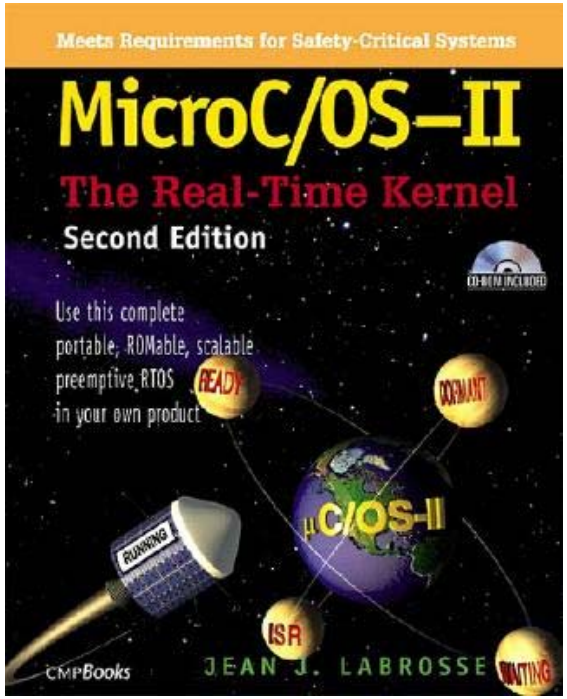


Chapter 3
Kernel Structure
Part I

Olli Wang <olliwang@ollix.com>

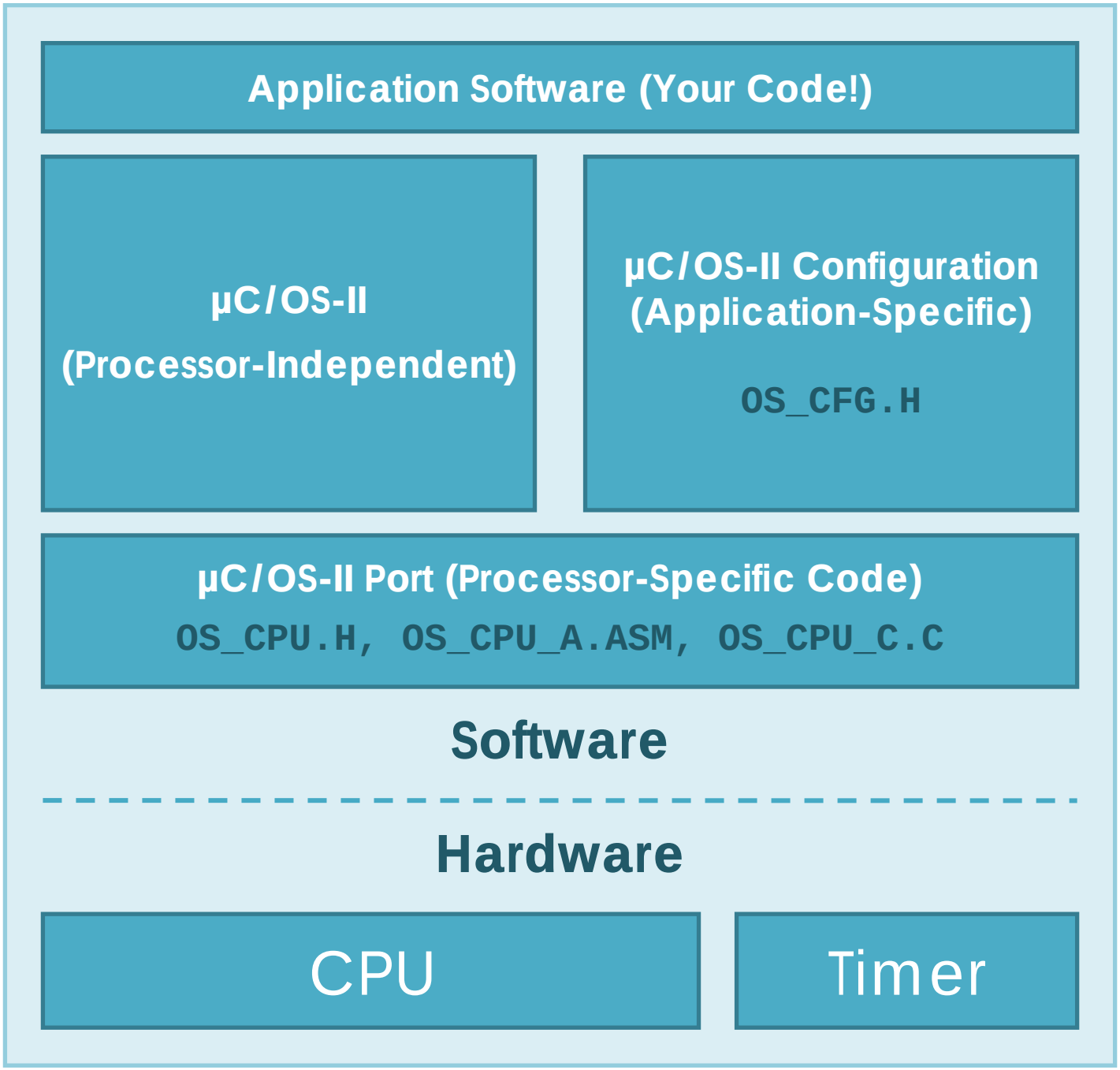
Powered by Ollix

July 15th, 2008

Contents

Sec. 1

µC/OS-II File Structure


Application Software (Your Code!)
µC/OS-II
(Processor-Independent)
µC/OS-II Configuration
(Application-Specific)
OS_CFG.H
µC/OS-II Port (Processor-Specific Code)
OS_CPU.H, OS_CPU_A.ASM, OS_CPU_C.C
Software
Hardware
CPU
Timer

Sec. 2

Critical Sections

Sec. 2

Critical Sections

Interrupts

Sec. 2

Critical Sections

Processors & C

Sec. 2

Critical Sections

Micros

Sec. 2

Critical Sections - Macros

OS_ENTER_CRITICAL()

OS_EXIT_CRITICAL()

Sec. 2

Critical Sections

OS_CPU.H

Sec. 2

Critical Sections

```
{
    …
    OS_ENTER_CRITICAL()
    /* µC/OS-II critical
       code section */
    OS_EXIT_CRITICAL()
    …
}
```

