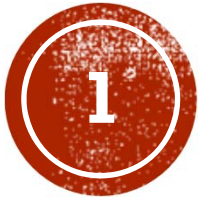
Kernel Structure of uC/OS-II

Da-Wei Chang
CSIE, NCKU



Code and Pictures on the slides are from
MicroC/OS-II, The Real-Time Kernel (2nd Edition), by Jean J. Labrosse

Outline

- Critical section
- Task
- Task scheduling
- Interrupt service routine
- Clock tick
- uC/OS-II initialization

Critical Section

- Shared resource access
- Race condition
- Mutual exclusion
  - Protect critical code from be entered simultaneously
- A typical implementation
  - Disable interrupt when entering
  - Enable interrupt when exiting
- Interrupt disable time should be kept minimum
  - One of the most important spec. for an RTOS

Critical Section

- 2 macros (in OS_CPU.h) for critical section
  - **OS_ENTER_CRITICAL**()
  - **OS_EXIT_CRITICAL**()
  - Code example

```
{
    ….
    OS_ENTER_CRITICAL();

    Code in critical section…

    OS_EXIT_CRITICAL();
}
```

Critical Section

- Do **NOT** block/sleep when you are in a critical section
  - e.g., do not call OSTimeDly() since no timer interrupts…
- Rule
  - Call OS services with interrupt enabled…

Critical Section

- 3 methods to implement OS_**ENTER_CRITICAL**() and OS_**EXIT_CRITICAL**()
  - Defined in the constant **OS_CRITICAL_METHOD**
  - Values: 1,2,3

- **Method 1**
  - Enter: disable interrupt
  - Exit: enable interrupt
  - Problem: always enable interrupt when exiting the critical section!!!
    - Maybe interrupt was disabled before entering the critical section….
  - So, we should **SAVE/RESTORE** interrupt status

Critical Section

- **Method 2**
  - Use stack to save/restore the interrupt status
  - The status can be restored

```
#difine OS_ENTER_CRITICAL
        asm("PUSH PSW")
        asm("DI")

#define OS_EXIT_CRITICAL()
        asm("POP PSW")

PSW = processor state word
```

Critical Section

- **Method 2**
  - Problem
    - When stack-relative address is used
      - Local variables are accessed via the OFFSET to the stack pointer (SP)
      - Compiler is not smart enough to know that the stack pointer has changed!!!
      - C code generally does not explicitly change SP

Critical Section

- **Method 3**
  - Obtain the processor status word (PSR)
  - Save PSR into a local variable

```
#difine OS_ENTER_CRITICAL
        cpu_sr = get_processor_psw();
        disable_interrupts();

#define OS_EXIT_CRITICAL()
        set_processor_psw(cpu_sr);
```

Note

1. No matter what method is used, users just call OS_**ENTER_CRITICAL**() and OS_**EXIT_CRITICAL**() !!
2. Change the method by changing the definition of the constant **OS_CRITICAL_METHOD**

Tasks

- Typically, a task is an infinite loop function

```
void YourTask (void *pdata)  // you can pass any types of data here
{
      for (;;)    {           // infinite loop
                              …Call one of uC/OS-II's services
      }
}
```

- A task can delete itself upon completion
  - Not actually deleted, but OS does not know the task anymore

```
void YourTask (void *pdata)  {
   ….
   OSTaskDel(OS_PRIO_SELF);
}
```

Tasks

- uC/OS-II supports up to 64 tasks
  - Two system tasks
    - Idle task → lowest priority
    - Statistic task → lowest priority - 1
  - Others can be used by applications…
- Each task has a priority
  - You assign it…
  - Low priority number = high priority
- Each priority has 0 or 1 task
  - Task priority = task ID (in the current version)

