WAIT_CLR_ALL:
          flags_rdy = ~pgrp->OSFlagFlags
               & pnode->OSFlagNodeFlags;
          if (flags_rdy ==
              pnode->OSFlagNodeFlags) {
                if (OS_FlagTaskRdy(pnode,
                    flags_rdy) == TRUE) {
                      sched = TRUE;
                }
           }
           break;
```

Setting or Clearing Event(s) in an Event Flag Group, OSFlagPost() (cont.)

```
 case OS_FLAG_WAIT_CLR_ANY:
      flags_rdy = ~pgrp->OSFlagFlags
              & pnode->OSFlagNodeFlags;
      if (flags_rdy != (OS_FLAGS)0)
      {
          if (OS_FlagTaskRdy(pnode,
                    flags_rdy) == TRUE)
         {
                     sched = TRUE;
         }
      }
      break;
#endif
   }
       pnode = (OS_FLAG_NODE *)pnode-
      >OSFlagNodeNext;
 }
```

```
  OS_EXIT_CRITICAL();
  if (sched == TRUE) {
     OS_Sched();
  }
  OS_ENTER_CRITICAL();
     flags_cur = pgrp->OSFlagFlags;
  OS_EXIT_CRITICAL();
  *err      = OS_NO_ERR;
  return (flags_cur);
}
```

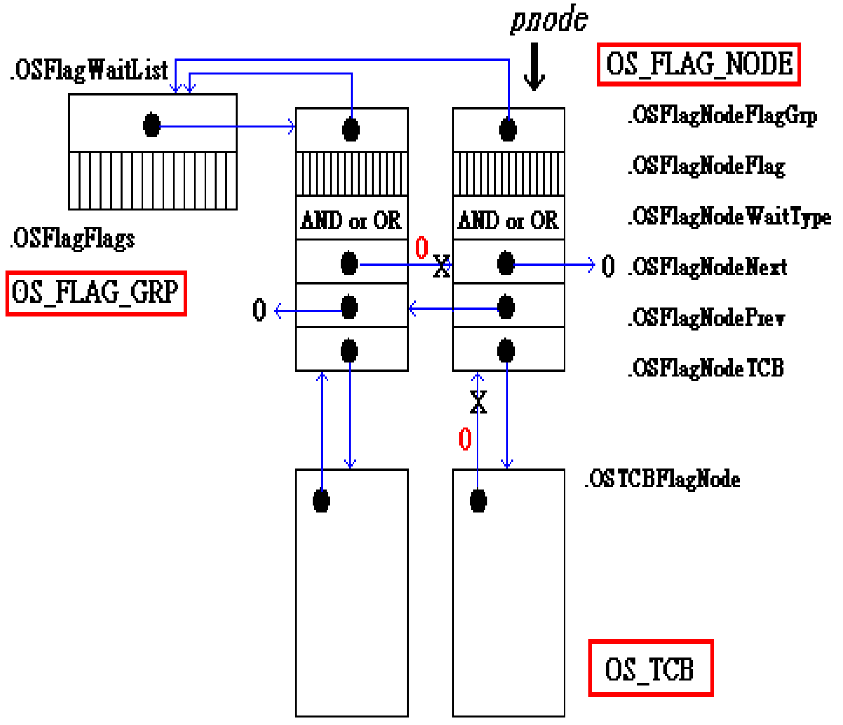
Make a waiting task ready to run, OS_FlagTaskRdy()

```
static  BOOLEAN  OS_FlagTaskRdy (OS_FLAG_NODE *pnode, OS_FLAGS flags_rdy)
{
    OS_TCB   *ptcb;
    BOOLEAN   sched;

    ptcb                = (OS_TCB *)pnode->OSFlagNodeTCB;
    ptcb->OSTCBDly      = 0;
    ptcb->OSTCBFlagsRdy = flags_rdy;
    ptcb->OSTCBStat    &= ~OS_STAT_FLAG;
    if (ptcb->OSTCBStat == OS_STAT_RDY) {
        OSRdyGrp               |= ptcb->OSTCBBitY;
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        sched                   = TRUE;
    } else {
        sched                   = FALSE;
    }
    OS_FlagUnlink(pnode);
    return (sched);
}
```

