

2 Kernel Structure

Contents – 1/2

- uC/OS-II
- uC/OS-II File Structure
- Critical Sections
- Task
- Task States
- Task Control Block
- Ready List
- Task Scheduling
- Task Level Context Switching

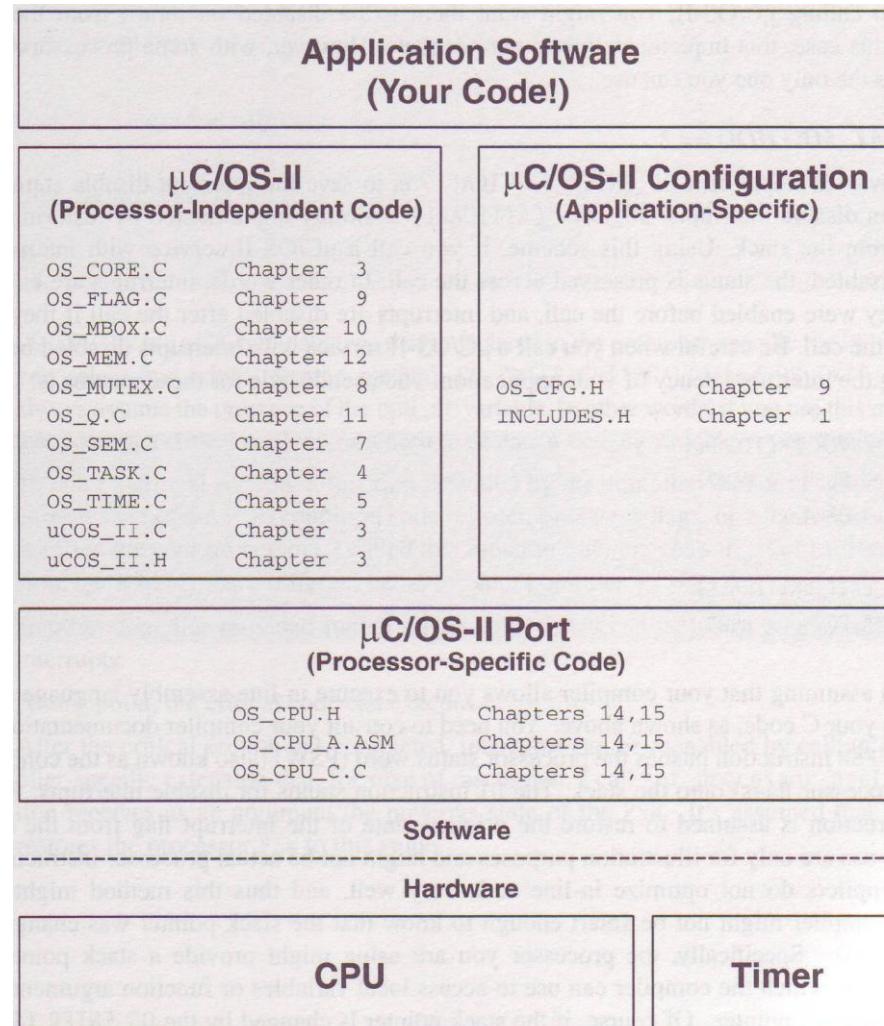
Contents – 2/2

- Locking and Unlocking the Scheduler
- Idle Task
- Statistics Task
- Interrupt under uC/OS-II
- Clock Tick
- uC/OS-II Initialization
- Starting uC/OS-II
- Obtaining the Current uC/OS-II Version

uC/OS-II

- What is μC/OS-II?
 - μC/OS-II is a portable, ROMable, scalable, preemptive, real-time deterministic multitasking kernel for microprocessors, microcontrollers and DSPs.
- Applications
 - Avionics - used in the Mars Curiosity Rover!
 - Medical Equipment/Devices
 - Data Communications Equipment
 - White Goods (Appliances)
 - Mobile Phones, PDAs, Mobile Internet Devices
 - Industrial Controls
 - Consumer Electronics
 - Automotive
 - A wide range of other safety critical embedded applications

uC/OS-II File Structure



Critical Sections

- uC/OS-II provides 2 macros in `os_cpu.h`
 - `OS_ENTER_CRITICAL()`
 - `OS_EXIT_CRITICAL()`
- Notes
 - Be sure not to call some kernel functions (eg: `OSTimeDly`) when interrupts are disabled
 - If you call a kernel function with interrupts disabled, interrupts are enabled on return

Critical Sections - Method 1

- OS_CRITICAL_METHOD == 1 (os_cpu.h)
- Code example

```
#define OS_ENTER_CRITICAL() asm("DI");  
#define OS_EXIT_CRITICAL() asm("EI");
```

- With this method, interrupt status is not preserved between before and after the macros

Critical Sections - Method 2

- OS_CRITICAL_METHOD == 2 (os_cpu.h)
- Code example

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

- With this method, interrupt status is preserved between before and after the macros

Critical Sections - Method 3

- OS_CRITICAL_METHOD == 3 (os_cpu.h)
- Use of a local variable OS_CPU_SR cpu_sr;
- Code example

```
#define OS_ENTER_CRITICAL() \
    cpu_sr = get_processor_psw(); \
    disable_interrupt(); \
#define OS_EXIT_CRITICAL() \
    set_processor_psw(cpu_sr);
```

- With this method, interrupt status is preserved between before and after the macros

Critical Sections - Method 3 on ARM

```
#define OS_CRITICAL_METHOD 3

#define OS_ENTER_CRITICAL() \
    cpu_sr = OS_CPU_SR_Save();

#define OS_EXIT_CRITICAL() \
    OS_CPU_SR_Restore(cpu_sr);
```