TASK STATE

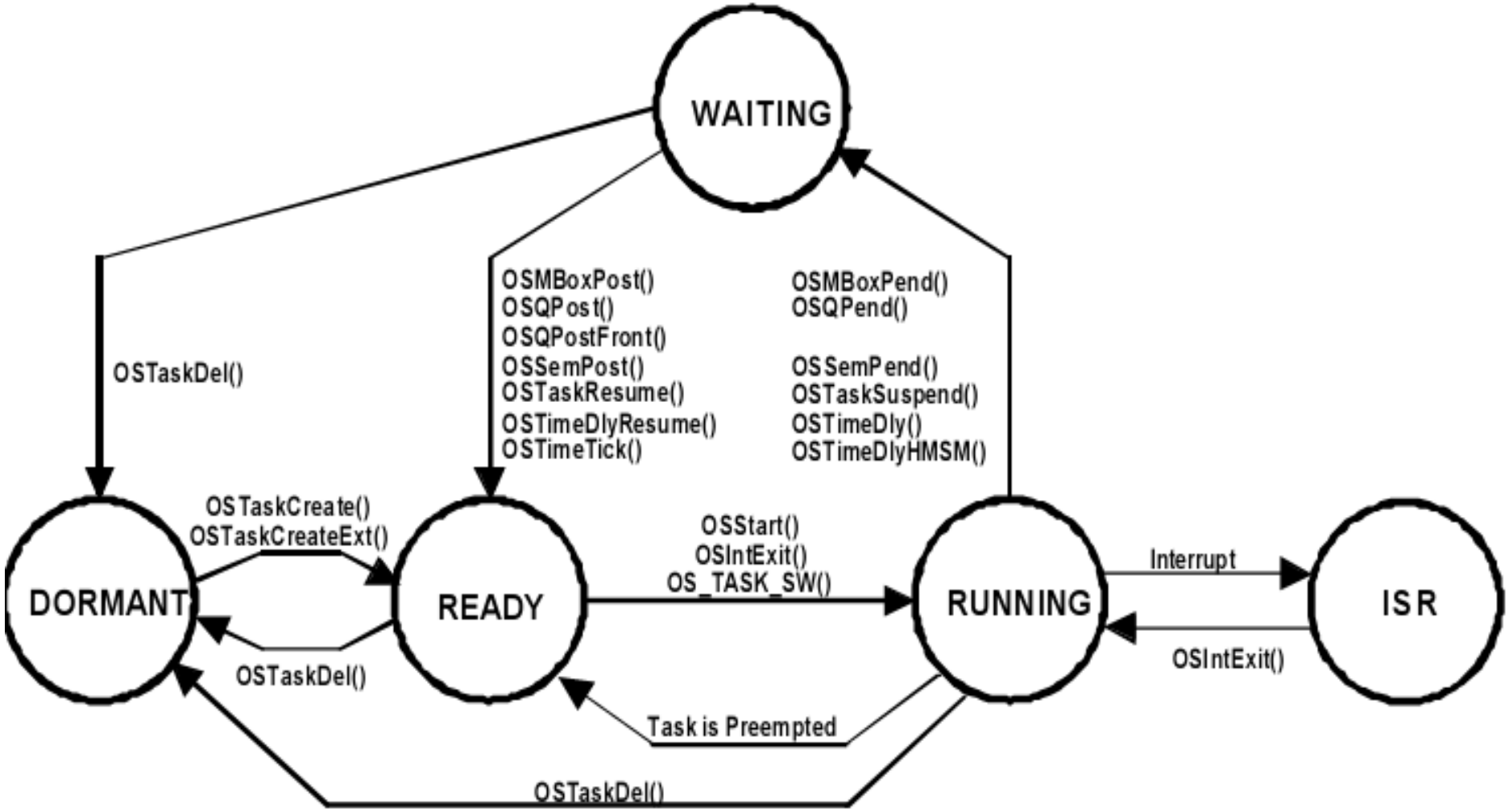


Figure 3-1, Task States

Task State

- At any given time, a task can be in one state
  - Dormant
    - Task resides in program space
    - Not been made available for the OS
  - Ready
    - OK to run
    - Task create, wakeup
  - Running
    - Own the CPU
    - The highest priority ready task
    - How many tasks can be in this state?

Task State

- At any given time, a task can be in one state
  - Waiting
    - Delay a period of time
    - Wait events that are triggered by other tasks

Task Control Blocks (TCB)

- Each task has a TCB after it has been created
- Data structure for managing that task
  - Reside in RAM
  - used for saving task state when context switch
  - the size of OS_TCB is determined by your application

Fields of a Task Control Block

```
typedef struct os_tcb {
   OS_STK       *OSTCBStkPtr;          // stack pointer in Top

#if OS_TASK_CREATE_EXT_EN
   void         *OSTCBExtPtr;          // for extend TCB
   OS_STK       *OSTCBStkBottom;       // stack point in Bottom
   INT32U       OSTCBStkSize;          // # of stack points, not size
   INT16U       OSTCBOpt;              // check, clear, floating
   INT16U       OSTCBId;               // task ID
#endif

   struct os_tcb *OSTCBNext;           // TCB → double-linked-list
   struct os_tcb *OSTCBPrev;

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
   OS_EVENT     *OSTCBEventPtr;
#endif
```

```
#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void        *OSTCBMsg;
#endif

INT16U       OSTCBDly;              //  the time waited for a event
INT8U        OSTCBStat;             //  OS_STATE
INT8U        OSTCBPrio;
INT8U        OSTCBX;                // To accelerate the process
INT8U        OSTCBY;                        //1  to run in ready state
INT8U        OSTCBBitX;                     //2  to run in waiting state
INT8U        OSTCBBitY;

#if OS_TASK_DEL_EN
    BOOLEAN      OSTCBDelReq;   // someone wants to kill me??
#endif

} OS_TCB;
```

Fields of a Task Control Block

- OSTCBStkPtr
  - top of the stack (where the push/pop happens)
  - uC/OS-II allows
    - Per-task stack
    - Different stack size
      - Save space
- OSTCBExtPtr
  - Extension to TCB
  - You can define your own extension
    - E.g., task name, task execution time, ctx switch count

Fields of a Task Control Block

- OSTCBStkBottom
  - Point to the bottom of the stack
- OSTCBStkSize
  - The size of the stack in elements
    - 100 32-bit elements = 400 bytes
    - 100 16-bit elements = 200 bytes
- OSTCBOpt
  - Hold the options specified when task creation
  - Three options
    - Stack check, stack clear, save floating point state
    - *described later…*

Fields of a Task Control Block

- OSTCBId
  - Task ID
- OSTCBNext / OSTCBPrev
  - Doubly-linked list for TCB
- OSTCBDly
  - Contains the number of ticks the task is allowed to wait…
- OSTCBStat, OSTCBPrio
  - State and priority
- OSTCBDelReq
  - Someone wants to kill me ?

