



MicroC-OS-II

(synchronization 2)



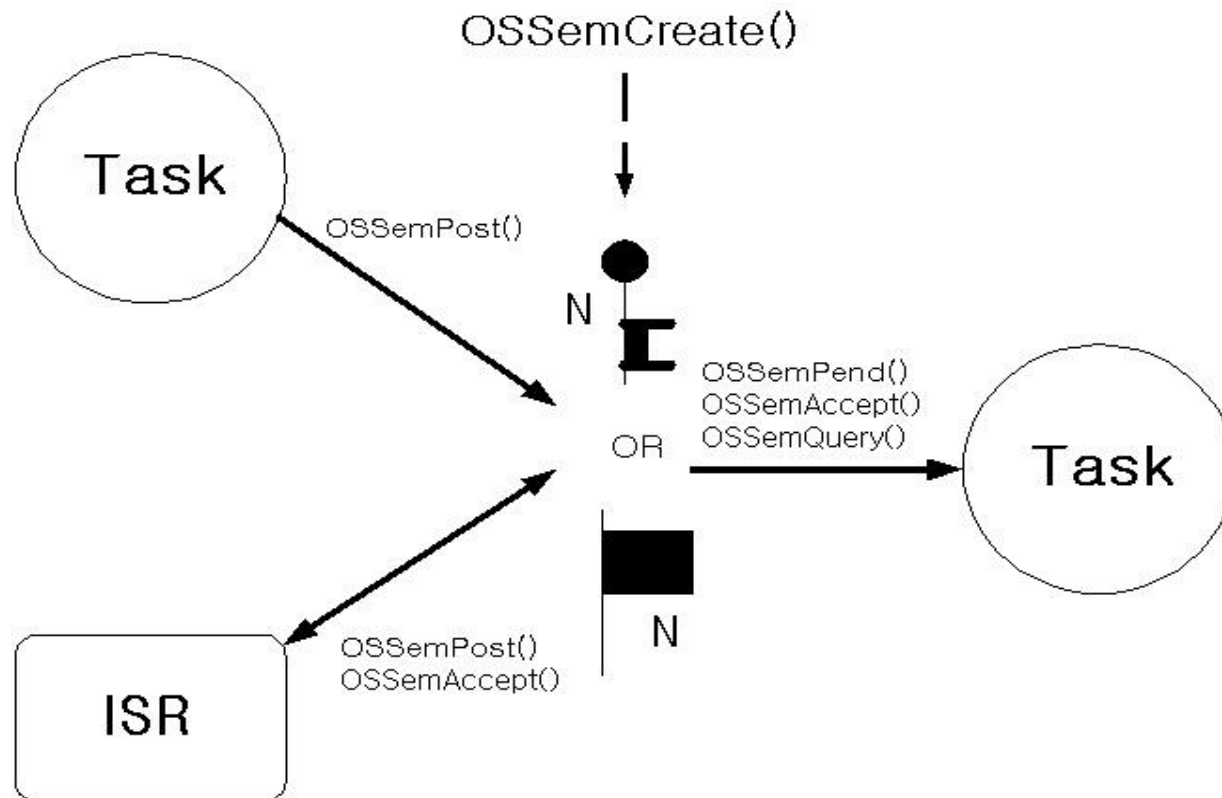
Outline

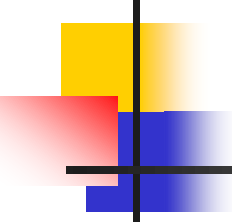
- Task synchronization
 - **Event control block**
 - **Mailbox**
 - **Message queue**
 - Semaphore
 - (Mutex semaphore)
 - Event flag



Semaphore (counting semaphore)

- **OSSemCreate()**
 - 세마포어 생성
- **OSSemPost()**
 - 세마포어에 이벤트 발생을 알림(signal)
 - 세마포어를 기다리는 태스크 중 가장 우선 순위가 높은 태스크가 실행
- **OSSemPend()**
 - 세마포어의 이벤트 발생을 기다림(wait)
 - 세마포어 값을 1 감소
 - 세마포어 값이 양수: **success**
 - 세마포어 값이 0: 다른 태스크가 깨워 줄 때까지 대기상태
 - **timeout값을 지정해 둔다면, timeout시간까지만 대기상태**
 - **timeout 시간이 지나면 에러 값을 가지고 리턴**
- **OSSemAccept()**
 - **OSSemPend()와 비슷하나, 세마포어가 사용가능하지 않더라도 대기하지 않음**
- **OSSemQuery()**
 - 현재 세마포어의 상태를 알 수 있음





```

OS_EVNET *OSSemCreate(INT16U cnt)
{
    OS_EVENT *pevent;

    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;
        OSEventWaitListInit(pevent);
    }
    return (pevent);
}

```

(0 to 65535)

자유 ECB 리스트로부터 ECB를 하나 얻는다.
OSEventFreeList의 ECB 포인터는 다음 ECB를
가리키도록 조정

ECB type을 세마포어로 지정한다
세마포어 초기값을 지정한다
wait list를 초기화한다.