Sec. 2

Critical Sections

The Problem

Sec. 2

Critical Sections

Implements

Sec. 2
Critical Sections - Implements

OS_CRITICAL_METHOD

within OS_CPU.H

Sec. 2

Critical Sections – Implements

```
OS_CRITICAL_METHOD == 1
```

Sec. 2
Critical Sections - Implements `OS_CRITICAL_METHOD == 1`

The Problem

Sec. 2

Critical Sections - Implements `OS_CRITICAL_METHOD == 1`

Interrupt Disabled

µC/OS-II Function

Interrupt Enabled

Sec. 2
Critical Sections - Implements

```
OS_CRITICAL_METHOD == 2
```

Sec. 2

Critical Sections - Implements OS_CRITICAL_METHOD == 2

Stack

Sec. 2

Critical Sections - Implements OS_CRITICAL_METHOD == 2

```
#define OS_ENTER_CRITICAL()  \
        asm("PUSH   PSW")    \
        asm("DI")
#define OS_EXIT_CRITICAL()   \
        asm("POP    PSW")
```

Sec. 2

Critical Sections - Implements

```
OS_CRITICAL_METHOD == 3
```

Sec. 2
Critical Sections - Implements OS_CRITICAL_METHOD == 3

Local Variable within C

Sec. 2

Critical Sections - Implements OS_CRITICAL_METHOD == 3

```
void Some_uCOS_II_Service (args) {
    OS_CPU_SR cup_sr;
    …
    cpu_sr = get_processor_psw();
    disable_interrupts();
    …
    /* Critical section of code */
    …
    set_processor_psw(cpu_sr)
    …
}
```

Sec. 3

Tasks

Sec. 3
Tasks

An Infinite Loop Function

Sec. 3
Tasks

```
void YourTask (void *pdata) {
    for (;;) {
        /* USER CODE */
        OSFlagPend();
        OSFlagPend();
        OSMboxPend();
        OSTaskDel(OS_PRIO_SELF);
        …
    }
}
```

Sec. 3
Tasks

64 Tasks & Priority

0, 1, 2, 3, ... , OLP-3, OLP-2, OLP-1, OLP

OLP = OS_LOWEST_PRIO

Sec. 3
Tasks

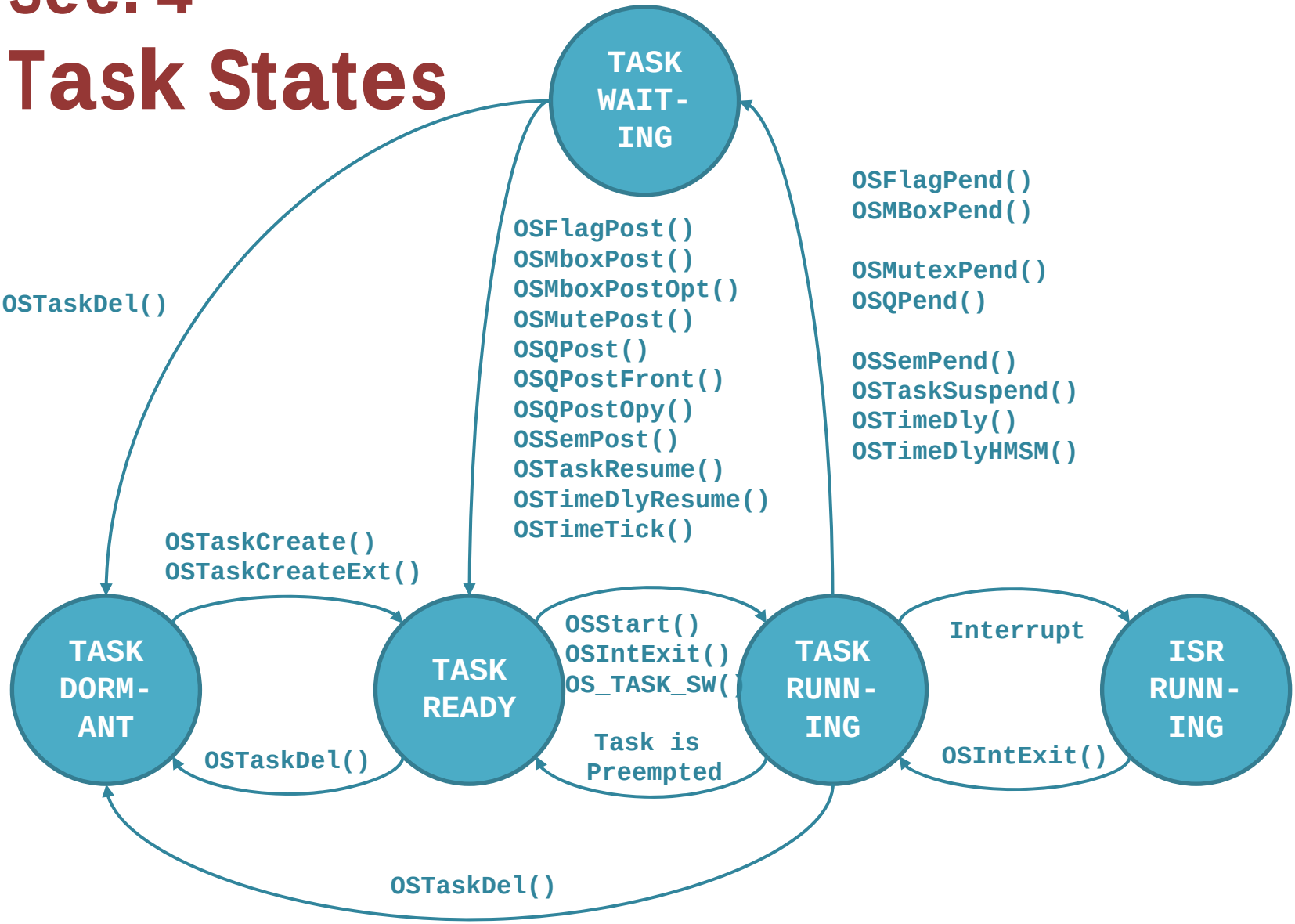
OSTaskCreate()