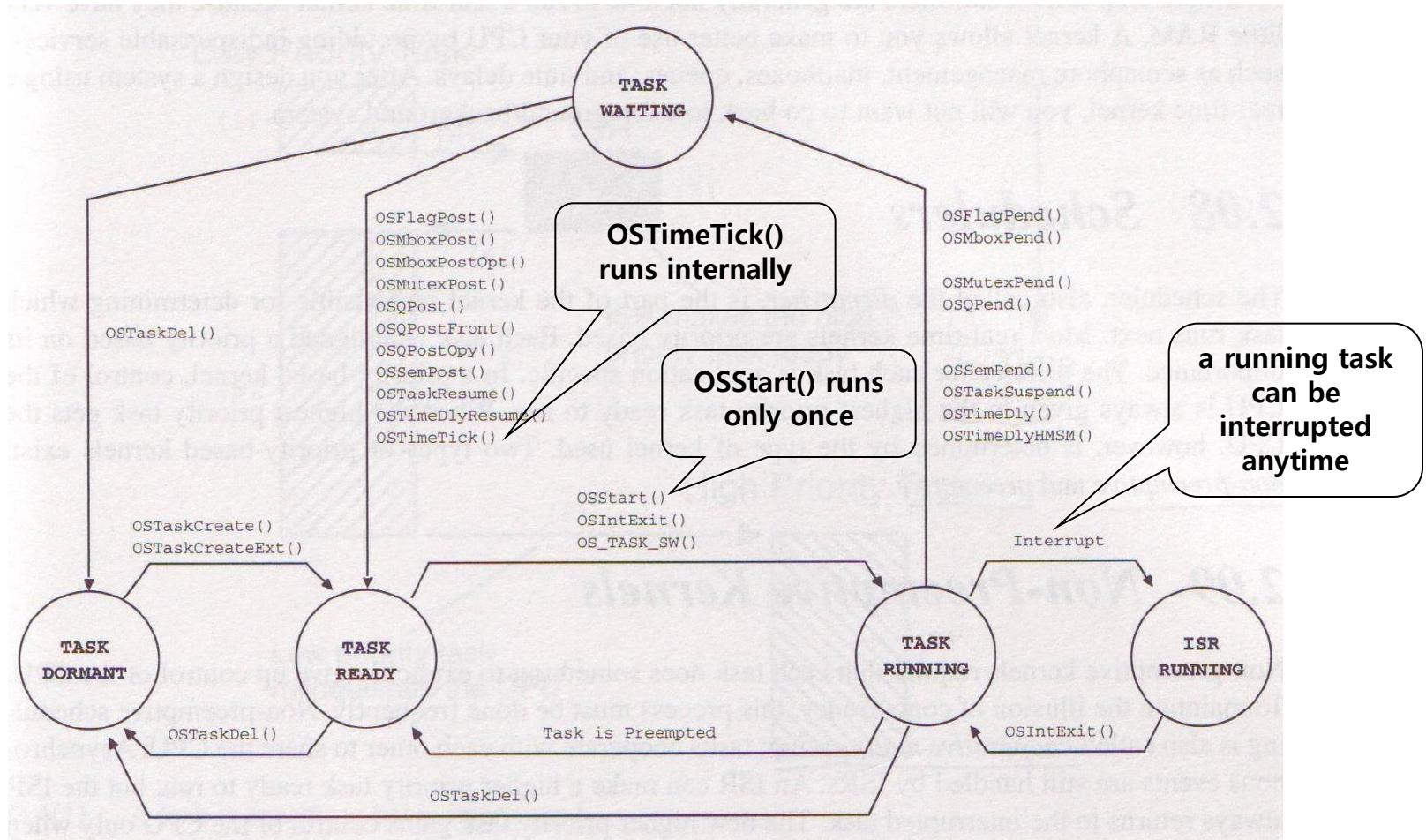
Tasks

Listing 3.2 A task is an infinite loop.

```
void YourTask (void *pdata) (1)
{
    for (;;) {
        /* USER CODE */

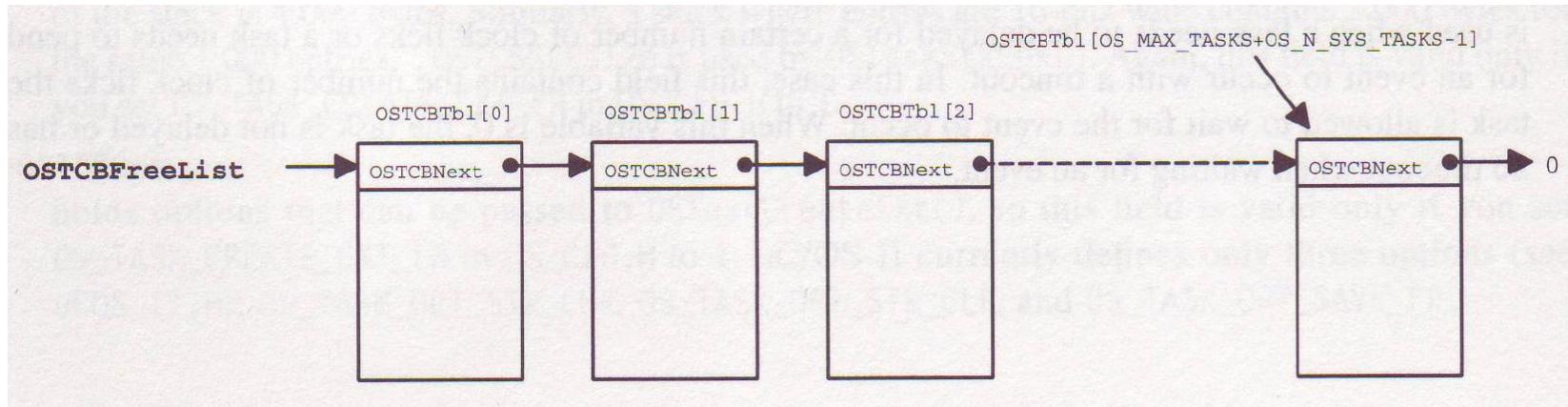
        Call one of uC/OS-II's services:
        OSFlagPend();
        OSMboxPend();
        OSMutexPend();
        OSQPend();
        OSSemPend();
        OSTaskDel(OS_PRIO_SELF);
        OSTaskSuspend(OS_PRIO_SELF);
        OSTimeDly();
        OSTimeDlyHMSM();
        /* USER CODE */
    }
}
```

Task States



Task Control Blocks (OS_TCB)

- When a task is created, it is assigned a task control block (OS_TCB)
- OS_TCBs reside in RAM
- Structure fields are organized to allow for structure packing



typedef struct os_tcb - 1/2

```
typedef struct os_tcb {  
    OS_STK *OSTCBStkPtr;           /* Pointer to current top of stack */  
  
#if OS_TASK_CREATE_EXT_EN > 0  
    void *OSTCBExtPtr;            /* Pointer to user definable data for TCB extension */  
    OS_STK *OSTCBStkBottom;        /* Pointer to bottom of stack */  
    INT32U OSTCBStkSize;          /* Size of task stack (in number of stack elements) */  
    INT16U OSTCBOpt;              /* Task options as passed by OSTaskCreateExt() */  
    INT16U OSTCBId;               /* Task ID (0..65535) */  
#endif  
  
    struct os_tcb *OSTCBNext;       /* Pointer to next TCB in the TCB list */  
    struct os_tcb *OSTCBPrev;       /* Pointer to previous TCB in the TCB list */  
  
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0) || (OS_SEM_EN > 0) || (OS_MUTEX_EN > 0)  
    OS_EVENT *OSTCBEVENTptr;        /* Pointer to event control block */  
#endif  
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0)  
    void *OSTCBMsg;                /* Message received from OSMboxPost() or OSQPost() */  
#endif
```

tick ISR uses
this links

typedef struct os_tcb - 2/2

```
#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (OS_MAX_FLAGS > 0)
#if OS_TASK_DEL_EN > 0
    OS_FLAG_NODE *OSTCBFlagNode; /* Pointer to event flag node */
#endif
    OS_FLAGS OSTCBFlagsRdy;      /* Event flags that made task ready to run */
#endif

INT16U OSTCBDly;                /* Nbr ticks to delay task or, timeout waiting for event */
INT8U OSTCBStat;                /* Task status */
INT8U OSTCBPrio;                /* Task priority (0 == highest, 63 == lowest) */

{ INT8U OSTCBX;                 /* Bit position in group corresponding to task priority (0..7) */
  INT8U OSTCBY;                 /* Index into ready table corresponding to task priority */
  INT8U OSTCBBitX;              /* Bit mask to access bit position in ready table */
  INT8U OSTCBBitY;              /* Bit mask to access bit position in ready group */

#if OS_TASK_DEL_EN > 0
    BOOLEAN OSTCBDelReq;        /* Indicates whether a task needs to be deleted */
    .OSTCBY = priority >> 3;
    .OSTCBBitY = OSMapTbl[priority >> 3];
    .OSTCBX = priority & 0x07;
    .OSTCBBitX = OSMapTbl[priority & 0x07];
} OS_TCB;
```

to make a task ready or wait fast (computed when a task is created or a task's priority is changed)