- OSSemCreate()이 리턴하기 직전의 ECB

OS_EVENT

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

.OSEventTbl []

ALL
initialized
to
0x00



```
void OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
```

```
{
```

1. pevent가 가리키는 ECB가 세마포어인지 검사한다. (pevent->OSEventType)

2. 세마포어가 사용가능한 경우 (cnt > 0) count를 1 감소시키고 리턴

3. 세마포어가 사용가능하지 않은 경우 (cnt = 0)

다른 태스크가 세마포어에 시그널을 보낼 때까지 sleep

3.1 sleep 하도록 하기 위해 태스크의 TCB의 상태를 OS_STAT_SEM로 만들

3.2 timeout값의 지정 (OSTCBCur->OSTCBDly = timeout)

3.3 태스크를 sleep 상태로 만든다.(OS_EventTaskWait())

4. OS_Sched() 호출하여 우선 순위 높은 태스크를 실행시킨다.

5. 다시 이 태스크가 실행될 때 timeout이 발생한 것이면,
OSEventTO()함수가 호출되고, ECB에의 링크가 끊어진다.

```
}
```



```
void OSSemPost (OS_EVENT *pevent)
```

```
{  
    1. pevent가 가리키는 ECB가 세마포어 인지 검사한다. (pevent->OSEventType)  
    2. 세마포어를 기다리는 태스크가 있다면  
        2.1 그중 우선순위가 가장 높은 태스크를 ECB 대기리스트에서 제거하고는  
            실행 상태를 만든다. <OS_EventTaskRdy() 함수 호출>  
        2.2 OS_Sched()을 호출한다.  
    3. 세마포어를 기다리는 태스크가 없다면 세마포어 counter를 1 증가시킨다.  
}
```

```
INT16U OSSemAccept (OS_EVENT *pevent)
```

```
{  
    1. 세마포어 count가 0보다 클 때만 1감소되고 return한다.  
    2. OSSemAccept()를 호출한 부분에서는 return값을 확인하여, 0 이 아닌 경우  
       에만 자원을 할당받아 사용한다.  
}
```

NO Wait

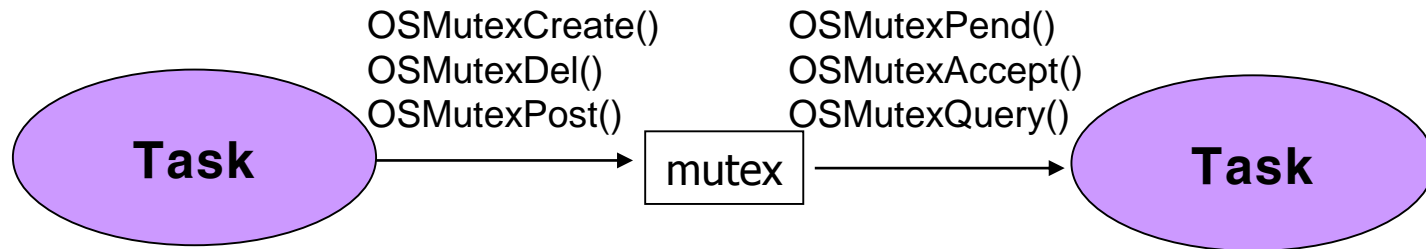
기타: 세마포어 삭제 함수: OSSemDel(), 세마포어 상태 얻기 함수: OSSemQuery()



Mutual Exclusion Semaphores

(for later)

Mutex semaphore



우선 순위 전도 현상의 해결
=> Priority Inheritance Protocol, Priority Ceiling Protocol

MicroC/OS II 의 특징
한 개의 태스크는 한 개의 우선 순위를 갖는다.
⇒ 현재 사용되지 않는 높은 우선 순위를
우선 순위 상속 순위 (Priority Inheritance Priority (PIP)) 로 세팅



■ Priority Inheritance Protocol in MicroC/OS-II

- For example,

- OSMutexCreate(9,&err);

pip

Task1: priority 10

Task2: priority 12

Task3: priority 15

If Task1 (priority 10) is blocked by Task3 (priority 15) due to mutex, then Task3's priority is enhanced to 9.



Mutex creation

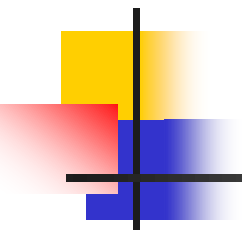
PIP

```
OS_EVENT *OSMutexCreate(INT8U prio, INT8U *err)
```

```
{
```

1. ISR에서의 호출인지 체크, PIP가 OS에서 제공하는 범위에 있는지 체크
2. PIP의 태스크가 OS에서 사용되고 있지 않은지 확인하고, OSTCBPrioTbl[prio]에 NULL 값이 아닌 값을 넣어서 예약
3. EventFreeList에서 ECB 받음
4. ECB->OSEventType = OS_EVENT_TYPE_MUTEX;
ECB->OSEventCnt = (prio << 8) | OS_MUTEX_AVAILABLE;
ECB->OSEventPtr = (void *) 0;
OSEventWaitListInit(pevent);
5. Return ECB;

```
}
```

- 
- OSMutexCreate()이 리턴하기 직전의 ECB

OS_EVENT_TYPE_MUTEX							
PIP				0xFF			
(void *) 0							
0x00							
7	6	5	4	3	2	1	0
...							
63	62	61	60	59	58	57	56



Mutex deletion

```
OS_EVENT *OSMutexDel(OS_EVENT *pevent, INT8U opt, INT8U *err)
```

```
{  
    1. ISR에서의 호출인지 체크, 삭제할 ECB가 존재하거나 뮉텍스가 맞는지 여부 확인  
    2. 뮉텍스를 기다리는 태스크가 있는지 검사한다. (tasks_waiting 값 갱신)  
    3. case OS_DEL_NO_PEND: // 아무 태스크도 기다리지 않는 경우 삭제  
        if( tasks_waiting == TRUE)  
            *err = OS_ERR_TASK_WAITING;  
        else  
            ECB 반환;  
    4. case OS_DEL_ALWAYS: // 항상 삭제  
        모든 대기중인 태스크를 ready 상태로 변경;  
        ECB 반환;  
        if(tasks_waiting == TRUE)  
            OS_Sched();  
        return;  
}
```

Mutex Pend

```
void OSMutexPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
{
```