```
#define  OS_STAT_RDY      0x00
#define  OS_STAT_SEM      0x01
#define  OS_STAT_MBOX     0x02
#define  OS_STAT_Q        0x04
#define  OS_STAT_SUSPEND  0x08
#define  OS_STAT_MUTEX    0x10
#define  OS_STAT_FLAG     0x20
```

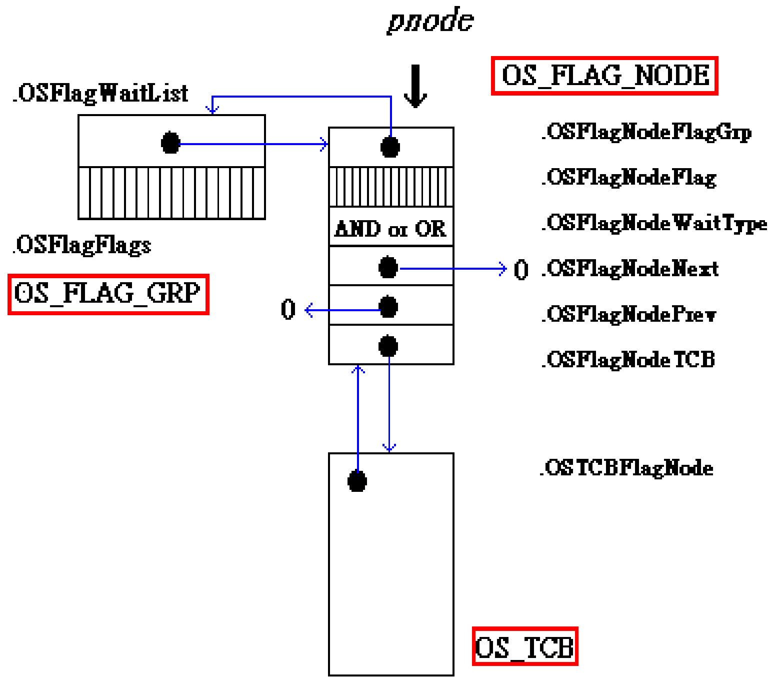
Make a waiting task ready to run , OS_FlagUnlink()

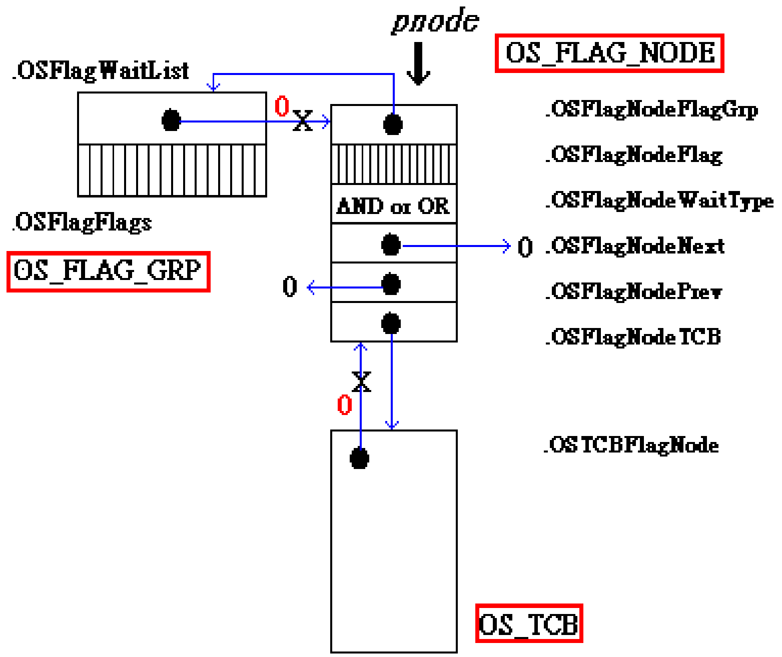