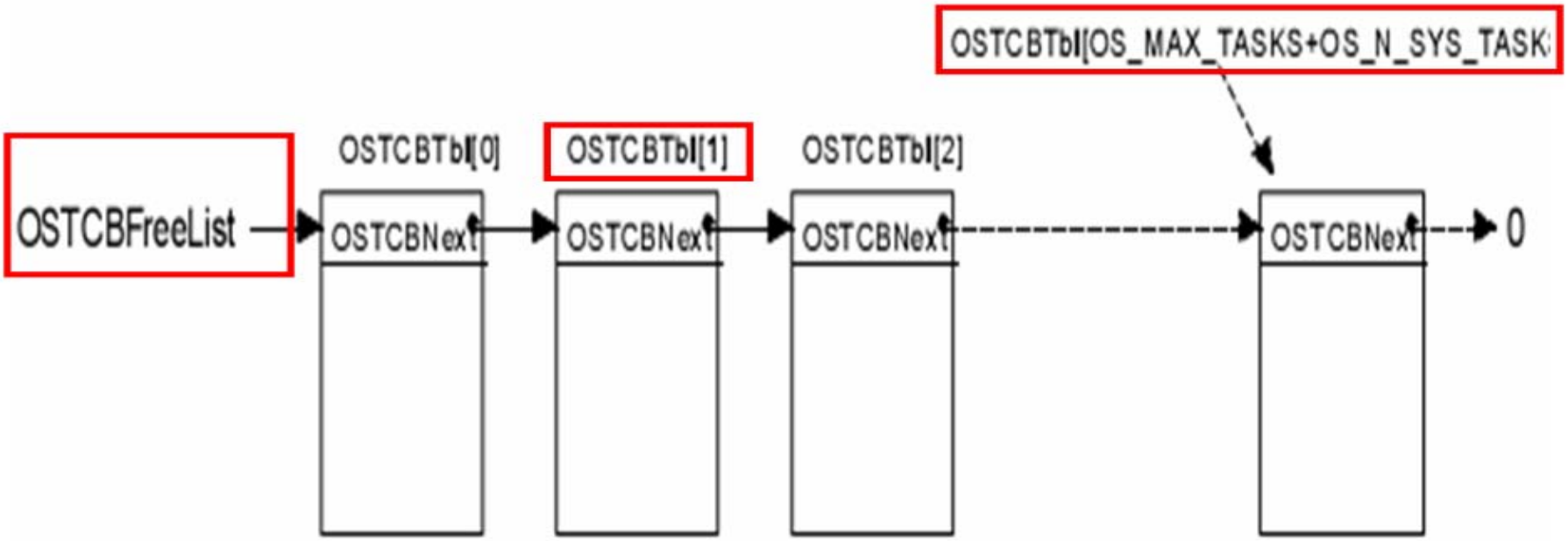
Task and TCB

- Lots of configuration options in TCB
  - Define the constants in OS_CFG.H
    - OS_MAX_TASKS : max # of **APPLICATION tasks**
      - Reduce task numbers, reduce memory requirements…
    - OS_N_SYS_TASKS : # of system tasks
      1. idle task
      2. statistic task

- All OS_TCBs are statically allocated in OSTCBTbl[]
  - At initialization, all TCBs are free

- OSTCBFreeList
  - a singly linked list of free TCB

TCB Free List



Get/Put a TCB from/to the free list

Questions:

1. Why singly linked list?
2. Get from where ? Put to where ?

TCB

- When to get a free TCB?
  - Task creation
- When to free a TCB ?
  - Task termination
- A TCB should be initialized before being used
  - OS_TCBInit()

```
INT8U  OS_TCBInit (INT8U prio, OS_STK *ptos, OS_STK *pbos, INT16U  id, INT32U
   stk_size, void *pext, INT16U opt ) {


#if OS_CRITICAL_METHOD == 3                    // allocate storage for PSW
   OS_CPU_SR  cpu_sr;
#endif
   OS_TCB    *ptcb;
   OS_ENTER_CRITICAL();          // WHY?
   ptcb = OSTCBFreeList;                              // get a free TCB from the free list


   if ( ptcb != (OS_TCB *)0 ) {                       // check TCB ?= NULL
      OSTCBFreeList       = ptcb->OSTCBNext;          // update TCB free list pointer
      OS_EXIT_CRITICAL();                             // exit critical when TCB is gotten
      ptcb->OSTCBStkPtr    = ptos;                    //Load Stack pointer
      ptcb->OSTCBPrio      = (INT8U) prio;            // load task priority
      ptcb->OSTCBStat      = OS_STAT_RDY;             //  ready to run
      ptcb->OSTCBDly       = 0;
```

```
#if OS_TASK_CREATE_EXT_EN > 0

    ptcb->OSTCBExtPtr    = pext                  // Store pointer to TCB extension
    ptcb->OSTCBStkSize   =  stk_size;            // Store stack size
    ptcb->OSTCBStkBottom = pbos;                 // bottom of stack
    ptcb->OSTCBOpt       =  opt;                 // task options
    ptcb->OSTCBId        = id;                   //task ID

#else                                            // Prevent from compiler warning
       pext              = pext;
       stk_size          = stk_size;
       pbos              = pbos;
       opt               = opt;
       id                = id;
#endif

#if OS_TASK_DEL_EN > 0
       ptcb->OSTCBDelReq    = OS_NO_ERR;
#endif
```