1. validity check

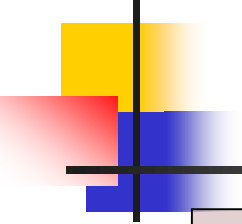
ISR에서 호출한 것인가?, 해당 ECB를 OSMutexCreate()가 생성한 것인가? 등

2. pevent->OSEventCnt 하위 8비트가 0xFF라면 뮉텍스 사용 가능

```
if (뮉텍스의 사용이 가능한가?) {
    pevent->OSEventCnt &= 0xFF00;
    pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;
    pevent->OSEventPtr = (void *)OSTCBCur;
    return;
}
```

아닐 경우 3. 우선 순위 계승 프로토콜

OS_EVENT_TYPE_MUTEX								
PIP				OSTCBCur->OSTCBPrio				
(void *)OSTCBCur								
0x00								
7	6	5	4	3	2	1	0	
...								
63	62	61	60	59	58	57	56	



```
if ( 뮤텍스 소유자 태스크 우선순위 < 호출한 태스크 우선순위 )  
{
```

```
    뮤텍스 소유자 태스크 우선 순위 = PIP;  
    새로운 우선 순위 할당 후 상태 결정 위해  
    if (뮤텍스 소유자 태스크 준비상태 인가?) {  
        준비 상태 해제;  
        rdy = TRUE;  
    }  
    else rdy = FALSE;  
    새로운 PIP를 위해서 TCB 내용 계산;  
    if(rdy == TRUE) PIP 의 태스크를 준비상태로 설정;  
}
```

```
    호출 태스크 상태 -> 뮤텍스 대기 상태
```

```
    OSTCBCur -> OSTCBStat |= OS_STAT_MUTEX;
```

```
    OSTCBCur -> OSTCBDly = timeout;
```

```
    OS_EVENTTaskWait(pevent); // 실제 대기 상태로 만듦
```

```
    OS_Sched(); // 준비상태가 아니므로 다음 우선순위의 태스크 호출
```

```
    // 대기 시간 만료로 인한 준비 상태로 상태 변화
```

```
    if (태스크가 뮤텍스를 기다리는가?)
```

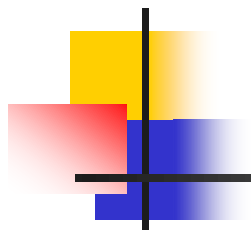
```
        OS_EVNETO(pevent); // 뮤텍스 대기 리스트에서 삭제
```

```
}
```




Mutex Post

```
INT8U OSMutexPost ( OS_EVENT *pevent)
{
    if ( 현 태스크가 실제 뮤텍스를 소유하는가?)
        뮤텍스 소유자의 우선순위 -> PIP 또는 자체 저장된 우선순위여야 함
    if ( 상위 태스크가 뮤텍스 소유자의 우선순위를 PIP로 변경한 경우) {
        PIP레벨의 호출 태스크 준비리스트에서 삭제;
        원래 우선순위로 준비리스트에 삽입;
    }
    if (뮤텍스를 기다리는 태스크가 있나?)
    {
        OS_EventTaskRdy();
        OS_Sched();
    }
    OSEventCnt의 하위 8비트 = 0xFF; // 뮤텍스 즉시 사용 가능
    OSEvnetPtr = 0;
}
```



```
INT8U OSMutexAccept (OS_EVENT *pevent, INT8U *err)
{
    if (뮤텍스 사용 가능한가?)
    {
        pevent->OSEventCnt &= 0xFF00;
        pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;
        pevent->OSEventPtr = (void *)OSTCBCur;
    }
    return (0); // 뮤텍스 얻지 못함
}
```



