Embedded Software
(MicroC/OS-II kernel)

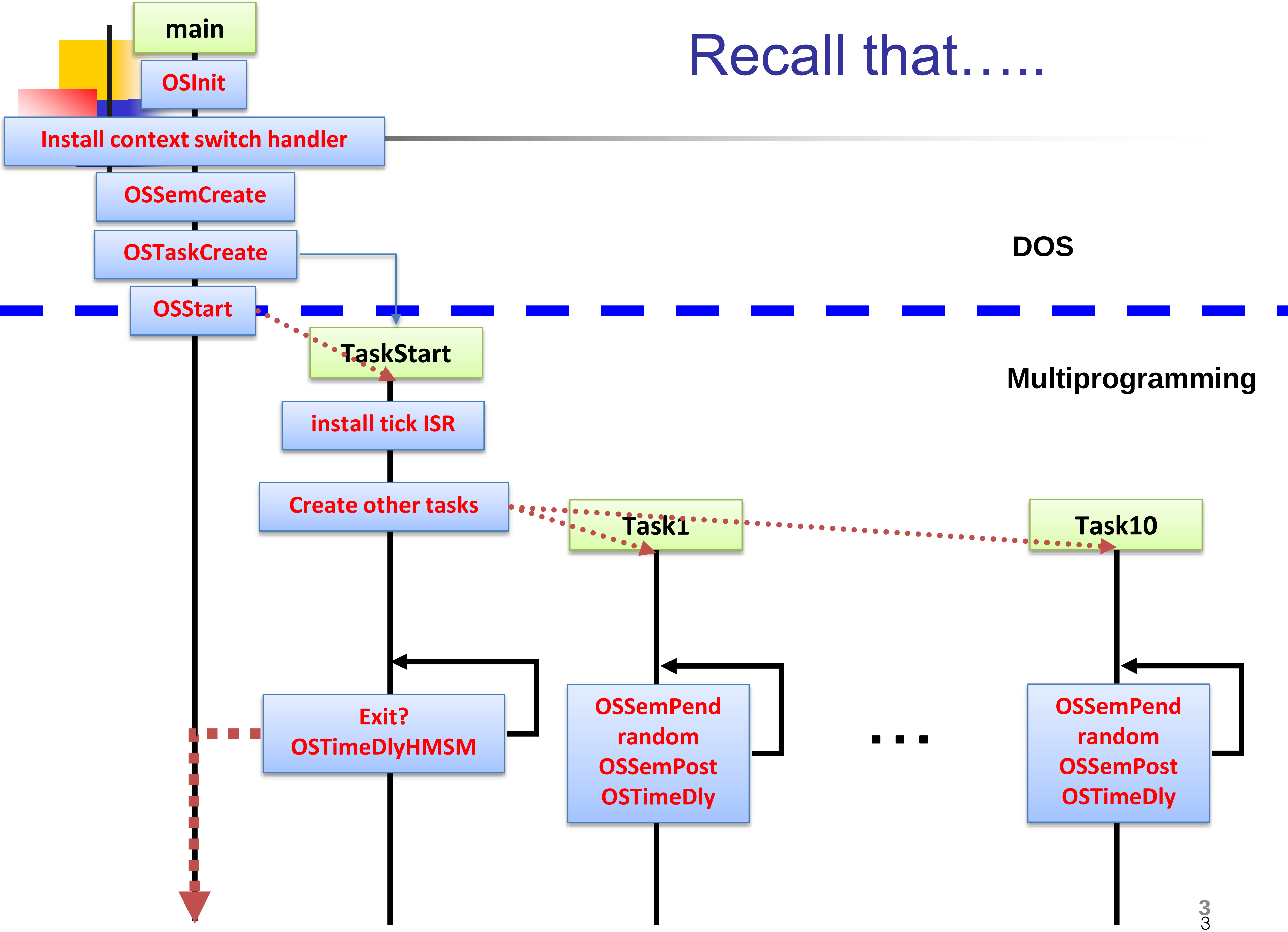
MicroC/OS-II kernel(scheduler)

Fall, 2018

Outline

- Kernel
  - Task types
  - Task Scheduling
    - Overview
    - State
    - TCB (Task Control Block)
    - Ready list
    - Scheduling
    - ISR
  - Clock ticks

Recall that…..


main
OSInit
Install context switch handler
OSSemCreate
OSTaskCreate
OSStart
DOS
TaskStart
Multiprogramming
install tick ISR
Create other tasks
Task1
Task10
Exit?
OSTimeDlyHMSM
OSSemPend
random
OSSemPost
OSTimeDly
. . .
OSSemPend
random
OSSemPost
OSTimeDly

Task types

- Two types
  - Infinite loop
  - After work, task may delete itself

- Infinite loop

```
void MyTask(void *pdata)
{
    while(1)
    {
      Code…
      OSMboxPend();
      OSQPend();
      OSSemPend();
      OSTaskDel(OS_PRIO_SELF);
      OSTaskSuspend(OS_PRIO_SELF);
      OSTimeDly();
      OSTimeDlyHMSM();
      Code…
    }
}
```

- After work, task may delete itself

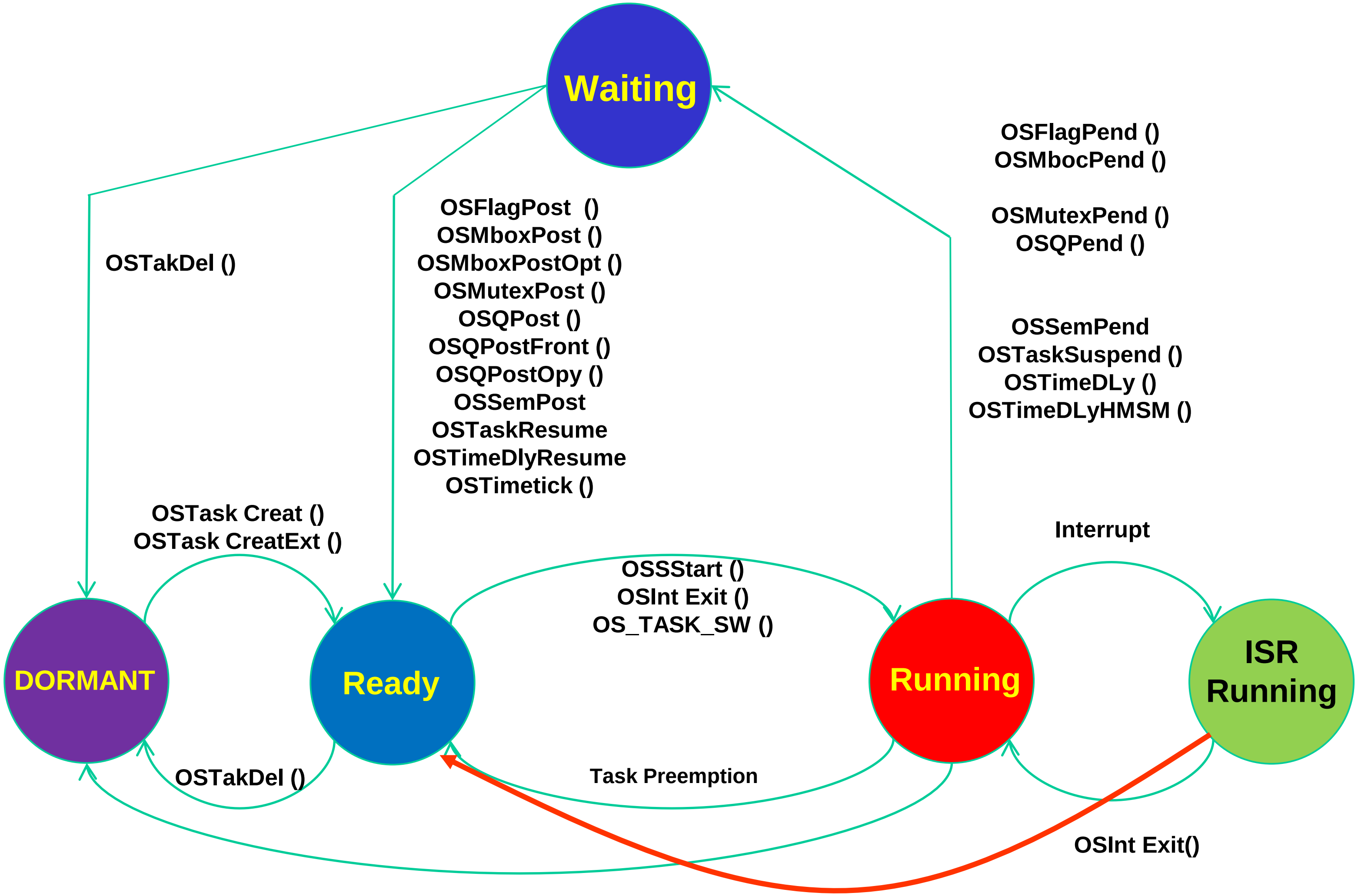
```
void MyTask(void *pdata)
{
   Code…
   OSTaskDel(OS_PRIO_SELF);
}
```