OSTCBStat

- OS_STAT_RDY (=0x00): Ready to run
- OS_STAT_SEM (=0x01): Pending on semaphore
- OS_STAT_MBOX (=0x02): Pending on mailbox
- OS_STAT_Q (=0x04): Pending on queue
- OS_STAT_SUSPEND (=0x08): Task is suspended
- OS_STAT_MUTEX (=0x10): Pending on mutual exclusion semaphore
- OS_STAT_FLAG (=0x20): Pending on event flag group
- OS_STAT_PEND_ANY (=0x3E): (OS_STAT_SEM | OS_STAT_MBOX | OS_STAT_Q | OS_STAT_MUTEX | OS_STAT_FLAG)

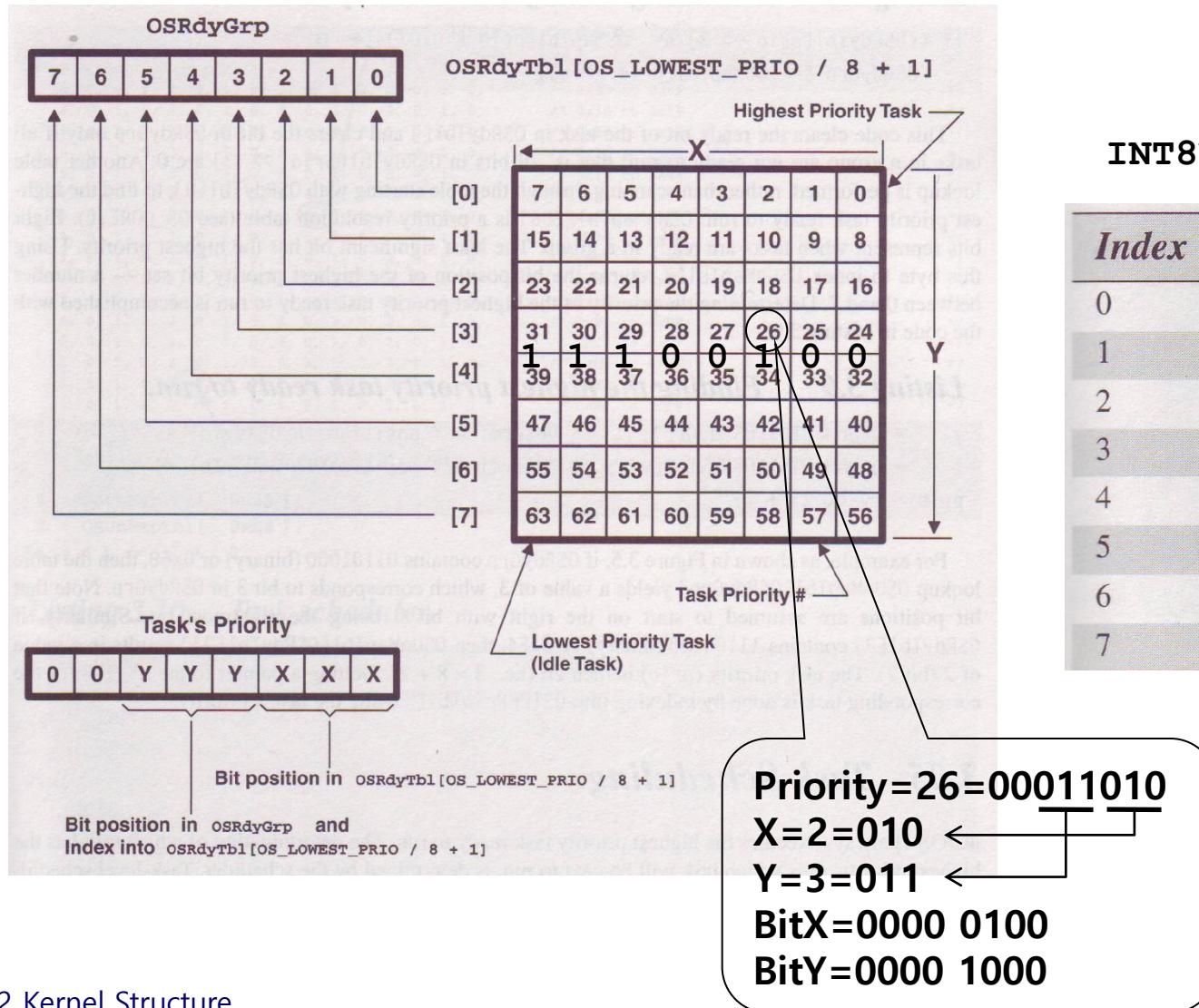
INT8U OS_TCBInit()

- OS_TCB is initialized by this function
- OSTaskCreate() or OSTaskCreateExt() calls this function
- OS_TCBInit() has seven arguments
 - INT8U prio: the task priority
 - OS_STK *ptos: a pointer to the top of stack
 - OS_STK *pbos: a pointer to the bottom of stack
 - INT16U id: task identifier
 - INT32U stk_size: total size of stack
 - void *pext: value of OSTCBExtPtr
 - INT16U opt: value of OSTCBOpt

Ready List

- Each task has unique priority level between 0 and OS_LOWEST_PRIO(=63) inclusive (totally 64 tasks)
- Ready list is maintained by two variables
 - INT8U OSRdyGrp
 - INT8U OSRdyTbl [8]

Ready List - Figure



INT8U OSMapTbl []

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

Operations on Ready List

Listing 3.7

Making a task ready to run.

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

**OSRdyGrp |= BitY
OSRdyTbl[Y] |= BitX**

Listing 3.8

Removing a task from the ready list.