Example-MutexSemaphore

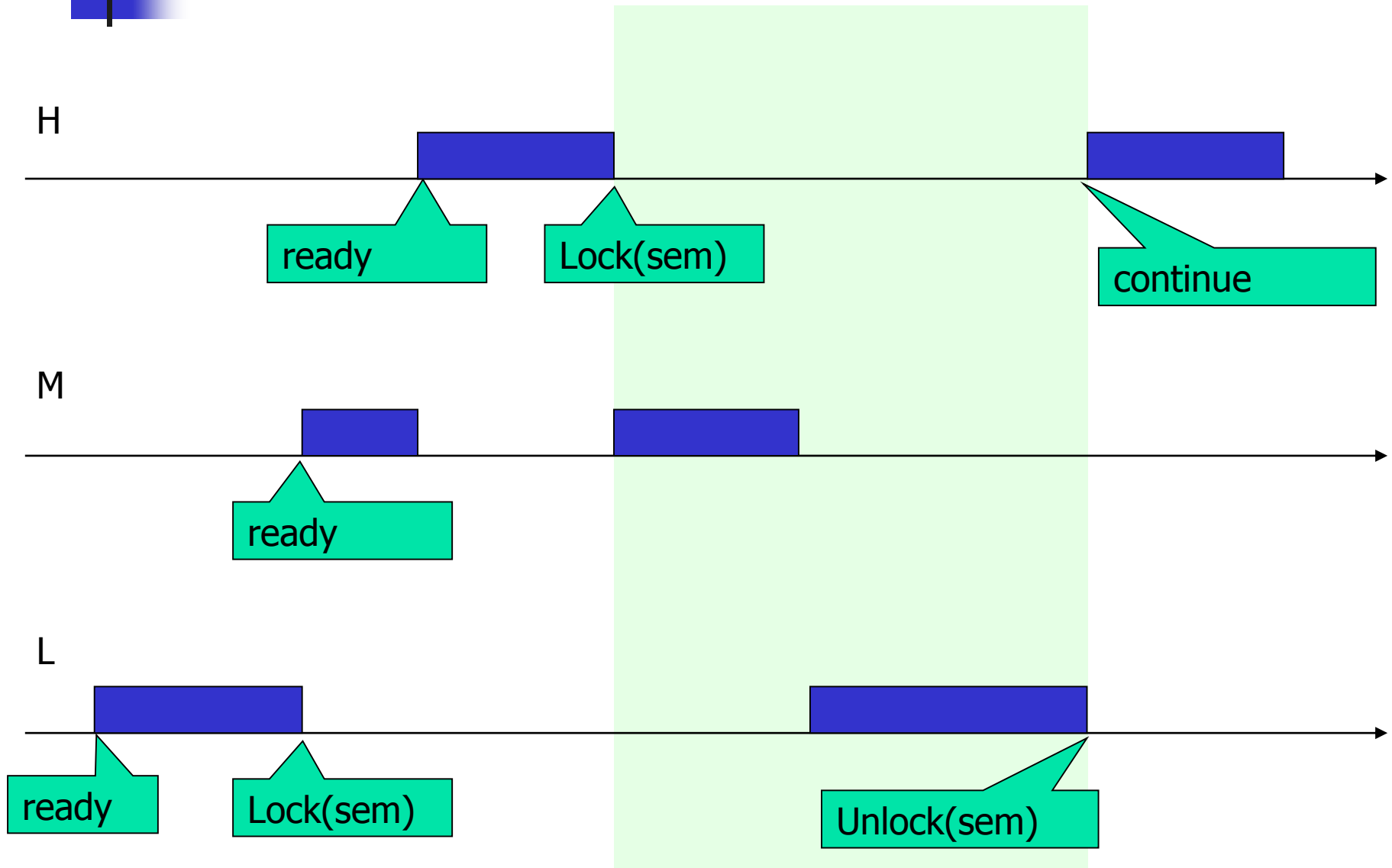
```
OSMutexCreate(VH, &err);
```

```
void taskPrioH {  
    while(1) {  
        /*...*/  
        OSMutexPend(Mutex, 0, &err);  
        /*...*/  
        OSMutexPost(Mutex);  
    }  
}
```

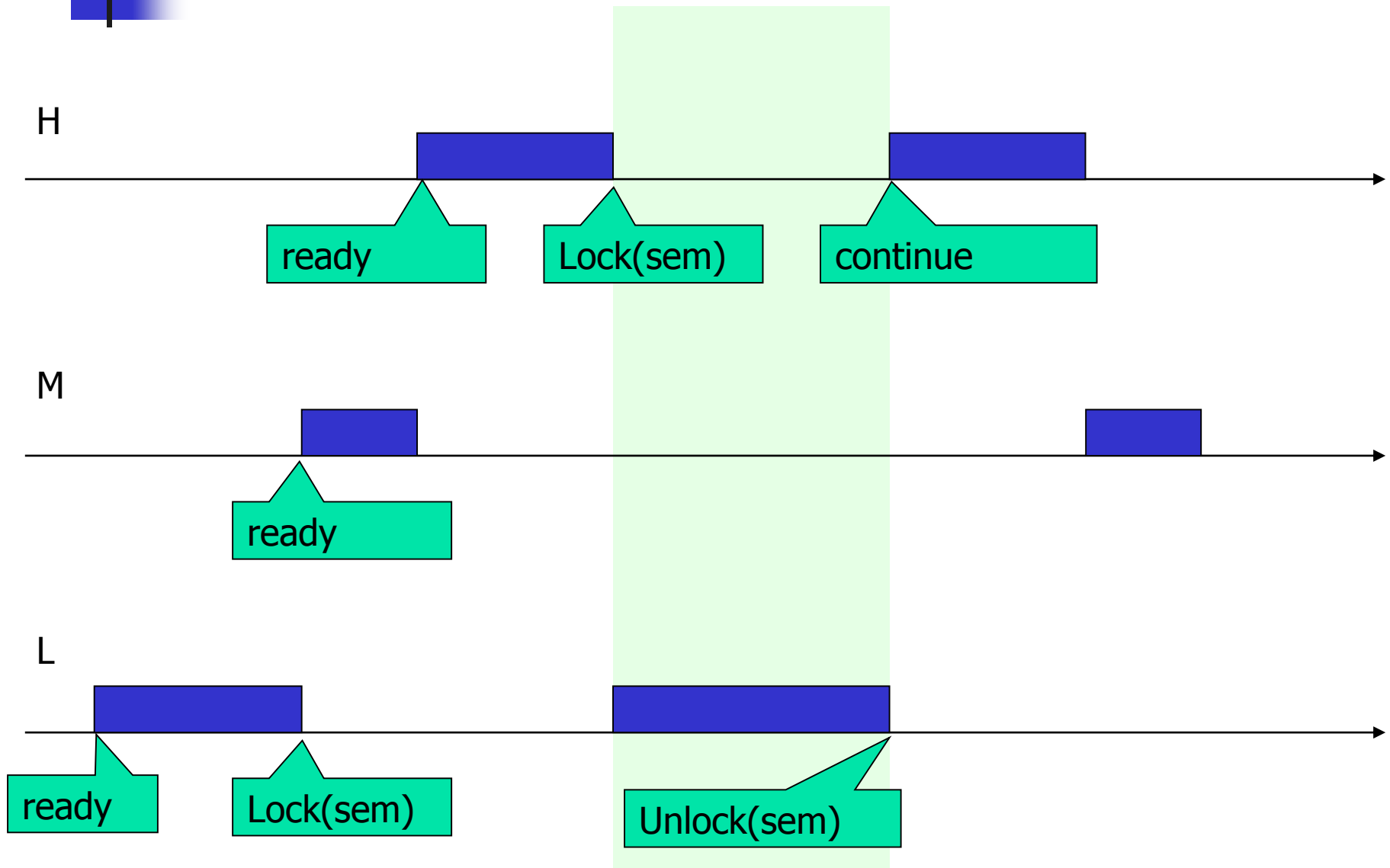
```
void taskPrioL {  
    while(1) {  
        /*...*/  
        OSMutexPend(Mutex, 0, &err);  
        /*...*/  
        OSMutexPost(Mutex);  
    }  
}
```

```
void taskPrioM {  
    while(1) {  
        /*...*/  
    }  
}
```