Task scheduling - overview

- Number of tasks
  - 64
- Static priority scheduling
- Priority
  - Unique identifier of tasks
  - 0 ~ 63
  - 0 : The highest priority,
  - 63: The lowest priority
  - Priority may be changed using OSTaskChangePrio() function
  - Assignment
    - 0, 1, 2, 3, …, OS_LOWEST_PRIO-3, OS_LOWEST_PRIO-2, OS_LOWEST_PRIO-1, OS_LOWEST_PRIO

Task scheduling – task state

- Task state transition



- Task state
  - DORMANT
    - Task is not registered to the OS
    - OSTaskCreate() or OSTaskCreateExt() changes the state of tasks to READY
  - READY
    - Task can be a candidate to be scheduled
  - RUNNING
    - CPU currently executes the task
  - Delayed
    - Task is temporarily stopped during some period
    - After expiration, its state is returned to READY
  - WAIT
    - Task is waiting for some event to occur
    - It cannot be scheduled
  - ISR
    - Interrupt routine is executed

Task scheduling –TCB

- Task control block (TCB)
  - Data structure that contains various information about task (one TCB per task)
  - When the task is scheduled, its information can be used for execution
  - Reside in main memory

- OS_TCB structure (* :OSTaskCreateExt())
  - OSTCBStkPtr
    - A pointer that indicates the top-of-stack
  - OSTCBExtPtr*
    - A pointer that indicates the extended TCB defined by a user
  - OSTCBStkBottom*
    - A pointer that indicates bottom-of-stack
  - OSTCBStkSize
    - Stack size

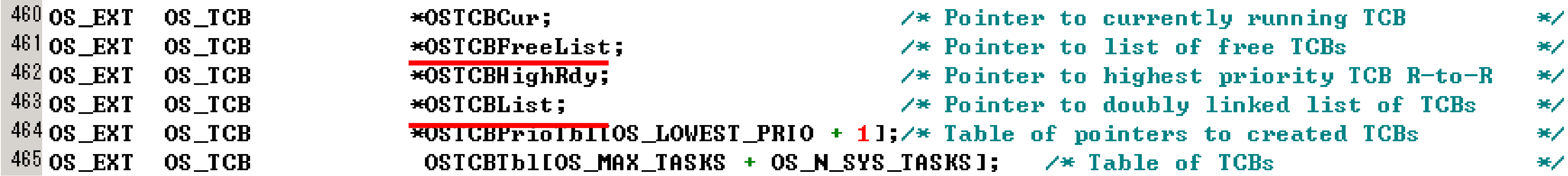
- OS_TCB structure (* :OSTaskCreateExt())
  - OSTCBNext, OSTCBPrev
    - Pointers used to maintain OS_TCB linked list
  - OSTCBOpt*
    - Optional flag used for task creation
      - *OS_TASK_OPT_STK_CHK*
        - *Investigate stack area when task is created*
      - *OS_TASK_OPT_STK_CLR*
        - *Make stack area with 0*
      - *OS_TASK_OPT_SAVE_FP*
        - *A flag indicating that task uses floating point operation*
  - OSTCBDly
    - Number of clock ticks until the timeout
  - OSTCBStat
    - Task state
  - Others
    - …

TCB특성

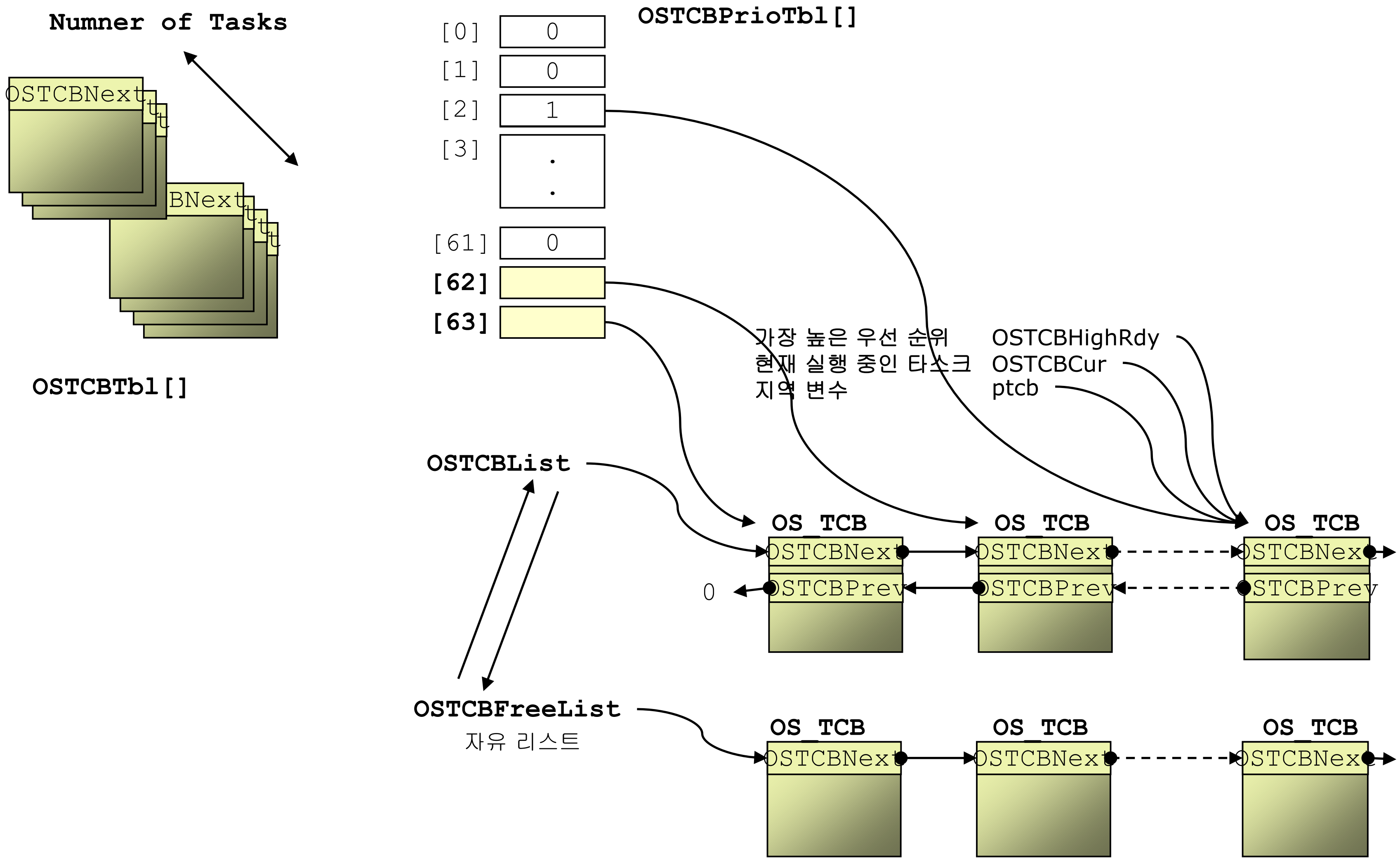
- 기타특징
  - All valid TCB’s are ***doubly linked***.
  - Free TCB’s are linked in a free list.