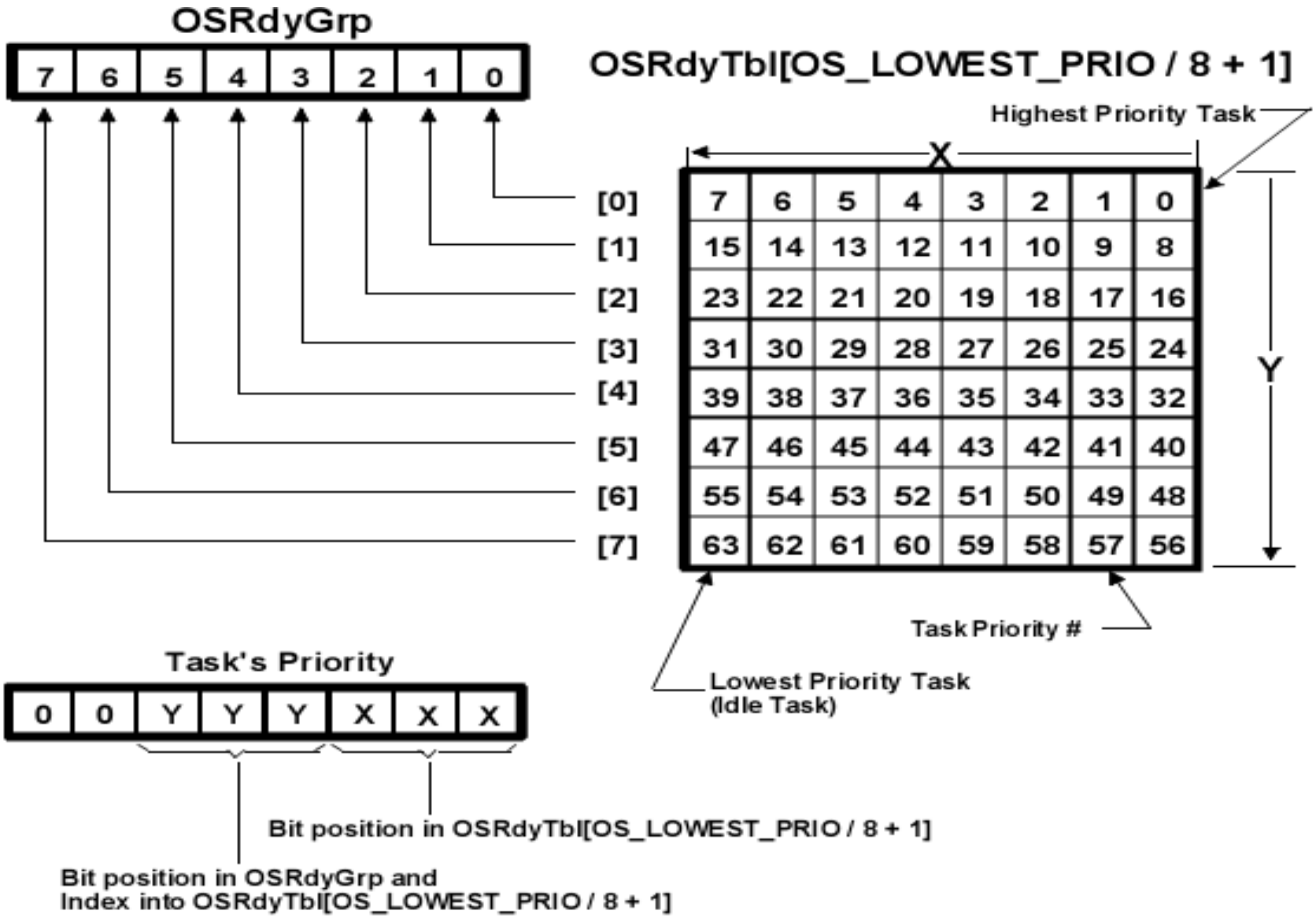
```
    ptcb->OSTCBY      = prio >> 3;          //for scheduling, described later…
    ptcb->OSTCBBitY   = OSMapTbl[ptcb->OSTCBY];
    ptcb->OSTCBX      = prio & 0x07;
    ptcb->OSTCBBitX   = OSMapTbl[ptcb->OSTCBX];

#if OS_EVENT_EN > 0                                  // for event related functions
    ptcb->OSTCBEventPtr  = (OS_EVENT *)0;
#endif

#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (OS_MAX_FLAGS > 0) &&
   (OS_TASK_DEL_EN > 0)
    ptcb->OSTCBFlagNode  = (OS_FLAG_NODE *)0;
#endif                                               // for event flags

#if (OS_MBOX_EN > 0) || ((OS_Q_EN > 0) && (OS_MAX_QS > 0))
    ptcb->OSTCBMsg    = (void *)0;          // No message is received now
#endif

#if OS_VERSION >= 204                  // allow to expand TCB
    OSTCBInitHook(ptcb);               // e.g. initial and save floating register
#endif                                   MMU register ….etc.
```

```
        OSTaskCreateHook(ptcb);                         // TaskCreate hook
        OS_ENTER_CRITICAL();
        OSTCBPrioTbl[prio] = ptcb;
        ptcb->OSTCBNext    = OSTCBList;                 // Link into the ALL TCB chain
        ptcb->OSTCBPrev    = (OS_TCB *)0;

        if ( OSTCBList  !=  (OS_TCB *)0 ) {
            OSTCBList->OSTCBPrev = ptcb;
        }
        OSTCBList              = ptcb ;

        // make the task ready to run, for scheduling…
        OSRdyGrp               |= ptcb->OSTCBBitY;
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

        OS_EXIT_CRITICAL();
        return (OS_NO_ERR);
    }
    // if ptcb == NULL, no free TCB can be used
    OS_EXIT_CRITICAL();
    return (OS_NO_MORE_TCB);
}
```