Example - without PIP

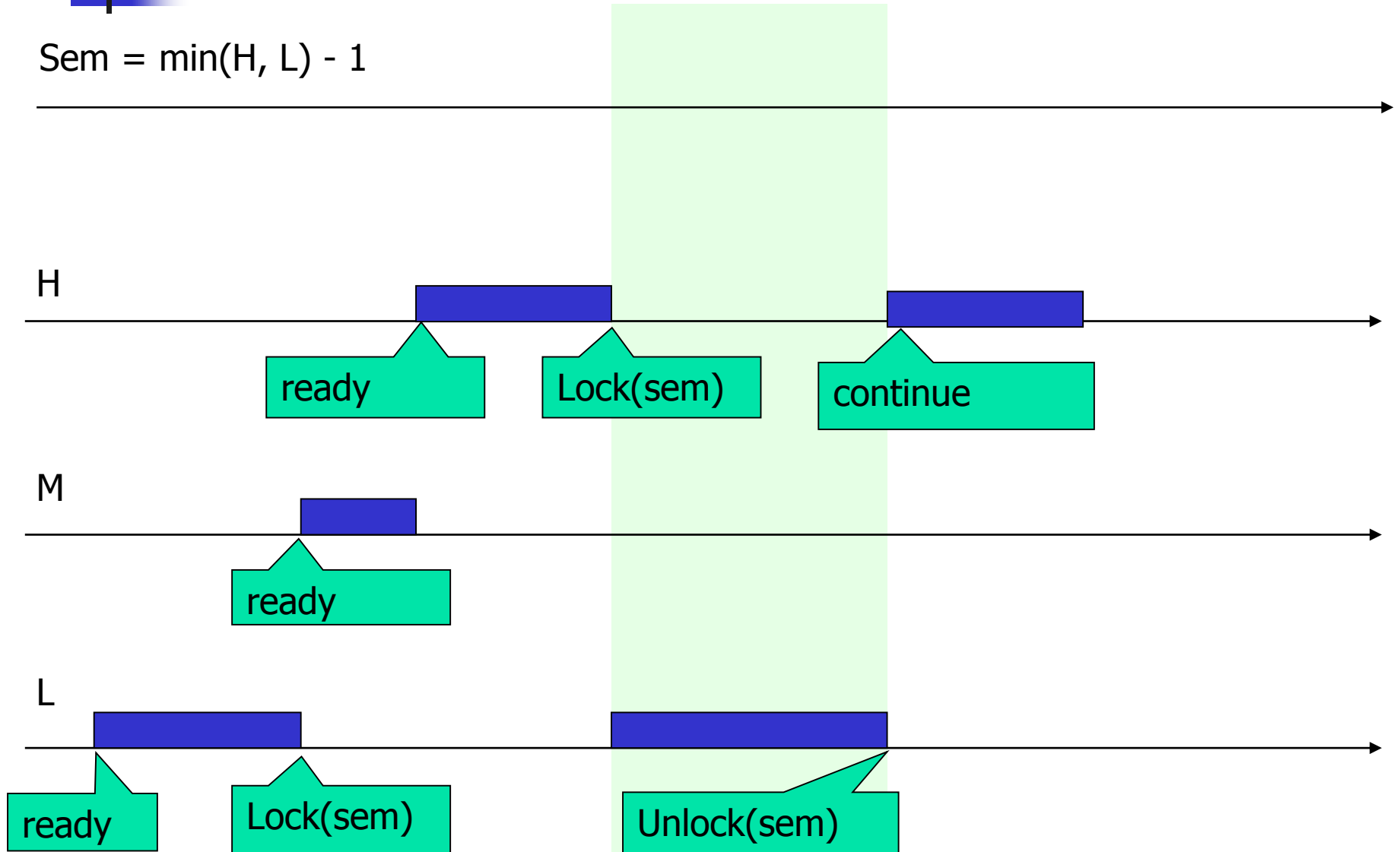


Example - with PIP



Example - μ C/OS-II

$$\text{Sem} = \min(H, L) - 1$$





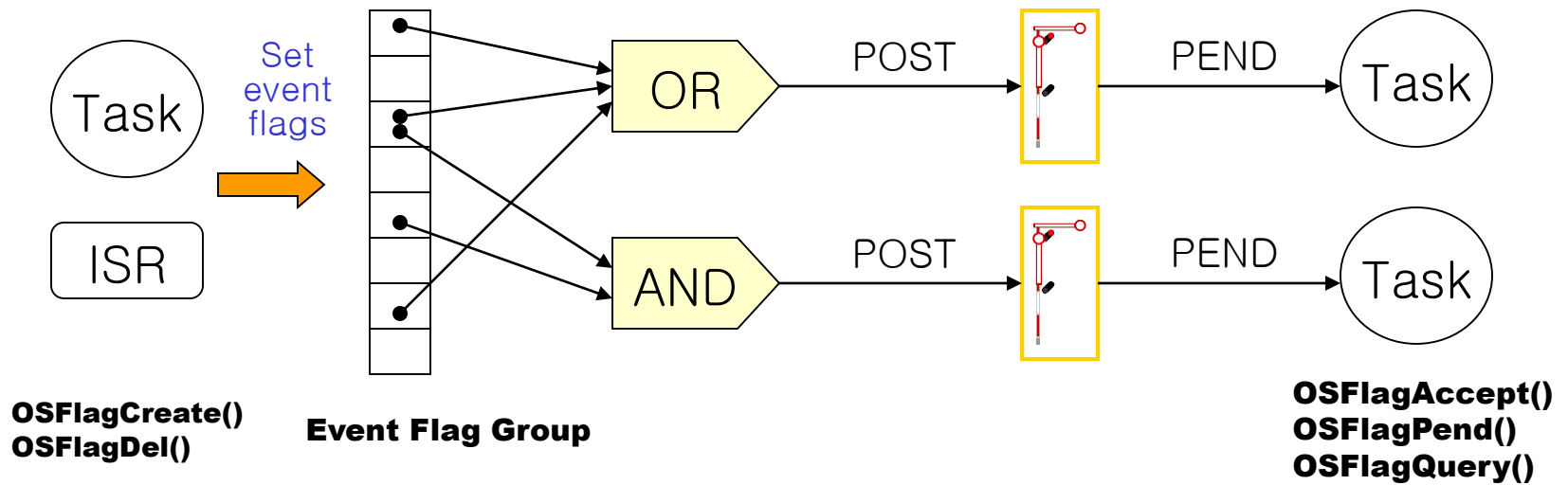
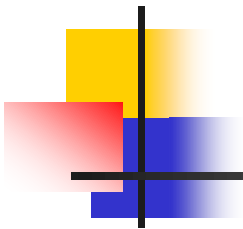
Event Flag



Event flag

- 이벤트 플래그

- 일종의 자료구조로서, 각 비트는 특정한 사건(event)에 대응됨.
- 이벤트가 발생하면 특정 비트를 1 또는 0으로 설정
- 하나 혹은 여러 이벤트에 대한 동기화 가능
- 연산
 - OSFlagCreate()
 - OSFlagDel()
 - OSFlagPend()
 - OSFlagPost()
 - OSFlagAccept()
 - OSFlagQuery()