uCOS_II.h

.
.
.



```
OS_EXT  OS_TCB          *OSTCBCur;                        /* Pointer to currently running TCB      */
OS_EXT  OS_TCB          *OSTCBFreeList;                   /* Pointer to list of free TCBs          */
OS_EXT  OS_TCB          *OSTCBHighRdy;                    /* Pointer to highest priority TCB R-to-R */
OS_EXT  OS_TCB          *OSTCBList;                       /* Pointer to doubly linked list of TCBs */
OS_EXT  OS_TCB          *OSTCBPrioTbl[OS_LOWEST_PRIO + 1];/* Table of pointers to created TCBs     */
OS_EXT  OS_TCB           OSTCBTbl[OS_MAX_TASKS + OS_N_SYS_TASKS];    /* Table of TCBs              */
```



Numner of Tasks

OSTCBNext

BNext

OSTCBTbl[]

OSTCBPrioTbl[]

[0]	0
[1]	0
[2]	1
[3]	. .
[61]	0
[62]	
[63]	

가장 높은 우선 순위 OSTCBHighRdy

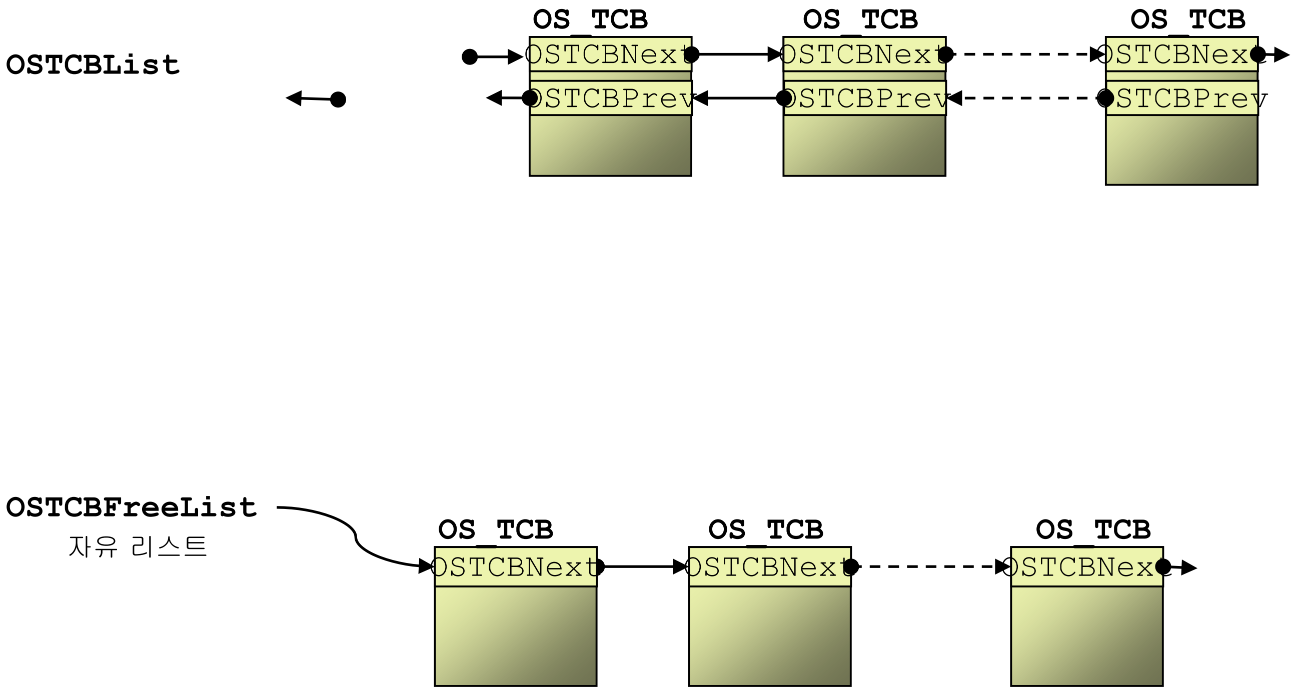
현재 실행 중인 타스크 OSTCBCur

지역 변수 ptcb

OSTCBList

OSTCBFreeList

자유 리스트



OSTaskCreate

Ready List

- Ready list is a special bitmap to reflect which task is currently in the ready state.
  - Each task is identified by its unique priority in the bitmap.
- A primary design consideration of the ready list is how to efficiently locate the highest-priority ready task.
  - The designer could trade some ROM space for an improved performance.
- If a linear list is adopted, it takes O(n) to locate the highest-priority ready task.
  - It takes O(log n) if a heap is adopted.
  - Under the design of ready list of μC/OS-II, it takes only O(1).
    - Note that the space consumption is much more than other approaches, and it also depends on the bus width.