OSTaskCreateExt()

Sec. 4

Task States

Sec. 4
Task States



Sec. 5

Task Control Blocks

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

The Current Top-of-Stack For the task

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

A user-definable Task control block extension

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

Used by
OSTaskStkChk()

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

Holds options that can be passed to `OSTaskCreateExt()`

(µCOS_II.H)

`OS_TASK_OPT_STK_CHK`
`OS_TASK_OPT_STK_CLR`
`OS_TASK_OPT_SAVE_FP`

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

Hold an Identifier

But.. not used

Sec. 5

Task Control Blocks

```
1  Typedef struct os_tab {
2    OS_STK   *OSTCBStkPtr;
3
4  #if OS_TASK_CREATE_EXT_EN > 0
5    void     *OSTCBExtPtr;
6    OS_STK   *OSTCBStkBottom;
7    INT32U    OSTCBStkSize;
8    INT16U    OSTCBOpt;
9    INT16U    OSTCBId;
10 #endif
11
12  struct os_tcb *OSTCBNext;
13  struct os_tcb *OSTCBPrev;
```

Used to doubly link **OS_TCB**s

Sec. 5
Task Control Blocks

```
14 #if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) ||
       (OS_MBOX_EN > 0) || (OS_SEM_EN > 0) ||
       (OS_MUTEX_EN > 0)
15   OS_EVENT *OSTCBEventPtr;
16 #endif
17
18 #if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) ||
       (OS_MBOX_EN > 0)
19   void *OSTCBMsg;
20 #endif
```

An event control block

Sec. 5
Task Control Blocks

```
14 #if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) ||
       (OS_MBOX_EN > 0) || (OS_SEM_EN > 0) ||
       (OS_MUTEX_EN > 0)
15   OS_EVENT *OSTCBEventPtr;
16 #endif
17
18 #if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) ||
       (OS_MBOX_EN > 0)
19   void *OSTCBMsg;
20 #endif
```

A message sent to a task

Sec. 5
Task Control Blocks

```
21 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0)
       && (OS_MAX_FLAGS > 0)
22 #if OS_TASK_DEL_EN > 0
23   OS_FLAG_NODE *OSTCBFlagNode;
24 #endif
25   OS_FLAGS OSTCBFlagsRdy;
26 #endif
27
28   INT16U OSTCBDly;
29   INT8U  OSTCBStat;
30   INT8U  OSTCBPrio;
```

An event flag node. Only used by `OSTaskDel()`

Sec. 5
Task Control Blocks

```
21 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0)
       && (OS_MAX_FLAGS > 0)
22 #if OS_TASK_DEL_EN > 0
23   OS_FLAG_NODE *OSTCBFlagNode;
24 #endif
25   OS_FLAGS OSTCBFlagsRdy;
26 #endif
27
28   INT16U OSTCBDly;
29   INT8U  OSTCBStat;
30   INT8U  OSTCBPrio;
```

The event flags that made the task ready to run when the task was waiting on an event flag

Sec. 5
Task Control Blocks

```
21 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0)
       && (OS_MAX_FLAGS > 0)
22 #if OS_TASK_DEL_EN > 0
23   OS_FLAG_NODE *OSTCBFlagNode;
24 #endif
25   OS_FLAGS OSTCBFlagsRdy;
26 #endif
27
28   INT16U OSTCBDly;
29   INT8U  OSTCBStat;
30   INT8U  OSTCBPrio;
```

Delayed for a certain number of clock ticks

Sec. 5
Task Control Blocks

```
21 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0)
       && (OS_MAX_FLAGS > 0)
22 #if OS_TASK_DEL_EN > 0
23   OS_FLAG_NODE *OSTCBFlagNode;
24 #endif
25   OS_FLAGS OSTCBFlagsRdy;
26 #endif
27
28   INT16U OSTCBDly;
29   INT8U  OSTCBStat;
30   INT8U  OSTCBPrio;
```

Sec. 5
Task Control Blocks

```
21 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0)
       && (OS_MAX_FLAGS > 0)
22 #if OS_TASK_DEL_EN > 0
23   OS_FLAG_NODE *OSTCBFlagNode;
24 #endif
25   OS_FLAGS OSTCBFlagsRdy;
26 #endif
27
28   INT16U OSTCBDly;
29   INT8U  OSTCBStat;
30   INT8U  OSTCBPrio;
```

The task priority

Sec. 5
Task Control Blocks

```
31    INT8U  OSTCBX;
32    INT8U  OSTCBY;
33    INT8U  OSTCBBitX;
34    INT8U  OSTTCBitY;
35
36 #if OS_TASK_DEL_EN > 0
37    BOOLEAN OSTCBDelReq;
38 #endif
39 } OS_TCB;
```

Used to accelerate the process of making a task ready to run or to make a task wait for an event

Sec. 5
Task Control Blocks

```
31    INT8U  OSTCBX;
32    INT8U  OSTCBY;
33    INT8U  OSTCBBitX;
34    INT8U  OSTTCBitY;
35
36 #if OS_TASK_DEL_EN > 0
37    BOOLEAN OSTCBDelReq;
38 #endif
39 } OS_TCB;
```

```
.OSTCBY = priority >> 3;
.OSTCBBitY = \
OSMapTbl[priority >> 3];

.OSTCBX = priority & 0x07;
.OSTTCBBitX = \
OSMapTbl[priority && 0x07];
```

Sec. 5
Task Control Blocks

```
31    INT8U  OSTCBX;
32    INT8U  OSTCBY;
33    INT8U  OSTCBBitX;
34    INT8U  OSTTCBitY;
35
36 #if OS_TASK_DEL_EN > 0
37    BOOLEAN OSTCBDelReq;
38 #endif
39 } OS_TCB;
```

Indicate to be deleted or not

Sec. 5
Task Control Blocks

OS_MAX_TASKS

Sec. 5

Task Control Blocks

OSTCBTbl[]

Sec. 5
Task Control Blocks

OS_N_SYS_TASKS

(µCOS_II.H)