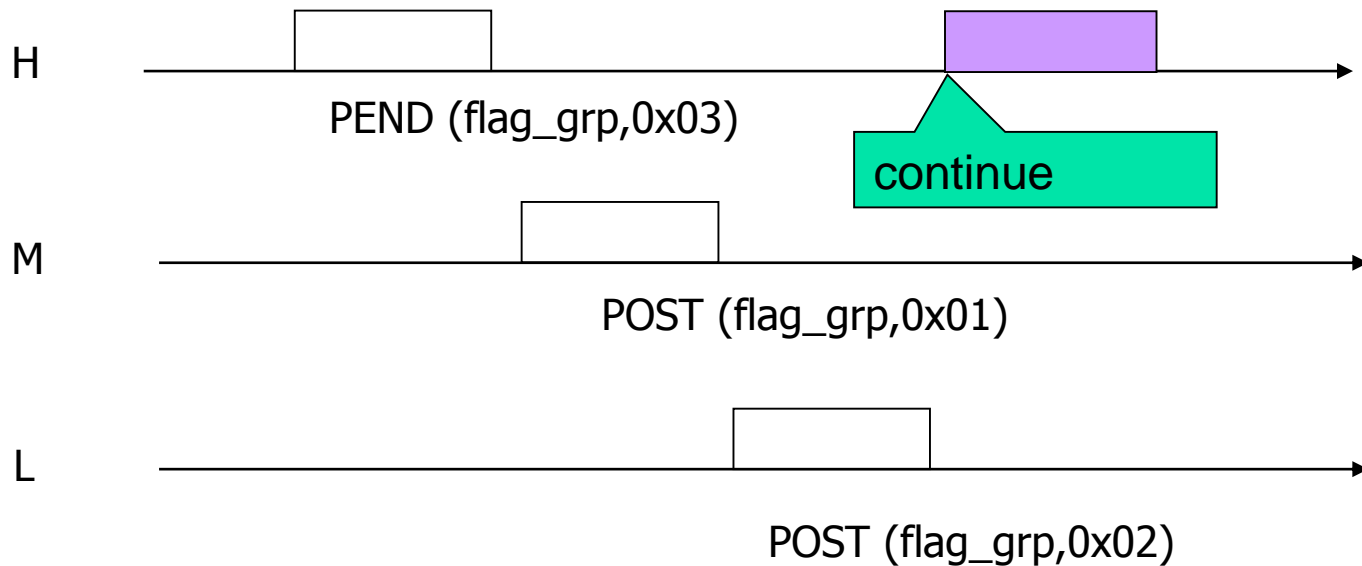


메시지 전달은 불가함

■ 사용 예-1

- 태스크 동기화에 유용

- 우선 순위 높은 태스크를 우선 순위 낮은 2개의 태스크가 특정 시점까지 블록킹하는 경우

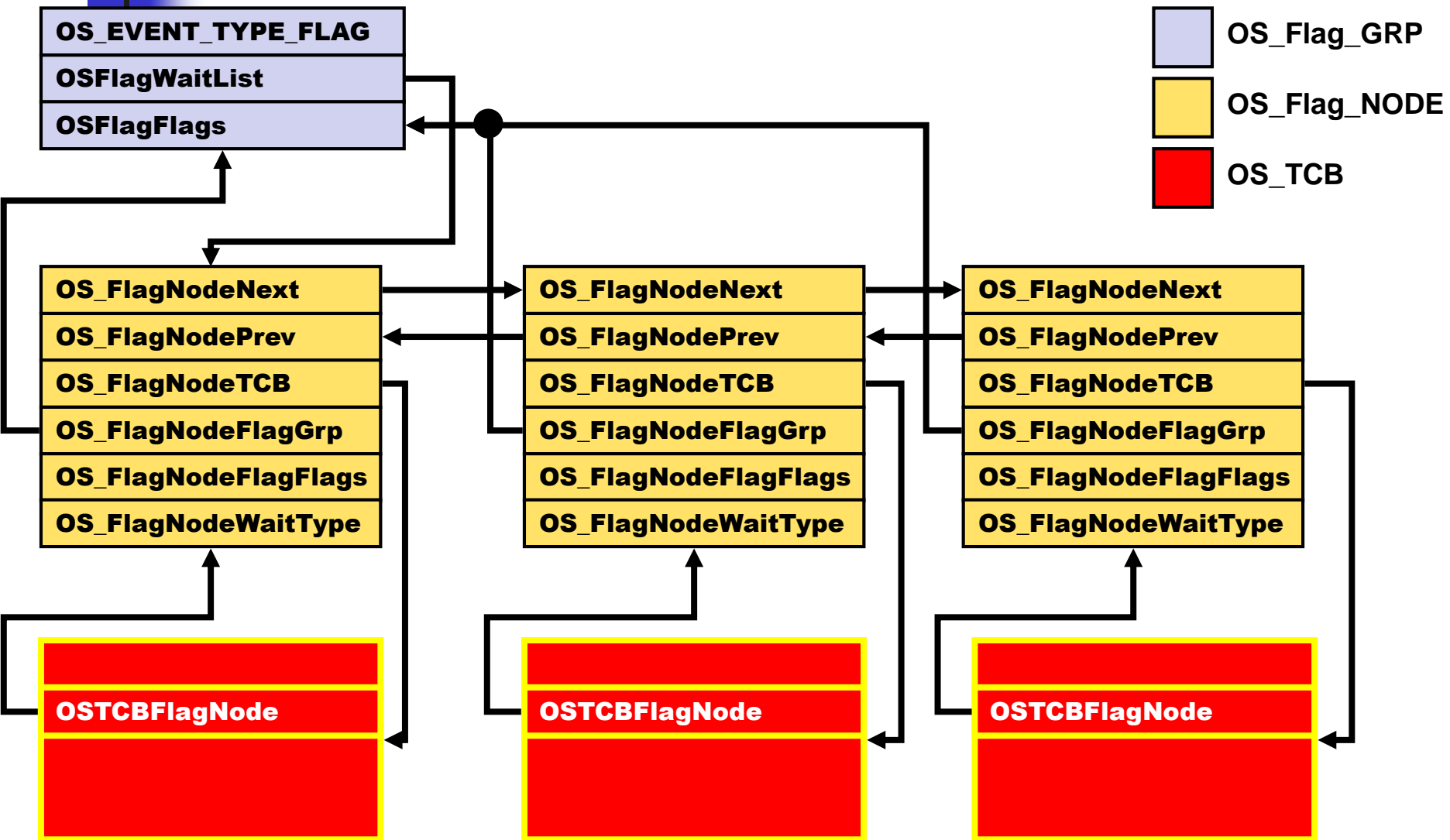


Example – 2

우선 순위 낮은 2개의 태스크를 우선 순위 높은 태스크가 특정 시점까지 블록킹하는 경우



Event flag group, event flag nodes and TCBs





Event Flags Group Structure

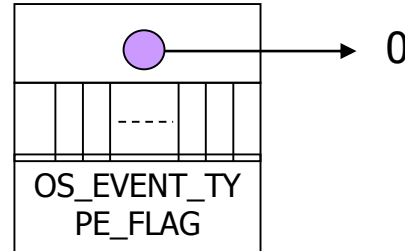
```
typedef struct {                                /* Event Flag Group */
    INT8U      OSFlagType;                      /* Should be set to OS_EVENT_TYPE_FLAG */
    void       *OSFlagWaitList;                /* Pointer to first NODE of task waiting on
                                              event flag */
    OS_FLAGS    OSFlagFlags;                   /* 8, 16 or 32 bit flags */
} OS_FLAG_GRP;
```

Event Flags Group Node Structure

```
typedef struct {                                /* Event Flag Wait List Node */