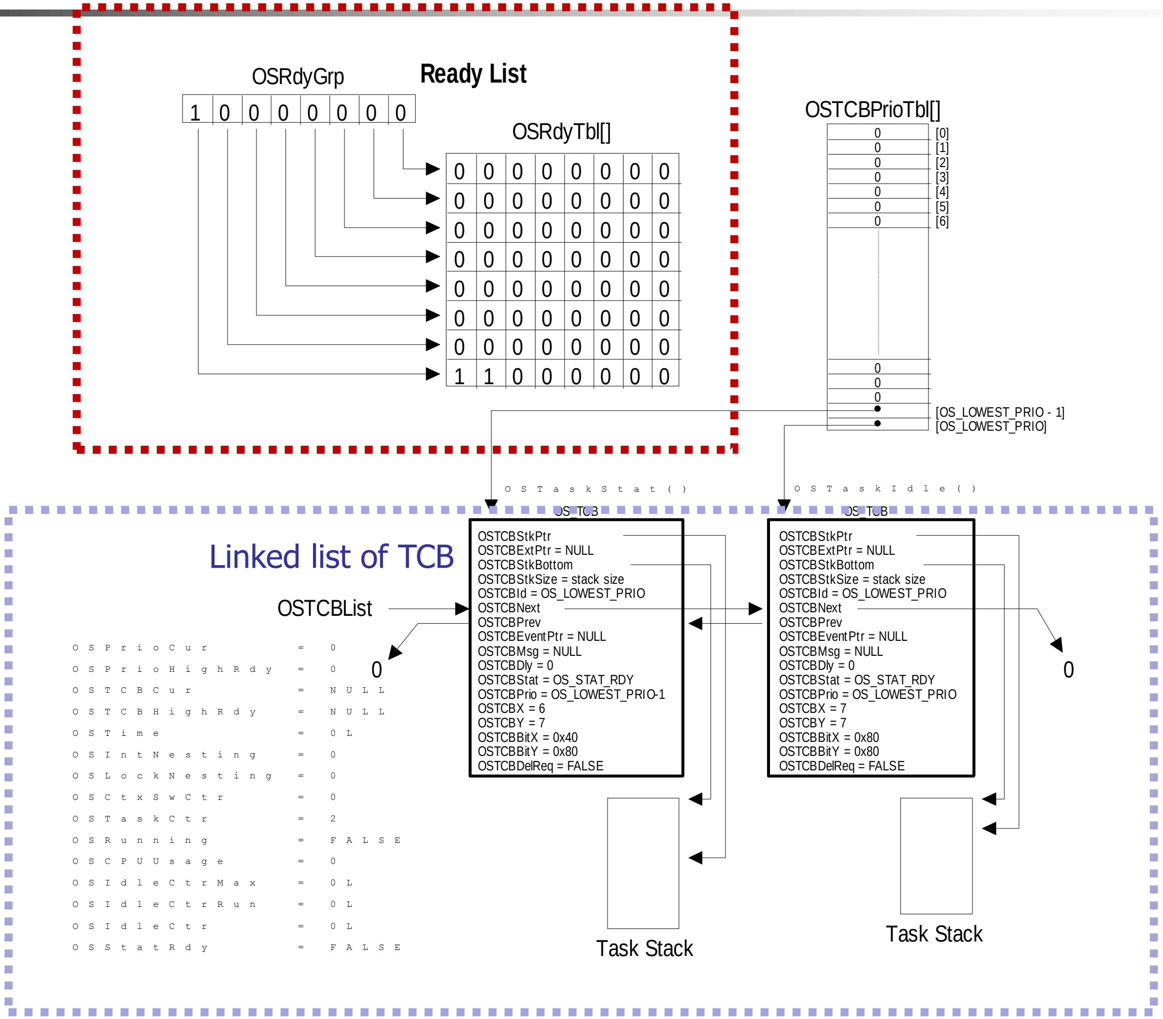
Task scheduling–ready list

- Why Bitmap?
  - To find a prior task in a constant time

- Ready list
  - Data structure
    - OSRdyGrp
      - Bit indicating group
    - OSRdyTbl[]
      - 8 task in the same group
    - OSMapTbl[]
      - Array used for bit mask
    - OSUnMapTbl[]
      - Array used to search the highest priority in ready list

  - Recall that there are total 64 tasks max !!

Scheduler가 관리하는 ready list



Ready List

OSRdyGrp

1	0	0	0	0	0	0	0

OSRdyTbl[]

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0

OSTCBPrioTbl[]

0	[0]
0	[1]
0	[2]
0	[3]
0	[4]
0	[5]
0	[6]
0	
0	
0	
●	[OS_LOWEST_PRIO - 1]
●	[OS_LOWEST_PRIO]

Linked list of TCB

OSTCBList

```
OSPrioCur          = 0
OSPrioHighRdy      = 0
OSTCBCur           = NULL
OSTCBHighRdy       = NULL
OSTime             = 0L
OSIntNesting       = 0
OSLockNesting      = 0
OSCtxSwCtr         = 0
OSTaskCtr          = 2
OSRunning          = FALSE
OSCPUUsage         = 0
OSIdleCtrMax       = 0L
OSIdleCtrRun       = 0L
OSIdleCtr          = 0L
OSStatRdy          = FALSE
```

OSTaskStat()

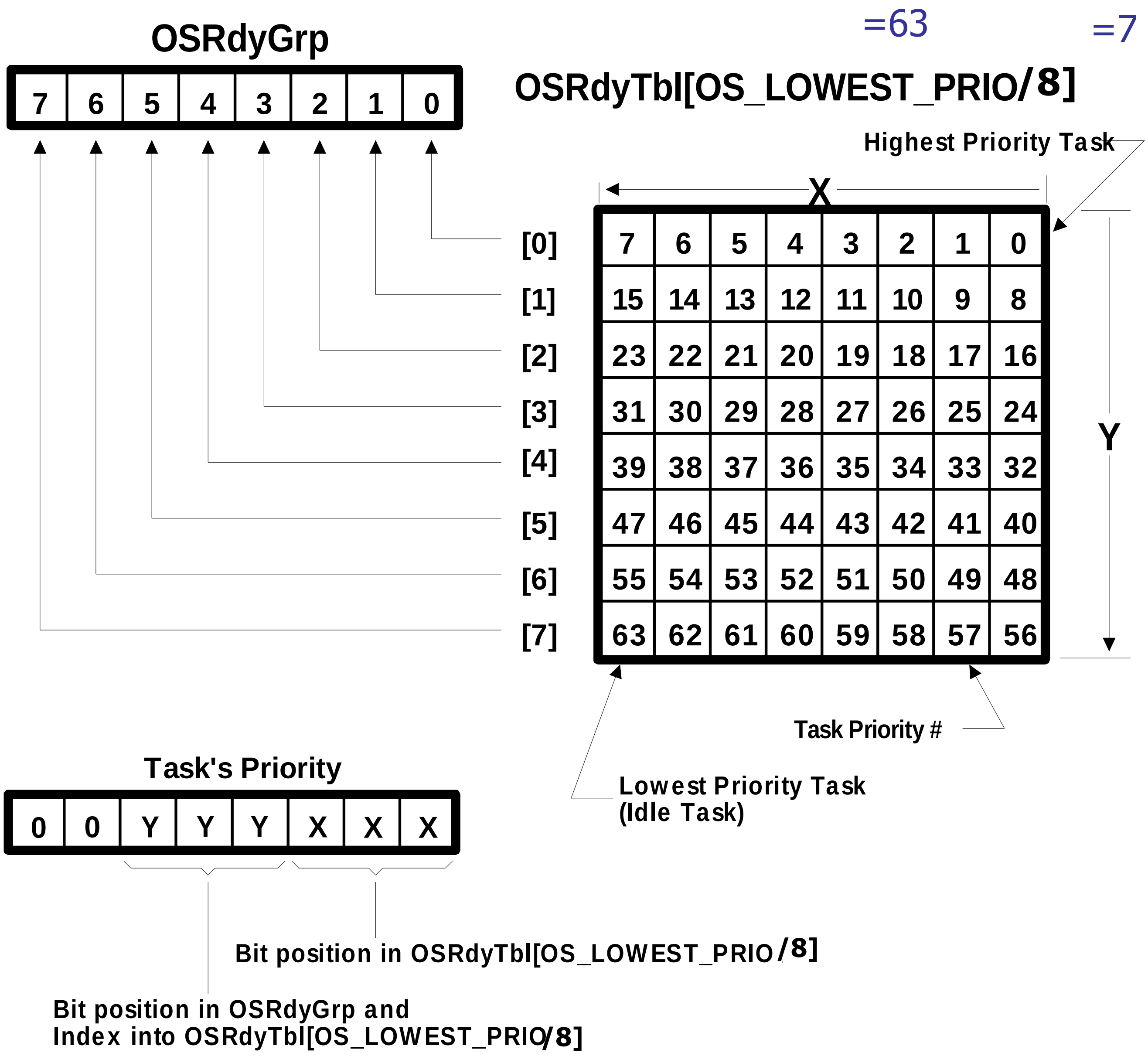
OS_TCB

```
OSTCBStkPtr
OSTCBExtPtr = NULL
OSTCBStkBottom
OSTCBStkSize = stack size
OSTCBId = OS_LOWEST_PRIO
OSTCBNext
OSTCBPrev
OSTCBEventPtr = NULL
OSTCBMsg = NULL
OSTCBDly = 0
OSTCBStat = OS_STAT_RDY
OSTCBPrio = OS_LOWEST_PRIO-1
OSTCBX = 6
OSTCBY = 7
OSTCBBitX = 0x40
OSTCBBitY = 0x80
OSTCBDelReq = FALSE
```

OSTaskIdle()

OS_TCB

```
OSTCBStkPtr
OSTCBExtPtr = NULL
OSTCBStkBottom
OSTCBStkSize = stack size
OSTCBId = OS_LOWEST_PRIO
OSTCBNext
OSTCBPrev
OSTCBEventPtr = NULL
OSTCBMsg = NULL
OSTCBDly = 0
OSTCBStat = OS_STAT_RDY
OSTCBPrio = OS_LOWEST_PRIO
OSTCBX = 7
OSTCBY = 7
OSTCBBitX = 0x80
OSTCBBitY = 0x80
OSTCBDelReq = FALSE
```



=63 =7

OSRdyGrp

7	6	5	4	3	2	1	0

OSRdyTbl[OS_LOWEST_PRIO/8]

Highest Priority Task

X

[0]	7	6	5	4	3	2	1	0
[1]	15	14	13	12	11	10	9	8
[2]	23	22	21	20	19	18	17	16
[3]	31	30	29	28	27	26	25	24
[4]	39	38	37	36	35	34	33	32
[5]	47	46	45	44	43	42	41	40
[6]	55	54	53	52	51	50	49	48
[7]	63	62	61	60	59	58	57	56

Y

Task Priority #

Lowest Priority Task
(Idle Task)

Task's Priority

0	0	Y	Y	Y	X	X	X

Bit position in OSRdyTbl[OS_LOWEST_PRIO/8]

Bit position in OSRdyGrp and
Index into OSRdyTbl[OS_LOWEST_PRIO/8]

Quiz

OSRdyGrp

?	?	?	?	?	?	?	?