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

**if((OSRdyTbl[Y] &= ~BitX) == 0)
OSRdyGrp &= ~BitY;**

**means no ready
task in the row**

Finding the Highest Priority Task

Listing 3.9 Finding the highest priority task ready to run.

```
y = OSUnMapTbl[OSRdyGrp]; /* Determine Y position in OSRdyTbl[] */  
x = OSUnMapTbl[OSRdyTbl[y]]; /* Determine X position in OSRdyTbl[Y] */  
prio = (y << 3) + x;
```

OSRdyGrp = 0x68

```
INT8U const OSUnMapTbl[] = {  
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 2, 0, 1, 0, /* 0x00 to 0x0F */  
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 0, 2, 0, 1, 0, /* 0x10 to 0x1F */  
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 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, 0, 1, 0, 2, 0, 1, 0, /* 0x50 to 0x5F */  
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 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] = 0xE4

OSRdyGrp = 0110 1000

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

OSRdyTbl[3] = 1110 0100

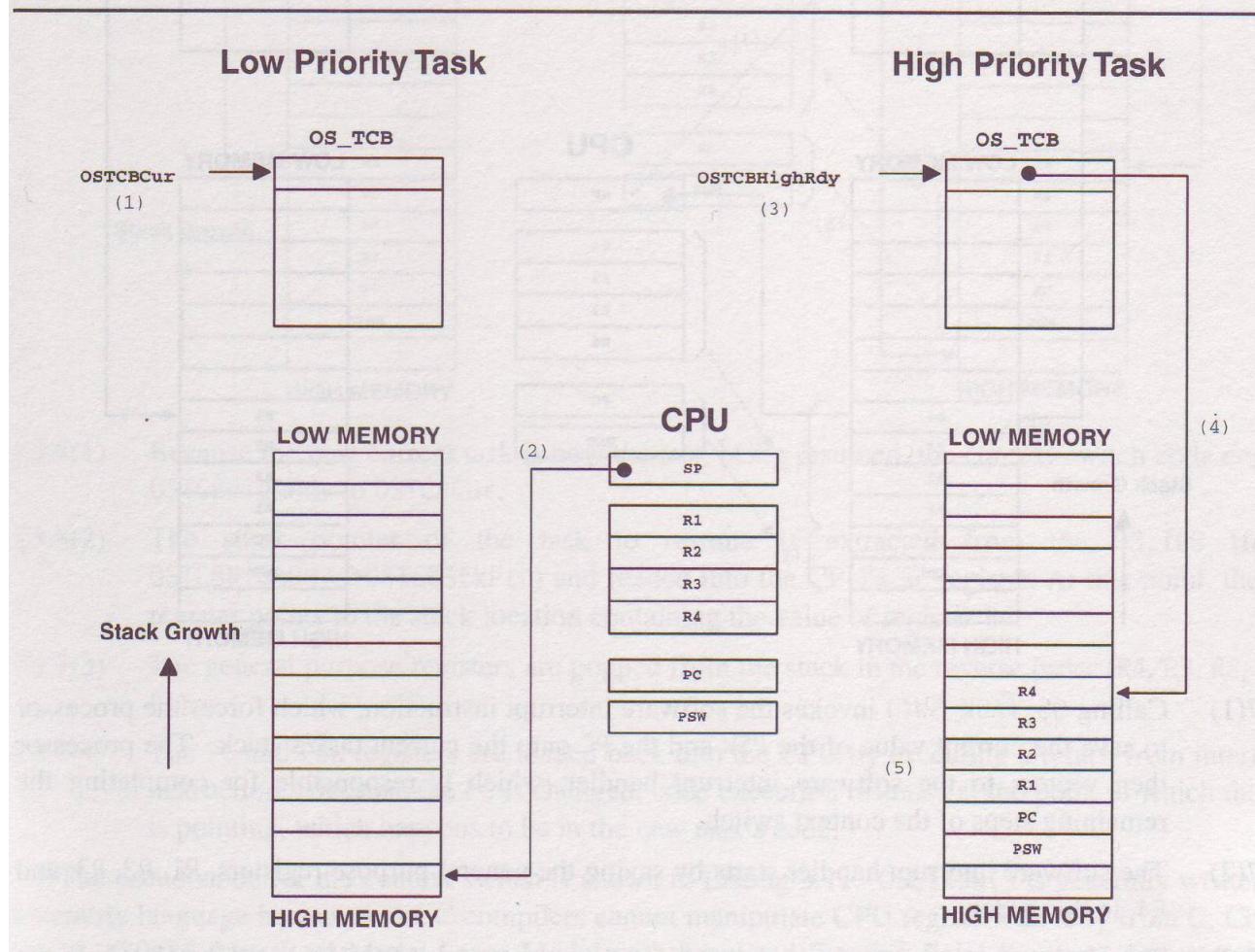
Task Scheduler - void OS_Sched()

```
void OS_Sched(void)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    INT8U y;