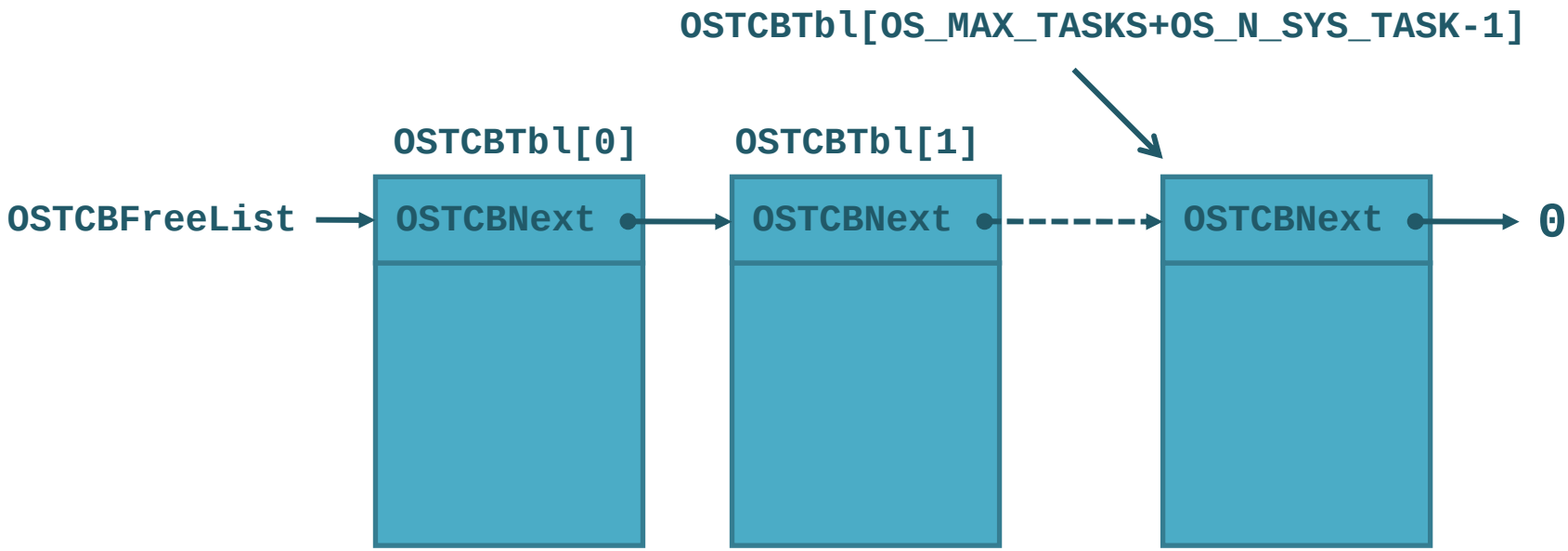
Sec. 5
Task Control Blocks

OS_TASK_STAT_EN

(OS_CFG.H)

Sec. 5

Task Control Blocks



Sec. 5
Task Control Blocks

OS_TCBInit()

Sec. 5
Task Control Blocks

`OSTaskCreate()` `OSTaskCreateExt()`

`OS_TCBInit()`

Sec. 5

Task Control Blocks

`OS_TCBInit()`


```
prio
ptos
pbos
id
stk_size
pext
opt
```

Sec. 5

Task Control Blocks

```
1 INT8U OS_TCBInit (INT8U prio, OS_STK *ptos,
                    OS_STK *pbos, INT16U id,
                    INT32U stk_size, void *pext,
                    INT16U opt) {
2 #if OS_CRITICAL_METHOD == 3
3   OS_CPU_SR cup_sr;
4 #endif
5   OS_TCB *ptcb;
6
7   OS_ENTER_CRITICAL();
8   ptcb = OSTCBFreeList;
```

Sec. 5
Task Control Blocks

```
1 INT8U OS_TCBInit (INT8U prio, OS_STK *ptos,
                    OS_STK *pbos, INT16U id,
                    INT32U stk_size, void *pext,
                    INT16U opt) {
2 #if OS_CRITICAL_METHOD == 3
3   OS_CPU_SR cup_sr;
4 #endif
5   OS_TCB *ptcb;
6
7   OS_ENTER_CRITICAL();
8   ptcb = OSTCBFreeList;
```

Sec. 5
Task Control Blocks

```
9     if (ptcb != (OS_TCB *)0) {
10      OSTCBFreeList  = ptcb->OSTCBNext;
11      OS_EXIT_CRITICAL();
12      ptcb->OSTCBStkPtr = ptos;
13      ptcb->OSTCBPrio   = (INT8U)prio;
14      ptcb->OSTCBStat   = OS_STAT_RDY;
15      ptcb->OSTCBDly    = 0;
```

Sec. 5

Task Control Blocks

```
16 #if OS_TASK_CREATE_EXT_EN > 0
17     ptcb->OSTCBExtPtr    = pext;
18     ptcb->OSTCBStkSize   = stk_size;
19     ptcb->OSTCBStkBottom = pbos;
20     ptcb->OSTCBOpt       = opt;
21     ptcb->OSTCBId        = id;
22 #else
23     pext     = pext;
24     stk_size = stk_size;
25     pbos     = pbos
26     opt      = opt;
27     id       = id;
28 #endif
```

Sec. 5
Task Control Blocks

```
29 #if OS_TASK_DEL_EN > 0
30     ptcb->OSTCBDelReq = OS_NO_ERR;
31 #endif
32
33     ptcb->OSTCBY    = prio >> 3;
34     ptcb->OSTCBBitY = OSMapTbl[ptcb->OSTCBY];
35     ptcb->OSTCBX    = prio & 0x07;
36     ptcb->OSTCBBitX = OSMapTbl[ptcb->OSTCBX];
37
38 #if OS_EVENT_EN >0
39     ptcb->OSTCBEventPtr = (OS_EVENT *)0;
40 #endif
```

Sec. 5
Task Control Blocks

```
29 #if OS_TASK_DEL_EN > 0
30     ptcb->OSTCBDelReq = OSO_NO_ERR;
31 #endif
32
33     ptcb->OSTCBY    = prio >> 3;
34     ptcb->OSTCBBitY = OSMapTbl[ptcb->OSTCBY];
35     ptcb->OSTCBX    = prio & 0x07;
36     ptcb->OSTCBBitX = OSMapTbl[ptcb->OSTCBX];
37
38 #if OS_EVENT_EN >0
39     ptcb->OSTCBEventPtr = (OS_EVENT *)0;
40 #endif
```