OSRdyTbl

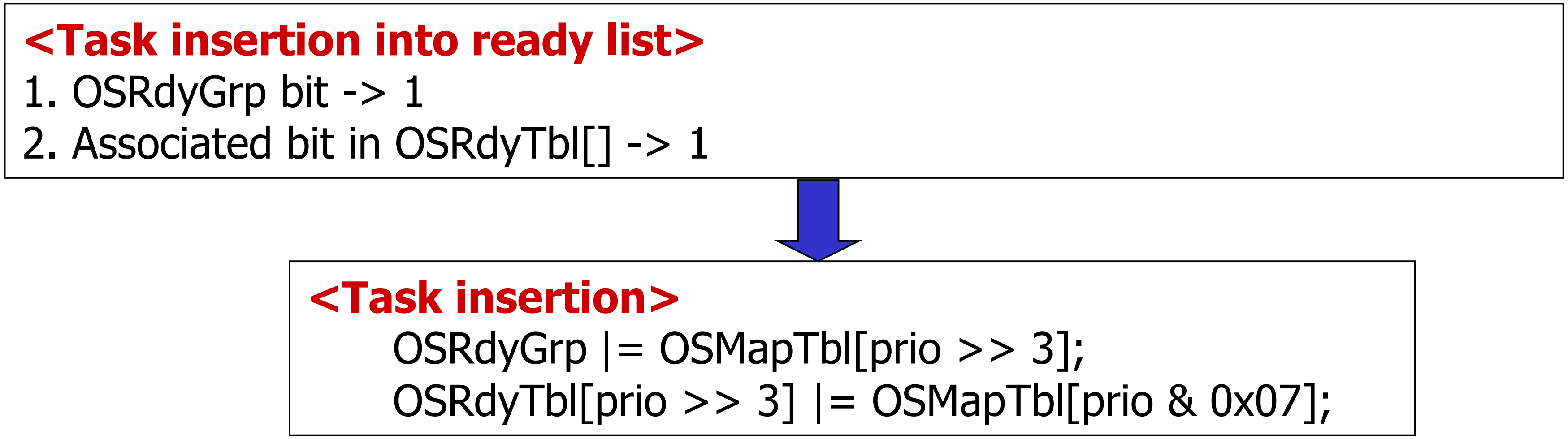
	1	1	1				
		1	1				
1	1						

OSMapTbl

Index	Bit mask (Binary)
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

Bit 0 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[0]** is 1.
Bit 1 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[1]** is 1.
Bit 2 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[2]** is 1.
Bit 3 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[3]** is 1.
Bit 4 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[4]** is 1.
Bit 5 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[5]** is 1.
Bit 6 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[6]** is 1.
Bit 7 in **OSRdyGrp** is 1 when any bit in **OSRdyTbl[7]** is 1.

- Make a task ready to run: (set proper bit in rdygrp&table)



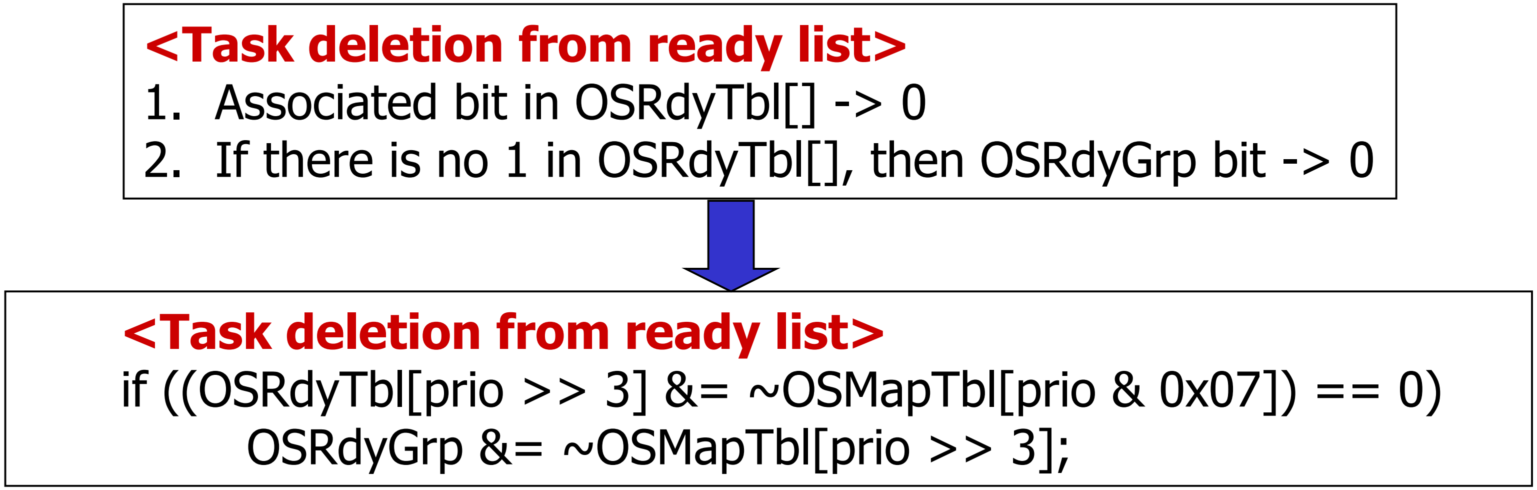
<Task insertion into ready list>

1. OSRdyGrp bit -> 1
2. Associated bit in OSRdyTbl[] -> 1

<Task insertion>

```
    OSRdyGrp |= OSMapTbl[prio >> 3];
    OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];
```

- Remove a task from the ready list:



<Task deletion from ready list>

1. Associated bit in OSRdyTbl[] -> 0
2. If there is no 1 in OSRdyTbl[], then OSRdyGrp bit -> 0

<Task deletion from ready list>

```
if ((OSRdyTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07]) == 0)
        OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

```
char x,y,mask;

  x = prio & 0x07;
  y = prio >> 3;
  mask = ~(OSMapTbl[x]);          // a mask for bit clearing
  if((OSRdyTbl[y] &= mask) == 0)  // clear the task's bit
  {                            // the group bit should be cleared too
     mask = ~(OSMapTbl[y]);       // another bit mask…
     OSRdyGrp &= mask;                       // clear the group bit
  }
```

To Find the highest priority task

- Finding the highest-priority task ready to run:

```
y    = OSUnMapTbl[OSRdyGrp];
x    = OSUnMapTbl[OSRdyTbl[y]];
prio = (y << 3) + x;
```

This matrix is used to locate the first LSB which is ‘1’, by given a value.

For example, if 00110010 is given, then ‘1’ is returned.