    OS_ENTER_CRITICAL();
                    /* if all ISRs done & not locked */
    if ((OSIntNesting == 0) && (OSLockNesting == 0)) {
        y = OSUnMapTbl[OSRdyGrp];
        OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
        if (OSPrioHighRdy != OSPrioCur) {
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
            OSCtxSwCtr++;           /* Increment context switch counter */
            OS_TASK_SW();           /* Perform a context switch */
        }
    }
    OS_EXIT_CRITICAL();
}
```

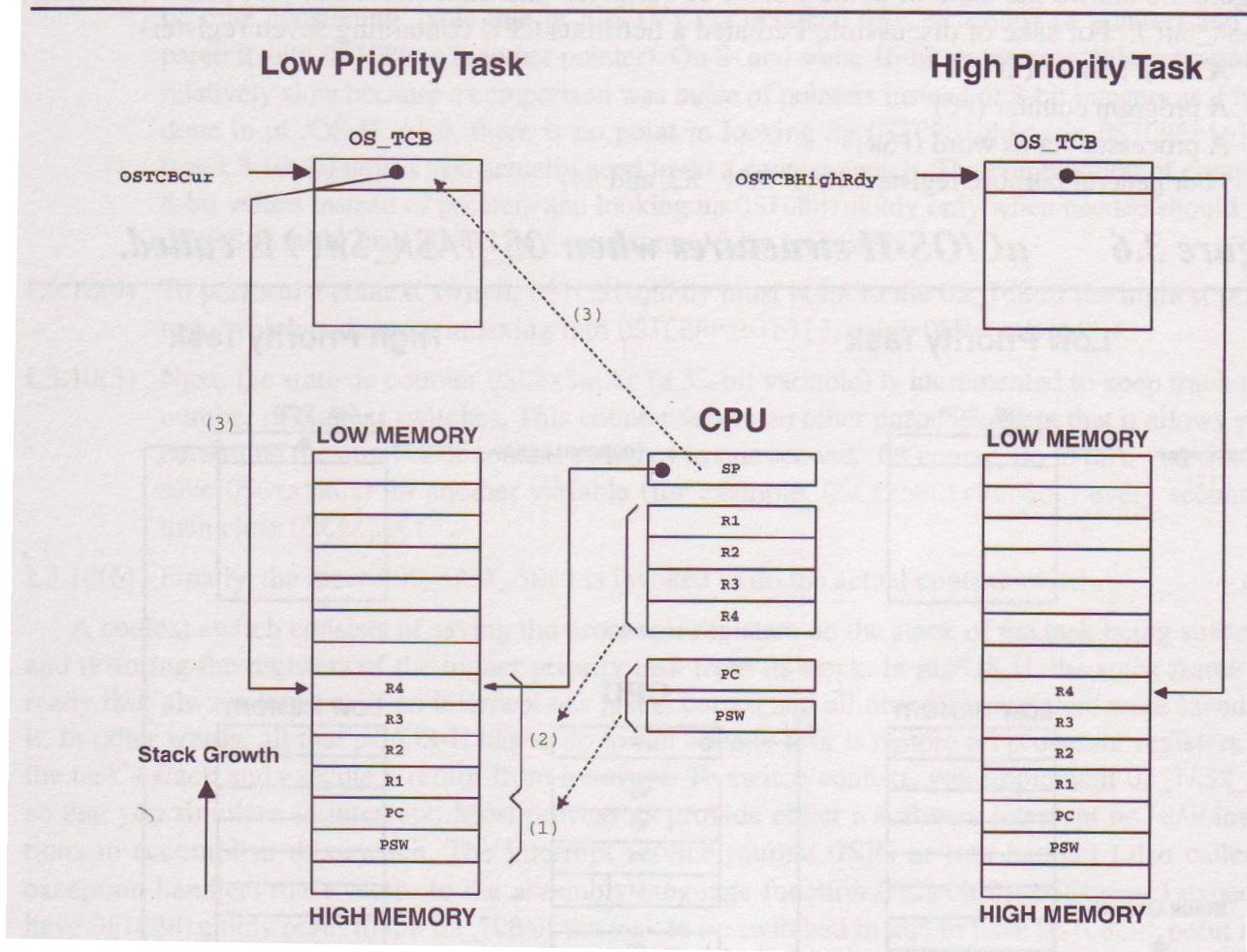
OS_TASK_SW() - 1/3

Figure 3.6 *μ C/OS-II structures when OS_TASK_SW() is called.*



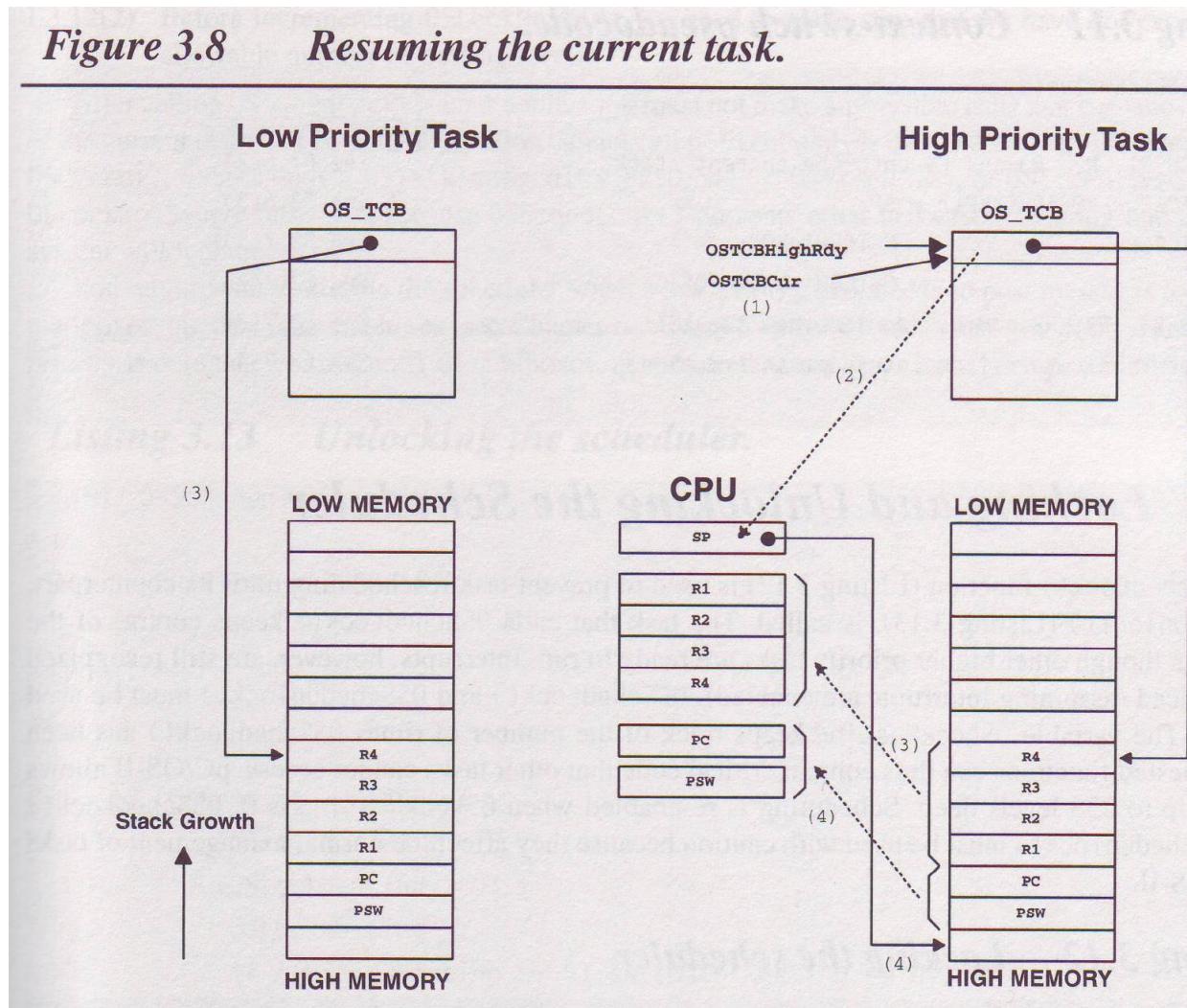
OS_TASK_SW() - 2/3

Figure 3.7 Saving the current task's context.



OS_TASK_SW() - 3/3

Figure 3.8 Resuming the current task.



Macro OS_TASK_SW()

```
#define OS_TASK_SW() asm("swi 2")
```

Listing 3.11 Context-switch pseudocode.

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

PUSH PC
PUSH PSW

POP PSW
POP PC

Locking the Scheduler - OSSchedLock()

Listing 3.12 Locking the scheduler.

```
void OSSchedLock (void)
{
    #if OS_CRITICAL_METHOD == 3
        OS_CPU_SR  cpu_sr;
    #endif

    if (OSRunning == TRUE) {
        OS_ENTER_CRITICAL();
        if (OSLockNesting < 255) {
            OSLockNesting++;
        }
        OS_EXIT_CRITICAL();
    }
}
```

**OSLockNesting > 0 when
scheduler is locked**

Unlocking the Scheduler - OSSchedUnlock()

Listing 3.13 Unlocking the scheduler.

```
void OSSchedUnlock (void)
{
#ifndef OS_CRITICAL_METHOD
    OS_CPU_SR  cpu_sr;
#endif

    if (OSRunning == TRUE) {
        OS_ENTER_CRITICAL();
        if (OSLockNesting > 0) {
            OSLockNesting--;
            if ((OSLockNesting == 0) && (OSIntNesting == 0))
                OS_EXIT_CRITICAL();
            OS_Sched();
        } else {
            OS_EXIT_CRITICAL();
        }
    } else {
        OS_EXIT_CRITICAL();
    }
}
```

Idle Task - void OS_TaskIdle(void *pdata)

Listing 3.14 The µC/OS-II idle task.