    void       *OSFlagNodeNext;                /* Pointer to next NODE in wait list */
    void       *OSFlagNodePrev;                /* Pointer to previous NODE in wait list */
    void       *OSFlagNodeTCB;                 /* Pointer to TCB of waiting task */
    void       *OSFlagNodeFlagGrp;             /* Pointer to Event Flag Group */
    OS_FLAGS    OSFlagNodeFlags;               /* Event flag to wait on */
    INT8U      OSFlagNodeWaitType;             /* Type of wait */
} OS_FLAG_NODE;
```

■ API

- OSFlagCreate



- OSFlagDel

- OSFlagPend

- Option

- OS_FLAG_WAIT_SET_ALL, OS_FLAG_WAIT_SET_ANY
- OS_FLAG_WAIT_CLR_ALL, OS_FLAG_WAIT_CLR_ANY
- OS_FLAG_CONSUME

- 호출하는 함수

- OS_FlagBlock
 - 플래그 대기 리스트에 추가
- 이후 OS_sched() 호출
 - 호출 태스크는 더 이상 실행할 수 없음
- OSFlagUnlink ()호출
 - Timeout 에 의한 만료
 - Event flag node의 링크 해제



■ API

- OSFlagPost()

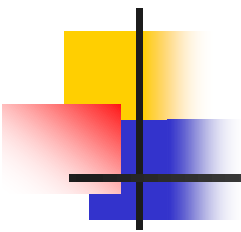
- 지정된 플래그를 켜거나 끄는 역할
- OS_FlagTakRdy() 호출