Ready List

- A list of READY-to-run tasks (TCBs)
  - Also called run queue
  - If a simple list is used: O(n)
- Not simply a list
  - **OSRdyGrp**
    - Tasks are grouped in OSRdyGrp
    - Each bit in OSRdyGrp indicates
      - at least one task in the group is ready to run !
  - **OSRdyTbl[]**
    - When a task is ready to run, the corresponding bit in the ready table is set
  - The size of OSRdyTbl[] depend on OS_LOWEST _PRIO.

Making a Task Ready-to-Run



Contents of OSMapTbl[]

index	value
0	0 0 0 0 0 0 0 1
1	0 0 0 0 0 0 1 0
2	0 0 0 0 0 1 0 0
3	0 0 0 0 1 0 0 0
4	0 0 0 1 0 0 0 0
5	0 0 1 0 0 0 0 0
6	0 1 0 0 0 0 0 0
7	1 0 0 0 0 0 0 0

Map a value (from 0 to 7) to the corresponding bitmap
e.g., k = 3 ➔ OSMapTbl[k] = 00001000

Making a Task Ready-to-Run

- The following code add the task to ready list

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

- Example

```
If priority == 44                       → 00101100
prio >> 3                               → 5 (row)
OSRdyGrp    |= OSMapTbl[5];             → 0X20
prio & 0x07                             → 4 (column)
OSRdyTbl[5]  |= OSMapTbl[4];            → 0X10
```

How to Accelerate ?

- 4 fields in the TCB can store the ready list related values

```
// for Y
OSTCBY      = priority >> 3;
OSTCBBitY   = OSMapTbl[priority >> 3];
//for X
OSTCBX      = priority & 0x07;
OSTCBBitX   = OSMapTbl[priority & 0x07];
```

- Do not need to compute the values every time
  - when the task is inserted into the ready list

Removing a Task from The Ready List

- Clear the corresponding bit in the X direction (row)
- If this makes the whole row become 0, clear Y

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

Finding the Highest Priority Ready Task to Run

- How to find the highest priority task?
  - Scanning the table from OSRdyTbl[0] ? **NO**
    - It requires O(n)
  - Another table lookup is used
    - E.g. OSRdyGrp = 0x68 → y = 3 , OSRdyTb1[3] = 0xE4 → x = 2
    - How to achieve this ? Check the code below

```
y    = OSUnMapTbl[OSRdyGrp];    // the highest (LSB) priority bit set
x    = OSUnMapTbl[OSRdyTbl[y]];
prio = (y << 3) + x;                    // (3 << 3) + 2   = 26
```

OSUnMapTbl[] : **map from a bitmap to a value**
- reverse function of the OSMapTbl

Finding the Highest Priority Ready Task to Run

```
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
```

Task Scheduling

- Select the highest priority task to run
- O(1)
  - Constant time, independent of the task #
- Task_level scheduling
  - performed by OS_Sched()
- ISR-level scheduling
  - handled by OSIntExit()