```
if (pnode_prev == (OS_FLAG_NODE *)0) {
     pgrp              = (OS_FLAG_GRP *)pnode->OSFlagNodeFlagGrp;
     pgrp->OSFlagWaitList = (void *)pnode_next;
     if (pnode_next != (OS_FLAG_NODE *)0) {
         pnode_next->OSFlagNodePrev = (OS_FLAG_NODE *)0;
     }
 } else {
     pnode_prev->OSFlagNodeNext = pnode_next;
     if (pnode_next != (OS_FLAG_NODE *)0) {
         pnode_next->OSFlagNodePrev = pnode_prev;
     }
 }
```

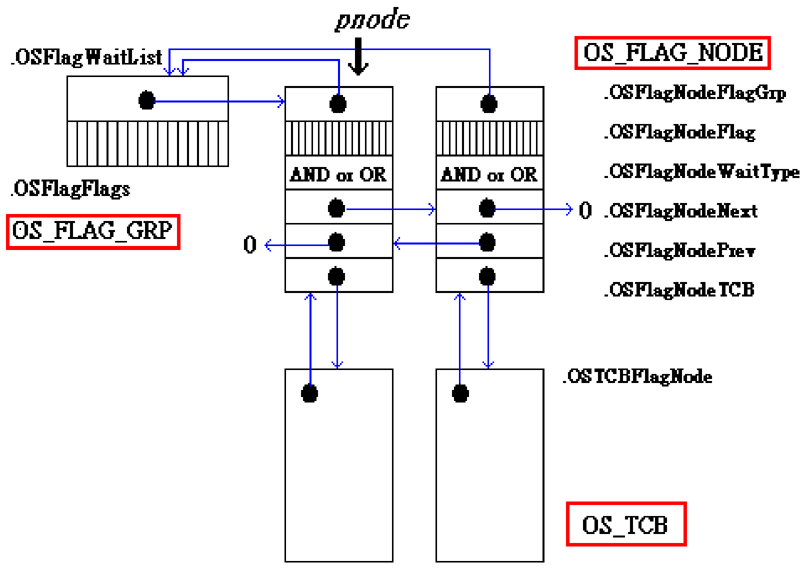
Make a waiting task ready to run , OS_FlagUnlink() (cont.)

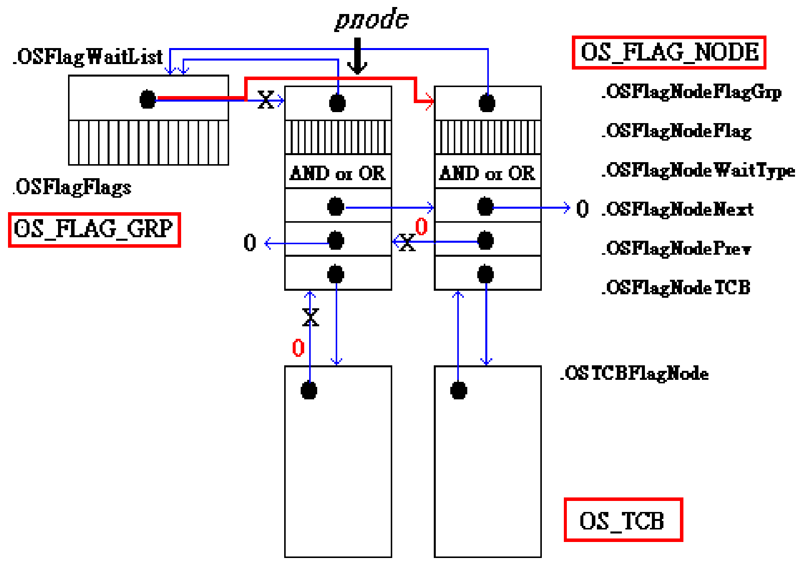
```
#if OS_TASK_DEL_EN > 0
    ptcb              = (OS_TCB *)pnode-
      >OSFlagNodeTCB;
    ptcb->OSTCBFlagNode =
      (OS_FLAG_NODE *)0;
#endif
}