```
void OS_TaskIdle (void *pdata)
{
    #if OS_CRITICAL_METHOD == 3
        OS_CPU_SR cpu_sr;
    #endif

    pdata = pdata;
    for (;;) {
        OS_ENTER_CRITICAL();
        OSIdleCtr++;
        OS_EXIT_CRITICAL();
        OSTaskIdleHook();
    }
}
```

a lowest priority task
OS_LOWEST_PRI

32 bits counter

user supplied function

Statistic Task

- uC/OS-II provides a task with `OS_LOWEST_PRIO-1` that provides run-time statistic called `OS_TaskStat()` when `OS_TASK_STAT_EN` macro is 1
- This task executes every second and computes the percentage of CPU usage
- The computed value is placed in the variable `INT8U OSCPUUsage`
- If you use the statistic task, you must call `OSStatInit()` from the first and only task created before calling `OSStart()`

Initializing the Statistic Task

```
void main (void)
{
    OSInit();                  /* Initialize uC/OS-II
    /* Install uC/OS-II's context switch vector */
    /* Create your startup task (for sake of discussion, TaskStart())
    OSStart();                 /* Start multitasking
}

void TaskStart (void *pdata)
{
    /* Install and initialize µC/OS-II's ticker
    OSStatInit();              /* Initialize statistics task
    /* Create your application task(s) */
    for (;;) {
        /* Code for TaskStart() goes here! */
    }
}
```

void OSStatInit (void)

```
void OSStatInit (void)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

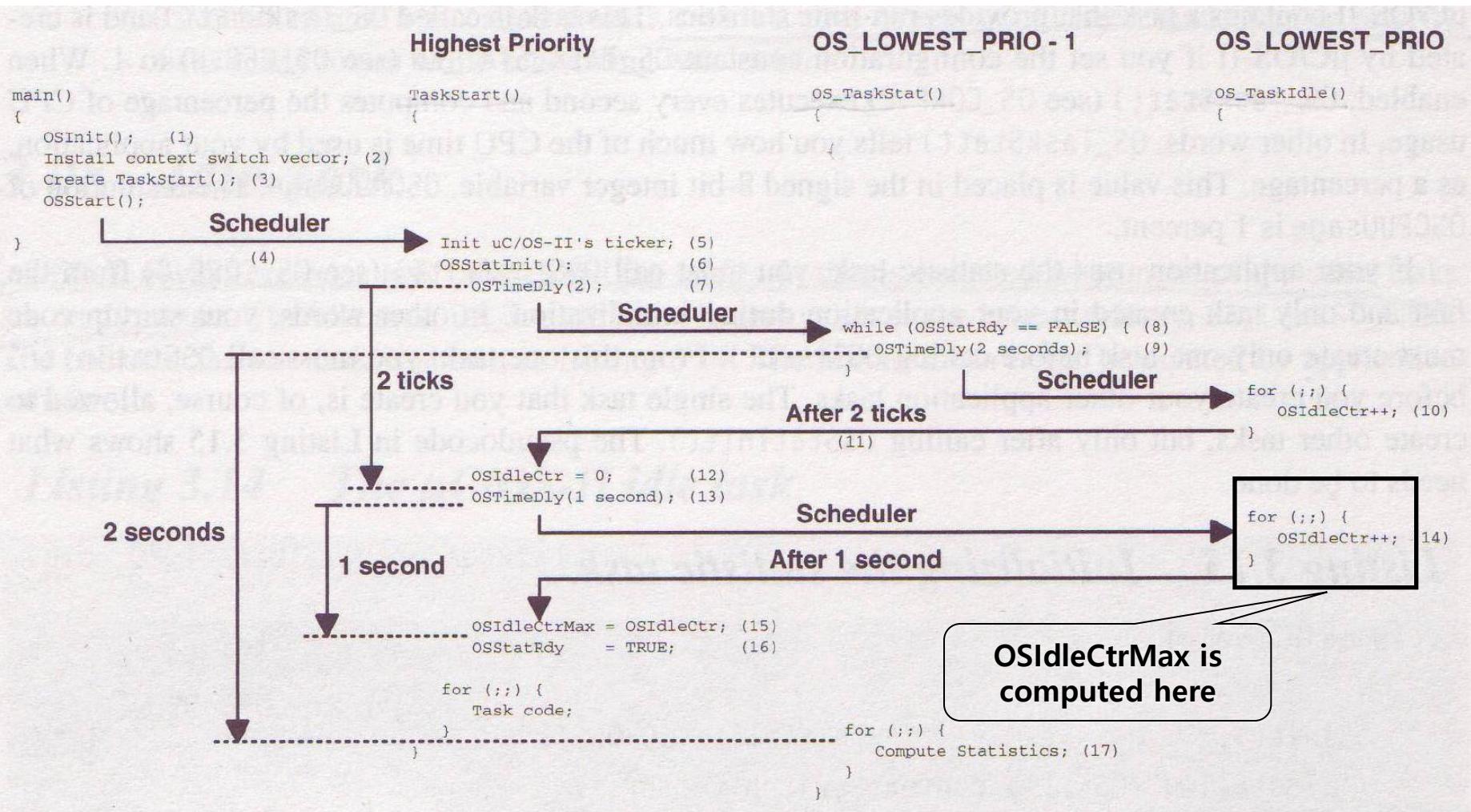
OSTimeDly(2);                                /* Synchronize with clock tick */
OS_ENTER_CRITICAL();
OSIdleCtr = 0L;                            /* Clear idle counter */
OS_EXIT_CRITICAL();
                                                /* Determine MAX. idle counter value for 1 second */
OSTimeDly(OS_TICKS_PER_SEC);
OS_ENTER_CRITICAL();
OSIdleCtrMax = OSIdleCtr;
OSStatRdy = TRUE;
OS_EXIT_CRITICAL();
}
```

Statistic Task

```
void OS_TaskStat (void *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    INT32U      run;
    INT32U      max;
    INT8S       usage;
    OSCPUUsage = 100*(1-(OSIdleCtr / OSIdleCtrMax));
    pdata = pdata;
    while (OSStatRdy == FALSE) {
        OSTimeDly(2 * OS_TICKS_PER_SEC);
    }
    max = OSIdleCtrMax / 100L;
```