Sec. 5
Task Control Blocks

```
29 #if OS_TASK_DEL_EN > 0
30     ptcb->OSTCBDelReq = OSO_NO_ERR;
31 #endif
32
33     ptcb->OSTCBY    = prio >> 3;
34     ptcb->OSTCBBitY = OSMapTbl[ptcb->OSTCBY];
35     ptcb->OSTCBX    = prio & 0x07;
36     ptcb->OSTCBBitX = OSMapTbl[ptcb->OSTCBX];
37
38 #if OS_EVENT_EN >0
39     ptcb->OSTCBEventPtr = (OS_EVENT *)0;
40 #endif
```

Sec. 5
Task Control Blocks

```
41 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) &&
42     (OS_MAX_FLAGS > 0) && (OS_TASK_DEL_EN > 0)
43     ptcb->OSTCBFlagNode = (OS_FLAG_NODE *)0;
44 #endif
45
46 #if (OS_MBOX_EN || (OS_Q_EN && (OS_MAX_QS >= 2))
47     ptcb->OSTCBMsg = (void *)0;
48 #endif
49
50 #if OS_VERSION >= 204
51     OSTCBInitHook(ptcb);
52 #endif
```

Sec. 5
Task Control Blocks

```
41 #if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) &&
42     (OS_MAX_FLAGS > 0) && (OS_TASK_DEL_EN > 0)
43     ptcb->OSTCBFlagNode = (OS_FLAG_NODE *)0;
44 #endif
45
46 #if (OS_MBOX_EN || (OS_Q_EN && (OS_MAX_QS >= 2))
47     ptcb->OSTCBMsg = (void *)0;
48 #endif
49
50 #if OS_VERSION >= 204
51     OSTCBInitHook(ptcb);
52 #endif
```

Sec. 5

Task Control Blocks

```
53      OSTaskCreateHook(ptcb);
54
55      OS_ENTER_CRITICAL();
56      OSTCBPrioTbl[prio] = ptcb;
57      ptcb->OSTCBNext    = OSTCBList;
58      ptcb->OSTCBPrev    = (OS_TCB *)0;
59      if (OSTCBList != (OS_TCB*)0) {
60        OSTCBList->OSTCBPrev = ptcb;
61      }
62      OSTCBList                = ptcb;
63      OSRdyGrp                |= ptcb->OSTCBBitY;
64      OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
65      OS_EXIT_CRITICAL()
66      return (OS_NO_ERR);
67    }
```

Sec. 5

Task Control Blocks

```
53      OSTaskCreateHook(ptcb);
54
55      OS_ENTER_CRITICAL();
56      OSTCBPrioTbl[prio] = ptcb;
57      ptcb->OSTCBNext    = OSTCBList;
58      ptcb->OSTCBPrev    = (OS_TCB *)0;
59      if (OSTCBList != (OS_TCB*)0) {
60        OSTCBList->OSTCBPrev = ptcb;
61      }
62      OSTCBList                = ptcb;
63      OSRdyGrp                |= ptcb->OSTCBBitY;
64      OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
65      OS_EXIT_CRITICAL()
66      return (OS_NO_ERR);
67    }
```

Sec. 5

Task Control Blocks

```
53      OSTaskCreateHook(ptcb);
54
55      OS_ENTER_CRITICAL();
56      OSTCBPrioTbl[prio] = ptcb;
57      ptcb->OSTCBNext    = OSTCBList;
58      ptcb->OSTCBPrev    = (OS_TCB *)0;
59      if (OSTCBList != (OS_TCB*)0) {
60        OSTCBList->OSTCBPrev = ptcb;
61      }
62      OSTCBList                = ptcb;
63      OSRdyGrp                |= ptcb->OSTCBBitY;
64      OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
65      OS_EXIT_CRITICAL()
66      return (OS_NO_ERR);
67    }
```

Sec. 5
Task Control Blocks

```
68    }
69    OS_EXIT_CRITICAL();
70    return (OS_NO_MORE_TCB);
71 }
```