Task Scheduler

```
void OSSched (void)
{
   INT8U y;
   OS_ENTER_CRITICAL();
   // check if scheduling is disabled or OSSched is called from ISR
   if ( (OSLockNesting | OSIntNesting) == 0  ) {

      y  = OSUnMapTbl[OSRdyGrp];     // find the highest priority ready task
      OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);

      if (OSPrioHighRdy != OSPrioCur) {          // not current task
         OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
         OSCtxSwCtr++;                           // counter : # of context switch per sec
         OS_TASK_SW();                           // do context switch
      }
   }
   OS_EXIT_CRITICAL();
}
```

critical section

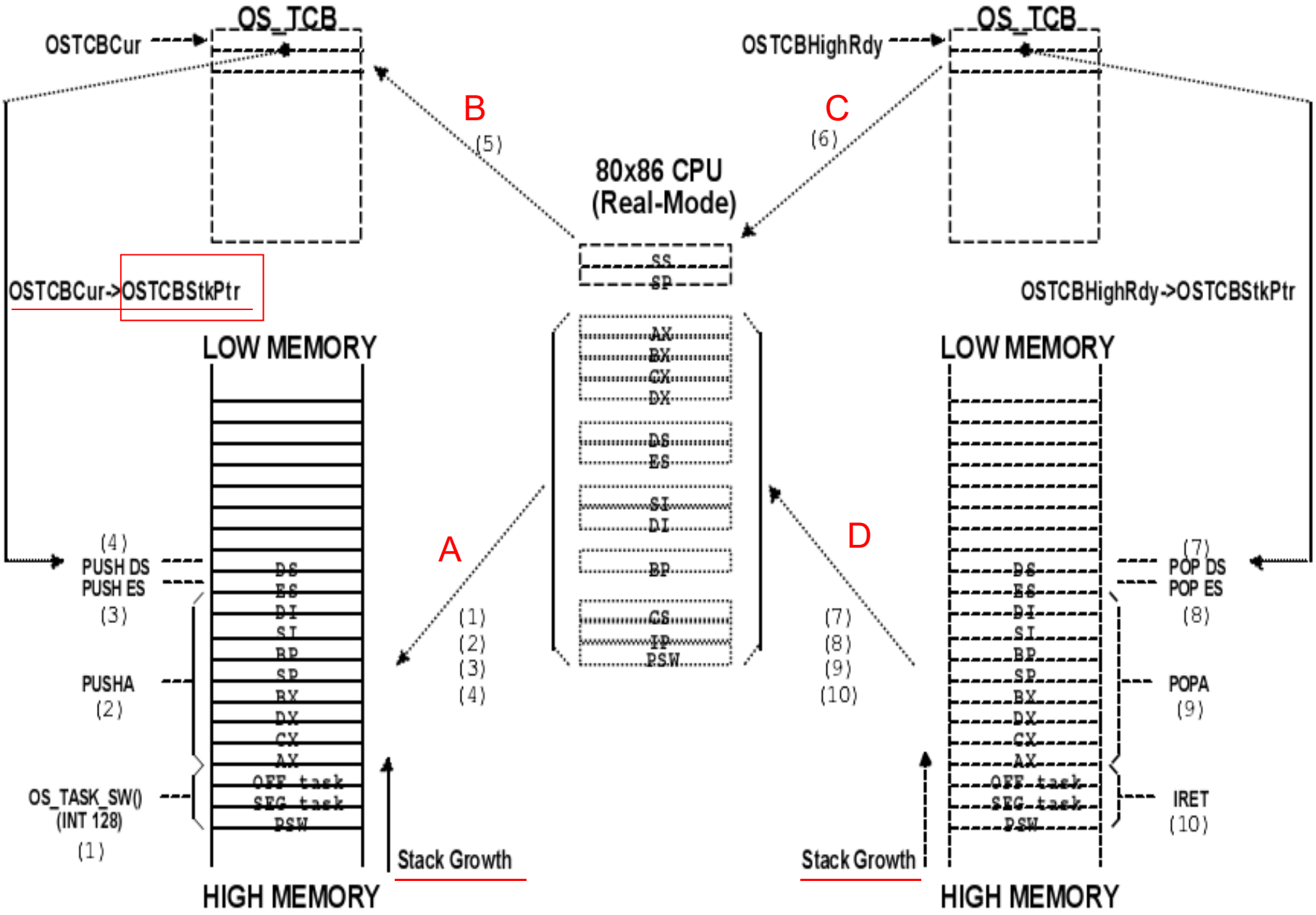
Switch from whom to whom ?
Next 2 slides....

Task Scheduler

- Enter critical section before finding xxxHIGHRDY
  - i.e., OSPrioHighRdy and OSTCBHighRdy
  - Prevents ISRs to set the new xxxHIGHRDY
- OS_Sched() could be written in assembly code to improve performance
  - But how about
    - Portability, and
    - Readability?

Task Level Context Switch

- Performed by **OS_TASK_SW()**
  - From ***OSTCBCur*** to ***OSTCBHighRdy***
- Steps
  - Save the context of the current task
    - Context: all the CPU registers
      - In ARM7: r0-r12, r14, cpsr, spsr (Note: ARM7 != ARMv7)
  - Save SP to current TCB's OSTCBStkPtr
  - Load SP from HIGHRDY TCB's OSTCBStkPtr
  - Restore the context of the HIGHRDY task
    - In uC/OS-II, the stack of a ready task looks as if an interrupt just occurred and the context was saved onto it