```
INT8U const OSUnMapTbl[] = {
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
};
```

Link

EX

LSB 기준 최초 1인 비트 번호를 표시

OSRdyGrp contains 0x68 = **01101000**

```
INT8U  const  OSUnMapTbl[] = {
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F                        */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x20 to 0x2F                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x30 to 0x3F                        */
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x40 to 0x4F                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F                        */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F                        */
    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x90 to 0x9F                        */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xA0 to 0xAF                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xB0 to 0xBF                        */
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xC0 to 0xCF                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xD0 to 0xDF                        */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xE0 to 0xEF                        */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0       /* 0xF0 to 0xFF                        */
};
```

OSRdyTbl[3] contains 0xE4 = **11100100**

```
 3 = OSUnMapTbl[  0x68 ];
 2 = OSUnMapTbl[  0xE4 ];
26 = ( 3  << 3) +  2;
```

<Searching the highest priority >

<example> OSRdyGrp: 01101000, 0x68 OSRdyTbl[3]: 11100100, 0xE4

3 << 3 + 2 = 26

Link

<Searching the highest priority >

```
    y = OSUnMapTbl[OSRdyGrp];
    x = OSUnMapTbl[OSRdyTbl[y]];
    prio = (y << 3) + x;
```

Task scheduling–scheduler

- Task scheduling
  - Task 수준의 Scheduling은 OS_Sched() 함수에 의해 수행
  - ISR 수준의 Scheduling은 OSIntExit() 함수에 의해 수행

- OS_Sched()

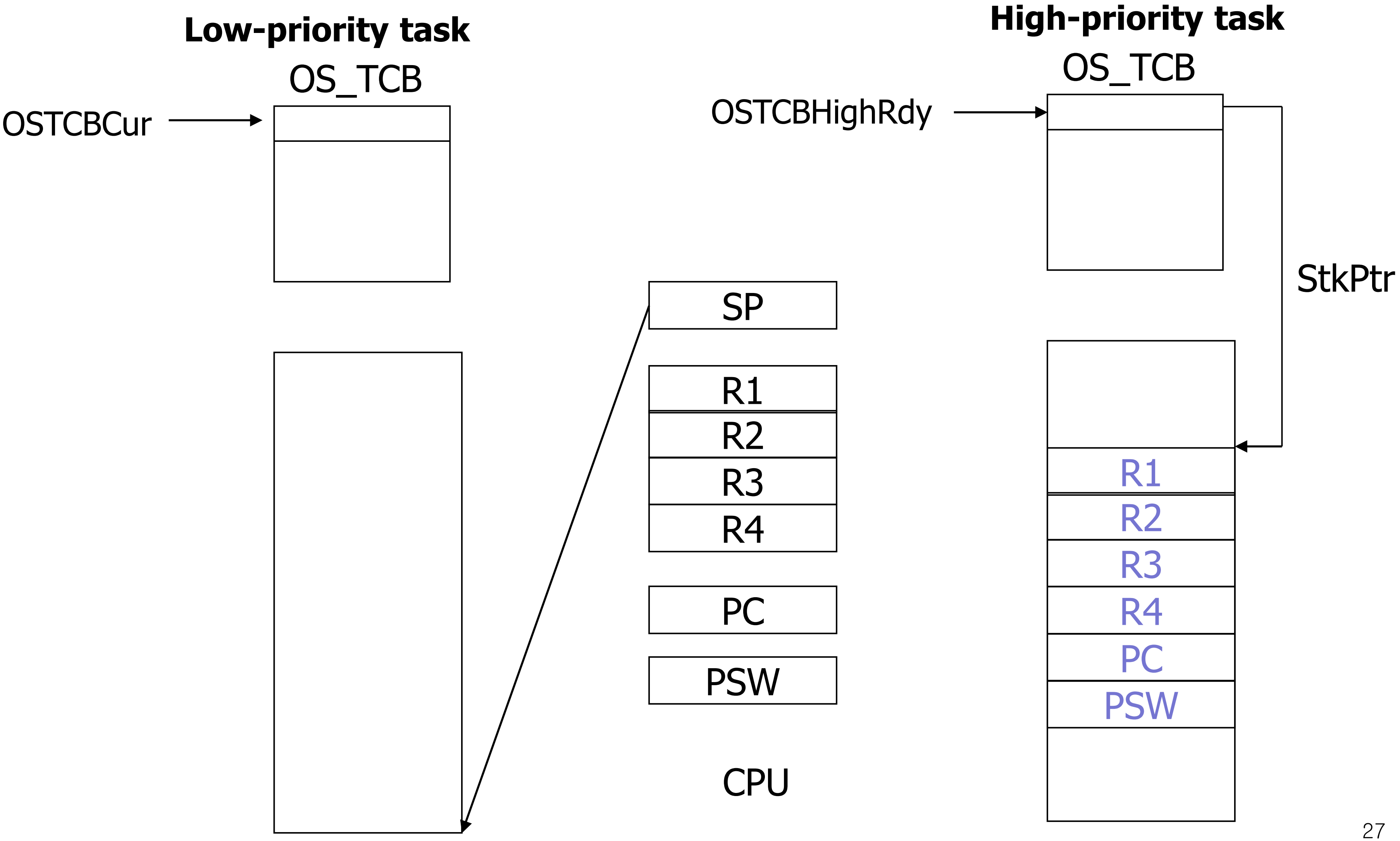
다른 task가 lock을 걸어 강제로 schedule을 막은 상태가 아님

Interrupt가 호출되지 않았음

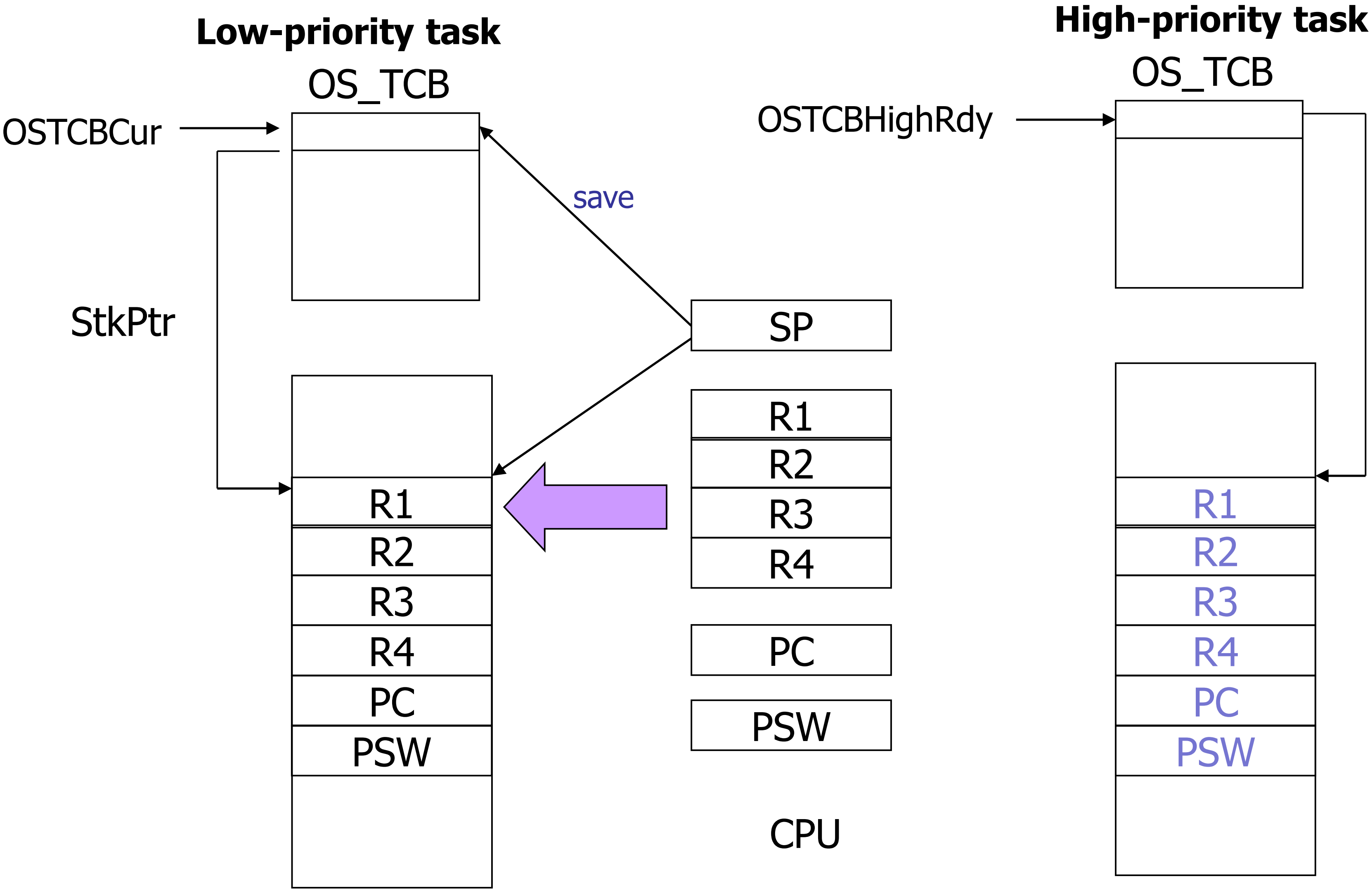
```
void OSSched (void)
{
   INT8U y;
   OS_ENTER_CRITICAL();
   if ((OSLockNesting | OSIntNesting) == 0) {
      y = OSUnMapTbl[OSRdyGrp];
      OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
      if (OSPrioHighRdy != OSPrioCur) {
         OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
         OSCtxSwCtr++;
         OS_TASK_SW();
      }
   }
   OS_EXIT_CRITICAL();
}
```

현재task가 priority가 제일 높으면 switching 없음

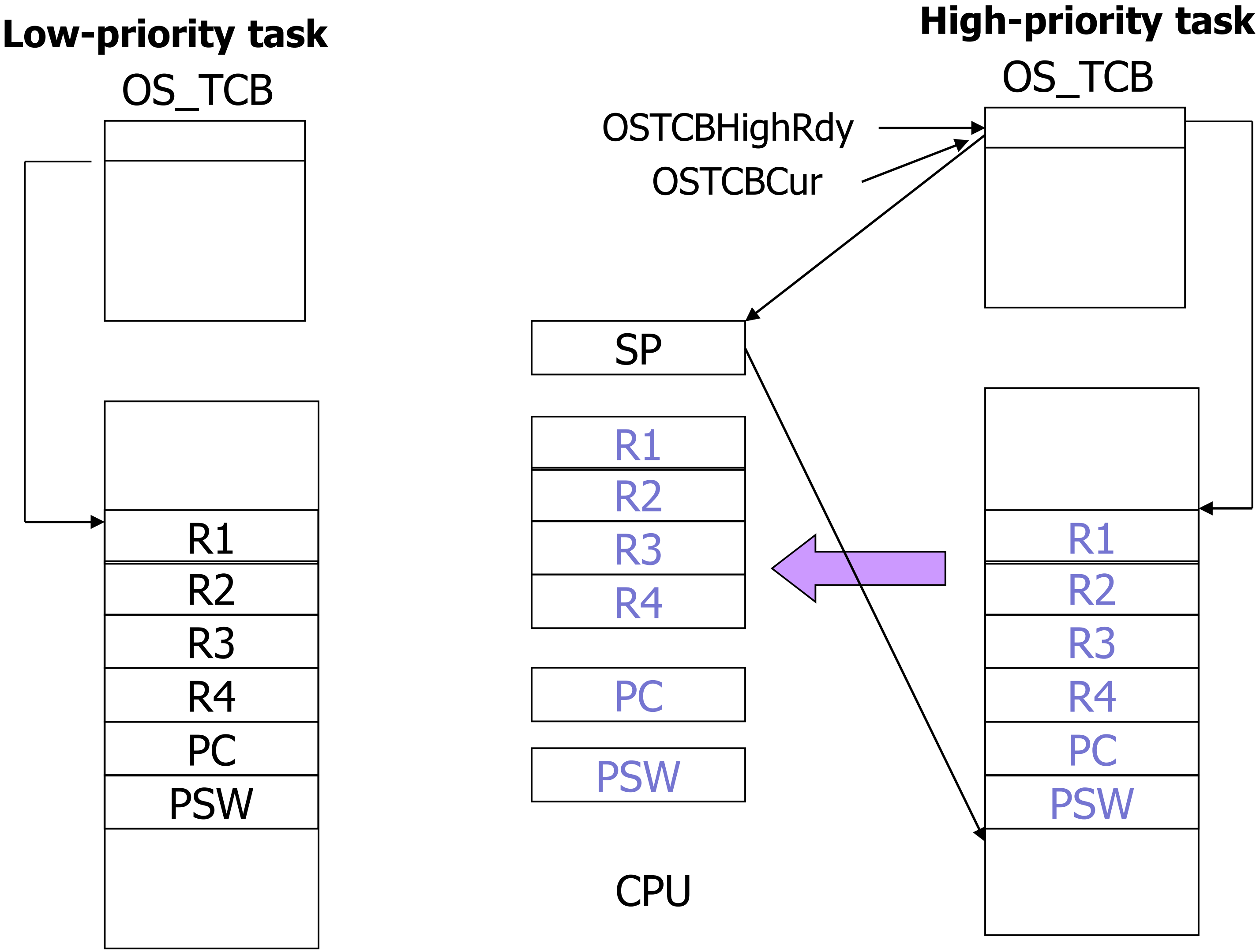
Before OS_TASK_SW()



OS_TASK_SW()


Low-priority task
OS_TCB
OSTCBCur
StkPtr
R1
R2
R3
R4
PC
PSW
save
SP
R1
R2
R3
R4
PC
PSW
CPU
High-priority task
OS_TCB
OSTCBHighRdy
R1
R2
R3
R4
PC
PSW

OS_TASK_SW()


Low-priority task
OS_TCB
R1
R2
R3
R4
PC
PSW
OSTCBHighRdy
OSTCBCur
SP
R1
R2
R3
R4
PC
PSW
CPU
High-priority task
OS_TCB
R1
R2
R3
R4
PC
PSW

Task 수준에서의 context switch