Sec. 6

Ready List

Sec. 6
Ready List

OS_LOWEST_PRIO

Sec. 6
Ready List

OS_MAX_TASKS

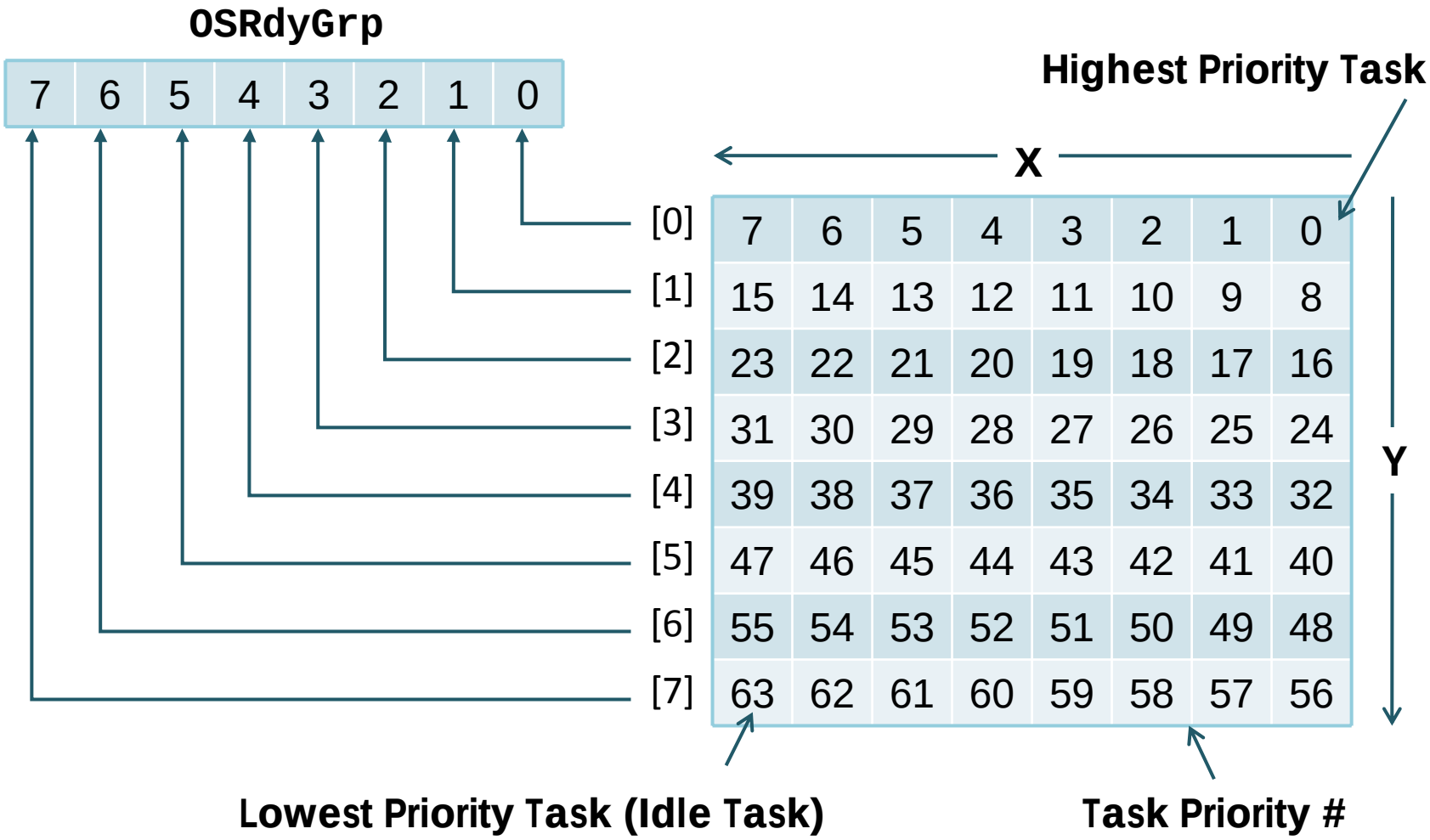
Sec. 6
Ready List

OSRdyGrp

&

OSRdyTbl[]

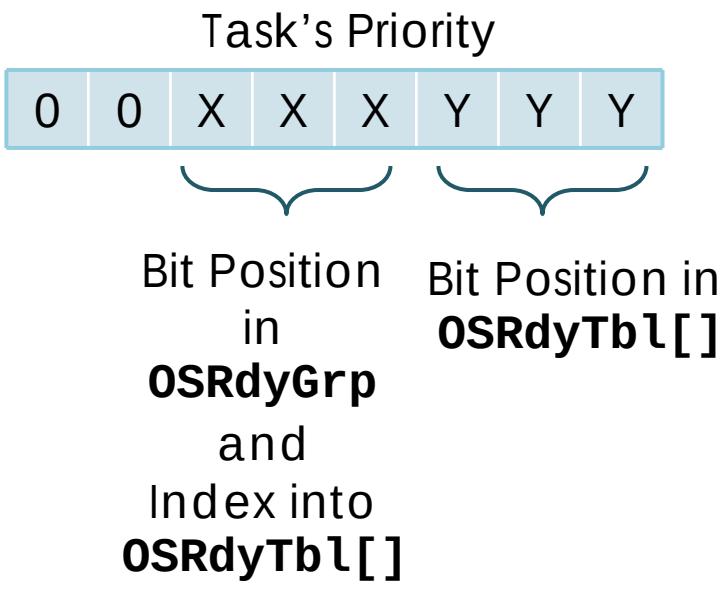
Sec. 6
Ready List



Sec. 6
Ready List

Making a task ready to run:

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



OSMapTbl[]

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

Sec. 6
Ready List

Removing a task from the ready list

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

Sec. 6
Ready List

Finding the highest priority task ready to run

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

Sec. 6
Ready List

```
OSRdyGrp = 01101000 (0x68)
OSRdyTbl[3] = 11100100 (0xE4)
```

```
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 */
}
```

Sec. 6
Ready List

```
OSrdyGrp = 01101000 (0x68)
OSRdyTbl[3] = 11100100 (0xE4)
```

```
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,
  6,0,1,0,2,0,1,0,3,0,1,0,
  4,0,1,0,2,0,1,0,3,0,1,0,
  5,0,1,0,2,0,1,0,3,0,1,0,
  4,0,1,0,2,0,1,0,3,0,1,0,
}
```

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

Sec. 7

Task Scheduling

Sec. 7
Task Scheduling

OS_Sched()

Sec. 7
Task Scheduling

OSIntExt()

Sec. 7

Task Scheduling

```
1 void OS_Sched (void) {
2 #if OS_CRITICAL_METHOD == 3
3   OS_CPU_SR cpu_sr;
4 #endif
5
6   INT8U y;
7
8   OS_ENTER_CRITICAL();
9   if ((OSIntNesting == 0) && (OSLockNesting == 0))
10  {
11    y = OSUmMapTbl[OSRdyGrp];
12    OSPrioHighRdy = (INT8U)((y << 3) + \
13                      OSUnMapTbl[OSRdyTbl[y]]);
```

Sec. 7

Task Scheduling

```
1 void OS_Sched (void) {
2 #if OS_CRITICAL_METHOD == 3
3   OS_CPU_SR cpu_sr;
4 #endif
5
6   INT8U y;
7
8   OS_ENTER_CRITICAL();
9   if ((OSIntNesting == 0) && (OSLockNesting == 0))
10  {
11    y = OSUmMapTbl[OSRdyGrp];
12    OSPrioHighRdy = (INT8U)((y << 3) + \
13                      OSUnMapTbl[OSRdyTbl[y]]);
```

Sec. 7
Task Scheduling

```
14      if (OSPrioHighRdy != OSPrioCur) {
15          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
16          OSCtxSwCtr++;
17          OS_TASK_SW();
18      }
19    }
20
21    OS_EXIT_CRITICAL();
22 }
```

Sec. 7
Task Scheduling

```
14      if (OSPrioHighRdy != OSPrioCur) {
15          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
16          OSCtxSwCtr++;
17          OS_TASK_SW();
18      }
19    }
20
21    OS_EXIT_CRITICAL();
22 }
```