OS_TCB
OSTCBCur
OSTCBCur->OSTCBStkPtr
LOW MEMORY
HIGH MEMORY
Stack Growth
(4)
PUSH DS
PUSH ES
(3)
PUSHA
(2)
OS_TASK_SW()
(INT 128)
(1)
DS
ES
DI
SI
BP
SP
BX
DX
CX
AX
OFF_task
SEG_task
PSW
B
(5)
A
(1)
(2)
(3)
(4)
80x86 CPU
(Real-Mode)
SS
SP
AX
BX
CX
DX
DS
ES
SI
DI
BP
CS
IP
PSW
C
(6)
D
(7)
(8)
(9)
(10)
OS_TCB
OSTCBHighRdy
OSTCBHighRdy->OSTCBStkPtr
LOW MEMORY
HIGH MEMORY
Stack Growth
(7)
POP DS
POP ES
(8)
POPA
(9)
IRET
(10)

Locking & Unlocking the Scheduler

- Scheduler Lock
  - Used to temporarily prevent occurrence of rescheduling
  - NOT GOOD for real-time systems
    - But, sometimes you need it, and you know what you are doing
      - e.g., a low priority task wants to post messages to many msg queues without the interruption of other tasks…
  - The API: **OSSchedLock()**
    - No scheduling can happen
    - ISR still works
    - ISR can make a high priority task to become ready, BUT….rescheduling cannot happen…
  - Remember to call **OSSchedUnLock()**

OSSchedLock()

```
void OSSchedLock (void)
{
    if (OSRunning == TRUE) {      // true after calling OSStart()..
        OS_ENTER_CRITICAL();

            if (OSLockNesting < 255 )
                OSLockNesting++;                    // count of lock nesting

        OS_EXIT_CRITICAL();
    }
}
```

After the invocation of the OSSchedLock(),
do NOT call any service functions that can suspend yourself!!!

All other tasks are waiting for your invocation of OSSchedUnLock()!!

Scheduling is re-enabled when OSLockNesting == 0

OSSchedUnlock ()

```
void OSSchedUnlock (void)
{
    if (OSRunning == TRUE)
    {
        OS_ENTER_CRITICAL();
        if (OSLockNesting > 0)
        {
            OSLockNesting--;                                    //decrease the counter
            if ((OSLockNesting | OSIntNesting) == 0)
            {
                OS_EXIT_CRITICAL();
                OSSched();                  // someone may wait for you, check it!!
            } else
                OS_EXIT_CRITICAL();
        } else
            OS_EXIT_CRITICAL();
    }
}
```

Idle Task

- Executed when no other tasks are ready
- Always set to the lowest priority
  - OS_LOWEST_PRIO
- Can never be deleted by application
  - CPU needs something to run….
  - In modern CPUs, you can usually switch to the low power mode
- Can never be suspended
  - CPU needs something to run….
- OSTaskIdle (void *pdata)

OSTaskIdle ()

```
void OSTaskIdle (void *pdata)
{
    pdata = pdata;                  // for compiler warning…
    for (;;) {
        OS_ENTER_CRITICAL();        // for the reason mentioned in the next line
          OSIdleCtr++;              // may be not atomic in 8-bit/16-bit systems
        OS_EXIT_CRITICAL();
        OSTaskIdleHook();           //call the Idle hook
    }
}
```

- **OSIdleCtr**
  - can be used to compute the CPU utilization
  - will be used by the statistic task
- **OSTaskIdleHook**()
  - for you to do something when the system becomes idle
    - e.g., can make CPU enter the low-power mode
  - do not suspend in this function!!
    Otherwise, scheduler may find no task to run

Statistics Task

- Computes the CPU usage (**re-compute for each second**)
  - stored in ***OSCPUUsage***
- OS_TASK_STAT_EN = 1