```



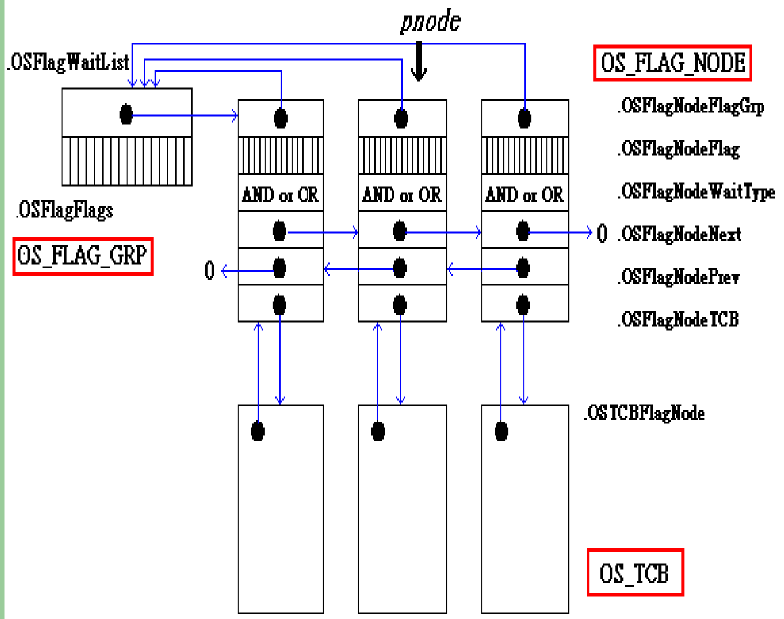


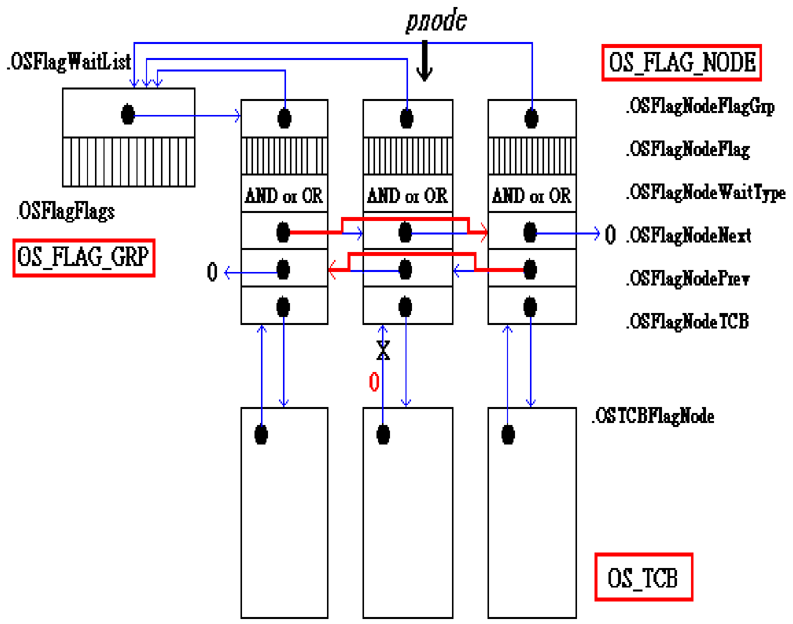
Make a waiting task ready to run, OS_FlagUnlink() (cont.)



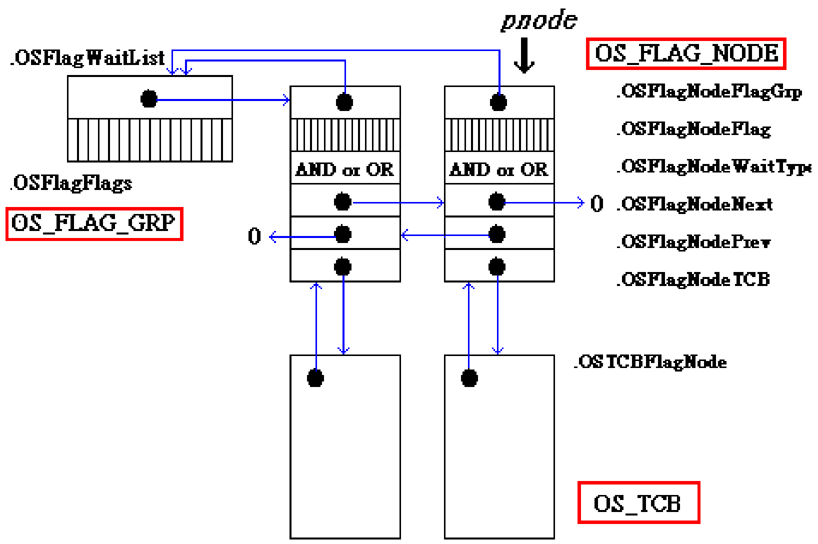


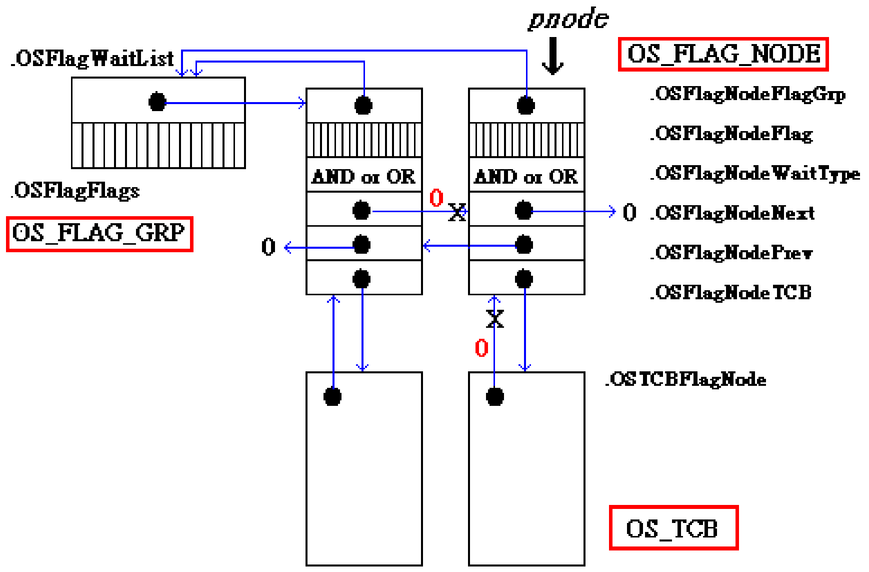
Make a waiting task ready to run, OS_FlagUnlink() (cont.)





Make a waiting task ready to run, OS_FlagUnlink() (cont.)





Looking for Event(s) of an Event Flag Group, OSFlagAccept()

- OSFlagAccept()
  - Is similar to OSFlagPend()
  - Caller is not suspended (ie. blocked)
  - Two differences
    - OSFlagAccept() can be called by an ISR
    - If conditions are not met , the call doesn’t block but returns an error code

Looking for Event(s) of an Event Flag Group, OSFlagAccept()