Sec. 8

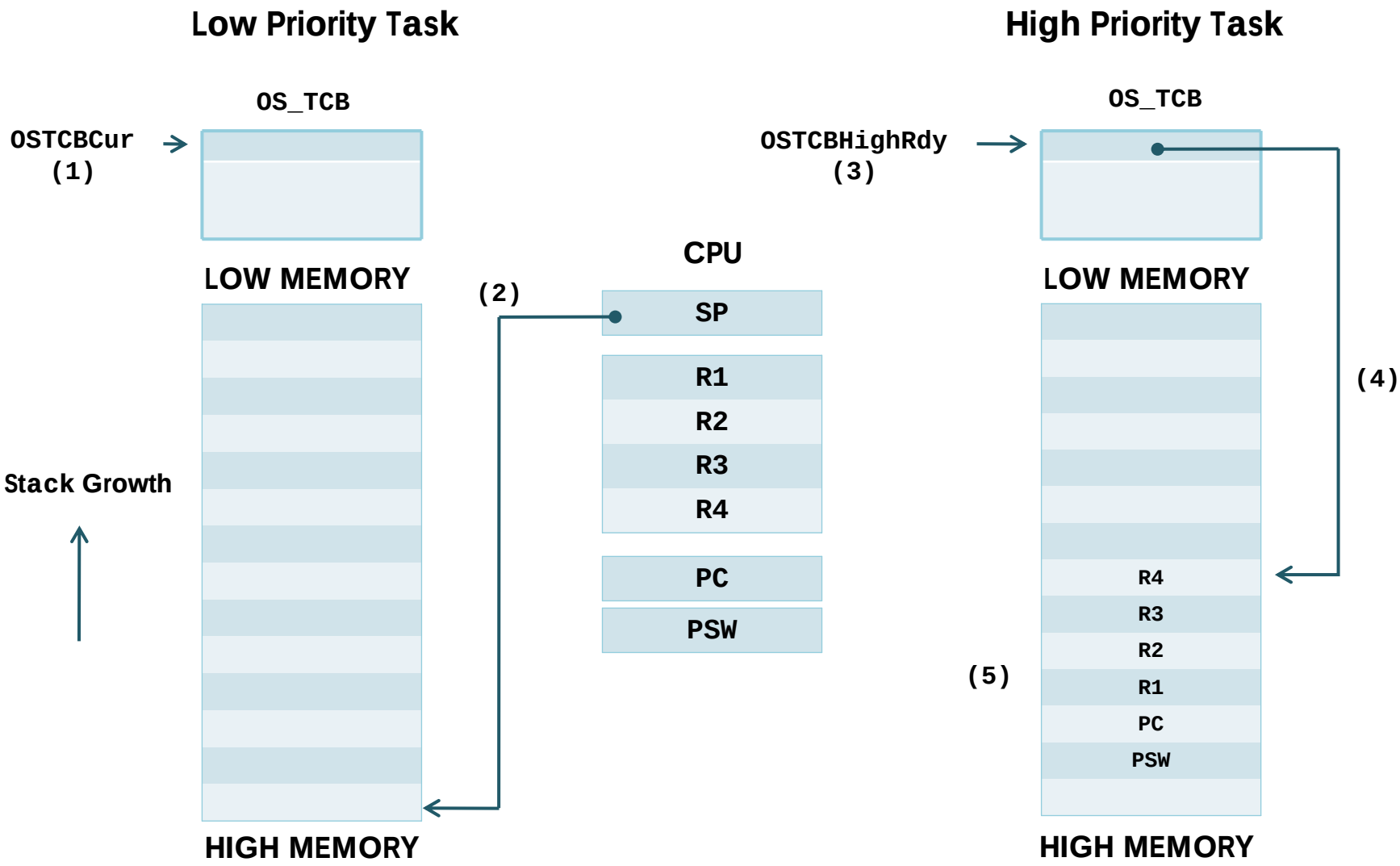
Task Level Context Switch

Sec. 8
Task Level Context Switch

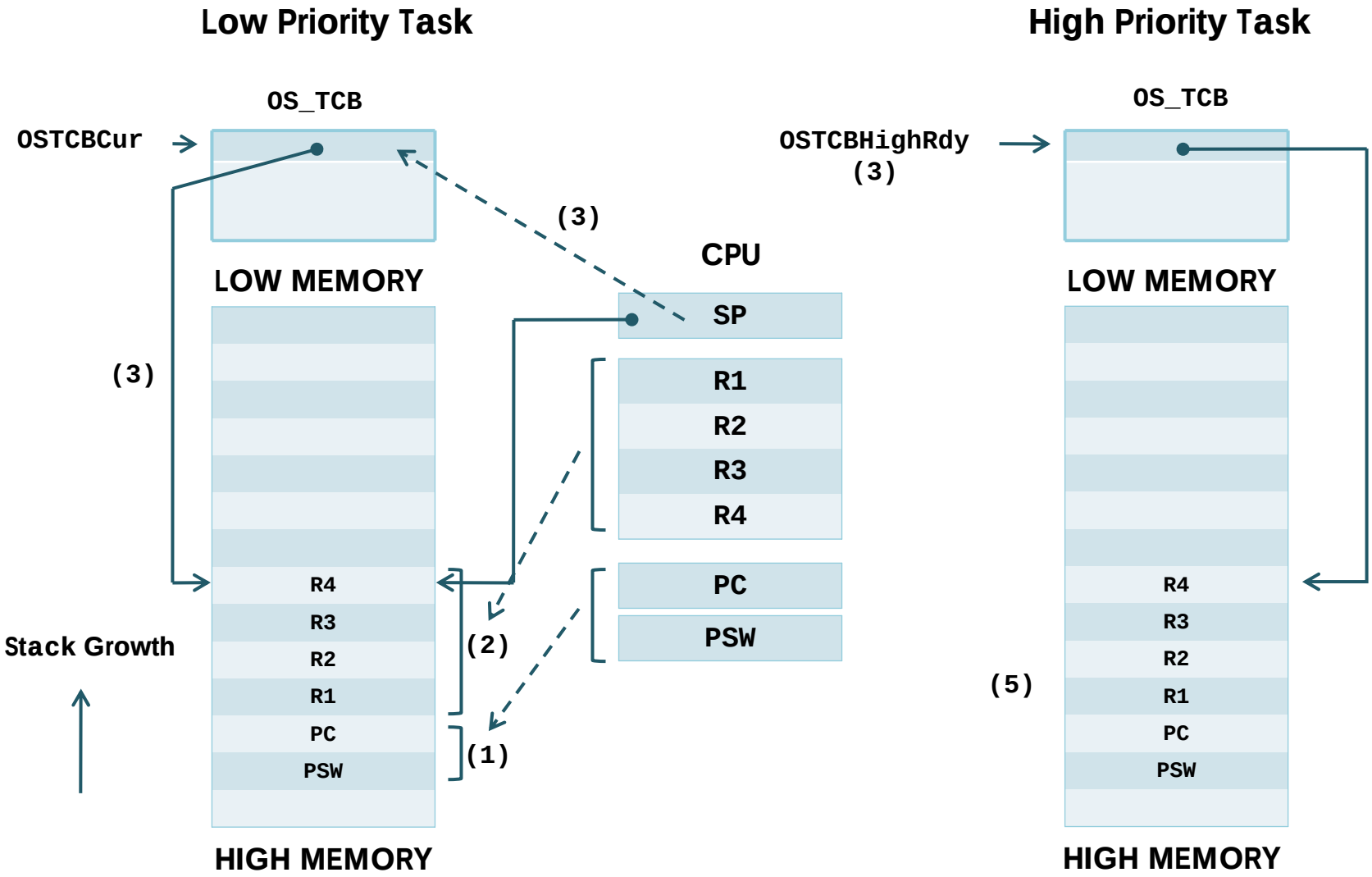
OS_TASK_SW()