```
void OSCtxSW (void)
{
   Values of R1, R2, R3, R4  and so on are pushed to stack;
   OSTCBCur->OSTCBStkPtr= SP;
   OSTCBCur              = OSTCBHighRdy;
   SP                    = OSCBHighRdy->OSTCBStkPtr;
   Values of R4, R3, R2, R1 and so on are popped;
}
```

Locking and Unlocking the Scheduler

Locking scheduler

```
void OSSchedLock (void)
{
    if (OSRunning == TRUE) {
      OS_ENTER_CRITICAL();
      OSLockNesting++;
      OS_EXIT_CRITICAL();
    }
}
```

Unlocking scheduler

```
void OSSchedUnlock (void)
{
    if (OSRunning == TRUE) {
        OS_ENTER_CRITICAL();
        if (OSLockNesting > 0) {
            OSLockNesting--;
            if ((OSLockNesting | OSIntNesting) == 0) {
                OS_EXIT_CRITICAL();
                OSSched();
            } else {
                OS_EXIT_CRITICAL();
            }
        } else {
            OS_EXIT_CRITICAL();
        }
    }
}
```

Task scheduling -ISR

- ISR
  - μC/OS-II requires an ISR being written in assembly if your compiler does not support in-line assembly!

```
MyISR:
    Save all register values to TCB
    Call OSIntEnter function

    Execute ISR program

    Call OSIntExit()
    Restore all register values to the CPU
```

```
void OSIntEnter (void)
{
   OS_ENTER_CRITICAL();
   OSIntNesting++;
   OS_EXIT_CRITICAL();
}
```

ISR 수준의 Scheduling은 OSIntExit() 함수에 의해 수행

```
void OSIntExit (void)
{
   OS_ENTER_CRITICAL();
   if ((--OSIntNesting | OSLockNesting) == 0) {
      OSIntExitY    = OSUnMapTbl[OSRdyGrp];
      OSPrioHighRdy = (INT8U)((OSIntExitY << 3) +
                  OSUnMapTbl[OSRdyTbl[OSIntExitY]]);
      if (OSPrioHighRdy != OSPrioCur) {
         OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
         OSCtxSwCtr++;
         OSIntCtxSw();
      }
   }
   OS_EXIT_CRITICAL();
}
```

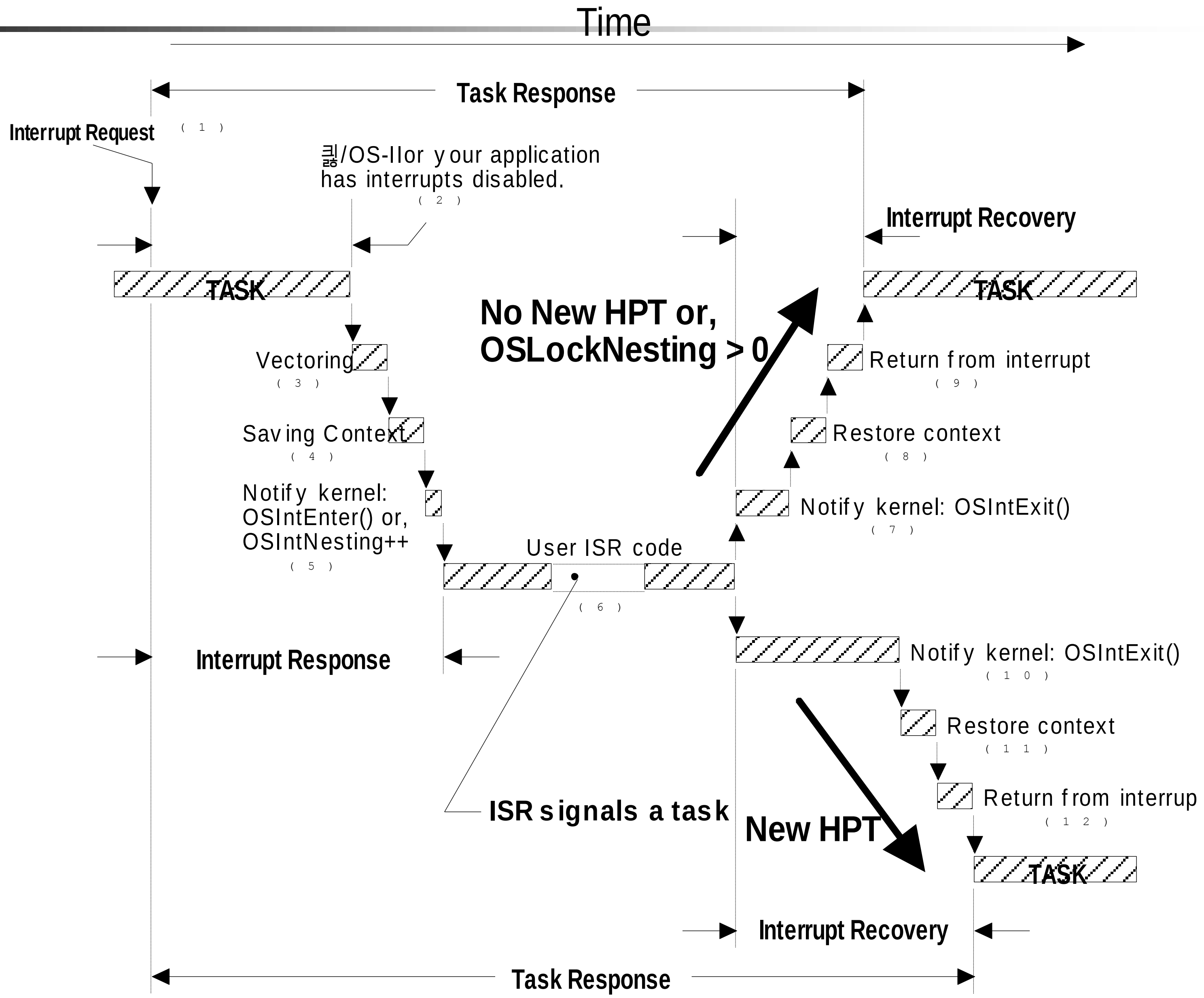
If no scheduler locking and no interrupt nesting

If there is another high-priority task ready

A context switch is executed. Not OSCtxSW()

- OSIntCtxSw()
  - OSIntExit()에서 context switch 수행
  - OSCtxSw()와 유사
    - 차이점: ISR에서 task를 수행시키므로, 이전 task의 문맥 저장 (stroing all CPU regiseters onto stack) 이 없음

```
void  OSIntCtxSw (void)
{
    Call OSTaskSwHook();
    OSTCBCur    = OSTCBHighRdy;
    OSPriorCur = OSPrioHighRdy;
    SP          = OSTCBHighRdy->OSTCBStkPtr;
    POP R4, R3, R2 and R1 from the task’s stack;
    Execute a return from interrupt instruction;
}
```



Time

Task Response

Interrupt Request

(1)

쿪/OS-IIor your application
has interrupts disabled.

(2)

Interrupt Recovery

TASK

No New HPT or,
OSLockNesting > 0

TASK

Vectoring

(3)

Return from interrupt

(9)

Saving Context

(4)

Restore context

(8)

Notify kernel:
OSIntEnter() or,
OSIntNesting++

(5)

Notify kernel: OSIntExit()

(7)

User ISR code

(6)

Interrupt Response

Notify kernel: OSIntExit()

(10)

Restore context

(11)

ISR signals a task

Return from interrup

(12)

New HPT

TASK

Interrupt Recovery

Task Response

Interrupts under μC/OS-II

```
void OSIntExit (void)
{
    OS_ENTER_CRITICAL();
    if ((--OSIntNesting | OSLockNesting) == 0) {
        OSIntExitY    = OSUnMapTbl[OSRdyGrp];
        OSPrioHighRdy = (INT8U)((OSIntExitY << 3) +
                        OSUnMapTbl[OSRdyTbl[OSIntExitY]]);
        if (OSPrioHighRdy != OSPrioCur) {
            OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
            OSCtxSwCtr++;
            OSIntCtxSw();
        }
    }
    OS_EXIT_CRITICAL();
}
```

If no scheduler locking an
d no interrupt nesting

If there is another high-prior
ity task ready

A context switch is executed.

Note that OSIntCtxSw() is called, instead of OS_TASK_SW(), because the ISR already saves the CPU registers onto the stack.

```
void OSIntEnter (void)
{
    OS_ENTER_CRITICAL();
    OSIntNesting++;
    OS_EXIT_CRITICAL();
}
```

Clock ticks

- Time source in OS
- OSTickISR() (in OS_CPU_A.ASM)
    - uC/OS-II에서 타임 아웃 기능과 시간지연 기능 등을 위해 Clock Tick을 사용
    - Multitasking을 시작한 후, 즉 반드시 OSStart()를 호출한 뒤에 task내에서 Clock Tick Interrupt를 활성화 해야 함 (ex. in the TaskStart())
    - Tick ISR이 OSTimeTick()을 호출함

```
void OSTickISR(void) {
    Save processor registers;
    Call OSIntEnter()

    Call OSTimeTick();

    Call OSIntExit();
    Restore processor registers;
    Execute a return from interrupt instruction;
}
```

Tick ISR

```
void OSTimeTick (void)
{
   while (ptcb->OSTCBPrio != OS_IDLE_PRIO) {
     OS_ENTER_CRITICAL();
     if (ptcb->OSTCBDly != 0) {
        if (--ptcb->OSTCBDly == 0) {
           if (!(ptcb->OSTCBStat & OS_STAT_SUSPEND)) {
              OSRdyGrp              |= ptcb->OSTCBBitY;
              OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
           } else {
              ptcb->OSTCBDly = 1;
           }
        }
     }
     ptcb = ptcb->OSTCBNext;
     OS_EXIT_CRITICAL();
   }
}
```

Decrement delay-counter if needed

If the delay-counter reaches zero, make the task ready. Or, the task remains waiting.

```
   OS_ENTER_CRITICAL();
    OSTime++;
    OS_EXIT_CRITICAL();
```