$$\text{Usage} = 100 - \frac{100 * \text{OSIdleCtr}}{\text{OSIdleCtrMax}}$$

- * 100 makes overflow easier for fast CPU, so …

$$\text{Usage} = 100 - \frac{\text{OSIdleCtr}}{\text{OSIdleCtrMax} / 100}$$

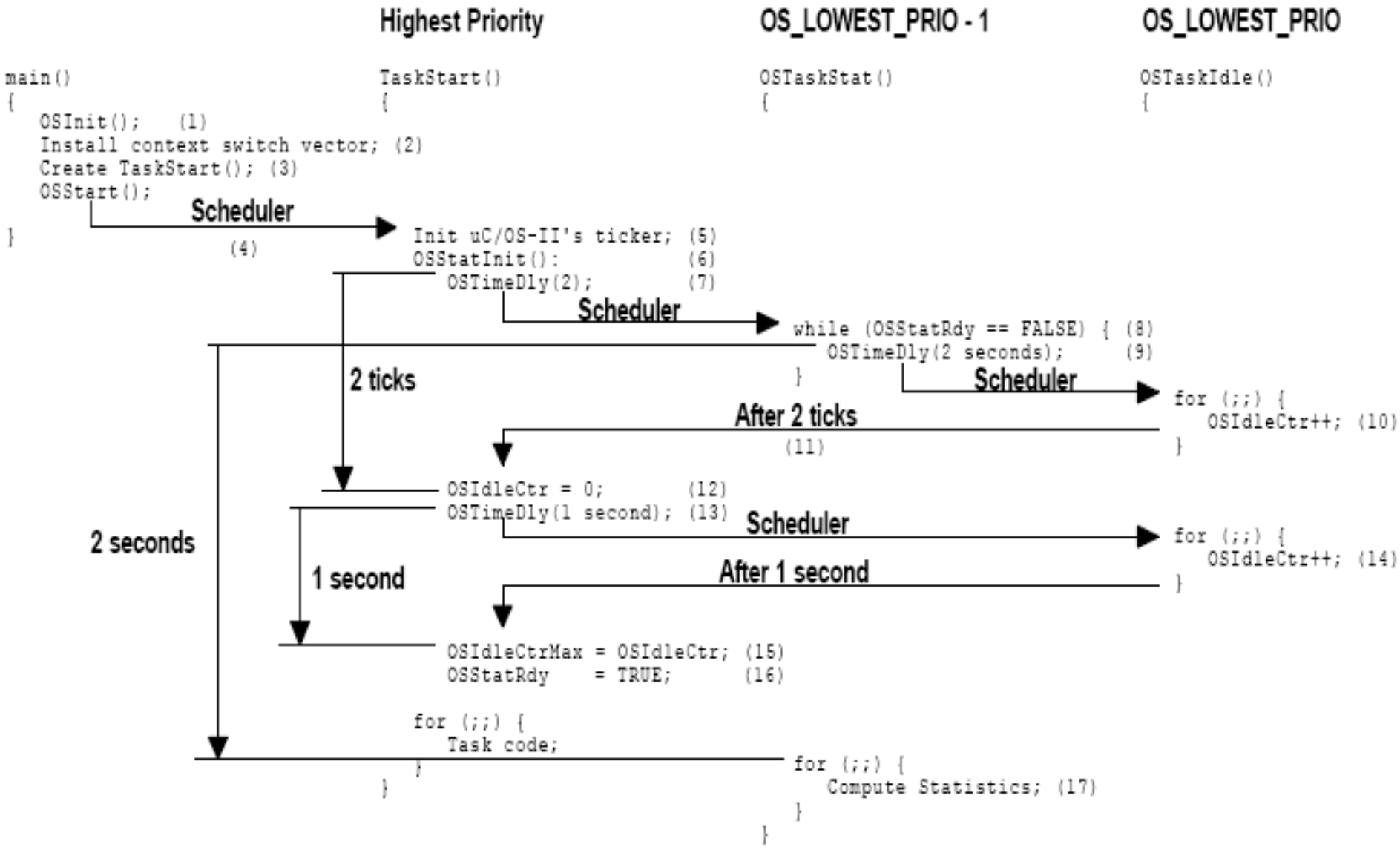
Statistic Task Initialization

- Using the statistic task
  - Create **only one** task first
  - Call OSStatInit() from the task
    - **For computing the OSIdleCtrMax!**
    - Make sure there are **only 3 tasks** when doing OSStatInit()
      - Your startup task, statistic task, and idle task
  - After the OSStatInit(), the OSIdleCtrMax has been computed, you can then create your other tasks

Code

```
void main (void)  {
    OSInit();                          // initialize uC/OS-II
    // Install uC/OS-II's context switch vector
    // create your startup task,  e.g. TaskStart()
    OSStart();        // Start multitasking: priority of TaskStart() > OS_TaskStat() >  OS_TaskIdle()
}

void TaskStart (void *pdata)  {
    //Install and initialize uC/OS-II's ticker
    OSStatInit();     // Initialize statistics task

    // Computation of OSIdleCtrMax is done!
    // Now, you can create your other tasks
    for (;;) {   ….}
}
```

Statistic Task Initialization



```
void OSStatInit (void)
{
    OSTimeDly(2);                          //wait 2 ticks

    OS_ENTER_CRITICAL();
    OSIdleCtr    = 0L;
    OS_EXIT_CRITICAL();

    OSTimeDly(OS_TICKS_PER_SEC);    //wait 1 second

    OS_ENTER_CRITICAL();
    OSIdleCtrMax = OSIdleCtr;
    OSStatRdy    = TRUE;
    OS_EXIT_CRITICAL();
}
```

allow **OSIdleCtr** to be accumulated in this one second

Computation of the OSIdleCtrMax is finished

```
void OSTaskStat (void *pdata) {          /* Compute the CPU usage (see slide 46) */
    INT32U run;     INT8S  usage;
    pdata = pdata;
    while (OSStatRdy == FALSE) {
        OSTimeDly(2 * OS_TICKS_PER_SEC);     //wait 2 seconds
    }
    OSIdleCtrMax /= 100uL;
    for (;;) {
        OS_ENTER_CRITICAL();
        OSIdleCtrRun = OSIdleCtr;          // Obtain the idle counter for the past second
        OSIdleCtr    = 0L;                 // Reset the idle counter for the next second
        OS_EXIT_CRITICAL();
        OSCPUUsage   = (INT8U)(100uL - OSIdleCtrRun / OSIdleCtrMax);
        OSTaskStatHook();                              // Invoke user definable hook
        OSTimeDly(OS_TICKS_PER_SEC);  // Accumulate OSIdleCtr for the next second
    }
}
```