```
        for (;;) {
            OS_ENTER_CRITICAL();
            OSIdleCtrRun = OSIdleCtr;
            run         = OSIdleCtr;
            OSIdleCtr   = 0L;
            OS_EXIT_CRITICAL();
            if (max > 0L) {
                usage = (INT8S)(100L - run / max);
                if (usage >= 0) {
                    OSCPUUsage = usage;
                } else {
                    OSCPUUsage = 0;
                }
            } else {
                OSCPUUsage = 0;
                max       = OSIdleCtrMax / 100L;
            }
            OSTaskStatHook();
            OSTimeDly(OS_TICKS_PER_SEC);
        }
    }
```

Computing OSIdleCtrMax



Interrupt Under uC/OS-II

Your ISR:

**must be written in
assembly**

Save all CPU registers;

**onto the current
task stack**

Call OSIntEnter() or, increment OSIntNesting directly;

if (OSIntNesting == 1) {

 OSTCBCur->OSTCBStkPtr = SP;

}

Clear interrupting device;

Re-enable interrupts (optional)

**if this is the first
level interrupt**

Execute user code to service ISR;

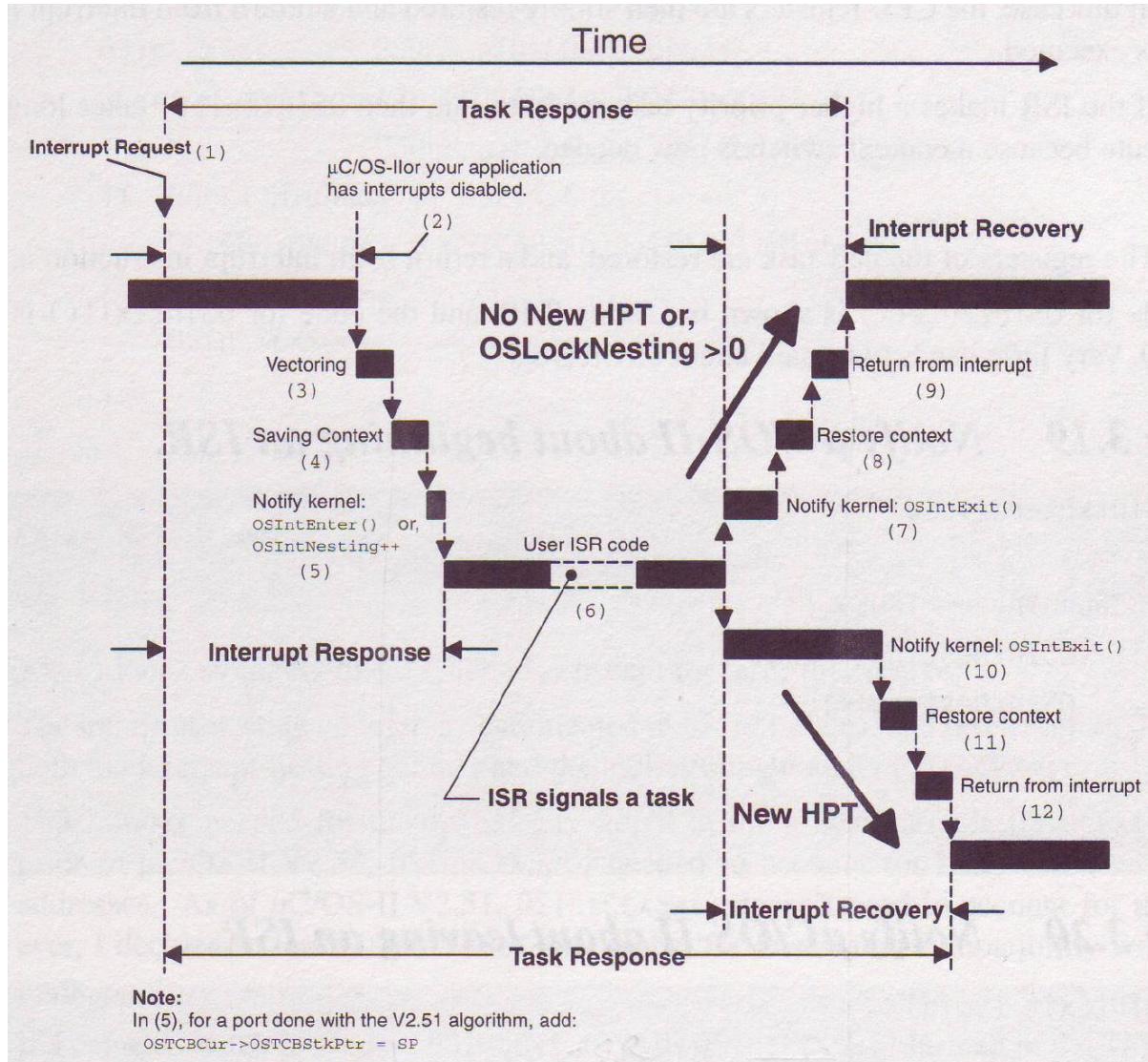
**if you want nested
interrupts**

Call OSIntExit();

Restore all CPU registers;

Execute a return from interrupt instruction;

Servicing an Interrupt



Tick ISR

- uC/OS-II require you provide a periodic time source to keep track of time delay and timeout
- A tick should be between 10 to 100 times per seconds or Hertz
- Tick ISR - void OSTimeTick()

Tick ISR - OSTimeTick()

```
void OSTimeTick (void)
{
    #if OS_CRITICAL_METHOD == 3
        OS_CPU_SR  cpu_sr;
    #endif

    OS_TCB     *ptcb;

    OSTimeTickHook();

    #if OS_TIME_GET_SET_EN > 0
        OS_ENTER_CRITICAL();
        OSTime++;
        OS_EXIT_CRITICAL();
    #endif

    if (OSRunning == TRUE) {
        ptcb = OSTCBLList;
        while (ptcb->OSTCBPrio != OS_IDLE_PRIO) {
            OS_ENTER_CRITICAL();
            if (ptcb->OSTCBDly != 0) {
                if (--ptcb->OSTCBDly == 0) {
                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0x00)
                        OSRdyGrp      |= ptcb->OSTCBBitY;
                    OSRdyTb1[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
                } else {
                    ptcb->OSTCBDly = 1;
                }
            }
        }
    }
}
```

why critical section?

increment tick counter

scan each task

check tick delay

make the task ready

suspended task

created task list

void TickTask(void *pdata)

- If you want a task-based tick ISR, make Tick ISR post a mail and supply TickTask with highest priority

```
void TickTask (void *pdata)
{
    pdata = pdata;
    for (;;) {
        OSMboxPend(...); /* Wait for signal from Tick ISR */
        OSTimeTick();
        OS_Sched();
    }
}
```