Sec. 8

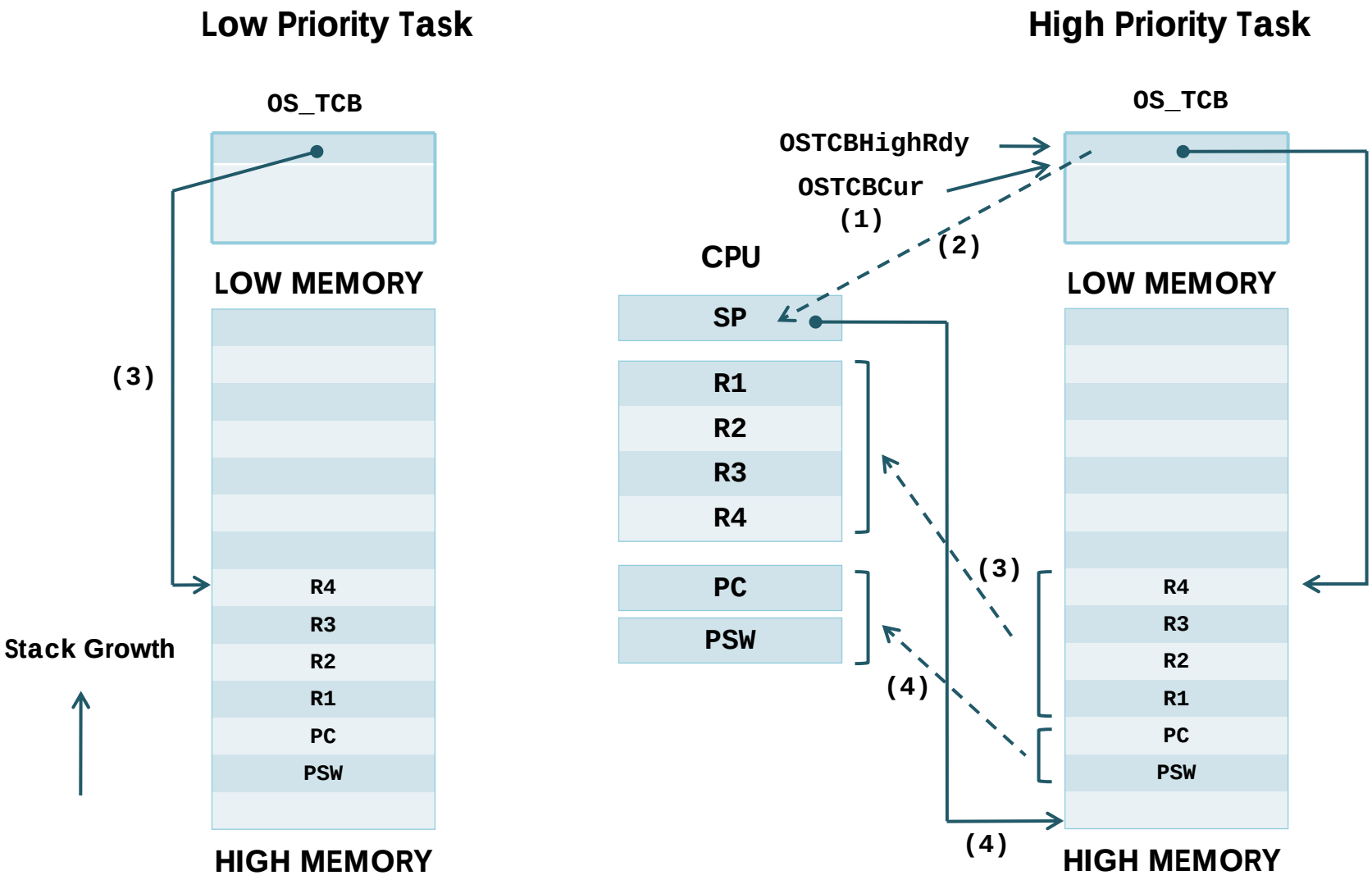
µC/OS-II structures when OS_TASK_SW( ) is called



Sec. 8 Saving the current task's context



Sec. 8 Resuming the current task



Sec. 8

Task Level Context Switch

```
Void OSCtxSw (void)
{
  PUSH R1, R2, R3 and R4 onto the current stack;
  OSTBCur->OSTCBStkPtr = SP;
  OSTCBCur             = OSTCBHighRdy;
  SP                   = OSTCBHighRdy->OSTCBStkPtr;
  POP R4, R3, R2 and R1 from the new stack;
  Execute a return from interrupt instruction;
}
```

終わり
ありがとう