```
OS_FLAGS  OSFlagAccept
      (OS_FLAG_GRP *pgrp, OS_FLAGS
      flags, INT8U wait_type, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
      OS_CPU_SR     cpu_sr;
#endif

   OS_FLAGS      flags_cur;
   OS_FLAGS      flags_rdy;
   BOOLEAN       consume;
#if OS_ARG_CHK_EN > 0
   if (pgrp == (OS_FLAG_GRP *)0) {
      *err = OS_FLAG_INVALID_PGRP;
      return ((OS_FLAGS)0);
   }
 if (pgrp->OSFlagType !=
      OS_EVENT_TYPE_FLAG) {
       *err = OS_ERR_EVENT_TYPE;
       return ((OS_FLAGS)0);
   }
#endif
   if (wait_type & OS_FLAG_CONSUME) {
       wait_type &= ~OS_FLAG_CONSUME;
       consume    = TRUE;
   } else {
       consume    = FALSE;
   }

 *err = OS_NO_ERR;
OS_ENTER_CRITICAL();
   switch (wait_type) {
```

Looking for Event(s) of an Event Flag Group, OSFlagAccept() (cont.)

```
case OS_FLAG_WAIT_SET_ALL:
            flags_rdy = pgrp->OSFlagFlags &
      flags;
            if (flags_rdy == flags) {
                if (consume == TRUE) {
                 pgrp->OSFlagFlags &= ~flags_rdy;
                }
            } else {
              *err  = OS_FLAG_ERR_NOT_RDY;
            }
            flags_cur = pgrp->OSFlagFlags;
            OS_EXIT_CRITICAL();
            break;
```

```
case OS_FLAG_WAIT_SET_ANY:
            flags_rdy = pgrp->OSFlagFlags &
      flags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags &=
      ~flags_rdy;
                }
            } else {
                *err  =
      OS_FLAG_ERR_NOT_RDY;
            }
            flags_cur = pgrp->OSFlagFlags;
            OS_EXIT_CRITICAL();
            break;
```

Looking for Event(s) of an Event Flag Group, OSFlagAccept() (cont.)

```
#if OS_FLAG_WAIT_CLR_EN > 0
        case OS_FLAG_WAIT_CLR_ALL:
             flags_rdy = ~pgrp->OSFlagFlags &
       flags;
             if (flags_rdy == flags) {
                 if (consume == TRUE) {
                     pgrp->OSFlagFlags |= flags_rdy;
                 }
             } else {
                *err  = OS_FLAG_ERR_NOT_RDY;
             }
             flags_cur = pgrp->OSFlagFlags;
             OS_EXIT_CRITICAL();
             break;
```

```
 case OS_FLAG_WAIT_CLR_ANY:
             flags_rdy = ~pgrp->OSFlagFlags &
       flags;
             if (flags_rdy != (OS_FLAGS)0) {
                 if (consume == TRUE) {
                     pgrp->OSFlagFlags |= flags_rdy;
                 }
             } else {
                *err  = OS_FLAG_ERR_NOT_RDY;
             }
             flags_cur = pgrp->OSFlagFlags;
             OS_EXIT_CRITICAL();
             break;
#endif
```

Looking for Event(s) of an Event Flag Group, OSFlagAccept() (cont.)

```
 default:
        OS_EXIT_CRITICAL();
        flags_cur = (OS_FLAGS)0;
        *err      = OS_FLAG_ERR_WAIT_TYPE;
             break;
    }
    return (flags_cur);
}
```

Querying an Event Flag Group, OSFlagQuery()

```
OS_FLAGS  OSFlagQuery (OS_FLAG_GRP
       *pgrp, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    OS_FLAGS   flags;


#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    } if (pgrp->OSFlagType !=
       OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
OS_ENTER_CRITICAL();
    flags = pgrp->OSFlagFlags;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
    return (flags);
}
```