```
void OSInit(void)
```

- OSInit() should be called before calling any uC/OS-II function
- OSInit() initialize all uC/OS-II variables and data structures
- OSInit() creates the idle task and the statistic task

Figure 3.11 Variables and data structures after calling OSInit().

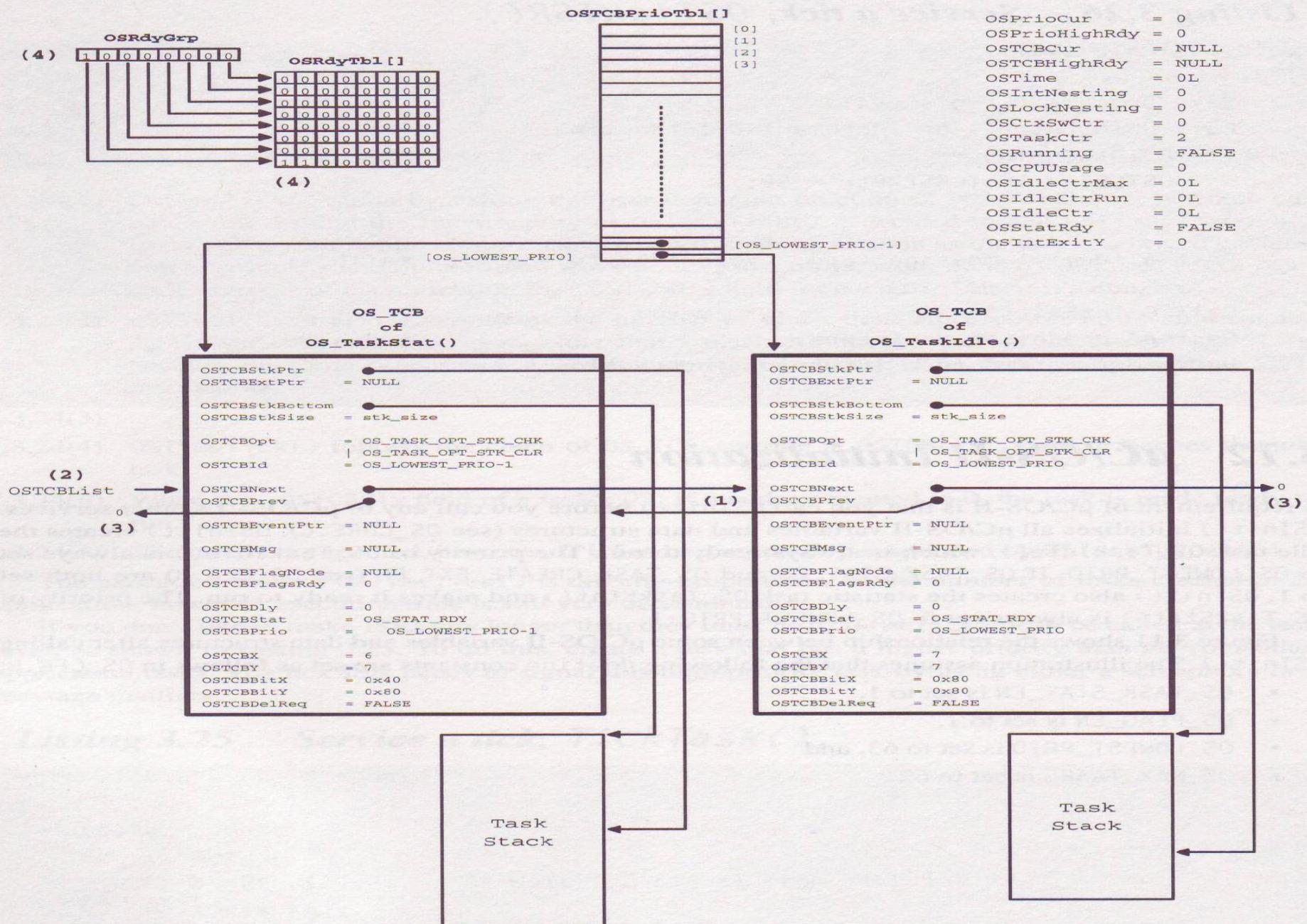
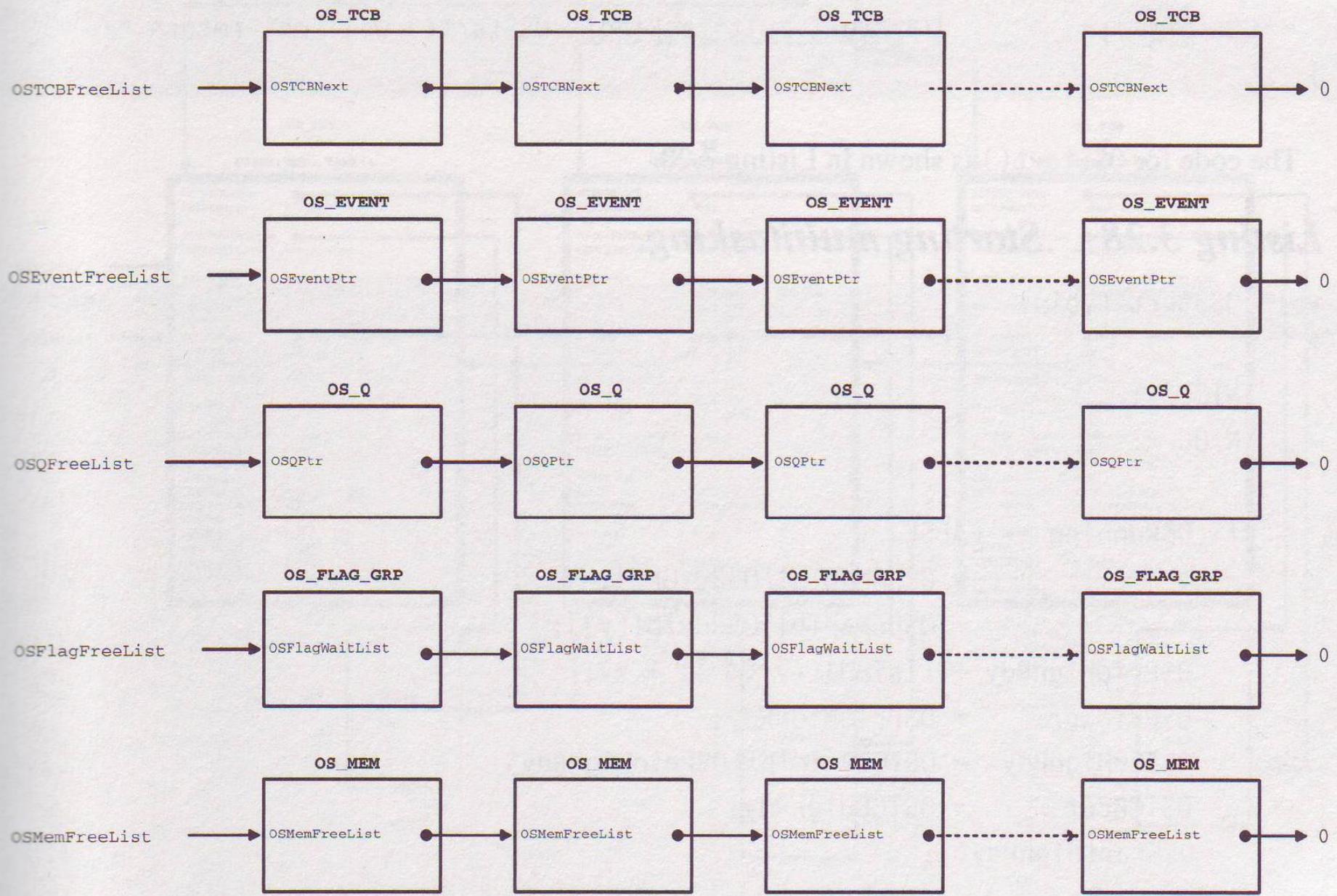


Figure 3.12 Free pools.



Starting uC/OS-II

```
void main (void)
{
    OSInit();          /* Initialize uC/OS-II */                      */
    .
    .
    Create at least 1 task using either OSTaskCreate() or OSTaskCreateExt();
    .
    .
    OSStart();         /* Start multitasking! OSStart() will not return */
}
```

Figure 3.13 Variables and data structures after calling OSStart().