- OSFlagAccept()

- 대기하지 않고 플래그 그룹으로부터 원하는 이벤트가 발생했는지 알아보는 역할

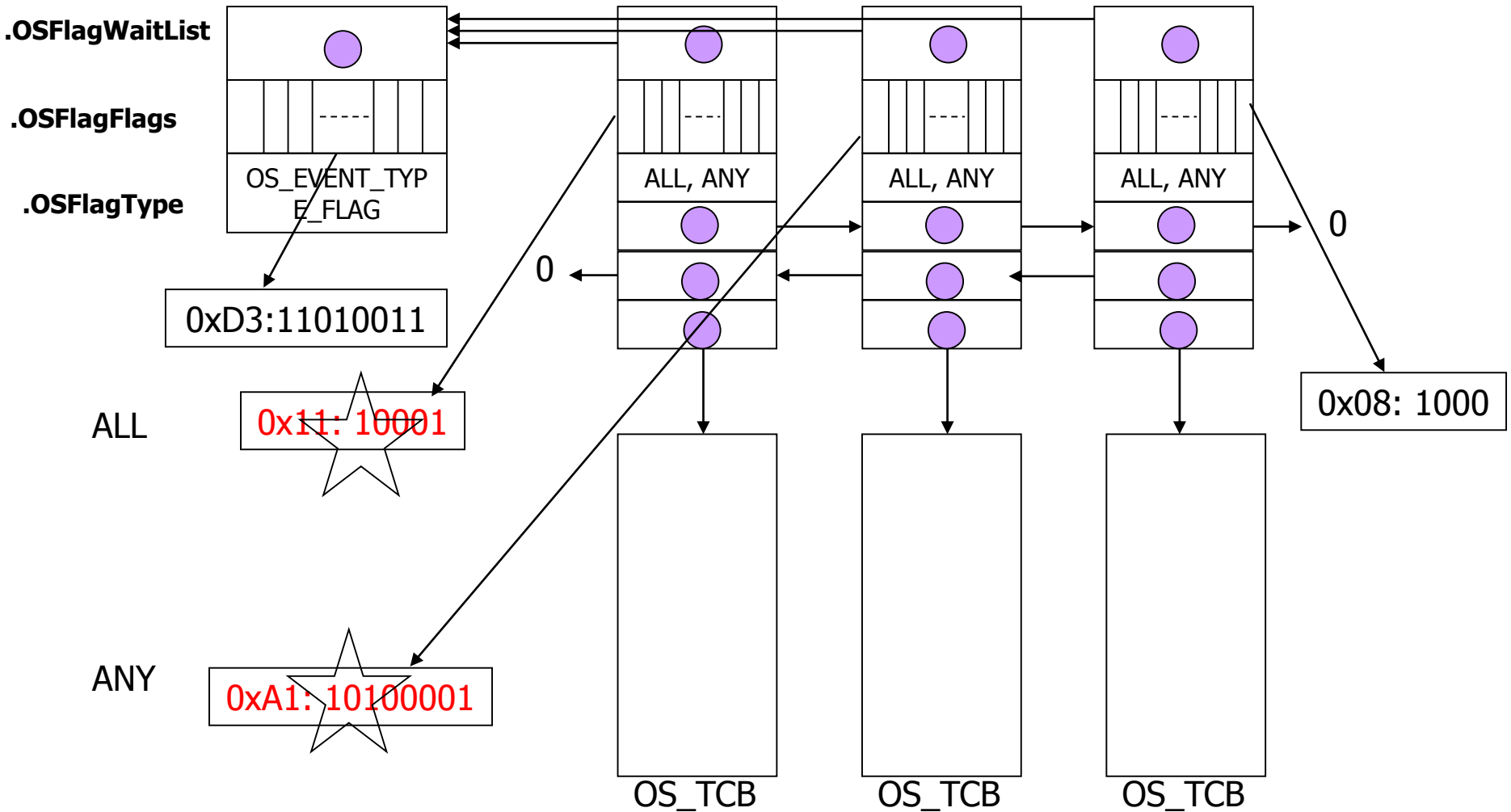
- OSFlagQuery()

- 이벤트 플래그 그룹의 현재값



OS_FLAG_GRP

OS_FLAG_NODE





1. Declaration:

```
OS_FLAG_GRP *e_grp;  
INT8U err;
```

2. Creation

```
e_grp = OSFlagCreate(0x00, &err);
```

3. Pend

```
OSFlagPend(e_grp, 0x01, OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME, 0, &err);
```

4. Post

```
OSFlagPost(e_grp, 0x01, OS_FLAG_SET, &err);
```

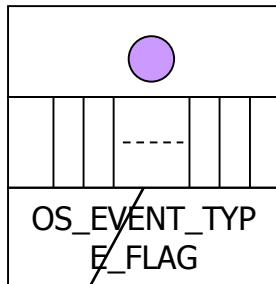
OS_FLAG_GRP

OS_FLAG_NODE

.OSFlagWaitList

.OSFlagFlags

.OSFlagType

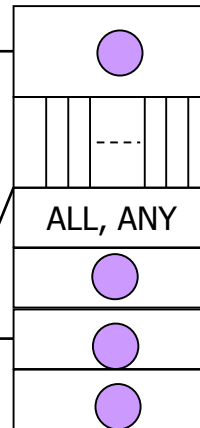


0x00:00000000

ALL

0x01: 0000001

0



OS_TCB

1. e_grp = OSFlagCreate(0x00, &err);

2. OSFlagPend(e_grp, 0x01, OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME, 0, &err);

3. OSFlagPost(e_grp, 0x01, OS_FLAG_SET, &err);